ВЛАДСТОН ФЕРРЕЙРА ФИЛО, МОТО ПИКТЕТ

ТЕОРЕТИЧЕСКИЙ МИНИМУМ ПО COMPUTER SCIENCE

СЕТИ, КРИПТОГРАФИЯ И DATA SCIENCE

COMPUTER SCIENCE UNLEASHED

HARNESS THE POWER OF COMPUTATIONAL SYSTEMS

Wladston Ferreira Filho
Moto Pictet

Las Vegas

ВЛАДСТОН ФЕРРЕЙРА ФИЛО, МОТО ПИКТЕТ

ТЕОРЕТИЧЕСКИЙ МИНИМУМ ПО COMPUTER SCIENCE

СЕТИ, КРИПТОГРАФИЯ И DATA SCIENCE

ПИТЕР®

Санкт-Петербург • Москва • Минск

2022

ББК 32.973.23-018
УДК 004.3
Ф54

Феррейра Фило Владстон, Пиктет Мото

Ф54 Теоретический минимум по Computer Science. Сети, криптография и data science. — СПб.: Питер, 2022. — 288 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-2945-4

Хватит тратить время на занудные учебники! Это краткое и простое руководство предназначено для читателей, не заботящихся об академических формальностях.

Большинство технологических прорывов нашей эпохи происходят в цифровой среде, создаваемой программистами. Ученые-компьютерщики объединяют различные области исследований и расширяют возможности этого нового мира. Чтобы научиться плавать в океане информации, необходимо разбираться в основах сетевых технологий, криптографии и науке о данных.

Вы узнаете, как эффективно манипулировать данными, освоите машинное обучение и современные концепции безопасности.

Раскройте мощь Computer Science и станьте гуру цифровой эпохи!

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.23-018
УДК 004.3

ISBN 780-997316032 англ.

ISBN 978-5-4461-2945-4

Оглавление

Нашим друзьям Кристофу и Матеусу — один из них поспорил, что мы закончим эту книгу к концу года.

Computer science имеет много общего с физикой. Обе науки о том, как мир устроен на довольно фундаментальном уровне.

Различие в том, что в физике вы изучаете, как устроен мир, а в компьютерных науках вы мир создаете. В математике, как и в программировании, все работает, пока система самосогласована. У вас может быть система уравнений, где три плюс три равно двум. Возможно все.

Линус Торвальдс,
из объяснения, откуда пошла его любовь к компьютерам

ПРЕДИСЛОВИЕ

> Мне никогда не нравился термин computer science, и главная причина состоит в том, что ничего подобного не существует. Computer science — это сборная солянка слабо связанных между собой областей, по воле случая оказавшихся рядом, прямо как Югославия.
>
> *Пол Грэм*

Большинство технологических прорывов нашей эпохи происходят в новом цифровом мире, создаваемом программистами. Ученые-компьютерщики объединяют различные области исследований и расширяют возможности этого нового мира. В книге мы рассмотрим основы некоторых из этих областей, включая сетевые технологии, криптографию и науку о данных.

Начнем с истории о том, как можно связать два компьютера для обмена информацией, а потом обсудим все, что происходило до эпохи расцвета

Рис. П.1. «Данные — это новая нефть», Амит Дангле и Ивано Нардаччионе

электронной почты и интернета. Погрузимся в криптографию и поймем, как защищают межсетевые и прочие системы, имеющие дело с личными данными. Затем узнаем, как программисты манипулируют данными и как научить машины прогнозировать будущее.

Надеемся, что через эти истории вы познакомитесь с важными концепциями, которые принесут пользу как программистам, так и техническим энтузиастам. Мы хотели бы рассказать все, что нужно знать новичкам, чтобы быстрее освоить работу в Сети, разобраться в безопасности и науке о данных, без излишней академической строгости, которая иногда делает эти темы попросту невыносимыми.

Эта книга появилась на свет благодаря всем тем, кто поддерживал предыдущую — *Computer Science Distilled*[1]. Мы написали свою первую книгу, чтобы разъяснить фундаментальные принципы computer science. Многие читатели с таким энтузиазмом просили еще, что мы вернулись к работе. Так что теперь изучим новые миры, которые computer science позволила создать.

ТАК ДЛЯ КОГО ЭТА КНИГА

Если вы начинающий программист, эта книга для вас. Опыта программирования не требуется, здесь объясняются идеи и механизмы: мы хотим, чтобы вы узнали, как работают всякие классные штуки. Если вы хотите понять, как устроен интернет, как хакеры атакуют компьютерные системы или почему данные — это золото XXI века, смело читайте дальше. А тем, кто уже изучал computer science, эта книга позволит закрепить знания и опыт.

ОТ ИЗДАТЕЛЬСТВА

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

---

[1] *Фило В. Ф.* Теоретический минимум по Computer Science. Все, что нужно программисту и разработчику. — СПб.: Питер, 2020.

БЛАГОДАРНОСТИ

Мы очень благодарны всем за поддержку. Хотели бы поблагодарить Абнера Марчиано, Андре Ламберта, Кайо Магно, Карлотту Фабрис, Дамиана Хирша, Даниэля Стори, Эдуардо Барбозу, Габриэля Пикте, Гильерме Маттара, Жаклин Уилсон, Леонардо Конегундеса, Ллойда Кларка, Майкла Уллмана, Рафаэля Алмейду, Рафаэля Виотти и Руана Бидарта. Наконец, спасибо Клэр Мартин, нашему корректору, и Педро Нетто, иллюстратору, за то, что они помогли сделать книгу лучше.

Да создадите вы многие миры,
Влад и Мото

Глава 1

СВЯЗИ

> Это полностью распределенная система, в которой нет централизованного управления. Единственная причина, по которой она работает, состоит в том, что все решили использовать один и тот же набор протоколов.
>
> *Винт Серф*[1]

Люди жаждут связей, и цифровая революция позволила нам быть более связанными, чем когда-либо прежде. Интернет дал миллиардам людей беспрецедентную экономическую и политическую свободу, а также мощные средства контроля и господства. При этом подавляющее большинство из нас понятия не имеет о его внутреннем устройстве.

Специалисты, способные запрограммировать компьютеры для работы в интернете, — это авангард цифровой революции. В этой главе вы узнаете, как работает интернет, что позволит вам влиться в группу просветленных. Вы научитесь:

соединять компьютеры в сеть;

объединять сети, используя **интернет**-протокол;

находить получателя по его интернет-**адресу**;

[1] Винтон Грей Серф — американский ученый в области теории вычислительных систем, один из разработчиков стека протоколов TCP/IP. — *Примеч. ред.*

находить **маршрут** к нему через интернет;

передавать данные между удаленными приложениями.

До появления интернета связь между двумя сторонами требовала прямого физического подключения. В 1950-х годах у каждого телефона был провод, ведущий прямо к центральной станции. Чтобы звонок прошел, оператору нужно было физически соединить провода двух телефонов. Для междугородных звонков между удаленными станциями прокладывались провода и нескольким операторам в разных местах приходилось физически формировать цепочку проводов, соединяющую два телефона.

Всему этому интернет положил конец. Теперь мы не формируем физические соединения из проводов, связываясь напрямую. Информация передается шаг за шагом по цепочке связанных устройств, пока не достигнет места назначения. Это устраняет необходимость в операторах связи и централизованной координации. Провода больше не ограничены обслуживанием единственного соединения — множество одновременных соединений могут использовать один и тот же провод. Это делает глобальную связь мгновенной, дешевой и доступной.

Однако современные сетевые технологии сложнее, чем телефония тех дней. Они состоят из множества слоев, и каждый располагается поверх предыдущего. Давайте рассмотрим, как устанавливаются связи на этих разных уровнях, начиная с самого базового.

1.1. КАНАЛЬНЫЙ УРОВЕНЬ

Прямое соединение между двумя компьютерами достигается посредством **среды передачи**: физического канала, по которому проходят сигналы. Это может быть медный провод, по которому проходит электрический ток, оптоволоконный кабель, направляющий свет, или воздух, переносящий радиоволны. У каждого подключенного компьютера имеется **сетевой интерфейс**, позволяющий отправлять и получать сигналы в среде передачи. Например, в мобильных телефонах есть радиочип и антенна для обработки радиосигналов, распространяющихся по воздуху (рис. 1.1).

Рис. 1.1. Соединение устанавливается между двумя сетевыми интерфейсами, если они совместно используют среду передачи и согласованные правила связи

Для связи сетевые интерфейсы должны согласовать правила, которым нужно следовать при отправке и получении сигналов. Такой набор правил называется **канальным уровнем**.

Когда среда соединяет два компьютера *исключительно*, мы говорим, что они поддерживают соединение «точка — точка», а их канальный уровень основан на самом базовом наборе правил: **P**oint-to-**P**oint-**P**rotocol **(PPP)**. Он просто гарантирует, что два компьютера могут идентифицировать друг друга и без потерь обмениваться данными.

Однако подключенные компьютеры не всегда могут воспользоваться такой исключительной связью. Чаще они должны делить среду передачи с несколькими другими компьютерами.

ОБЩИЕ СВЯЗИ

Один из способов связать компьютеры в офисе — подключить каждый из них проводом к сетевому концентратору, или хабу. Концентратор физически соединяет все идущие к нему провода, поэтому сигнал, отправленный одним компьютером, будет обнаружен *всеми* остальными! Это происходит и с вашим домашним Wi-Fi, поскольку одна и та же радиочастота используется всеми подключенными устройствами (рис. 1.2). Связь может стать запутанной, если все они начнут использовать среду одновременно.

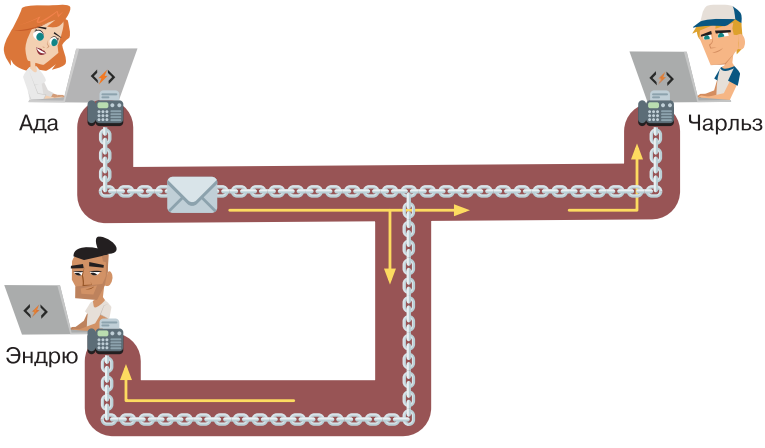


Рис. 1.2. Сообщение, отправленное по общей связи, получат все

Канальный уровень содержит набор правил, определяющих то, как компьютеры должны совместно использовать свои средства связи, и назван «управление доступом к среде» (**M**edium **A**ccess **C**ontrol, **MAC**). Эти правила решают две основные проблемы.

КОЛЛИЗИИ Если два компьютера одновременно отправляют сигнал через одну и ту же среду, возникающие в результате помехи искажают обе передачи. Такие ситуации называются **коллизиями**. Аналогичная проблема возникает, когда группа людей одновременно разговаривает друг с другом, так что невозможно понять, что именно говорит каждый (рис. 1.3, 1.4).

Есть способы избежать коллизий. Во-первых, начинать передачу сигналов следует только тогда, когда никакой другой компьютер не занят тем же самым. Во-вторых, требуется следить за своей связью — если происходит коллизия, нужно подождать короткое, но случайное время, прежде чем пытаться передавать снова.

У этих методов есть некоторые ограничения. Если будет делаться слишком много попыток передачи через среду, коллизии будут происходить безостановочно. Мы говорим, что соединение **нарушено** (saturated), когда чрезмерное количество коллизий нарушает связь. У вас было такое, что на многолюдном мероприятии ваш телефон вдруг переставал отправлять сообщения или дозваниваться? Такое происходит, когда слишком много телефонов пытаются установить связь одновременно и сотовая связь становится перегруженной.

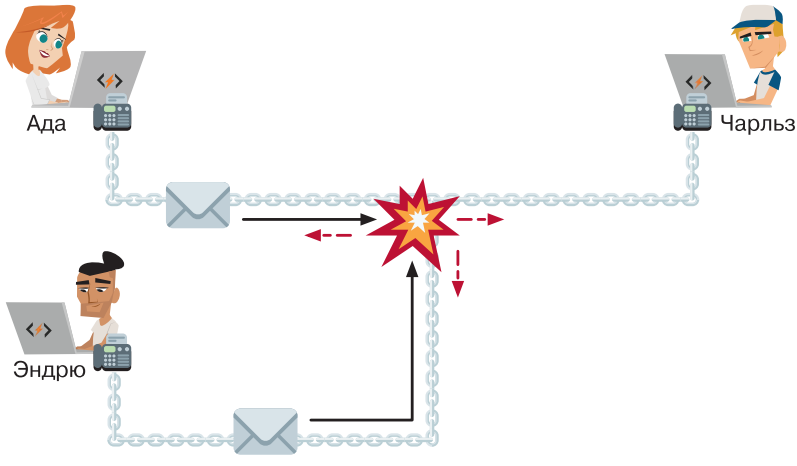


Рис. 1.3. Коллизия между Адой и Эндрю

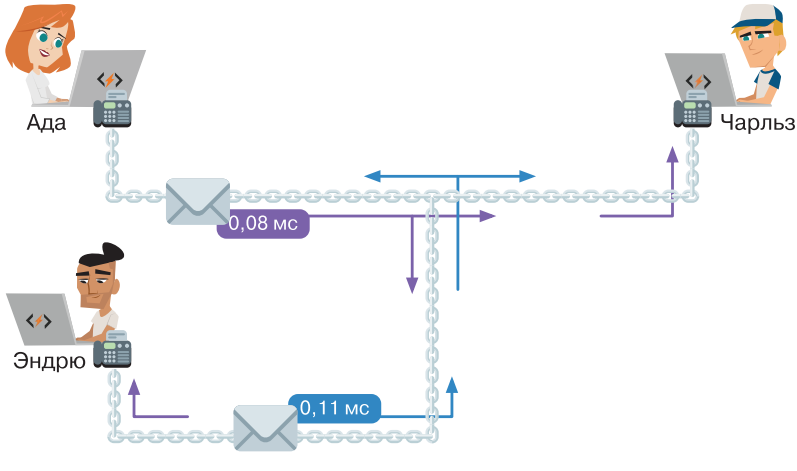


Рис. 1.4. Ада и Эндрю отправляют сигнал повторно через случайное время

ФИЗИЧЕСКИЙ АДРЕС У Ады и Чарльза имеется прямая связь между их компьютерами. Ада хочет поговорить с Чарльзом, поэтому передает сигнал со своим сообщением через среду. Тем не менее среда является общей, поэтому все, кто подключен с ней, получают сообщение (рис. 1.5). Как другие компьютеры узнают, что принятый сигнал предназначен не им?

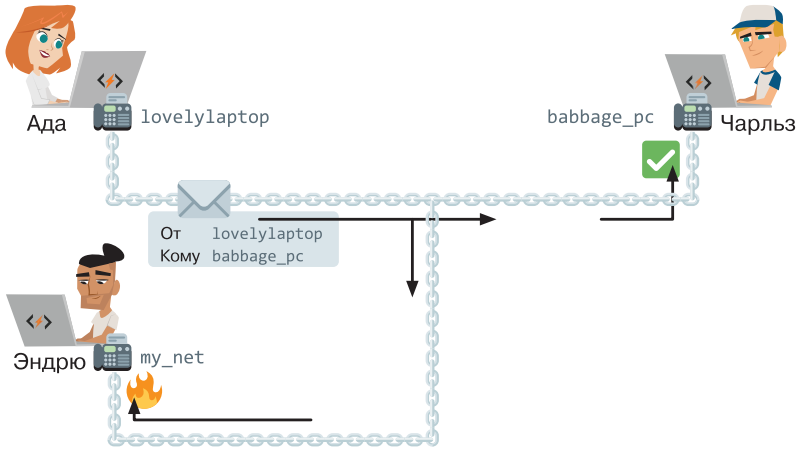


Рис. 1.5. Сетевой интерфейс Эндрю отклоняет сообщение

Сетевой интерфейс каждого компьютера имеет идентификатор, известный как **физический**, или **аппаратный**, **адрес**. Передача в совместно используемой среде должна начинаться с двух таких адресов: адреса получателя и адреса отправителя. Получив сообщение, компьютер поймет — следует его проигнорировать или принять, а также на какой адрес он должен ответить.

Это работает только в том случае, если физические адреса уникальны: если два компьютера используют `my_netinterface`, мы оказываемся в исходной ситуации. По этой причине практически все сетевые интерфейсы следуют схеме именования, определенной в правилах управления доступом к среде (MAC). Эти стандартные физические адреса называются **MAC-адресами**.

MAC-АДРЕСАЦИЯ

Компьютеры, смартфоны, смарт-часы и умные телевизоры могут иметь сетевые интерфейсы Wi-Fi, Bluetooth и Ethernet. Каждый сетевой интерфейс имеет собственный уникальный MAC-адрес, присвоенный оборудованию на этапе производства. Вам не стоит беспокоиться о присвоении вашему компьютеру MAC-адреса: вы всегда можете использовать тот, который был в комплекте с его сетевым интерфейсом.

Поскольку MAC-адреса — это просто большие случайные числа, производители сетевых интерфейсов по всему миру должны согласовывать свои действия, дабы избежать случайного присвоения одного и того же адреса двум разным устройствам. В этом они полагаются на Институт инженеров по электротехнике и электронике (**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers, **IEEE**), который назначает каждому из них свой диапазон MAC-адресов.

MAC-адрес задается шестью парами шестнадцатеричных чисел[1], разделенных двоеточиями. Первая половина адреса — это идентификатор, присвоенный IEEE конкретному производителю. Затем этот производитель выбирает уникальную вторую половину для каждого сетевого интерфейса.

```
60:8B:0E:C0:62:DE
```

Здесь `608B0E` — это номер производителя. Этот конкретный номер IEEE присвоил компании Apple, поэтому такой MAC-адрес должен принадлежать устройству Apple[2]. MAC-адрес устройства часто пишется на этикетке, приклеенной к упаковке, или на самом устройстве, рядом с серийным номером (рис. 1.6).

Для передачи на все компьютеры в среде зарезервирован специальный адрес. Он называется **широковещательным адресом** и выглядит как `FF:FF:FF:FF:FF`. Его используют при попытке подключиться к неизвестному устройству. Например, когда Wi-Fi вашего смартфона не отключен, он постоянно транслирует на `FF:FF:FF:FF:FF` в поисках точки доступа. Обнаруживаемые точки доступа возвращают в ответ свой собственный MAC-адрес, чтобы вы могли установить с ними связь.

Подобные широковещательные сообщения с целью обнаружения, как и все другие передачи, содержат MAC-адрес отправителя. Поэтому прогулку со смартфоном можно сравнить с прогулкой с громкоговорителем,

---

[1] В реальной жизни мы почти всегда записываем числа в десятичной форме, где каждая цифра является одним из десяти символов: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Ученые-компьютерщики, в свою очередь, любят записывать числа в шестнадцатеричной форме, где каждая цифра может быть одним из 16 символов: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, а, б, в, г, д, е. Дополнительную информацию об основаниях систем счисления см. в приложении I.

[2] Можно узнать, кто произвел устройство, введя первые шесть цифр его MAC-адреса здесь: http://code.energy/mac-lookup.

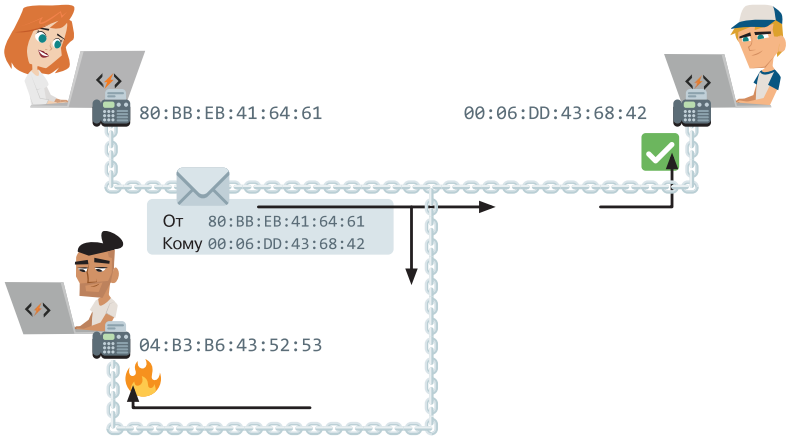


Рис. 1.6. Каждый MAC-адрес уникален

из которого постоянно доносится ваше имя, только с помощью радиоволн вместо звука и MAC-адреса вместо имени. В 2013 году Эдвард Сноуден сообщил, что АНБ[1] отслеживали передвижения людей, прослушивая передачи Wi-Fi в больших городах и сохраняя записи о том, где был обнаружен каждый MAC-адрес.

Вы также можете перевести свой собственный сетевой интерфейс в **неизбирательный режим**, и он начнет принимать все передачи независимо от их предполагаемого получателя. Это позволяет обнаруживать скрытые сети Wi-Fi, составлять перечень MAC-адресов вокруг вас, а иногда даже читать содержимое сообщений других людей. Поэтому интернет-серфинг через незащищенную сеть Wi-Fi может быть небезопасным: связь транслируется всем, кто находится в пределах досягаемости. Вот почему для канального уровня Wi-Fi так важно шифрование[2].

Будьте осторожны: сетевой интерфейс можно настроить так, чтобы передача начиналась как с MAC-адреса получателя, *так и с адреса отправителя*. Ничто не мешает злоумышленнику выдавать себя за вас, используя ваш физический адрес в своих передачах. Такой тип атаки известен как **подмена MAC (MAC-спуфинг)**. Когда канальный уровень только

[1] Агентство национальной безопасности (National Security Agency), разведывательная организация правительства США.

[2] В результате шифрования злоумышленники получают искаженные сообщения.

изобрели, о безопасности еще не нужно было заботиться. Протоколы постоянно развиваются, делаясь более безопасными, что нейтрализует такие атаки, но этому процессу не видно конца.

КАДРЫ

Иногда передача должна содержать много данных, и отправка одного большого сообщения нецелесообразна. Не все сетевые интерфейсы и компьютеры поддерживают одинаковые скорости передачи. Более того, что будет, если в середине передачи произойдет коллизия? Всю передачу придется отменить, поскольку отправителю и получателю будет сложно точно определить, какие части сообщения были получены, а какие — нет.

Для решения подобных проблем длинные сообщения всегда разделяются на небольшие части, каждая из которых отправляется как независимая передача. Время между передачами может варьироваться в зависимости от возможностей обоих компьютеров: более медленным устройствам требуются более длительные перерывы. Если возникает ошибка, необходимо отменить и повторно отправить лишь небольшую передачу, которая не удалась (рис. 1.7).

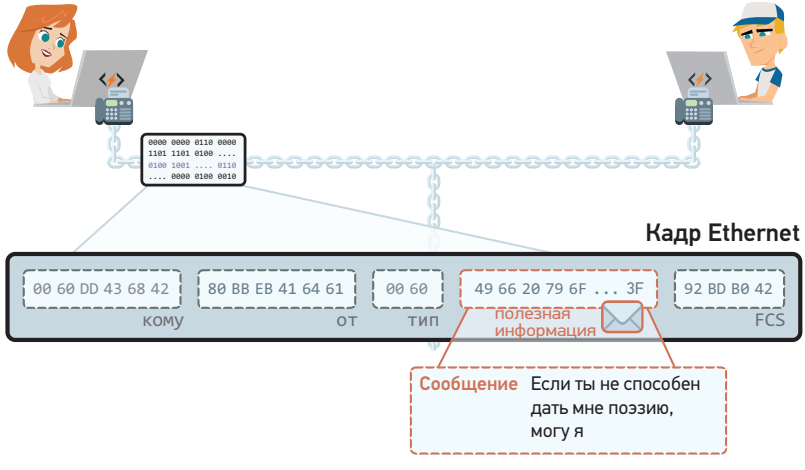


Рис. 1.7. Кадр Ethernet. Как только он передается по медному проводу, он превращается в серию электрических сигналов, которые кодируют число. Протокол Ethernet указывает, как интерпретировать это число. Например, первые 12 шестнадцатеричных цифр кодируют MAC-адрес назначения

Каждая независимая передача называется **кадром**. Стандартные протоколы Wi-Fi ограничивают размер кадра 2346 байтами. Тридцать четыре байта необходимы для MAC-адресов и кодов обнаружения ошибок. Таким образом, кадр Wi-Fi в итоге может нести до 2312 байт данных, называемых полезной информацией[1]. В проводных сетях максимальный размер кадра обычно составляет 1526 байт, где полезной информации — 1500 байт.

В редких случаях помехи в среде мешают передаче, и получатель улавливает сигналы, которые не кодируют в точности ту же информацию, которую отправитель намеревался передать. Посмотрим на специальное поле, которое было добавлено для устранения этой проблемы.

FCS Последняя часть кадра, **FCS** (**F**rame **C**heck **S**equence, последовательность проверки кадра), гарантирует, что информация была передана точно. FCS не добавляет новую информацию к передаче: это просто результат вычисления с использованием содержимого всех остальных полей. Изменение любого содержимого до FCS должно приводить к изменению самого FCS.

Получив кадр, компьютер вычисляет *ожидаемый* FCS из полученной информации и сравнивает его с *полученным* FCS. Если они не совпадают, кадр отбрасывается. Если же совпадают, то мы понимаем, что сообщение не было искажено, а принятая полезная информация не содержит ошибок.

ТИП Кадр, показанный на рис. 1.7, содержит еще одно поле, о котором мы пока не говорили: **тип полезной информации**. Оно сообщает получателю, каким правилам следовать для интерпретации данных в кадре. В следующем разделе мы рассмотрим наиболее распространенный набор таких правил.

1.2. МЕЖСЕТЕВОЙ УРОВЕНЬ

Мы увидели, что канальный уровень позволяет напрямую подключенным компьютерам обмениваться сообщениями внутри кадров. В **межсетевом уровне**, также известном как **сетевой уровень**, указывается, как передавать эти сообщения между компьютерами, которые подключены *не* напрямую.

[1] Если мы используем один байт на символ, в кадре Wi-Fi уместится около 500 слов, чего достаточно для заполнения страницы текста.

Хитрость заключается в установке нескольких компьютеров, называемых **маршрутизаторами** (или **роутерами**), с несколькими сетевыми интерфейсами. Затем все компьютеры в сети подключаются как минимум к одному маршрутизатору, а каждый маршрутизатор связан как минимум еще с одним. Когда маршрутизатор получает сообщение на одном из своих сетевых интерфейсов, он может переслать его другому маршрутизатору через другой сетевой интерфейс.

ЛОКАЛЬНЫЕ СЕТИ Мы можем попросить маршрутизатор, с которым мы связаны, переслать сообщение на компьютер, с которым мы не связаны. Предположим, у вас дома есть проводная сеть, соединяющая маршрутизатор и настольный компьютер. Предположим, маршрутизатор также напрямую подключен к смартфону в другой, беспроводной сети.

Несмотря на то что настольный компьютер и смартфон напрямую не подключены к одной и той же сети, они могут отправлять сообщения друг другу, используя маршрутизатор в качестве ретранслятора. Компьютеры из разных сетей в непосредственной близости, способные общаться друг с другом через маршрутизаторы, образуют большую сеть, называемую локальной (**L**ocal **A**rea **N**etwork, **LAN**).

В доме или небольшом офисе одного маршрутизатора будет достаточно, чтобы связать все компьютерные сети на данной территории. При сборке локальной сети, охватывающей большую организацию, такую как университет или больница, может потребоваться множество маршрутизаторов для соединения всех компьютерных сетей в единую систему (рис. 1.8).

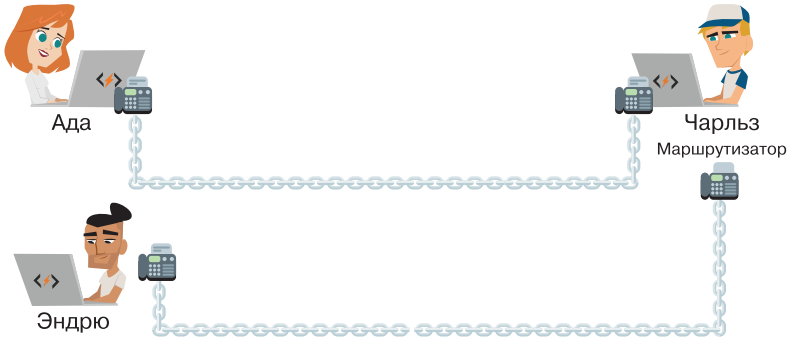


Рис. 1.8. В этой небольшой локальной сети Ада и Эндрю могут отправлять сообщения друг другу через свой маршрутизатор «Чарльз»

ГЛОБАЛЬНЫЕ СЕТИ Но зачем останавливаться на достигнутом? Если ваш маршрутизатор подключен к маршрутизатору за пределами вашего дома, который, в свою очередь, связан с маршрутизатором в университете, вы можете запросить пересылку вашего сообщения на компьютеры в локальной сети университета. Когда удаленные локальные сети соединяются друг с другом, они формируют глобальную сеть (**W**ide **A**rea **N**etwork, **WAN**) (рис. 1.9).

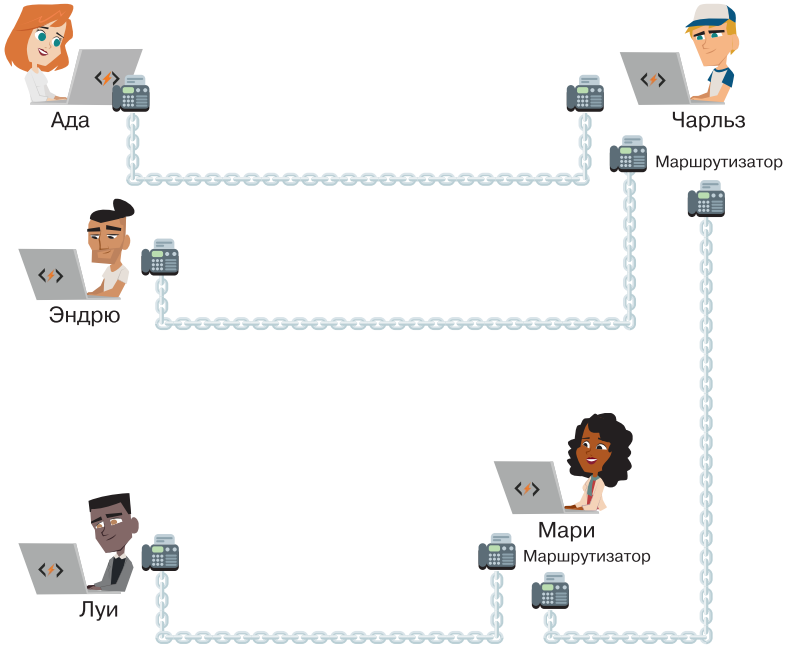


Рис. 1.9. Чарльз подключен к удаленному маршрутизатору Мари, и они оба пересылают сообщения по этой глобальной сети

Глобальная сеть может расти по мере того, как к ней подключается все больше локальных сетей. Различные глобальные сети также могут быть подключены друг к другу, формируя сеть еще большего размера. Самая большая глобальная сеть в мире — это собрание тысяч **взаимо**связанных **сетей**, которое мы называем **интернетом**. Это сеть, которую мы используем каждый день для отправки электронных писем и просмотра веб-страниц. В 2020 году в нее входило более миллиарда компьютеров. Узнаем, каким образом все они связаны.

МЕЖСЕТЕВОЕ ВЗАИМОДЕЙСТВИЕ

Самый простой способ подключить маршрутизатор к интернету — заплатить за это. Определенные организации в интернете свяжут один из своих маршрутизаторов с вашим и позволят вашим сообщениям проходить через свою сеть в обоих направлениях. Эта платная услуга называется **транзитом**, так как все ваши сообщения будут проходить *транзитом* через их сеть, прежде чем перейти к конкретному целевому маршрутизатору.

Однако проход через стороннюю сеть не всегда нужен для подключения к другому маршрутизатору в интернете. Если, например, два близлежащих университета много общаются, они могут связать свои маршрутизаторы, чтобы сообщения передавались между их сетями напрямую. Это позволит сэкономить деньги, поскольку в противном случае эти сообщения должны были бы проходить через платное соединение. Свободный обмен сообщениями между сетями разных организаций называется **пирингом**.

МАРШРУТИЗАЦИЯ

Любой компьютер, подключенный к маршрутизатору в интернете, может запросить пересылку его сообщений другим маршрутизаторам (рис. 1.10). Так сообщения могут передаваться на большие расстояния. Например, во многих прибрежных городах есть система подводных кабелей, соединяющих маршрутизаторы.

Прямого соединения между маршрутизаторами в Майами и Буэнос-Айресе нет. Однако Майами связан с Пуэрто-Рико, который связан с Форталезой, которая связана с Рио-де-Жанейро, который связан с Буэнос-Айресом. Майами и Буэнос-Айрес могут обмениваться сообщениями через эти кабели, если маршрутизаторы по пути пересылают сообщения туда и обратно. Сегодня подводные кабели соединяют сотни прибрежных городских маршрутизаторов по всему миру (рис. 1.11).

Практически любой другой город на Земле напрямую или опосредованно связан с этими прибрежными городами, часто с помощью кабелей, проложенных на земле. Спутники связи также имеют маршрутизаторы для установления беспроводных соединений с удаленными точками. Все маршрутизаторы могут пересылать сообщения, поэтому сообщение,

которое вы отправляете в интернете, можно направить на любой другой компьютер в интернете. Но только в том случае, *если* путь к нему может быть найден.

Рис. 1.10. Система SAm-1 связывает маршрутизаторы в 16 городах из 11 разных стран, используя более 15 тысяч миль подводных кабелей

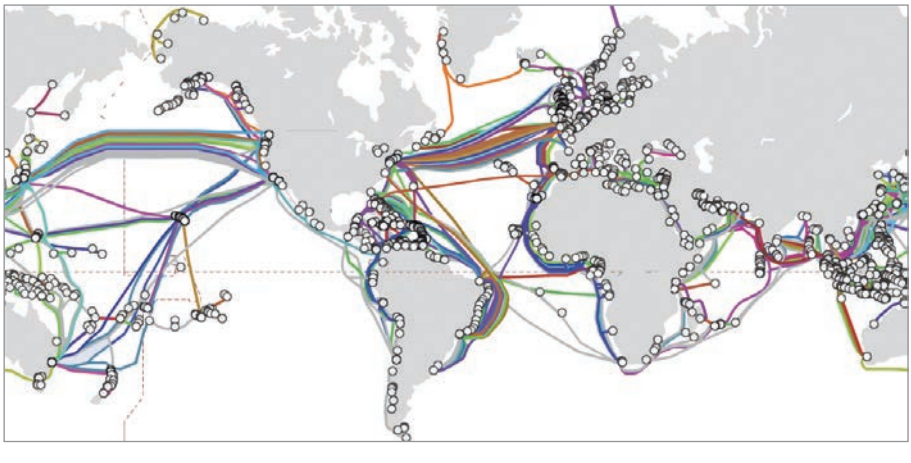


Рис. 1.11. В настоящее время используются подводные оптоволоконные кабели

АДРЕСАЦИЯ МЕСТОПОЛОЖЕНИЯ

На канальном уровне компьютеры идентифицируются по физическому адресу. Физические адреса однозначно идентифицируют компьютеры, но не дают никаких подсказок о том, *где именно* компьютер подключен и как до него добраться. Если компьютер перевезти на другой конец света, он сохранит свой физический адрес!

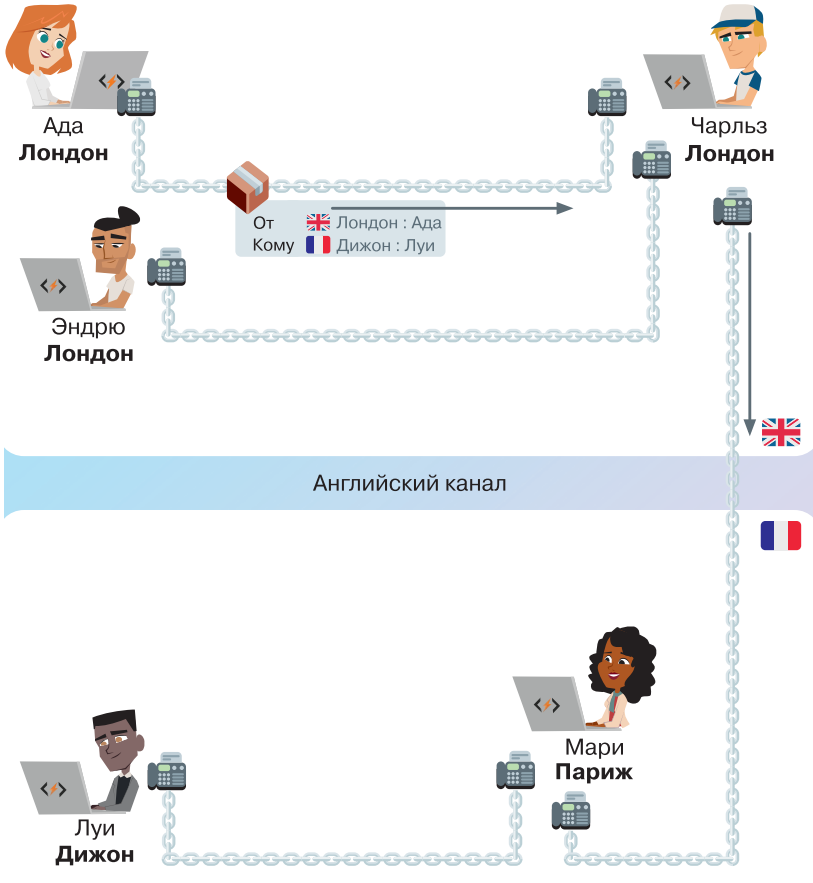


Рис. 1.12. Ада хочет отправить пакет Луи, поэтому просит свой маршрутизатор (у Чарльза) переслать его. Она пишет на пакете иерархический адрес Луи. В итоге Чарльз знает, что должен отправить пакет во Францию, поэтому отправляет его французскому маршрутизатору, с которым он связан: Мари

Предположим, вы отправили Луи по почте пакет, где вместо его адреса только его фотография. Этот пакет имеет определенное место назначения; однако международная почтовая служба не имеет возможности узнать, куда именно следует отправить пакет, чтобы доставить его Луи.

Почтовые отделения должны сначала узнать, в какую страну отправлять посылку. После этого первое почтовое отделение в этой стране должно знать, в какую область или штат ее необходимо переправить. Следующее почтовое отделение должно знать город, а последнее — почтовый адрес. Адрес, содержащий всю эту информацию, называется **иерархическим**. Как и в случае с почтовыми отделениями, маршрутизаторам необходим иерархический адрес местонахождения получателя пакета (рис. 1.12).

Чтобы такой механизм работал в глобальном масштабе, *все* задействованные компьютеры должны следовать одному и тому же набору правил для создания и обработки запросов на пересылку сообщений (рис. 1.13). Компьютер в Китае должен понимать запрос от компьютера в Нигерии, даже если они используют разные языки, операционные системы и оборудование.

Рис. 1.13. «До интернета». Любезно предоставлено http://xkcd.com

ИНТЕРНЕТ-ПРОТОКОЛ

Мы видели, что для установки соединения с другим компьютером необходимо следовать правилам Medium Access Control. Точно так же нужно

следовать **интернет-протоколу**, или **IP**[1], чтобы просить маршрутизаторы пересылать сообщения на другие компьютеры в вашей локальной сети или в интернете.

Запрос на пересылку сообщения, который следует правилам IP, называется **IP-пакетом**. IP-пакет, по сути, представляет собой большое число, где цифры в определенных позициях кодируют ключевую информацию. Практически все компьютеры, выпущенные в последние десятилетия, поддерживают IP-пакеты и умеют их пересылать. Это позволяет с легкостью передавать IP-пакет с одного компьютера на другой, пока тот не достигнет места назначения.

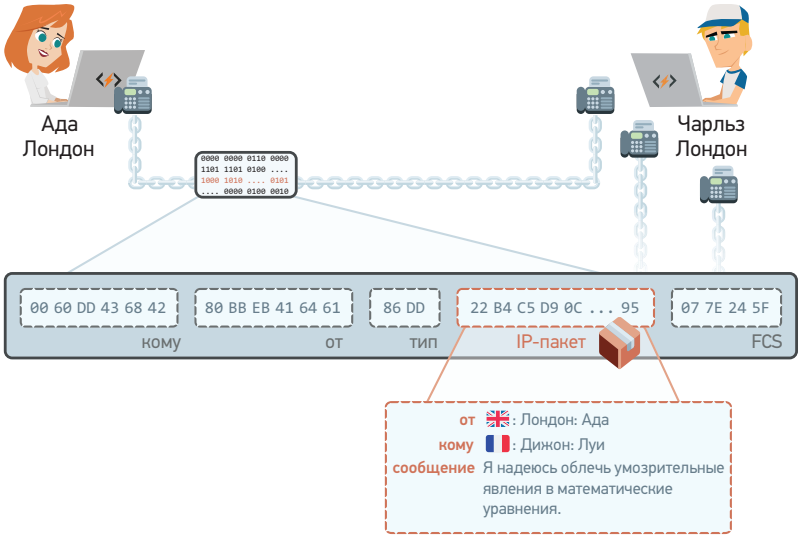


Рис. 1.14. Ада отправляет своему маршрутизатору (у Чарльза) кадр Ethernet, содержащий IP-пакет для Луи. Таким образом, кадр Ethernet содержит физический адрес Чарльза, а пакет — адрес Луи. После этого Чарльз перешлет пакет в новом собственном кадре, содержащем некий физический адрес во Франции

[1] Под IP мы подразумеваем его последнюю версию, IPv6. Устаревшая версия протокола, IPv4, до сих пор используется, хотя была выпущена в 1981 году. IPv4 способен поддерживать только около трех миллиардов компьютеров. IPv6, запущенный в 2012 году, может поддерживать практически неограниченное количество компьютеров. По состоянию на 2020 год треть компьютеров в интернете используют IPv6.

IP-пакет содержит *адреса местоположения* отправителя и получателя, после чего следуют любые данные, которые они хотят передать. Чтобы отправить IP-пакет, мы передаем кадр, в котором полезной информацией является IP-пакет, а тип такого кадра — `86DD`. Когда маршрутизатор получает кадр этого типа, IP-пакет повторно передается в другом кадре на следующий компьютер по пути к месту назначения пакета (рис. 1.14).

Чтобы IP-пакеты могли пересылаться повсеместно, необходимо согласовать стандарт для адресации местоположения. Мы видели, как производители выделяют физические адреса в соответствии с правилами контроля доступа к среде. Пришла пора узнать, как интернет-протокол делает то же самое для адресов местоположения. Затем мы увидим, как интернет-протокол определяет правила маршрутизации на основе этих адресов.

1.3. IP-АДРЕСАЦИЯ

Интернет-протокол устанавливает правила работы с адресами местоположения — поэтому они и называются **IP-адресами**. Компьютеры способны отправлять или получать IP-пакеты только после получения IP-адреса. Разрешение на использование группы IP-адресов в первую очередь предоставляется организации. Эти адреса затем назначаются компьютерам, которые прямо или косвенно связаны с этой самой организацией.

Чтобы понять суть процесса, определим, что такое IP-адреса и как они записываются[1]. IP-адрес — это число длиной 128 бит[2]. Обычно они записываются в шестнадцатеричном формате с двоеточиями, разделяющими восемь групп по четыре цифры. Это IP-адрес сервера Facebook:

```
2a03:2880:f003:0c07:face:b00c:0000:0002
```

[1] Мы будем представлять IP-адреса, как это определено в последней версии IP. Устаревшие адреса IPv4 по-прежнему используются. Они записываются в виде четырех групп до трех десятичных чисел, разделенных точками, например 192.168.0.1.

[2] Для записи такого числа нужно 128 нулей и единиц. Это означает, что число лежит в диапазоне от 0 до 340 282 366 920 938 463 463 374 607 431 768 211 456.

IP-адреса можно сокращать, опуская ведущие нули в любом четырехзначном блоке:

```
2a03:2880:f003:c07:face:b00c::2
```

Как и в случае с почтовым адресом с указанием страны, города и улицы, IP-адреса имеют иерархическую структуру для облегчения маршрутизации. В то время как самая общая часть почтового адреса — это страна, самая общая часть IP-адреса — это **префикс маршрутизации**:

```
2a03:2880:f003:c07:face:b00c::2
```
Префикс маршрутизации

Префикс представлен первыми цифрами IP-адреса. Как только организация получает такой префикс, она имеет право назначить своим компьютерам любой IP-адрес, который начинается с этого префикса. Префикс может иметь разную длину: организациям, под управлением которых больше компьютеров, выдаются более короткие префиксы, а некоторым предоставляется даже несколько.

Например, мы знаем, что все адреса, начинающиеся с префикса `2a03:2880`, принадлежат компьютерам в сети Facebook. Те, которые начинаются с `2c0f:fb50:4002`, находятся в сети Google в Кении. Для своего центра обработки данных в Сингапуре Google получил префикс `2404:6800`.

Для целей маршрутизации локальные и глобальные сети с одним и тем же префиксом организованы в небольшие сети, называемые **подсетями**. Между префиксом маршрутизации и серединой IP-адреса указано, в какой именно подсети находится компьютер.

```
2a03:2880:f003:c07:face:b00c::2
          Подсеть
```

Это означает, что в Facebook есть сеть, где все компьютеры имеют IP-адреса, начинающиеся с `2a03:2880:f003:c07`. Префикс маршрутизации и подсеть вместе образуют **идентификатор сети** IP-адреса. Идентификатор сети всегда состоит из 16 цифр (включая опущенные нули). Это означает, что в организации с более длинным префиксом маршрутизации может быть меньше подсетей.

Наконец, следующие 16 цифр IP-адреса называются **идентификатором интерфейса**, поскольку они указывают на конкретный сетевой интерфейс

в подсети. Многие сетевые администраторы попросту заполняют эту часть IP-адреса MAC-адресом устройства. Эти цифры могут быть любыми, если они используются только один раз для каждой подсети.

```
2a03:2880:f003:c07:face:b00c::2
```

Идентификатор сети — Идентификатор интерфейса

Чтобы эта система адресации работала повсеместно, необходим механизм, гарантирующий, что никакие две организации не используют один и тот же префикс маршрутизации (рис. 1.15). Как и в случае с MAC-адресами, инженеры решили эту проблему посредством международной координации.

Рис. 1.15. Не спрашивайте начальницу, где она живет!

IANA

Инженеры всего мира согласились с тем, что американская некоммерческая организация **I**nternet **A**ssigned **N**umbers **A**uthority (**IANA**) решает, кто и над каким префиксом IP-маршрутизации получает контроль. На практике IANA делегирует большую часть своих полномочий пяти некоммерческим организациям, которые называются **R**egional **I**nternet **R**egistries или **RIR**. Для этого он выделяет каждой RIR короткие шестнадцатеричные комбинации, которые они могут использовать в качестве первых цифр назначаемых ими префиксов маршрутизации (рис. 1.16, 1.17).

Чтобы получить префикс маршрутизации для вашей организации, придется сделать запрос в RIR того региона, где будут находиться ваши маршрутизаторы. Затем этот RIR назначит вам префикс, начинающийся с одной из своих комбинаций шестнадцатеричных цифр, которые им присвоила IANA.

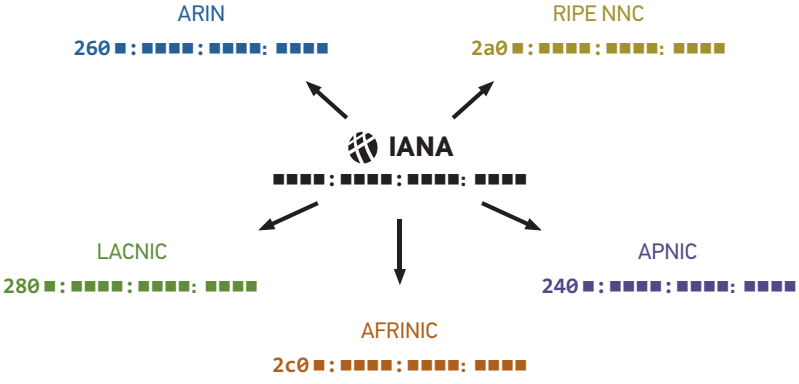


Рис. 1.16. Примеры распределения по каждой RIR

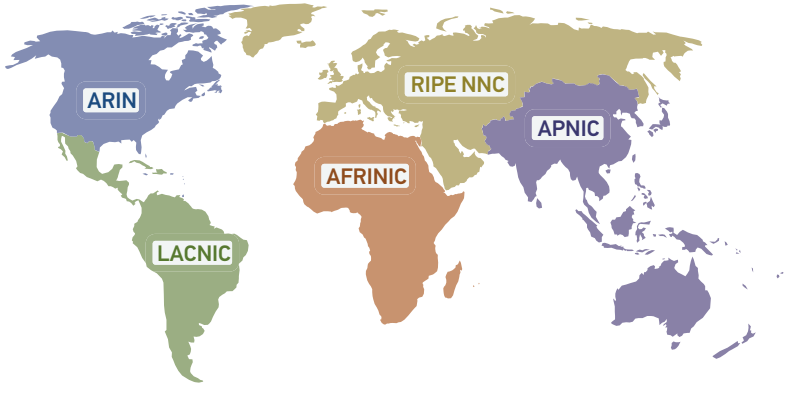


Рис. 1.17. IANA делегирует полномочия по IP-адресации по географическому принципу: каждая RIR отвечает за свой регион

Например, Facebook, штаб-квартира которого находится в Ирландии, префикс маршрутизации получил от RIPE NCC. Точно так же швейцарский банк Credit Suisse имеет филиал в Латинской Америке, и префикс маршрутизации ему предоставила LACNIC (рис. 1.18).

Это означает, что компьютерам в филиалах Credit Suisse в Латинской Америке могут быть назначены следующие IP-адреса:

2801:80:1380:■■■■:___:___:___:___

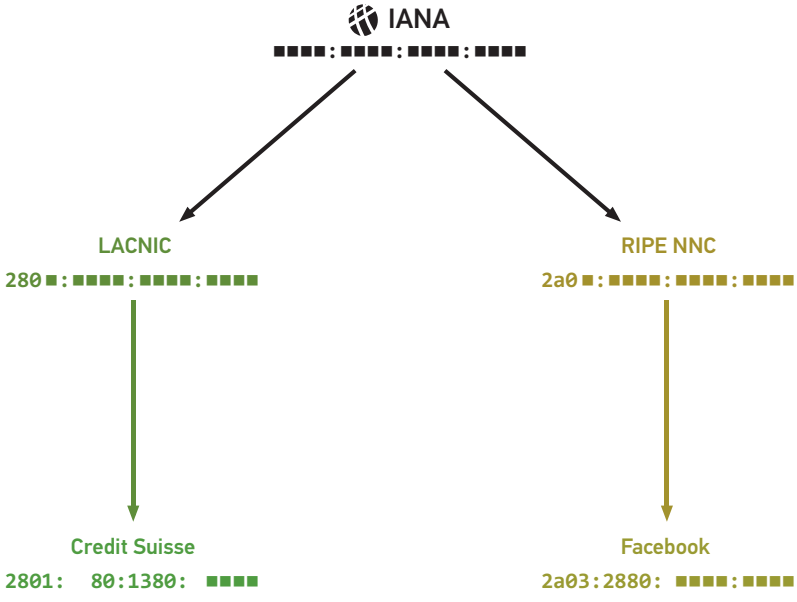


Рис. 1.18. Цепочка распределения IP-адресов для двух компаний

Сетевые администраторы в банке присваивают уникальную комбинацию шестнадцатеричных цифр каждой из своих подсетей, чтобы они умещались в оставшемся пространстве сетевой части ■■■■. Поскольку каждая шестнадцатеричная цифра может иметь 16 различных значений, у банка достаточно места для $16^4 = 65\,536$ разных подсетей. Facebook, как более крупной организации, был предоставлен префикс с местом более чем на 4 миллиарда подсетей!

Мы уже видели, что сетевые администраторы могут сами выбирать, как им заполнять 16 свободных символов идентификатора интерфейса для отдельных устройств. Такие устройства могут затем отправлять и получать IP-пакеты в интернет и из интернета, пока их маршрутизатор подключен.

ПРОВАЙДЕРЫ ИНТЕРНЕТ-УСЛУГ

Большинство частных лиц и небольших организаций не имеют непосредственного отношения к RIR и не поддерживают пиринговые связи с другими компьютерными сетями. Вместо этого они покупают подключение

к интернету у специализированных компаний, которые называются интернет-провайдерами (**ISP**, **I**nternet **S**ervice **P**roviders). Провайдеры устанавливают маршрутизаторы неподалеку от своих клиентов. Таким образом, они могут легко подключить один из своих маршрутизаторов к маршрутизатору в помещении любого клиента. Они же назначают и префикс маршрутизации для каждого из своих клиентов.

Посмотрим, как это работает на практике. В Великобритании интернет-провайдер Sky получил префикс маршрутизации `2a02:0c7f`. Sky работает во многих британских городах, поэтому префикс делится между их региональными базами. Например, они назначают `2a02:c7f:48` своей сети в Милтон-Кинсе и `2a02:c7f:7e` той, что в Ромфорде[1]. Предположим, Ада живет в Ромфорде и хочет создать сеть у себя дома. У нее имеются настольный компьютер и принтер, которые она хочет подключить с помощью кабеля Ethernet. Ей также нужна собственная сеть Wi-Fi для подключения смартфона, планшета и ноутбука.

Ада нанимает Sky, и они подключают свой маршрутизатор Romford к маршрутизатору в ее доме. Sky назначает маршрутизатору Ады 14-значный префикс маршрутизации на основе префикса из своей ромфордской базы. Каждой сети в доме Ады (проводной и беспроводной) назначается подсеть на основе префикса маршрутизации Sky, назначенного Аде. Рисунок 1.19 показывает полный процесс выделения IP-адресов от IANA к каждому из устройств Ada. Маршрутизатор Ады принимает IP-пакеты от нескольких компьютеров, но ее маршрутизатор способен с легкостью решить, по какому каналу пересылать каждый полученный пакет. Пакеты, адресованные компьютеру в одной из подсетей Ada, доставляются напрямую. Все остальные получаемые IP-пакеты пересылаются провайдеру.

Для маршрутизаторов, которые не полагаются на интернет-провайдера, это не так просто: с ними связываются несколько маршрутизаторов из нескольких компьютерных сетей. Но как же они решают, по какому каналу следует пересылать IP-пакет? И как они могут быть уверены, что пересылают его на маршрутизатор, расположенный ближе к конечному пункту назначения?

---

[1] Эта информация является общедоступной, вы можете найти сетевое расположение любого префикса маршрутизации. Практика называется геолокацией по IP, и именно так сайты определяют страну и город, из которых вы их просматриваете.

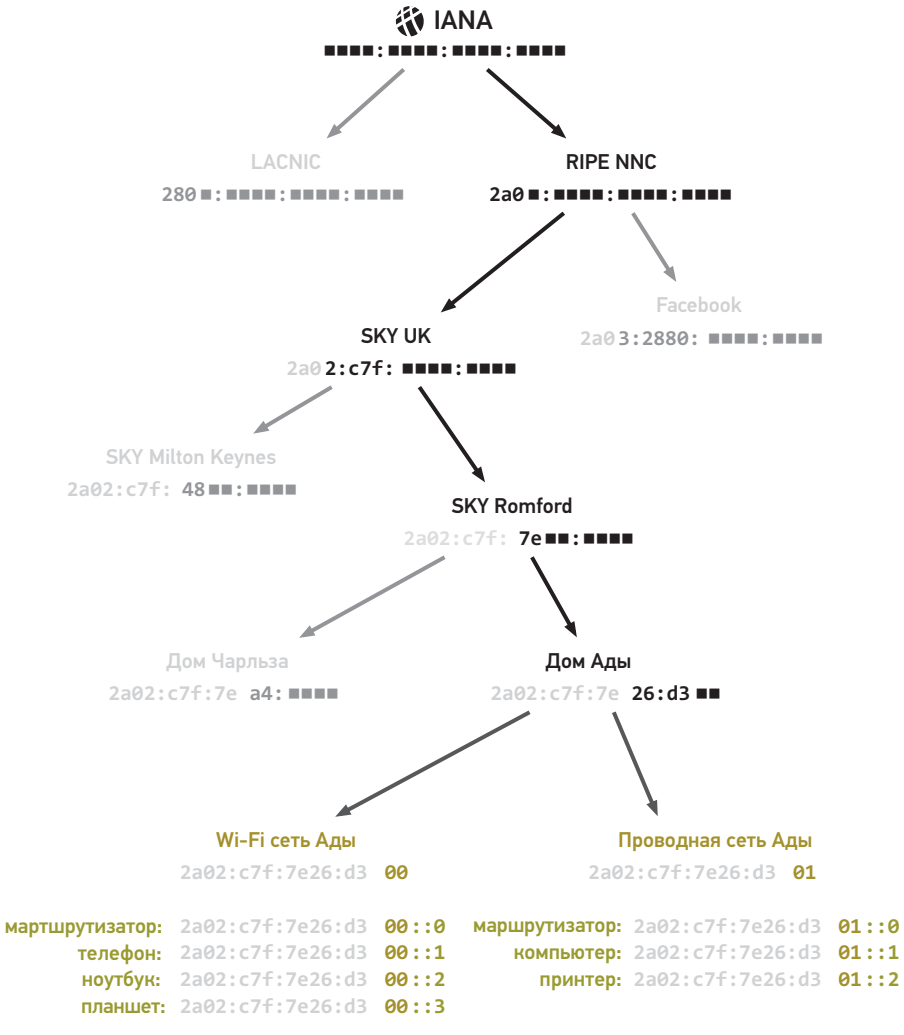


Рис. 1.19. Выделение IP-адресов от IANA устройствам Ada. Ее маршрутизатор использует разные подсети для проводных и беспроводных сетей и, следовательно, имеет разные IP-адреса для каждой из них

1.4. IP-МАРШРУТИЗАЦИЯ

Предположим, Ада хочет отправить сообщение в Facebook со своего ноутбука. Она будет использовать интернет-протокол, поэтому начинает с создания IP-пакета, который содержит ее собственный IP-адрес,

IP-адрес Facebook и ее сообщение в качестве полезной информации. Затем она передает пакет в составе кадра Wi-Fi со своего ноутбука на домашний маршрутизатор (рис. 1.20).

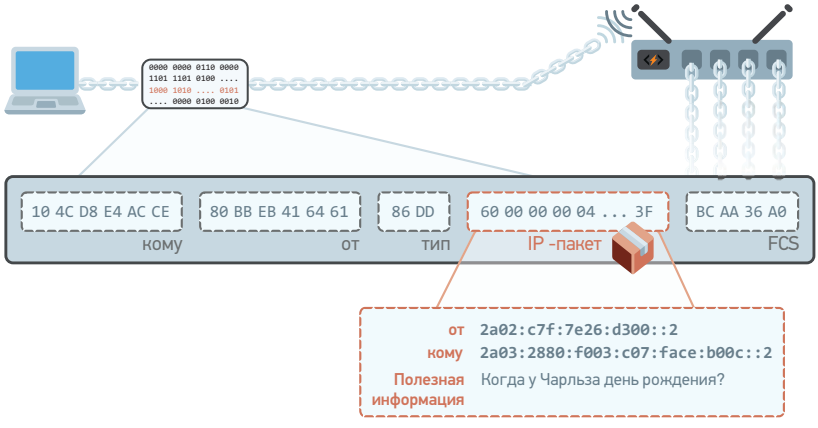


Рис. 1.20. IP-пакет, передаваемый по Wi-Fi[1]

Несколько маршрутизаторов, начиная с маршрутизатора в доме Ады, повторно передают пакет, пока он не достигнет Facebook. По пути каждый из этих маршрутизаторов должен выбрать, в каком направлении нужно «перебросить» пакет, чтобы он достиг следующего маршрутизатора. Затем последний маршрутизатор «перебросит» его к конечному компьютеру назначения.

ТАБЛИЦЫ АДРЕСОВ

Маршрутизаторы выбирают следующий пункт назначения пакета на основе его IP-адреса. Для этого у них есть таблица с адресами. В ее строках перечислены возможные IP-адреса, на распознавание которых настроен маршрутизатор. Для каждого адреса в таблице указано, какой компьютер должен быть следующим переходом пакета, направленного на этот адрес. У каждого маршрутизатора есть уникальная таблица, отражающая способ подключения маршрутизатора. Например, вот как подключен маршрутизатор Ады (рис. 1.21, 1.22).

[1] Мы включили поля кадра Wi-Fi, которые существуют и в кадрах Ethernet. В кадре Wi-Fi больше полей, но для упрощения они были скрыты.

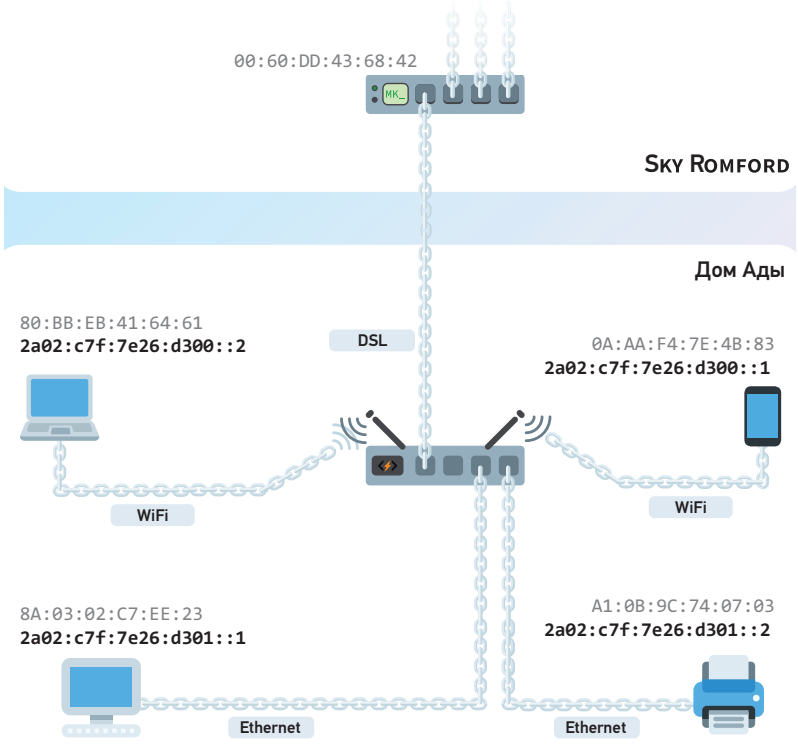


Рис. 1.21. Маршрутизатор Ады подключен к настольному компьютеру и принтеру через Ethernet, к ноутбуку и смартфону через Wi-Fi, а к интернет-провайдеру через DSL (Digital Subscriber Line, технология передачи цифровых данных по старым телефонным кабелям)

IP-адрес получателя	Интерфейс	MAC-адрес
2a02:c7f:7e26:d301::1	Ethernet	8A:03:02:C7:EE:23
2a02:c7f:7e26:d301::2	Ethernet	A1:0B:9C:74:07:03
2a02:c7f:7e26:d300::1	WiFi	0A:AA:F4:7E:4B:83
2a02:c7f:7e26:d300::2	WiFi	80:BB:EB:41:64:61
По умолчанию	DSL	00:60:DD:43:68:42

Рис. 1.22. Таблица, которая помогает маршрутизатору Ады правильно пересылать IP-пакеты на компьютеры, показанные на рис. 1.21

Если маршрутизатор получает пакет, IP-адрес назначения которого не соответствует ни одной строке в таблице, пакет пересылается на **маршрут по умолчанию**. Маршрутизация для маршрутизатора Ады проста: пакет либо доставляется непосредственно на компьютер в ее доме, либо пересылается ее провайдеру Sky Romford.

Маршрутизация для маршрутизатора интернет-провайдера сложнее. В дополнение к своим пиринговым и транзитным каналам он получает пакеты от множества разных клиентов. Для простоты предположим, что маршрутизатор Sky Romford обслуживает только двух клиентов и имеет два пиринговых канала: один к маршрутизатору Sky в Милтон-Кинсе, а другой — к Оксфордскому университету. Наконец, представим, что у него есть транзитное соединение с более крупной телекоммуникационной компанией (рис. 1.23, 1.24).

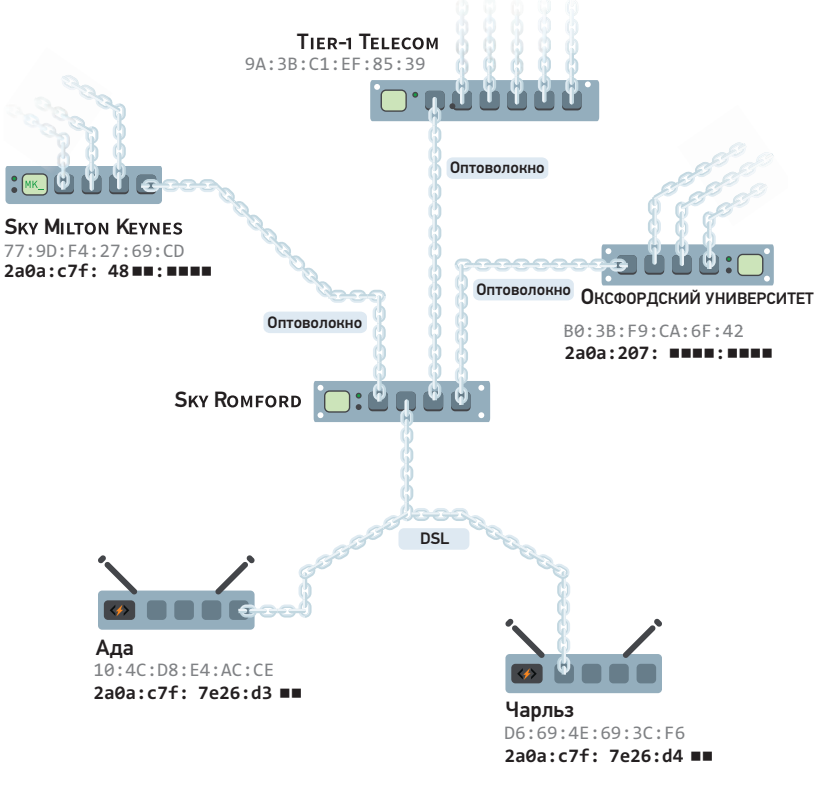


Рис. 1.23. Карта связи Sky Romford. Ада, Чарльз и Оксфордский университет могут общаться, используя только местную инфраструктуру Sky

IP-адрес получателя	Интерфейс	MAC-адрес
2a0a:c7f:7e26:d3■■:…	DSL	10:4C:D8:E4:AC:CE
2a0a:c7f:7e26:d4■■:…	DSL	D6:69:4E:69:3C:F6
2a0a:c7f:48■■:■■■■:…	Оптоволокно	77:9D:F4:27:69:CD
2a0a:207:■■■■:■■■■:…	Оптоволокно	B0:3B:F9:CA:6F:42
по умолчанию	Оптоволокно	9A:3B:C1:EF:85:39

Рис. 1.24. Таблица, которая помогает маршрутизатору Sky Romford правильно пересылать IP-пакеты в сети, показанные на рис. 1.23

Вот здесь и пригодится иерархия IP-адресов. В таблице пересылки на рис. 1.24 IP-адреса группируются по префиксу маршрутизации. Это работает, потому что все IP-адреса, начинающиеся с `2a0a:207`, относятся к компьютерам в Оксфордском университете, и все IP-адреса, начинающиеся с `2a02:c7f:48`, — к компьютерам, обслуживаемым Sky в Милтон-Кинсе.

ТОЧКИ ОБМЕНА ИНТЕРНЕТ-ТРАФИКОМ

Чтобы увеличить емкость и скорость, сетевые администраторы часто устанавливают пиринговые соединения с максимально возможным количеством других организаций. Самый дешевый способ — использовать места, называемые точками обмена интернет-трафиком (**I**nternet **E**xchange **P**oints, или **IXP**с). Организации присоединяются к IXP, подключая свои маршрутизаторы к зданию IXP. Каждая участвующая организация может затем установить индивидуальные пиринговые связи с другими организациями, подключенными к зданию[1].

На рис. 1.23 для ясности показаны только два пиринговых соединения. Типичный интернет-провайдер на самом деле поддерживает множество пиринговых связей на каждый IXP, к которому он подключен. Кроме того, в больших городах интернет-корпорации, такие как Netflix и Google, часто устанавливают пиринговые соединения напрямую с интернет-

[1] IXP очень важны для того, чтобы сделать интернет надежным и дешевым. Это видео объясняет почему: http://code.energy/IXP.

провайдерами, что позволяет им устанавливать более короткие и быстрые соединения со многими своими клиентами.

ИНТЕРНЕТ-МАГИСТРАЛЬ

Интернет-провайдеры и другие телекоммуникационные компании обычно максимально расширяют свои межсетевые соединения, устанавливая пиринговые связи везде, где только возможно. Однако для доступа к сетям, с которыми они не могут взаимодействовать, им приходится покупать транзит у других операторов.

В мире есть всего несколько компаний, которые никому не платят за транзит. Эти компании обслуживают *огромные* сети, которые взаимодействуют друг с другом, что позволяет региональным интернет-провайдерам быть глобально взаимосвязанными. Эти *огромные* сети называются **сетями уровня 1**, и они составляют основу интернета. Некоторые из них эксплуатируются компаниями AT&T, Verizon и Lumen[1].

ДИНАМИЧЕСКАЯ МАРШРУТИЗАЦИЯ

Крупные телекоммуникационные компании должны поддерживать связь, даже если некоторые из их транзитных или пиринговых каналов выходят из строя. Это означает, что они не могут полагаться лишь на одно соединение для каждого префикса маршрутизации в своей таблице адресов. В реальности у них есть **динамические маршрутизаторы**, которые отображают то, как другие сети связаны между собой, и это позволяет выбирать маршруты для определения приоритетов в их таблицах.

Динамические маршрутизаторы периодически обмениваются информацией с другими динамическими маршрутизаторами, с которыми связаны. Они сообщают друг другу, какие префиксы сети доступны по каждой из их ссылок. Это позволяет определять, на сколько переходов находится каждая ссылка от каждого префикса маршрутизации и где эти переходы происходят. Далее динамические маршрутизаторы могут определить

[1] Чтобы понять, насколько огромны эти сети, представьте, что только Lumen эксплуатирует 750 000 миль оптоволоконных кабелей. Длины такого кабеля более чем достаточно, чтобы трижды дотянуться до Луны!

лучший маршрут к каждому префиксу на основе таких показателей, как расстояние и скорость[1].

На основе этой информации динамические маршрутизаторы создают таблицу, охватывающую все префиксы маршрутизации. Для каждого префикса в таблице указывается, какой следующий переход принадлежит к наилучшему маршруту к конечному пункту назначения. Когда связь устанавливается или обрывается, динамические маршрутизаторы информируют об этом своих пиров. По мере распространения новостей все они обновляют свои таблицы, чтобы и дальше пересылать пакеты по самым оптимальным маршрутам.

Нет никакого центрального органа, координирующего обмен этой информацией: маршрутизаторы свободно и добровольно делятся информацией о канале со своими коллегами. Следовательно, часто возникают проблемы с маршрутизацией.

ПЕТЛЯ МАРШРУТИЗАЦИИ

Неправильно настроенные маршрутизаторы могут спровоцировать ошибки. В частности, ошибочные таблицы адресов могут отправлять пакет *назад* на несколько переходов, обрекая его на бесконечный цикл (рис. 1.25).

Если таблицы не исправить, то новые пакеты с тем же адресатом будут бесконечно пересылаться по кругу. Слишком большое количество пакетов может даже привести к насыщению и засорению соединений. Это явление известно под названием **«проблема петли маршрутизации»**. К счастью, интернет-протокол позволяет определить проблему, когда она возникает.

ЛИМИТ ПЕРЕХОДОВ Чтобы не попадать в бесконечные петли маршрутизации, все IP-пакеты содержат **лимит переходов, задаваемый числом** от 0 до 255. Он указывает, сколько раз пакет может быть переадресован маршрутизаторами. Обычно пакеты создаются с лимитом переходов 64. Всякий раз, когда маршрутизатор пересылает пакет, он уменьшает лимит переходов на единицу (рис. 1.26).

---

[1] Все пять RIR постоянно раскрывают информацию обо всех делегированных ими префиксах маршрутизации. Динамические маршрутизаторы внимательно отслеживают эти объявления и могут гарантировать, что в их таблицах имеется строка для каждого существующего префикса маршрутизации (см. рис. 1.26).

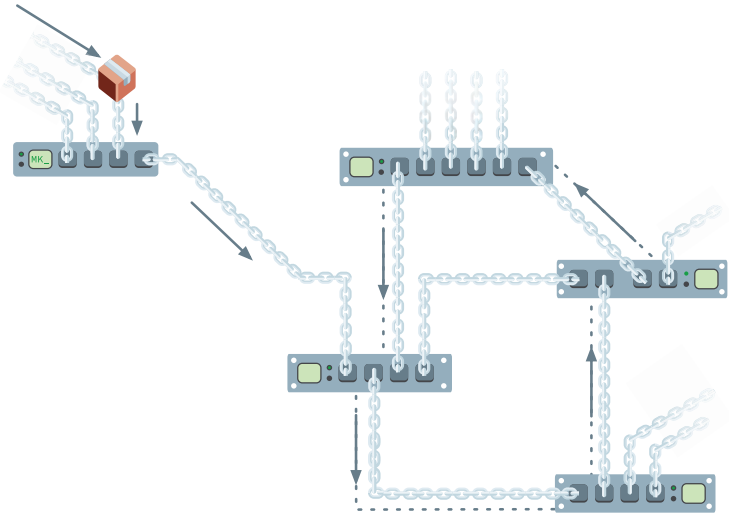


Рис. 1.25. Ошибочные таблицы, отправляющие пакет по кругу

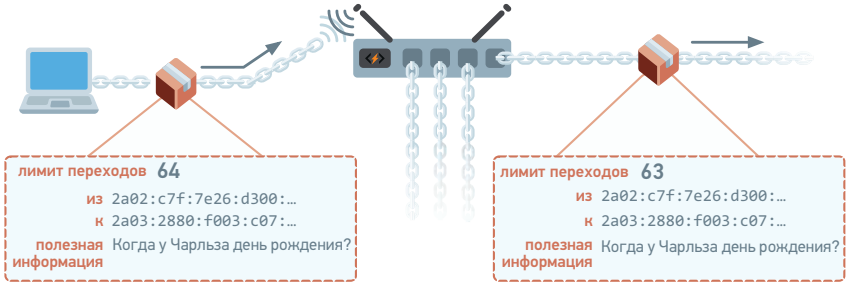


Рис. 1.26. Лимит переходов пакета — это единственный элемент IP-пакета, который маршрутизаторы изменяют при пересылке

Если пакет движется по кругу, его лимит переходов в итоге исчерпывается. Если маршрутизатор получает IP-пакет с нулевым лимитом, он может его отбросить. Затем последний маршрутизатор должен передать IP-пакет, содержащий сообщение об ошибке, обратно отправителю, указав, что пакет невозможно доставить из-за достижения лимита переходов.

Обратная связь через подобные сообщения об ошибках помогает администраторам сети исправлять критические ошибки, и петли маршрутизации — не единственные из них. Фактически интернет-протокол описывает, как решать различные проблемы маршрутизации.

ДИАГНОСТИКА

Маршрутизаторы отбрасывают IP-пакеты, которые не могут обработать. Когда это происходит, они отправляют информационное сообщение об инциденте отправителю пакета. Интернет-протокол определяет, как маршрутизаторы должны форматировать такие сообщения, чтобы их мог понять любой компьютер. Эти правила представляют собой подмножество интернет-протокола, называемое протоколом управления сообщениями в сети Internet (**I**nternet **C**ontrol **M**essage **P**rotocol, **ICMP**).

ICMP назначает коды ошибок наиболее распространенным проблемам маршрутизации. Чтобы сообщить о проблеме, маршрутизатор отправляет IP-пакет, содержащий код ошибки в теле сообщения, отформатированный в соответствии с правилами ICMP. Посмотрим на некоторые распространенные проблемы, о которых можно сообщить с помощью ICMP, начиная с проблемы петли маршрутизации.

ПРЕВЫШЕНИЕ ВРЕМЕНИ Если маршрутизатор получает IP-пакет с нулевым лимитом переходов, время его прохождения истекло. Либо пакет застрял в петле маршрутизации, либо отправитель назначил ему недостаточный лимит переходов.

В таких случаях сообщение ICMP с кодом ошибки, указывающим на исчерпание времени, отправляется обратно. Сообщение ICMP включает в себя первые байты отклоненного пакета, чтобы исходный отправитель мог знать, какой именно пакет не добрался до места назначения (рис. 1.27).

Обратите внимание, что IP-пакет, который Чарльз отправляет обратно, на рис. 1.27 включает поле протокола. Это двузначное шестнадцатеричное число, определяющее, как следует интерпретировать полезную нагрузку пакета. Последней версии ICMP был присвоен номер протокола 0x3A. Все IP-пакеты должны включать номер протокола. Больше об этом мы узнаем в следующем разделе, а пока рассмотрим другие распространенные проблемы маршрутизации.

МЕСТО НАЗНАЧЕНИЯ НЕДОСТИЖИМО Иногда маршрутизатору некуда отправить пакет. Это может произойти по разным причинам, например, если IP-адрес отсутствует в таблице адресов маршрутизатора и таблица не предлагает следующий переход по умолчанию. Иногда следующий переход указывает на выключенное устройство.

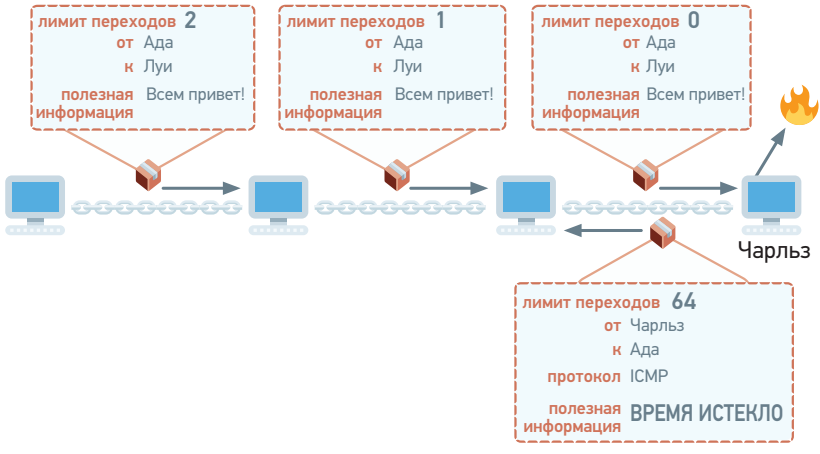


Рис. 1.27. Как только маршрутизатор получает пакет с нулевым лимитом переходов, он отбрасывается и отправителю пакета отправляется сообщение об ошибке ICMP

Когда маршрутизатор не знает, куда пересылать пакет, он возвращает ICMP-сообщение с кодом ошибки, указывающим, что **пункт назначения недоступен**, сопроводив его первыми байтами содержимого отброшенного пакета.

ПРЕВЫШЕНИЕ РАЗМЕРА Мы видели, что протоколы канального уровня ограничивают объем данных, которые могут быть отправлены в одном кадре. Кадры из разных типов сетевых каналов могут нести полезную информацию разного размера.

Максимальное количество байтов полезной информации, которое может быть перенесено в одном кадре, называется его максимальным передаваемым модулем данных (**M**aximum **T**ransmission **U**nit, **MTU**). Различные протоколы канального уровня имеют разные значения MTU. Для кадров Ethernet MTU он составляет 1500. Для кадров Wi-Fi это уже 2305.

Если маршрутизатор получает пакет большего размера, чем может обработать следующий переход, такой пакет не может быть перенаправлен в существующем виде. Вместо этого маршрутизатор возвращает сообщение ICMP с кодом ошибки, указывающим, что **пакет слишком большой**, первые байты проблемного пакета и MTU следующего перехода. Проинформированный таким образом отправитель может обрезать или разделить исходное сообщение на более мелкие пакеты перед повторной попыткой (рис. 1.28).

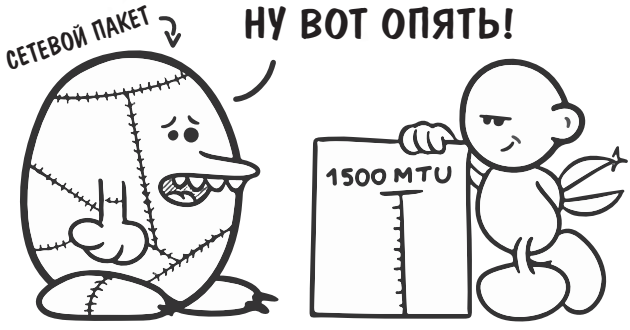


Рис. 1.28. MTU, любезно предоставлено Даниэлем Стори (http://turnoff.us)

ПРОБЛЕМА В ПАРАМЕТРАХ IP-пакет, помимо полезной нагрузки, содержит много дополнительной информации. Мы уже видели, что он содержит IP-адреса, лимит переходов и номер протокола. Он также включает в себя поле с указанием размера полезной информации и еще одно, содержащее поддерживаемую версию интернет-протокола. Есть также дополнительные поля, помогающие маршрутизаторам расставить приоритеты для важных пакетов.

Все они должны быть упорядочены и отформатированы в соответствии со строгими правилами. Когда маршрутизатор получает пакет, не соответствующий протоколу, он возвращает сообщение ICMP с кодом ошибки, указывающим на **проблему в параметрах** и место в пакете, где был обнаружен конфликт. Как обычно, сообщение ICMP также содержит несколько байтов отброшенного пакета для целей идентификации.

ИНФОРМАЦИОННЫЕ СООБЩЕНИЯ Отчеты об ошибках — не единственные сообщения, которые ICMP определяет для проверки и диагностики неисправных компьютерных сетей. В частности, широко используется пара информационных сообщений типа **«эхо-запрос»** и **«эхо-ответ»**. Когда компьютер получает эхо-запрос ICMP, он возвращает пакет, содержащий эхо-ответ ICMP.

Это полезно, чтобы проверить, подключен ли компьютер к сети. Есть программа под названием `ping`, которая отправляет сообщение с эхо-запросом ICMP и измеряет, сколько времени потребовалось на получение ответа[1]. Кроме того, отправляя эхо-запросы ICMP с различными началь-

[1] Здесь вы можете отправлять ICMP-пакеты: http://code.energy/ping.

ными лимитами переходов, вы можете отслеживать маршруты пакетов, по которым они достигают места назначения[1].

1.5. ТРАНСПОРТНЫЙ УРОВЕНЬ

Мы видели, что компьютеры в интернете могут обмениваться сообщениями в виде полезной информации в составе IP-пакетов. Например, они могут обмениваться сообщениями ICMP. Однако настоящая сила интернета раскрывается, когда не компьютеры, а *приложения* начинают отправлять друг другу данные в разделе полезной информации IP-пакета.

Компьютеры в сети часто называют **хостами**, потому что они попросту *владеют* (англ. host — «хозяин, владелец») приложениями, использующими сеть. Например, на смартфоне могут размещаться приложения для одновременной потоковой передачи музыки, просмотра веб-страниц и получения электронной почты.

Как правило, приложение использует свой хост совместно с другими приложениями, но не хочет перебирать все входящие IP-пакеты. Чтобы решить эту проблему, приложения просят свои хосты отправлять и получать пакеты от их имени. Таким образом, хосты направляют полезную информацию каждого пакета в нужное приложение, и каждое приложение получает только те данные, которые предназначены ему.

Это решение требует, чтобы данные, отправляемые приложением, сопровождались некоторой дополнительной информацией. **Транспортный уровень** определяет, как форматировать и интерпретировать эту дополнительную информацию в полезной информации IP-пакета.

ПРОТОКОЛ ПОЛЬЗОВАТЕЛЬСКИХ ДЕЙТАГРАММ

Простейшим протоколом транспортного уровня является протокол пользовательских дейтаграм (**U**ser **D**atagram **P**rotocol, **UDP**). Приложениям, использующим UDP, хостами назначаются **номера портов**. Сообщениям

[1] Объяснение того, как ICMP используется для отслеживания маршрутов, по которым проходят IP-пакеты, можно найти по адресу http://code.energy/traceroute.

для обмена предшествуют номера портов как отправляющего, так и принимающего приложений.

Для связи через UDP приложение сначала создает **сокет**. Сокет — это канал связи между приложением и его хостом. Предположим, на сервере Facebook запущено приложение-календарь, использующее UDP-порт номер 18. Приложение, работающее на хосте Ады, может отправить сообщение этому гипотетическому приложению Facebook следующим образом:

```
socket ← SOCKET.new(IP, UDP)
message ← "When is Charles' birthday?"
facebook_address ← 2a03:2880:f003:c07:face:b00c::2
app_port ← 18
socket.sendto(message, facebook_address, app_port)
```

При отправке этого сообщения хост Ады автоматически назначает неиспользуемый номер порта вновь созданному сокету. Предположим, он выбирает номер порта `54321`. IP-пакет, который передается при вызове `sendto()`, будет выглядеть так, как на рис. 1.29.

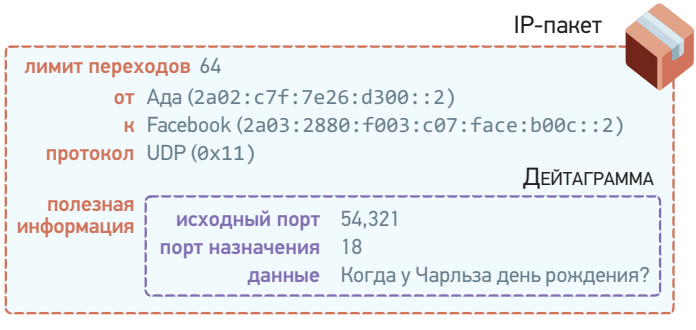


Рис. 1.29. Когда полезная нагрузка IP-пакета соответствует формату UDP, это называется дейтаграммой. Пакеты, содержащие дейтаграммы, имеют номер протокола 0x11

Хост может обслуживать тысячи различных сокетов для своих приложений. Одно приложение может даже создать несколько сокетов для параллельной работы нескольких каналов связи. Каждому сокету хост назначает свой номер порта.

Рассмотрим наше гипотетическое приложение-календарь на хосте Facebook. Получив первоначальный контакт Ады, он возвращает

дейтаграмму с портом назначения 54 321 в пакете для IP-адреса Ады. Приложение Ады ожидает эту дейтаграмму, вызывая другой метод сокета:

```
received ← socket.recv()
```

Он останавливает приложение Ады до тех пор, пока дейтаграмма не поступит на порт 54 321. Как только это произойдет, входящие данные сохранятся в `received` и приложение возобновит работу.

КЛИЕНТ И СЕРВЕР Чтобы два приложения могли взаимодействовать, одно должно ожидать контакта, а другое — инициировать его. Мы называем первое **сервером**, а второе — **клиентом**. В нашем примере приложение Facebook является сервером, а приложение Ады — клиентом. Однако приложение Ады также может выступать в роли сервера. Оно должно просто создать сокет, привязать его к порту, на котором ожидается получение дейтаграммы, и дождаться контакта:

```
socket ← Socket.new(IP, UDP)
socket.bind(17)
received ← socket.recv()
```

Вы сами выбираете номер порта вашего сервера. Если у другого приложения на хосте уже есть сокет, привязанный к этому порту, `bind()` выдаст ошибку. Когда вызывается `recv()`, приложение останавливается, пока не прибудет дейтаграмма с номером вашего порта. Наряду с данными дейтаграммы в переменной `received` хранятся IP-адрес отправителя и номер порта источника. Таким образом, приложение может ответить, если это необходимо:

```
received ← socket.recv()
x ← process_data(received.data)
socket.sendto(x, received.src_ip, received.src_port)
```

КОНТРОЛЬНАЯ СУММА UDP Мы видели, что каждый компьютер и маршрутизатор в сети могут проверять целостность полученных кадров Ethernet и Wi-Fi благодаря своему полю FCS. Дейтаграмма внутри IP-пакета передается в разных кадрах для каждого канала, поэтому FCS изменяется при каждом переходе. К сожалению, это означает, что хост, получающий дейтаграмму, не может узнать, все ли FCS были правильно сгенерированы и проверены.

Для решения этой проблемы каждая дейтаграмма содержит свой собственный модуль проверки целостности, называемый **контрольной**

суммой UDP. Эта контрольная сумма генерируется только один раз и проверяется один раз отправляющим и принимающим хостами соответственно. Поврежденные дейтаграммы автоматически отбрасываются принимающим узлом, поэтому его приложения могут быть уверены, что входящие данные не были случайно повреждены.

ОГРАНИЧЕНИЯ UDP Дейтаграмма должна быть достаточно короткой, чтобы поместиться в объем полезной информации одного IP-пакета. Поскольку MTU для большей части интернета составляет 1500 байт, большинство дейтаграмм разработано с учетом этого ограничения. «Тяжелое» сообщение, такое как большая фотография, должно быть отправлено посредством нескольких IP-пакетов.

Напомним, что любой IP-пакет на своем пути может наткнуться на перегруженный маршрутизатор и оказаться отброшенным. Иногда хосты даже не уведомляются об исчезновении их пакетов. Кроме того, обработка пакетов маршрутизаторами может привести к их беспорядочной или дублированной доставке. Это затрудняет восстановление данных, разделенных на несколько дейтаграмм.

UDP лучше всего подходит для приложений, запросы и ответы которых умещаются в одной дейтаграмме. В идеале приложение должно отправлять по одному запросу за раз и посылать следующую дейтаграмму только после получения ответа. Если ответ не приходит слишком долго, приложение должно либо отказаться, либо попытаться отправить ту же дейтаграмму еще раз.

Дейтаграммы также подходят для передачи потоков данных, в которых допустима случайная потеря данных, например, при передаче телефонного разговора в реальном времени, когда небольшие сбои звука не имеют особого значения.

ПРОТОКОЛ УПРАВЛЕНИЯ ПЕРЕДАЧЕЙ ДАННЫХ

Транспортный уровень способен на большее, нежели просто сопоставлять данные из IP-пакетов с соответствующими сокетами приложений. При использовании протокола управления передачей данных (**T**ransmission **C**ontrol **P**rotocol, **TCP**) вместо UDP хосты добавляют *еще больше* дополнительной информация к полезной нагрузке IP-пакета. Это позволяет TCP предлагать функции, повышающие надежность связи между приложениями.

С помощью TCP приложения обмениваются данными, как если бы существовала прямая линия связи с другим приложением, даже несмотря на то, что IP-пакеты имеют ограниченный размер и маршрутизаторам нельзя доверять их доставку. За кулисами хосты разделяют данные на более короткие фрагменты и обрабатывают потерянные, повторяющиеся и неупорядоченные пакеты. Приложение же просто запрашивает у своего хоста соединение с другим приложением, а затем запрашивает отправку или получение его данных.

UDP-сокеты можно сравнить с почтовыми ящиками, которые отправляют и получают короткие единичные сообщения в другие ящики и из них без гарантии доставки. Сокеты TCP — совсем другое дело. Представьте активный сокет TCP как один конец виртуального канала, ведущего к другому сокету. Все данные, отправленные через сокет TCP, будут доставлены без искажений. Посмотрим, как можно выстроить такую систему поверх ненадежной системы доставки IP-пакетов.

СЕГМЕНТЫ TCP

С помощью TCP данные приложения разделяются и передаются через фрагменты полезной информации, называемые **сегментами**. Как и в дейтаграммах UDP, сегменты TCP содержат данные, дополненные вспомогательной информацией (рис. 1.30).

Рис. 1.30. Упрощенное представление сегмента. Пакеты, содержащие сегменты, имеют номер протокола 0x6

Каждое из этих полей необходимо TCP, чтобы гарантировать целостность и полноту пересобранных данных.

НОМЕРА ПОРТА Хост обычно управляет несколькими каналами TCP для различных приложений. Как только сегмент прибывает, у хоста должен быть способ сопоставить его с каналом связи, которому он принадлежит. С этой целью сегменты содержат номера портов источника и назначения. Канал связи TCP может быть однозначно идентифицирован четырьмя числами: IP-адресами хостов и двумя номерами портов — по одному для каждого хоста.

ПОРЯДКОВЫЙ НОМЕР Хост-отправитель разбивает данные приложения на последовательность небольших фрагментов. Каждому фрагменту он присваивает **порядковый номер**, чтобы указать, как именно они должны быть упорядочены. Каждый фрагмент помещается в сегмент с соответствующим порядковым номером, и сегменты передаются один за другим в индивидуальных IP-пакетах. Если сегменты поступают не по порядку, принимающий хост упорядочивает их, используя их порядковые номера. Если сегмент поступает дважды, хост обнаружит повторяющийся порядковый номер и отбросит повторяющиеся данные.

НОМЕР ПОДТВЕРЖДЕНИЯ Связь с TCP двунаправленная: удаленные приложения должны обмениваться сегментами *одновременно*. Для отслеживания того, какой сегмент получил каждый хост, каждый из них включает **номер подтверждения**. Он соответствует порядковому номеру следующего сегмента, который хост ожидает получить от другой стороны. Например, если Чарльз отправляет сегмент с номером подтверждения `44` Аде, он тем самым признает, что получил все ее сегменты до порядкового номера `43`. Если у него нет данных для отправки, он все равно должен отправлять сегменты без данных для подтверждения (рис. 1.31).

После отправки непустого сегмента отправляющий хост запускает таймер для получения его подтверждения. Если получение подтверждения занимает слишком много времени, отправитель предполагает, что пакет был утерян, повторно передает неподтвержденный сегмент и сбрасывает таймер.

Хосты отслеживают время, необходимое для получения подтверждений. Когда время увеличивается, это верный признак того, что сеть перегружена, поэтому хосты снижают скорость, с которой отправляют сегменты. Когда время подтверждения уменьшается, это признак того, что доступна

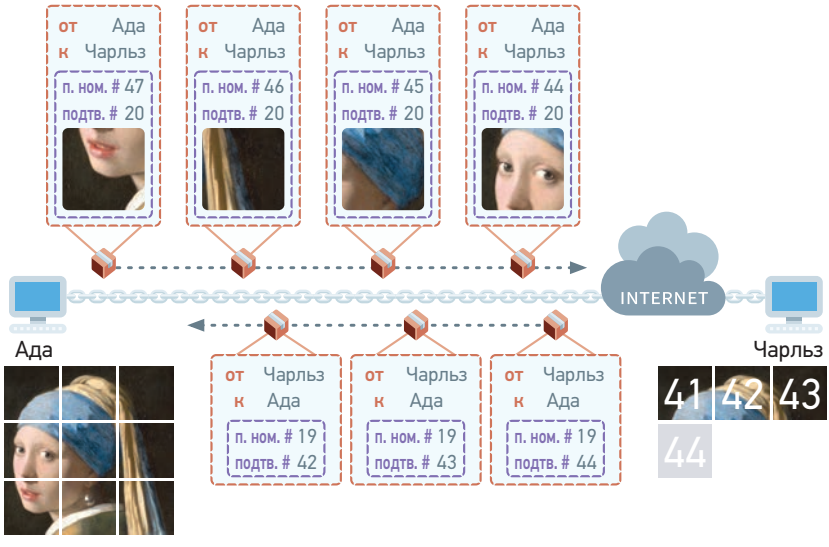


Рис. 1.31. Чарльз получает изображение от Ады. У него нет данных для нее, поэтому он отправляет пустые сегменты подтверждения. Обратите внимание, что порядковые номера его сегментов не увеличиваются. Отправляя свои данные, Ада сигнализирует, что готова принять сегмент # 20 Чарльза

более широкая пропускная полоса, так что хосты отправляют сегменты с большей скоростью. Это **контроль перегрузки**, и он гарантирует, что пропускная способность сети используется надлежащим образом.

РАЗМЕР ОКНА Если хост получает слишком много сегментов одновременно, его вычислительные ресурсы могут быть перегружены. Чтобы гарантировать, что они всегда способны обрабатывать входящие данные, хосты включают **размер окна** вместе с номером подтверждения. Он указывает количество байтов данных, которые они хотят получить после последнего подтвержденного сегмента.

Отправители приостанавливают передачу неподтвержденных сегментов, когда их общий размер достигает размера окна принимающей стороны. Это называется **управлением потоком** и гарантирует, что вычислительные возможности каждого хоста не превышаются. Используя TCP, приложениям не нужно беспокоиться об управлении перегрузкой и потоком, поскольку об этом заботятся их хосты. С другой стороны, приложения, использующие UDP, отвечают за адаптацию скорости отправки к пропускной способности сети и вычислительным ресурсам на другой стороне.

КОНТРОЛЬНАЯ СУММА TCP Как и в случае дейтаграмм UDP, сегменты TCP включают контрольную сумму. Она добавляется хостом-отправителем и проверяется хостом-получателем. Если получен поврежденный сегмент, он отбрасывается без подтверждения. Это гарантирует, что все фрагменты данных в итоге дойдут до места назначения без ошибок.

TCP-СОЕДИНЕНИЕ

Связь TCP всегда осуществляется через пару сегментных потоков, называемых **TCP-соединением**, как показано на рис. 1.31. Если два приложения хотят начать обмен данными через TCP-соединение, их хосты должны уметь распознавать и сортировать сегменты этого соединения. Они делают это, запоминая, какие номера портов соответствуют какому потоку. Каждый хост также выбирает начальный порядковый номер для запуска своей стороны соединения и подтверждает начальный порядковый номер другой стороны.

По этим причинам для установления TCP-соединения требуется обмен тремя сегментами между клиентом и сервером. Первые два сегмента несут специальный флаг SYN[1], который служит синхронизации порядковых номеров и номеров подтверждений потоков. Кроме того, флаг ACK переносится теми сегментами, чьи **номера подтверждения** (acknowledgment) действительны и синхронизированы. Третий сегмент уже может быть использован для данных приложения (рис. 1.32).

Прежде чем произойдет обмен сегментами на рис. 1.32, Чарльз должен попросить свой хост ожидать входящие запросы на TCP-соединение через порт номер 20. Когда Ада запрашивает у своего хоста новое соединение с Чарльзом через порт 20, хост Ады случайным образом выбирает номер порта для назначения соединению — в данном случае был выбран порт 61 311.

Пока соединение активно, хосты отслеживают порядковые номера, номера подтверждений и размеры окон. Они рассчитывают время получения подтверждений от другой стороны и решают, когда отправлять следующие сегменты. Приложение же, в свою очередь, пребывает в блаженном неведении касательно всей этой работы, поскольку оно просто вызывает методы сокета TCP для подключения, отправки и получения.

[1] Поля, закодированные единицей или нулем, называются флагами.

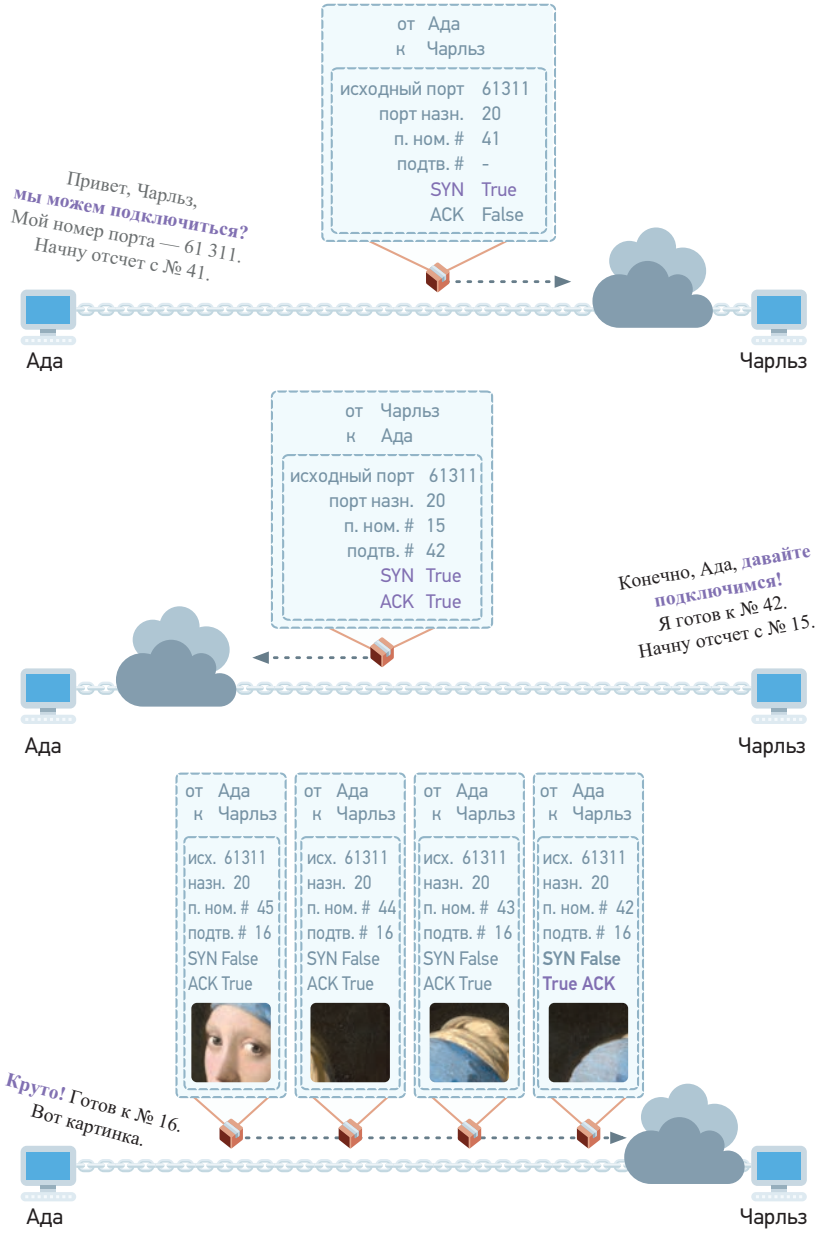


Рис. 1.32. Хосты инициируют TCP-соединения со случайными порядковыми номерами. Мы считаем, что соединение полностью установлено с момента получения третьего сегмента, поскольку оба хоста получили подтверждение своего порядкового номера

TCP-СОКЕТЫ

Сокеты TCP могут быть активными или пассивными. К соединению можно привязать только активные сокеты. Когда TCP-сокеты создаются, они активны по умолчанию. Вот как клиент запускает соединение:

```
active_socket ← Socket.new(IP, TCP)
port_number ← 80
server_ip ← 2a03:2880:f003:c07:face:b00c::2
active_socket.connect(server_ip, port_number)
active_socket.send("When is Charles' birthday?")
```

Клиенту нужно просто указать IP-адрес хоста-сервера и номер порта, который этот сервер прослушивает. Все процедуры по запуску и поддержанию TCP-соединения выполняются хостом за кулисами. После успешного выполнения `connect()` могут быть вызваны `send()` и `recv()` для обмена данными.

Чтобы ожидать и принимать запросы на соединение с заданным номером порта, необходимо использовать пассивный сокет. Сокет становится пассивным после привязки к номеру порта и вызова его метода `listen()`. Вот как сервер ожидает подключения к порту 80:

```
server_socket ← Socket.new(IP, TCP)
server_socket.bind(80)
server_socket.listen()
active_socket ← server_socket.accept()
```

Сервер выбирает номер своего порта. Если у другого приложения уже есть сокет, привязанный к тому же порту, `bind()` выдаст ошибку.

Когда вызывается `accept()`, сервер останавливается до тех пор, пока не поступит новый запрос на соединение, после чего создается новое TCP-соединение. Новое соединение привязывается к только что созданному активному сокету. Исходный пассивный сокет можно использовать для приема других подключений к другим хостам. Вновь созданный активный сокет можно использовать для связи с клиентом путем вызова для него `send()` и `recv()`.

Часто серверы продолжают работу `accept()` в цикле, чтобы установить множество TCP-соединений с несколькими клиентами и общаться со всеми ними одновременно. Именно так веб-сервер может одновременно обслуживать тысячи или даже миллионы различных клиентских приложений.

Это был очень упрощенный обзор того, как работают TCP-соединения. Мы не обсудили множество пограничных случаев. Сегменты несут дополнительные флаги, указывающие на срочность, состояния ошибок, перегрузку сети, завершение соединения и т. д. Хотя TCP был первоначально разработан вскоре после IP в 1974 году, его внутренняя работа меняется по сей день — например, для улучшения обработки перегрузок и оптимизации использования полосы пропускания.

Если вы не специалист по работе сетей, сокеты TCP позволят вам не думать обо всех этих деталях. Сокеты — хороший пример здоровой разработки систем: они предоставляют простой интерфейс для выполнения сложных операций с минимальным знанием внутренних особенностей.

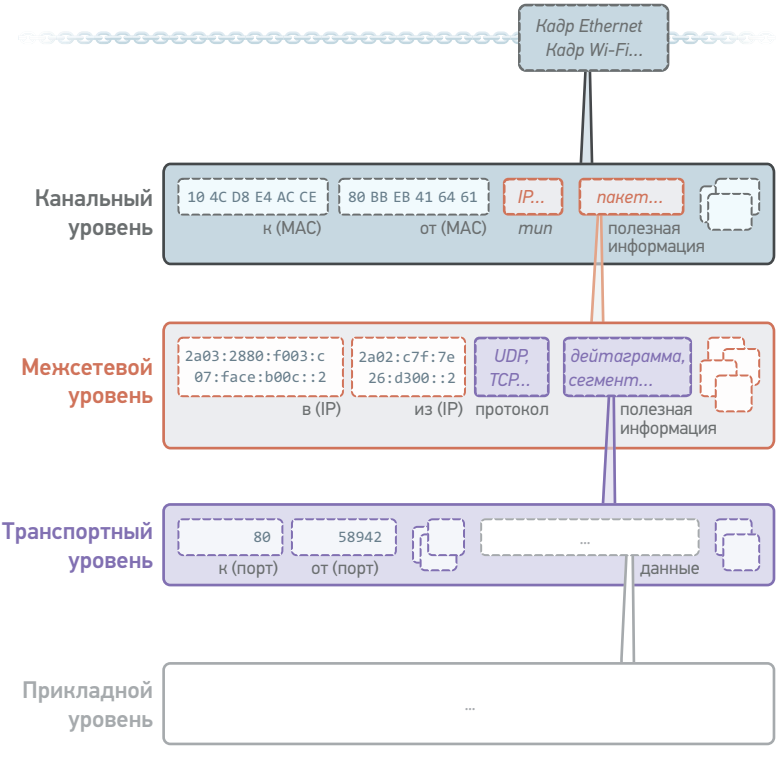
РЕЗЮМЕ

Интернет — одно из величайших технологических достижений человечества, результат огромного количества инженерных разработок и международного сотрудничества. Компьютерные программы практически в любой точке мира могут безошибочно и легко взаимодействовать друг с другом без централизованной координации.

Такие взаимодействия используют физические и виртуальные соединения на разных уровнях аппаратного и программного обеспечения. Поэтому, когда приложения обмениваются данными, вполне естественно, что их данные проходят несколько уровней упаковки (см. рис. на с. 61).

КАНАЛЬНЫЙ УРОВЕНЬ Антенны, кабели и другое оборудование для управления электромагнитными волнами позволяют парам компьютеров передавать сигналы друг другу. Протоколы канального уровня позволяют парам компьютеров использовать эти сигналы для обмена данными в кадрах. Помимо прочего, каждый кадр содержит MAC-адреса отправителя и получателя, полезную нагрузку и поле, указывающее правила обработки полезной нагрузки на уровне интернета.

МЕЖСЕТЕВОЙ УРОВЕНЬ Специальные компьютеры, называемые маршрутизаторами, способны объединять разные сети в более крупные, такие как интернет. Благодаря интернет-протоколу полезные данные, упакованные в виде IP-пакетов, могут переходить от одного кадра к другому и, таким образом, перемещаться между компьютерами, которые не имеют прямого физического соединения. Каждый пакет содержит IP-адреса



своего отправителя и конечного получателя, полезную нагрузку и поле, указывающее правила обработки этой полезной нагрузки на транспортном уровне.

ТРАНСПОРТНЫЙ УРОВЕНЬ На удаленных компьютерах размещены приложения, которым нужно обмениваться данными. Благодаря таким протоколам, как UDP и TCP, полезные данные, выраженные в виде дейтаграмм или сегментов, могут формировать потоки данных между удаленными приложениями. Помимо прочего, каждая дейтаграмма или сегмент данных содержит фрагмент данных приложения вместе с портами источника и назначения, через которые эти данные проходят.

И TCP, и UDP используются по-разному в зависимости от характера связи между приложениями. В главе 2 мы рассмотрим **ПРИКЛАДНОЙ УРОВЕНЬ**: как мы применяем IP, TCP и UDP для создания множества современных интернет-сервисов, которые мы используем каждый день, таких как электронная почта и World Wide Web.

Такая многоуровневая архитектура сетей существует уже давно. В 1975 году IP-пакеты, несущие полезную нагрузку TCP, уже передавались через страны между компьютерами в Стэнфорде и Лондоне. Протоколы развиваются — IP получил обновление в 2012 году, а TCP многократно подвергался настройке, — но фундаментальная механика интернета не изменялась на протяжении десятилетий.

В этой главе мы рассмотрели основные сетевые концепции, которые, вероятно, будут поддерживать нашу связь еще долгие годы. Теперь узнаем, как можно использовать подключение к интернету для цифровых услуг, таких как электронная почта и World Wide Web.

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

- Warriors of the Net. http://code.energy/net-warriors.
- *Severance C.* Introduction to Networking. http://code.energy/severance.

Глава 2

ОБМЕН ДАННЫМИ

> Интернет работает потому, что множество людей объединяются, чтобы делать что-то вместе.
>
> *Джон Постел*

Благодаря голосу мы можем посылать звуковые сигналы другим, но этого недостаточно для общения. Чтобы два человека могли обмениваться мыслями посредством речи, они должны использовать общий язык. Так и транспортные протоколы позволяют передавать данные между приложениями, но этого недостаточно, чтобы приложения понимали друг друга и работали в связке. Помимо подключений, приложениям нужны правила взаимодействия.

Многие правила связи, появившиеся на заре интернета, до сих пор широко используются приложениями. Чтобы ваши приложения работали в интернете, важно познакомиться с некоторыми из них. В этой главе вы научитесь:


давать читаемые **имена** онлайн-адресам;


синхронизировать часы с универсальным **временем**;


получать доступ и использовать удаленные компьютеры;


отправлять и получать **почту** в электронном виде;


просматривать **интернет** цифровых документов и услуг.

Каждый независимый набор правил связи образует протокол, а совокупность всех таких протоколов составляет **прикладной уровень**. Первый протокол прикладного уровня, который мы изучим, работает как адресная книга, к которой прямо или косвенно обращается почти каждое приложение, подключенное к интернету.

2.1. ИМЕНА

Мы можем отправить IP-пакет на хост, только если знаем его IP-адрес. Тем не менее мы никогда не запоминаем IP-адреса основных хостов, таких как Google, Amazon или Facebook. Инженеры поняли, что ссылаться на хосты с использованием больших чисел непрактично, поэтому создали Службу доменных имен (**D**omain **N**ame **S**ystem), или **DNS**, которая связывает удобочитаемые имена с IP-адресами. Благодаря DNS мы можем связываться с хостами, используя такие имена, как `google.com`, `amazon.com` или же `facebook.com`.

Когда вы печатаете `facebook.com` в браузере, он использует DNS для поиска IP-адреса, связанного с этим именем. Затем он может запросить веб-сайт, отправив IP-пакеты на этот адрес. Если вы напрямую введете IP-адрес Facebook в браузере, общения с реестром DNS не произойдет, но та же самая веб-страница все равно отобразится[1].

DNS зависит от взаимодействия множества компьютеров в интернете. Эти компьютеры обмениваются данными в соответствии с протоколом DNS для ведения этого реестра имен. Большинство интернет-приложений, таких как браузеры и почтовые клиенты, используют DNS для поиска IP-адресов хостов, с которыми им необходимо соединиться. Но прежде, чем приступить к изучению этого протокола, обратимся к некоторым основным концепциям, начав с иерархической структуры имен.

ДОМЕНЫ

Имена в DNS называются **доменными именами** или **доменами**. Они состоят из текстовых меток, разделенных точками. Метки могут использовать 26 букв от `a` до `z`, цифры от `0` до `9`, а также дефис (`-`). Например, `book-2.code.energy` — это домен с тремя метками.

[1] Убедитесь сами, посетив сайт http://[2a03:2880:f003:c07:face:b00c::2].

Точки обозначают иерархические поддомены. Это домены, контролируемые более коротким родительским доменом: `book-2.code.energy` является поддоменом `code.energy`. Кроме того, `code.energy` является поддоменом однокомпонентного домена `energy`. Домены нечувствительны к регистру: `code.energy` и `coDE.enERgy` одинаковы.

У каждого домена имеется владелец. Владельцы могут создавать поддомены собственных доменов. Право собственности на поддомен может быть передано в любое время владельцем родительского домена. Например, Verisign — американская компания, владеющая как `com`, так и `net`. В 1997 году по просьбе небольшой команды ботанов Verisign создала `google.com` и предоставила им право собственности на него.

Домены из одной метки, такие как `com`, `net`, а также `energy`, называются доменами высшего уровня (**T**op **L**evel **D**omains, **TLDs**). Благодаря своему устройству DNS наиболее эффективен при ограниченном количестве TLD. По этой причине создание новых TLD представляет сложность. Обычно, когда частным лицам и организациям нужен домен, они запрашивают поддомен у владельца существующего TLD, и чаще всего это `com`[1].

Тем не менее многие влиятельные организации обладают собственными TLD. В 2014 году Google получил в свое распоряжение TLD `Google`[2]. В 2016 году Motorola получила `moto`. Что интересно, Amazon пытается получить TLD `amazon`, но правительства Бразилии и Перу выступают против этого, поскольку считают, что название принадлежит в первую очередь их тропическим лесам.

ICANN

Чтобы доменные имена понимались одинаково, у всех должен быть способ договориться о том, кто и каким TLD владеет. Это требует как согласования процессов создания TLD, так и разрешения конфликтов по поводу их владения. С этой целью было решено, что американская некоммерческая организация **I**nternet **C**orporation for **A**ssigned **N**ames and **N**umbers (**ICANN**)[3] получает полный контроль над TLD.

---

[1] Вы можете найти список TLD на http://code.energy/tlds.

[2] Посетите сайт http://about.google, чтобы убедиться в этом.

[3] ICANN контролирует IANA, организацию, которая назначает префиксы IP-адресов.

Когда ICANN санкционирует создание TLD, она, как правило, требует, чтобы новый владелец разрешил широкой публике регистрировать поддомены. Так обстоит дело с большинством существующих TLD, включая `com`, `net`, а также `energy`. Чтобы Verisign сохранила свое право собственности на `com`, ICANN требует от них принимать запросы на создание `.com` поддоменов от кого угодно.

Иногда ICANN позволяет владельцам TLD использовать их в исключительных интересах. Например, `gov` и `mil` контролируются правительством США. Они создают домены только для американских государственных органов, таких как `nasa.gov`, а также `spaceforce.mil`. Точно так же владелец `edu` позволяет только американским образовательным учреждениям иметь домен `.edu`, например `mit.edu` или `caltech.edu`. А Google использует свой TLD исключительно для корпоративных целей, например `Grow.google` или `blog.google`.

Помимо TLD для общего и исключительного использования, ICANN назначает двухбуквенные **TLD**, кодирующие страны, и уступает контроль над ними их правительствам. Для Колумбии это `co`, в Соединенных Штатах `us`, а Британская территория в Индийском океане имеет домен `io`[1]. Иногда ICANN создает TLD для известных географических регионов. Например, `rio` был передан городскому совету Рио-де-Жанейро.

СЕРВЕРЫ ИМЕН

Записи DNS связывают данные с доменами. В домене может быть множество записей, каждая из которых связывает часть данных с доменом. Все записи имеют код, который указывает на тип информации, которую они несут. Например, записи `AAAA` связывают домен с IP-адресом[2]:

```
sprint.net. AAAA 2600::
```

[1] TLD назначаются в соответствии с кодами стран ISO 3166-1 alpha-2, за некоторыми исключениями, такими как Великобритания. Их код страны — GB, но им выдали домен uk.

[2] Обратите внимание на домен sprint.net. В официальных записях все домены заканчиваются точкой. В повседневном использовании последнюю точку чаще всего для удобства опускают.

Все записи DNS должны храниться на **серверах имен**. Это специальные хосты, которые призваны делиться своими записями с кем угодно в интернете в любое время. Владелец домена должен предоставить как минимум двум серверам имен право хранить свои записи DNS. Владельцы доменов могут предоставить эти серверы сами или нанять для этого третьих лиц.

Право собственности на домены осуществляется через записи DNS. Например, вспомним, что Verisign владеет `com`. В принципе, серверы имен Verisign должны содержать все записи для домена `.com`, в том числе `google.com`. Однако это не так, потому что Verisign хранит следующие записи на своих серверах имен:

```
google.com.        NS ns1.google.com.
google.com.        NS ns2.google.com.
ns1.google.com. AAAA 2001:4860:4802:32::a
ns2.google.com. AAAA 2001:4860:4802:34::a
```

Запись `NS` говорит о том, что данный сервер имен управляет данным доменом. Указанными выше записями Verisign делегирует ответственность за хранение всех записей, связанных с `google.com`. Verisign фактически передает контроль над доменом Google, где размещены указанные серверы имен. Это дает Google право устанавливать записи DNS для `google.com` на собственных серверах имен.

КОРНЕВЫЕ СЕРВЕРЫ Специальным серверам имен, называемым **корневыми**, ICANN дает право хранить записи DNS касательно TLD. Новые TLD создаются, когда ICANN запрашивает, чтобы на этих серверах была добавлена запись `NS`. Существует 13 корневых серверов — ICANN следит, чтобы их IP-адреса были хорошо известны. Университет Мэриленда, НАСА и армия США — вот некоторые из организаций, где размещены корневые серверы.

ЗАПРОС

Серверы имен имеют две функции: хранить записи и отвечать на запросы. Большинство интернет-приложений в своей работе полагаются именно на них. Например, ваш браузер работает только в том случае, если у него есть способ определять IP-адреса сайтов, на которые вы хотите перейти.

Серверы имен ожидают запросы на порт UDP 53. Обычно сообщение запроса DNS и ответ на него помещаются в одну дейтаграмму. Чтобы проиллюстрировать, как это работает, запросим серверы имен, используя программу `dig`[1]. Она отправляет на указанный нами IP-адрес дейтаграмму, содержащую запрос, соответствующий протоколу DNS. `dig` вызывается из командной строки следующим образом:

```
dig @[address] [domain] [type]
```

Эта строка отправляет дейтаграмму по указанному адресу на UDP-порт 53, запрашивая записи определенного типа, связанные с данным доменом. Когда в ответ приходит дейтаграмма, `dig` декодирует ее и отображает записи в виде обычного текста.

Предположим, мы хотим подключиться к `icmc.usp.br`, но нам известен только IP-адрес корневого сервера HACA: `2001:500:a8::e`. Давайте узнаем у HACA, какой сервер имен ответственен за `icmc.usp.br`:

```
dig @2001:500:a8::e icmc.usp.br. NS
```

Корневой сервер отвечает следующими записями:

```
br.                     NS a.dns.br.
a.dns.br.             AAAA 2001:12f8:6::10
```

Мы сразу видим, что у HACA нет `NS`-записи для `icmc.usp.br`. На самом деле не следует ожидать, что корневые серверы будут хранить эти записи: `br` — это домен верхнего уровня с кодом страны, находящийся в ведении правительства Бразилии. Корневой сервер помогает нам, указывая IP-адрес бразильского сервера, ответственного за `br`. Воспользуемся советом и отправим туда наш запрос:

```
dig @2001:12f8:6::10 icmc.usp.br. NS
```

На этот раз мы получаем конкретные записи, которые мы и запрашивали:

```
icmc.usp.br.            NS c.dns.usp.br.
c.dns.usp.br.         AAAA 2001:12d0::8
```

[1] Программа dig предустановлена на эмуляторы терминалов Linux и MacOS. Если у вас нет доступа к интерфейсу командной строки, используйте http://code.energy/dig. Если не знаете, что такое командная строка, не волнуйтесь! Поговорим о ней позже в этой главе.

Теперь мы можем запросить AAAA-запись, связанную с `icmc.usp.br`:

```
dig @2001:12d0::8 icmc.usp.br. AAAA
```

И вот наконец мы получаем желаемый IP-адрес:

```
icmc.usp.br.            AAAA 2001:12d0:2080::231:6
```

Если бы исходный запрос был поддоменом `icmc.usp.br`, возможно, потребовалось бы повторить наш запрос для `NS` записи еще раз. Это называется **итеративным запросом**, и он позволяет находить любую информацию DNS, начиная с любого IP-адреса корневого сервера.

РЕКУРСИВНЫЙ ЗАПРОС

По оценкам, в 2020 году *каждую секунду* выполняется несколько миллионов DNS-запросов. Если бы хосты выполняли только итеративные запросы, серверам имен было бы чрезвычайно сложно обрабатывать их все. К счастью, большинство DNS-запросов обрабатываются по-другому, что сводит к минимуму нагрузку на серверы имен.

Некоторые серверы имен, часто называемые **DNS-серверами**, позволяют просматривать любую запись DNS с помощью одного запроса. Обычно они поддерживаются крупными организациями с большой пропускной способностью, такими как интернет-провайдеры, которые почти всегда сообщают маршрутизаторам клиентов IP-адрес своего DNS-сервера. Маршрутизаторы клиентов часто используют этот сервер для всех своих DNS-запросов.

Сегодня большинство домашних маршрутизаторов могут иметь собственный DNS-сервер. Обычно персональные компьютеры настроены на использование своего маршрутизатора в качестве DNS-сервера по умолчанию. Если вызвать `dig` без аргумента `@`, то ваш компьютер, скорее всего, запросит ваш маршрутизатор. В свою очередь, ваш маршрутизатор, вероятно, перенаправит запрос на DNS-сервер интернет-провайдера. Попробуйте выполнить следующую команду:

```
dig icmc.usp.br. AAAA
```

Это намного быстрее, ведь ваш DNS-сервер ближе к вам, чем любой другой. Более того, с помощью одного запроса вы получите нужные

записи DNS. Как клиента, вас совершенно не заботит, запрашивал ли ваш DNS-сервер другой DNS-сервер или извлекал записи непосредственно с серверов имен, имеющих над ними контроль.

Мы называем этот процесс **рекурсивным запросом**, потому что единственный запрос клиента распространяется по цепочке DNS-серверов до тех пор, пока не будет найден ответ. Его преимущество заключается в **кэшировании**: серверы хранят результаты прошлых запросов в своем локальном хранилище, или *кэше*. Например, при первом запросе DNS-сервера о каком-либо домене `.io` он запрашивает у корневого сервера `NS`-записи `io`; однако в следующий раз, когда получен запрос о домене `.io`, он извлекает записи из своего кэша, а не тратит время корневого сервера.

Поскольку ICANN ограничивает количество TLD, интернет-провайдеры могут легко кэшировать все записи TLD на DNS-серверах и, следовательно, запрашивать корневые серверы только при необходимости. Такая схема гарантирует, что корневые серверы не окажутся перегружены.

С другой стороны, кэширование замедляет распространение изменений по сети: DNS-серверы только время от времени обновляют свои кэшированные записи NS. В частности, значение записи **T**ime-**T**o-**L**ive (**TTL**) указывает, как долго она может храниться в кэше.

В предыдущих примерах мы для простоты опускали значения TTL, но они есть у каждой записи. Обычно оно отображается перед типом записи:

```
br.             149836      NS   a.dns.br.
```

Эта запись имеет значение TTL 149 836, что указывает на то, что ее можно кэшировать на данное количество секунд. Примерно через 41 час после того, как эта запись получена DNS-сервером, она должна быть удалена из кэша. Установка низкого значения TTL ускоряет распространение будущих изменений в записи, а более высокие значения TTL позволяют серверам тратить меньше ресурсов на обновление кэша.

ТИПЫ ЗАПИСЕЙ

Мы видели два типа записей DNS: `AAAA`, которая связывает IP-адреса с доменами, а также `NS`, которая предоставляет серверам имен полномочия над доменами. Существует множество других типов записей. Обратимся к самым распространенным.

АДРЕС Когда DNS была изобретена в 1983 году, тип записи `A` использовался для IP-адресов. В 2012 году IP был обновлен, и адреса сделались в четыре раза длиннее. Для хранения новых адресов и был создан тип записи `AAAA`. Таким образом, устаревший IP-адрес не изменится при обновлении сети. Домены могут иметь обе записи, например:

```
one.one.one.one.     AAAA  2606:4700:4700::1111
one.one.one.one.        A  1.1.1.1
```

Если у вас устаревшее соединение, которое поддерживает только IPv4, вы можете запросить запись `A` и вместо пакетов Ipv6 отправлять на полученный адрес пакеты IPv4.

TLD не может иметь `A`- или `AAAA`-записи. В этом причина того, что нет сайта по адресу http://com/ или же http://google/. Имена с одной меткой используются только для адресации в локальных сетях. Самое распространенное из них — `localhost`, и оно всегда указывает на собственный сетевой интерфейс компьютера.

ПОЧТОВЫЙ ОБМЕН Если вы хотите оставить отзыв об этой книге, то можете направить электронное письмо на адрес hi@code.energy[1]. Но как ваша система электронной почты узнает, куда отправить письмо, чтобы оно достигло наших серверов?

Как и в случае с интернетом, почта зависит от DNS. `MX`-запись указывает хост, ответственный за получение почты для домена. Для отправки электронного письма по адресу maria@icmc.usp.br первым шагом будет поиск `MX`-записи для `icmc.usp.br`. Итак:

```
dig icmc.usp.br. MX
```

Каждая `MX`-запись включает имя хоста и номер приоритета:

```
icmc.usp.br.       MX 1 aspmx.l.google.com.
icmc.usp.br.       MX 5 alt1.aspmx.l.google.com.
icmc.usp.br.       MX 5 alt2.aspmx.l.google.com.
icmc.usp.br.       MX 10 alt3.aspmx.l.google.com.
```

Эти записи указывают на то, что владелец `icmc.usp.br` выбрал Google для получения почты от своего имени. Вы можете подключиться к любому

[1] Можно сделать это прямо сейчас!

из этих хостов, чтобы отправить письмо на `icmc.usp.br`, но лучше отдать предпочтение тому, у которого самый низкий приоритет.

Как и в случае с `A`- и `AAAA`-записями, TLD не могут иметь `MX`-записи. Вот почему нет адресов электронной почты вроде ada@io или larry@google.

КАНОНИЧЕСКОЕ ИМЯ На заре интернета люди начали настраивать серверы в своих компаниях для обработки запросов к веб-страницам. Такой сервер обычно назывался www, поэтому http://www.example.com/ стал ожидаемым веб-адресом для `example.com`. В последнее время многие убирают часть www из своих веб-адресов. Специальная `CNAME`-запись позволяет нам определить поддомен, который попросту выступает псевдонимом:

```
www.code.energy. CNAME code.energy.
```

Он дает указание любому, кто пытается получить запись для `www.code.energy`, вместо этого обратиться к записи для `code.energy`.

ТЕКСТ Можно связать произвольный текст с доменом, используя `TXT`-записи. Они часто применяются для подтверждения права собственности на домен, но вы можете использовать их для чего-нибудь еще. Мы оставили для вас особое сообщение в TXT-записи, связанной с `enigma.code.energy`. Сможете отыскать его? Разрешено использовать `dig` для запроса любого типа записи, связанной с любым доменом.

ОБРАТНЫЙ DNS-ЗАПРОС

DNS чаще всего используется для получения IP-адреса, связанного с данным доменным именем. Однако она также работает в обратном направлении: может найти, какой домен связан с данным IP-адресом. Хитрость заключается в том, чтобы преобразовать IP-адрес в поддомен специального TLD `arpa`, принадлежащего ICANN. Например:

```
icmc.usp.br.          AAAA 2001:12d0:2080::231:6
```

становится:

```
.0.8.2.0.d.2.1.1.0.0.2.ip6.arpa.        PTR icmc.usp.br.
```

обращенным IP-адресом
(32 шестнадцатеричные цифры)

Благодаря ICANN появился уникальный поддомен `arpa`, соответствующий каждому IP-адресу. `PTR`-запись может быть установлена для этого поддомена, чтобы связать его с доменным именем.

Поддомен `arpa` контролируется IANA. Когда организации передается блок IP-адресов, она также получает полномочия на соответствующие поддомены `arpa`. Когда компьютеру назначаются IP-адрес и доменное имя, рекомендуется записать эту пару в запись `PTR`, используя IP-адрес поддомена `arpa`.

С помощью `dig` вы можете запросить `PTR`-записи, соответствующие любому IP-адресу, используя аргумент `-x`. Попробуйте сами:

```
dig -x 2600::
```

Обращение DNS полезно для подтверждения того, откуда исходит IP-пакет. Например, предположим, что вы получили IP-пакет, содержащий сообщение от `apple.com`. Если обратный DNS-запрос IP-адреса отправителя не является доменом Apple, это уже подозрительно!

А все потому, что те, кто имеет право устанавливать DNS-записи для поддоменов `arpa`, очень осторожны. Большинство интернет-провайдеров вообще не позволяют своим клиентам устанавливать записи для поддоменов `arpa`, соответствующих их IP-адресам.

РЕГИСТРАЦИЯ ДОМЕНА

DNS не просто заменяет большие числа именами — она дает людям автономию в изменении конфигурации своих сетей. Представьте компьютер, на котором размещен веб-сайт. Если он перемещается, он получает новый IP-адрес. Записи DNS-сайта можно обновить на новый адрес, и посетители сайта даже не заметят этих изменений!

Этот подход требует от DNS универсальности. Владение доменом должно быть простым, безопасным и доступным для всех. Есть несколько правил, гарантирующих это. ICANN либерально относится к TLD, предназначенным для исключительного использования: Google и IBM используют `Google` и `ibm` по своему усмотрению. Кроме того, ICANN позволяет правительству каждой страны устанавливать правила для своего TLD с кодом страны. Однако ICANN налагает строгие правила на TLD общего

пользования, предназначенные для общественности, такие как `com`, `org`, `rocks` и `energy`.

ICANN называет организацию, которая контролирует один или несколько TLD, **реестром** или сетевым информационным центром (**N**etwork **I**nformation **C**enter, **NIC**). Реестром для `com` выступает Verisign. ICANN обязывает реестры TLD общего пользования принимать заявки на регистрацию доменов от кого бы то ни было. ICANN также оговаривает, что реестр не может напрямую получать заявки на регистрацию доменов.

Стремясь усилить конкуренцию и максимально расширить охват DNS, ICANN предусматривает, что заявки на регистрацию доменов для TLD общего назначения должны обрабатываться компаниями, называемыми **регистраторами**. По состоянию на 2020 год крупнейшим регистратором является GoDaddy: он уже обработал десятки миллионов регистраций доменов.

Регистраторы должны быть одобрены ICANN. ICANN устанавливает максимальную и минимальную плату, которую реестры и регистраторы могут взимать за регистрацию домена. Каждый раз, когда домен регистрируется, часть комиссии уходит в ICANN[1]. Технически домен по-прежнему принадлежит ICANN и сдается в аренду на срок не более десяти лет. Но, поскольку договор аренды может быть продлен, мы обычно называем арендатора собственником.

WHOIS Первоначально ICANN обязывала регистраторов раскрывать имена и контактную информацию владельцев доменов в общедоступном каталоге под названием WHOIS. Этот каталог работает через другой интернет-протокол, независимый от DNS. К каталогу можно обратиться, используя программу `whois` посредством командной строки[2]:

```
whois code.energy
```

Благодаря развитию норм и правил защиты данных ICANN теперь позволяет регистраторам изменять данные о владельцах доменов. В 2012 году ICANN пообещала «заново изобрести WHOIS», но по состоянию на 2020 год этот процесс все еще не завершен.

---

[1] С 2020 года ICANN взимает 0,18 доллара США в год за домен.

[2] Если у вас нет доступа к интерфейсу командной строки, обратитесь к WHOIS здесь: http://code.energy/whois.

DNS-ХОСТИНГ Теоретически при регистрации домена нужно указать имена и IP-адреса своих серверов имен. Эта информация пересылается регистратором в реестр для включения в серверы имен TLD. Но большинство людей даже не знают, что такое сервер имен. Чтобы упростить задачу, многие регистраторы предлагают услугу хостинга DNS: они используют свои собственные серверы имен от вашего имени и позволяют вам управлять своими записями DNS через веб-интерфейс.

РЫНОК ИМЕН Мы живем во времена процветающей онлайн-экономики. Доменные имена рассматриваются как виртуальная недвижимость и иногда продаются за миллионы. Например, в 2019 году GoDaddy выступила посредником при продаже `voice.com` за 30 миллионов долларов. В `.com` доменные имена настолько ценны, что разобраны уже все возможные четырехбуквенные комбинации.

Теперь, когда вы знаете, как работает DNS, мы будем обращаться к хостам, используя их доменные имена. Когда вы видите доменное имя, используемое в качестве адреса хоста, помните, что для поиска IP-адреса и отправки IP-пакета необходим DNS-запрос.

2.2. ВРЕМЯ

Тысячи лет назад люди начали разбираться, как астрономия и механика могут помочь количественно определить время. Древние цивилизации разработали различные лунные и солнечные календари для отслеживания времен года и изобрели устройства, такие как солнечные и водяные часы, для отслеживания времени в течение дня или ночи[1].

По мере того как в I веке до н. э. древнеримские военные подразделения становились все более профессиональными, они все чаще следили за временем, дабы максимально использовать потенциал своих кадров. Хронометраж позволил им *координировать* точные маневры и наносить сокрушительные удары врагам. Они также систематически записывали время сбора разведданных посыльными и разведчиками, чтобы иметь возможность *соотносить* события и получать представление о тактике врага.

[1] Солнечные часы показывают время, когда под действием солнечного света отбрасывают тень на размеченную поверхность. Водяные часы отслеживают прошедшее время от постепенного перетекания воды из одного контейнера в другой.

Сегодня мы в значительной степени полагаемся на хронометраж, поскольку грузовики, поезда, корабли и самолеты должны *координироваться*, перевозя непостижимое количество товаров по всему миру. Мы ведем журналы, иногда каждый раз, когда почтовая посылка переходит из рук в руки. Когда все согласны с текущим временем, возможно *соотносить* события, отслеживая происшествия и их последствия, а также привлекая людей к ответственности, если они сделали что-то не так.

Точно так же хронометраж позволяет соединенным между собой компьютерам объединяться в мощные команды. Если удаленные компьютеры могут договориться о том, сколько сейчас времени, они способны *координировать* свои действия со скоростью, недоступной для людей. Например, финансовые транзакции между сторонами на разных континентах могут быть подтверждены за секунды.

Компьютеры, в свою очередь, записывают в лог каждое свое действие, сопровождая каждую запись точным временем, когда оно произошло, что называется **отметкой времени**.

Например, финансовые операции между сторонами на разных континентах подтверждаются в течение нескольких секунд.

Кроме того, компьютеры сохраняют запись о каждом своем действии с указанием точного времени посредством так называемой **метки времени**. Согласование по времени позволяет взаимосвязанным компьютерам хронологически упорядочивать прошлые действия, записи и сообщения и, следовательно, *соотносить* события. В основном это помогает программистам находить ошибки, но также может, например, помочь специалистам по безопасности обнаружить и отследить вредоносную активность.

Совместное использование информации о времени обеспечивает координацию и корреляцию событий в невероятных масштабах. Возникает вопрос: как люди и компьютеры вообще договариваются о том, сколько сейчас времени?

КООРДИНИРОВАННОЕ УНИВЕРСАЛЬНОЕ ВРЕМЯ

В эпоху исследований европейские моряки совершенствовали свои методы навигации, основанные на наблюдениях за небом и измерении времени. К XVIII веку они уже могли точно определить свое местоположение

в море, используя таблицы, опубликованные Королевской Гринвичской обсерваторией в Лондоне. Это работало, пока они точно знали текущее время по меркам обсерватории, поэтому на кораблях держали механические часы[1], синхронизированные с Гринвичем. По мере того как такие надежные источники времени распространялись по миру, гринвичское время постепенно превратилось в универсальный стандарт времени.

Пару столетий спустя люди перешли от механических часов к кварцевым и атомным часам[2], дающим большую точность. Сегодня сотни атомных часов отслеживают мировое время. Всемирное скоординированное время (**C**oordinated **U**niversal **T**ime, или **UTC**)[3] является средним значением показаний этих часов, и в настоящее время это наш наилучший стандарт времени. Благодаря достижениям в астрономии и планетологии мы даже можем учесть замедление вращения Земли: иногда мы добавляем високосные секунды к UTC, чтобы стандарт оставался синхронизированным с солнечным временем в Лондоне.

Во всем мире официальное время вычисляется путем добавления определенного смещения к UTC. Смещение может меняться в зависимости от времени года. Например, Великобритания полгода на час опережает UTC из-за перехода на летнее время. Токио опережает UTC на девять часов весь год.

ЧАСОВЫЕ ПОЯСА Регионы, в которых официальное время соответствует одному и тому же смещению относительно UTC, образуют **часовой пояс**. IANA отслеживает постоянно меняющиеся часовые пояса и их смещения и публикует их в **базе данных часовых поясов**. Например, на большей части восточной части США, от Майами до Нью-Йорка, одинаковое смещение UTC. В базе данных этот часовой пояс имеет код `America/New_York`. Британия целиком находится в часовом поясе `Europe/London`, а Япония — в `Asia/Tokyo` (рис. 2.1).

---

[1] Механические часы обычно сохраняют энергию с помощью веса или пружины и содержат в себе хитрый механизм, который позволяет энергии высвобождаться через постоянные промежутки времени.

[2] Кварцевые часы используют умную электронику и специальный кристалл для создания вибраций с точно заданной частотой. Атомные часы похожи на кварцевые, но они непрерывно синхронизируют колебания с явлениями из области атомной физики.

[3] Четырнадцать англоязычных стран первоначально предложили сокращение CUT, в то время как франкоязычные страны предпочли TUC, сокращение от *temps universel coordonné*. Международно признанное сокращение UTC служит компромиссным решением.

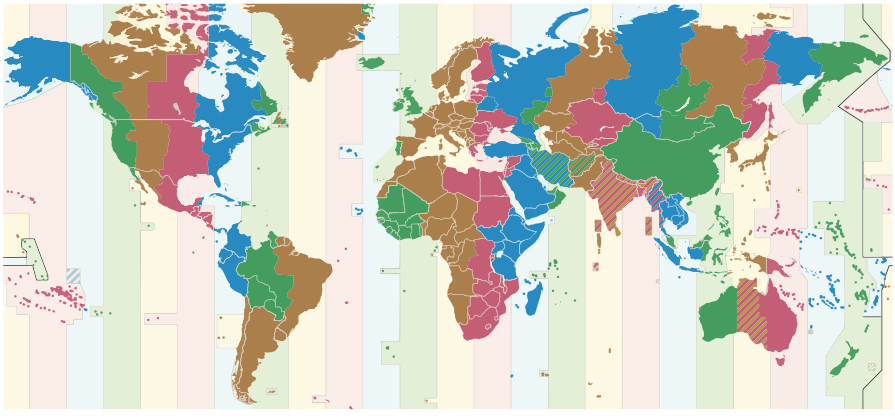


Рис. 2.1. Примерная схема часовых поясов мира по состоянию на 2020 год

Компьютеры обычно отслеживают время в UTC, но отображают его в соответствии со своими текущими часовыми поясами. Например, письмо, отправленное из Нью-Йорка в Токио, будет содержать метку времени создания в UTC. Эта метка времени будет отображаться по-разному для отправителя и получателя в зависимости от конфигурации часового пояса их компьютеров. И когда мы будем говорить о времени, то будем иметь в виду UTC.

ПРОТОКОЛ СЕТЕВОГО ВРЕМЕНИ

Люди могут проявлять невероятную пунктуальность, если их часы точны до секунды. Но компьютеры работают быстрее, и их часы нуждаются в большей точности. Например, группа компьютеров может регистрировать сотни записей в секунду. Чтобы сохранить эти записи в хронологическом порядке, мы должны синхронизировать часы с точностью до миллисекунды.

Не так-то просто настроить часы на такую точность. Наивно было бы отправлять сообщение через интернет с запросом о времени от кого-то, кто его знает, после чего устанавливать часы на метку времени, полученную в ответ. Это не сработает, ведь IP-пакеты передаются в течение неопределенного времени. Временная метка от удаленного компьютера неизбежно придет позже, чем нужно.

Но, имея некоторые дополнительные данные, можно приблизительно определить, сколько времени потребовалось пакету для прохождения

через интернет. Затем эту продолжительность можно добавить к указанной метке времени и после этого можно использовать ее для настройки наших часов. Протокол сетевого времени (**N**etwork **T**ime **P**rotocol, **NTP**) является широко используемым стандартом для синхронизации часов по этому принципу.

Метки времени NTP выражаются в секундах с полуночи 1 января 1900 года. Например, метка времени полуночи 1 января 2000 года будет выглядеть как 3 155 673 600. Число может иметь дробную часть, поэтому может фиксировать время с очень высокой точностью. Серверы NTP ожидают входящие сообщения на UDP-порте 123. Обмен сообщениями NTP работает так, как показано на рис. 2.2.

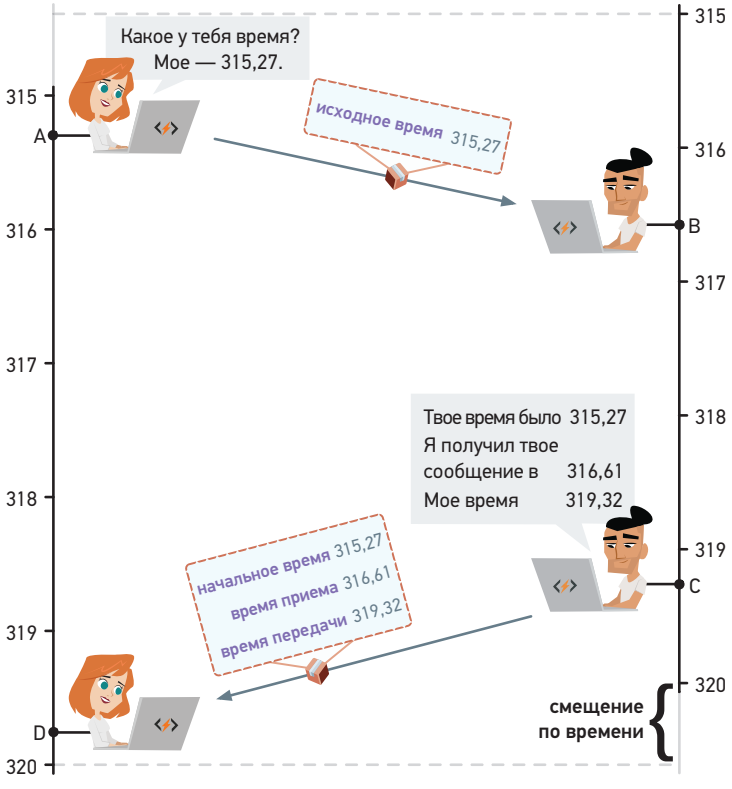


Рис. 2.2. Клиент запрашивает у сервера время, используя NTP. Для простоты давайте представим, что мы живем в 1900 году, так что временные метки — это пока еще небольшие числа. Время, хранящееся на любом компьютере, проходит сверху вниз

Обмен начинается, когда клиент отправляет свой запрос времени. Это сообщение содержит время своей отправки в соответствии с часами клиента (время A). Сервер регистрирует время поступления сообщения в соответствии с часами сервера (время B). При ответе сервер отправляет сообщение, содержащее три времени: время A, время B и время ответа (C). Клиент получает эти значения времени и подтверждает, что время A совпадает с ранее отправленным. Клиент также отмечает, когда приходит ответ, в соответствии со своими собственными часами (время D).

Предположим, что время D равно 319,42 (см. рис. 2.2). Зная моменты времени A, B, C и D, клиент может рассчитать время прохождения IP-пакетов через интернет:

время в пути = время в пути клиента – время обработки сервера =
= D – A – (C – B) =
= 319,42 – 315,27 – (319,32 – 316,61) =
= 4,15 – 2,71 =
= 1,44.

Потребовалось 1,44 секунды, чтобы сообщения прошли от клиента к серверу и обратно. Мы можем приблизительно подсчитать, что для передачи пакета от сервера к клиенту потребовалось вдвое меньше времени: 0,72 секунды. Клиент должен был получить ответ через 0,72 секунды после времени C, в 319,32 + 0,72 = 320,04. Однако клиент получил сообщение, когда его часы показывали только 319.42. Это означает, что часы клиента опаздывают, по сравнению с сервером, примерно на 320,04 – 319,42 = 0,62 секунды. NTP консервативен: он предписывает клиентам корректировать свои часы только на половину предполагаемого смещения. Таким образом, наш клиент переводит свои часы на 0,31 секунды вперед. Каждые десять минут клиент отправляет новый запрос и соответствующим образом подстраивает свои часы. Если время прохождения пакетов между клиентом и сервером одинаково в обоих направлениях, часы клиента в итоге синхронизируются с часами сервера. На типичных интернет-каналах NTP синхронизирует часы с ошибкой в десятки миллисекунд.

СЕРВЕРЫ ВРЕМЕНИ

Возможно, вам интересно, откуда серверы NTP сами получают свое время. Большинство серверов NTP фактически синхронизируются с другими

серверами NTP в интернете: нет никаких проблем с одновременной работой серверных и клиентских программ NTP.

Но это чревато проблемами. Если Ада получит свое время от Эндрю, Эндрю — от Чарльза, а Чарльз — от Ады, ни у кого из троих не будет показаний времени, откалиброванных по UTC. Даже если их часы изначально шли правильно, неточности будут накапливаться с течением времени и заставлять часы отклоняться от UTC.

Чтобы предотвратить подобную проблему, серверы NTP организованы в иерархию, называемую **часовыми уровнями** (clock strata). Сервер, который напрямую связан с надежным источником времени, таким как атомные часы или GPS-приемник, называется *stratum 1*. Компьютеры, которые синхронизируются с сервером уровня 1, относятся к *stratum 2*. Если компьютер получает время от сервера уровня 2, он становится уровнем 3 и т. д. По правилам NTP серверы должны сообщать о своем уровне при ответе на запросы времени, самым нижним уровнем иерархии является *stratum 15*.

Это позволяет компьютерам избежать циклического определения времени: если Чарльз — это stratum 2 и он сообщает время Эндрю, Эндрю становится stratum 3. Если Эндрю передает информацию о времени Аде, она становится stratum 4. Если Ада когда-нибудь сообщит Чарльзу время, он будет знать, что его следует игнорировать, потому что он — stratum 2, а она — stratum 4. Для синхронизации он должен найти сервер stratum 1.

Есть множество общедоступных серверов NTP, которые отвечают на запросы от кого угодно. Компьютеры Apple предварительно настроены на синхронизацию своих часов с time.apple.com, который является сервером stratum 2. Правительства и крупные организации часто предоставляют общедоступные серверы NTP. Например, правительство США содержит time.nist.gov, публичный сервер stratum 1.

NTP был создан в 1985 году и является одним из старейших протоколов интернета. Поскольку почти на каждом компьютере работает клиент NTP, люди часто считают само собой разумеющимся, что их компьютеры естественным образом знают время. Вы можете сверить собственные часы по адресу http://time.gov/. Скорее всего, отставание будет менее чем на секунду. Далее давайте рассмотрим интернет-протокол, который еще старше, чем NTP.

2.3. ДОСТУП

В XIX веке люди-операторы должны были декодировать электрические сигналы телеграфа на человеческий язык и наоборот. На рубеже веков их работу облегчило новое устройство, получившее название **«телетайп»**. Эти машины были подключены к телеграфным проводам и включали в себя механическую пишущую машинку, которая могла автоматически печатать входящие телеграфные символы.

Телетайпы также имели клавиатуру, поэтому символы, набранные их оператором, автоматически передавались по телеграфному проводу. Это позволяло людям общаться на больших расстояниях: сообщения, набранные на одной машинке, печатались второй машинкой на другой стороне линии.

ТЕРМИНАЛЫ

Когда в 1950-х годах появились первые коммерческие электронные компьютеры, у них еще не было экранов. Чтобы получать обратную связь от машин, люди должны были подключать свои входные/выходные соединения к телетайпам. Так они могли передавать данные и инструкции на компьютер, набирая текст, и ответ их компьютера печатался в режиме реального времени. Такие телетайпы, которые позволяют взаимодействовать с компьютерами через текст, называются **терминалами**.

Компьютерная программа, которая обрабатывает информацию и инструкции, поступающие в терминал и исходящие из него, называется **оболочкой**. Для того чтобы взаимодействовать с людьми, она использует интерфейс командной строки (**C**ommand-**L**ine **I**nterface, **CLI**). Когда пользователь терминала вводит команду в CLI, оболочка подтверждает каждый символ, прося терминал распечатать его. При этом пользователь может в режиме реального времени видеть, что именно он печатает. Пользователь может ввести непечатаемый символ (**N**on-**P**rinting **C**haracter, **NPC**), называемый *возвратом каретки*, и оболочка выполнит всю строку как команду, при необходимости запросив терминал для печати каких-то выходных данных, после чего будет ожидать следующей команды.

В 1960-х годах появились **стеклянные телетайпы**. Они работали точно так же, как и старые, за исключением того, что они отображали символы на экране вместо печати на бумаге. Вот почему и по сей день отображение текста на экране терминала называется *печатью*!

Экранам было разрешено создавать **курсор**: подвижный индикатор, указывающий место взаимодействия оболочки и экрана. Пользователи начали использовать непечатаемые символы, чтобы просить оболочку переместить курсор. Важно отметить, что курсоры позволяли заменять или удалять ранее напечатанный текст, что делало взаимодействие с компьютерами более эффективным. Это новшество вызвало всплеск развития компьютерных программ с интерактивными текстовыми интерфейсами.

Видеомониторы были представлены в конце 1960-х годов, а первый коммерческий компьютер с графическим интерфейсом (**G**raphical **U**ser **I**nterface, **GUI**) появился в 1979 году. Сегодня практически все персональные компьютеры, такие как ноутбуки и телефоны, работают с оболочкой, которая управляет графическим интерфейсом: мы взаимодействуем со значками и окнами с помощью мыши, трекпада и сенсорного экрана (рис. 2.3).

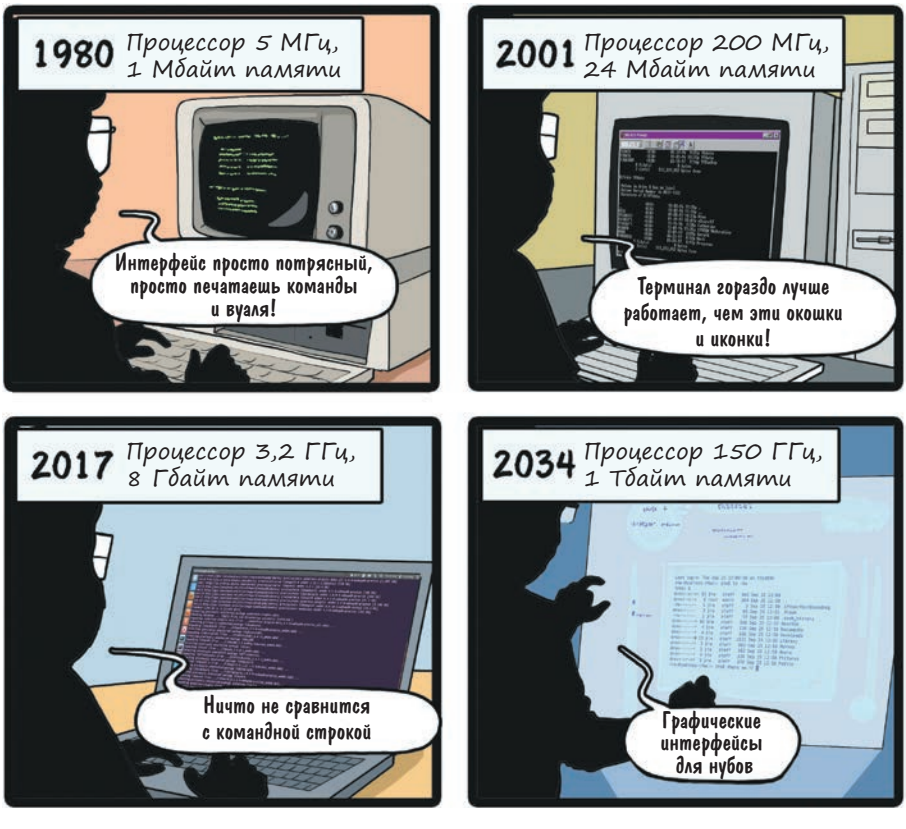


Рис. 2.3. «Командная строка», любезно предоставленная http://commitstrip.com

Но многие айтишники продолжают взаимодействовать со своими компьютерами через CLI. Это может быть необходимым, например, для работы с корпоративными серверами, на которых не поддерживается графический интерфейс, но также и по личным предпочтениям. Некоторые используют свой графический интерфейс только для того, чтобы открыть **эмулятор терминала**: приложение, которое подключается к оболочке CLI компьютера и превращает ваш экран и клавиатуру в стеклянный телетайп.

Эмуляторы терминалов — это мощные интерфейсы[1]. С помощью CLI можно выполнять практически любые текстовые задачи, такие как организация папок, отправка электронной почты, написание кода и компиляция ПО (рис. 2.4). Мы написали и отредактировали эту книгу в эмуляторах терминалов!

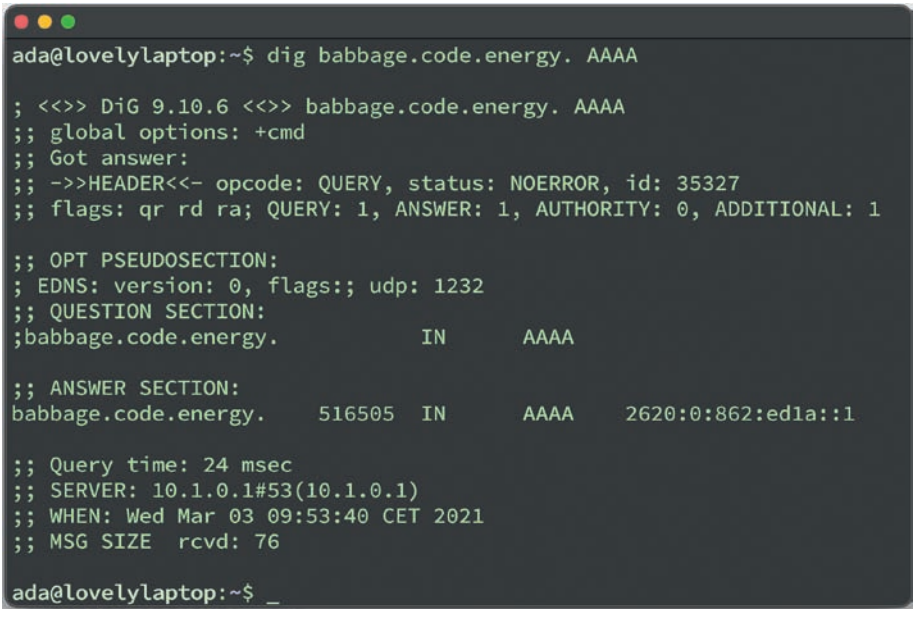


Рис. 2.4. Запуск dig в эмуляторе терминала macOS

[1] Некоторые особо упорные показали целые «Звездные войны» на терминале: http://code.energy/terminal-movie.

TELNET

В течение большей части 1950-х и 1960-х годов терминал должен был быть подключен непосредственно к компьютеру, чтобы взаимодействовать с его оболочкой. Однако все изменилось в 1969 году, когда инженеры разработали способ подключения терминалов к любому компьютеру в сети. Они называли эту технологию **tel**etype over **net**work или **telnet**.

Сегодня эмуляторы терминалов могут использовать telnet для доступа к удаленным компьютерным оболочкам через интернет. CLI удаленной оболочки должен быть чрезвычайно надежным, так как неупорядоченные или отсутствующие символы могут привести к выполнению ошибочных и потенциально разрушительных команд. По этой причине предпочтительным протоколом транспортного уровня для telnet является TCP, а не UDP. Серверы telnet, как правило, ожидают подключения через TCP-порт 21.

Когда через порт 21 установлено TCP-соединение, сервер telnet запускает оболочку и отправляет все входящие данные из соединения в оболочку. Вывод оболочки отправляется клиенту. На стороне клиента все символы, введенные пользователем, посимвольно передаются на сервер в режиме реального времени. И когда клиент получает один или несколько символов, они отображаются на эмуляторе терминала.

Обычно после установки соединения telnet удаленная оболочка запрашивает имя пользователя и пароль с целью аутентификации. После входа в систему клиент telnet работает как терминал, подключенный непосредственно к серверу. Если на компьютере работает сервер telnet, *любой компьютер, подключенный к интернету, может выступать в качестве терминала для этого сервера*[1]. Гики способны управлять компьютером на другом континенте, как если бы они сидели перед его экраном и клавиатурой!

---

[1] Широкое распространение интернета создает серьезные риски для безопасности, и telnet считается небезопасным. Сейчас мы используем аналогичный протокол, называемый Secure Shell, или SSH. Конечный пользователь работает почти так же, как через telnet, но при этом используется передовая криптография для противостояния атакам со стороны хакеров. Мы вернемся к этой теме в следующей главе.

Протоколы связи через интернет используются не только программистами. А теперь рассмотрим самые популярные приложения интернета, начиная с наиболее известного способа обмена письменными сообщениями.

2.4. ПОЧТА

В 1960-х годах немногие компьютеры совместно использовались множеством пользователей. В то время «почтовые» программы уже были популярны, но они позволяли оставлять сообщение только коллеге, работающему на том же самом компьютере. Эти программы просто добавляли текст к файлам, которые служили почтовыми ящиками. Каждому пользователю назначался один файл почтового ящика. Когда пользователи входили в систему, появлялись оповещения, если в их почтовых ящиках находились новые сообщения.

В 1970-х годах компьютерные сети выросли и появилась технология доступа к файлам на удаленных компьютерах. Почтовые программы эволюционировали для добавления текста в почтовые ящики уже удаленных компьютеров. Это позволяло пользователям удаленных компьютеров обмениваться сообщениями, как если бы они работали на одной машине. Со временем программисты улучшили эффективность этих систем путем постепенного принятия общего формата сообщений, который выглядел так:

```
From: White at SRI-ARC
Date: 24 JUL 1973 1527-PDT
Subject: Multi-Site Journal Meeting Announcement
NIC: 17996

At 10 AM Wednesday 25-JULY there will be a meeting
to discuss a Multi-Site Journal in the context of
the Utility. Y'all be here.
```

Сообщение начинается с **заголовков**, которые содержат общую информацию о сообщении, такую как его отправитель, получатель и тема. Каждый заголовок занимает одну строку, и двоеточие отделяет имя заголовка от его содержания. После заголовков пустая строка сигнализирует о начале тела сообщения. Эти сообщения широко использовались программистами, которые стали называть их **email** (**e** расшифровывается как **e**lectronic). По сей день письма, которыми мы обмениваемся, следуют такому формату.

В 1973 году на вышеприведенное письмо можно было ответить, добавив новое сообщение в почтовый файл Уайта на компьютере `SRI-ARC`. В то время TCP/IP и DNS еще не было, и компьютеры получали такие вот прозвища. Позже слово `at` уступило позиции символу `@`, поэтому почтовый ящик в примере был бы записан как `White@SRI-ARC`. Для справки, **SRI** расшифровывается как **S**tanford **R**esearch **I**nstitute — это одна из организаций, которая помогла создать и стандартизировать этот формат сообщений.

ПОЧТОВЫЕ СЕРВЕРЫ

Компьютеры постепенно становились менее дорогими, и к 1980-м годам уже не было ничего необычного в том, что кто-то работал на нескольких машинах. Но было бы легче связываться с людьми по электронной почте, если бы у каждого человека его основной адрес почтового ящика был в одном месте. Многие группы пользователей принимали решение держать свои основные ящики на одном хосте, как если бы все по-прежнему работали на одном компьютере. Организации помогали этому, назначая один из своих компьютеров **почтовым сервером**.

В то время студенты университетов и инженеры-технологи получали учетную запись на почтовом сервере своей организации, независимо от того, какие компьютеры они использовали. Основной адрес почты каждого указывал на этот почтовый сервер. С разных компьютеров люди использовали telnet для входа на почтовый сервер и чтения электронных писем, хранящихся в их отдельных файлах почтовых ящиков.

В 1985 году была запущена DNS, и она значительно улучшила систему электронной почты. Организации начали использовать доменные имена и создавать записи `MX` для публичного объявления своих почтовых серверов, и доменные имена стали стандартным способом адресации почтовых ящиков. Например, ящик с именем `White` на почтовом сервере SRI получил адрес `white@sri.com`.

Имея такой адрес, любой может использовать DNS для обнаружения почтового сервера, на котором хранится файл почтового ящика, и отправки IP-пакетов на этот компьютер. Но, чтобы сохранить новое сообщение на чужом сервере, сначала нужно было получить доступ к файлам его ящика. Это было непростой задачей, поэтому инженеры принялись изобретать способы получения писем без внешних манипуляций с какими-либо файлами на принимающем сервере.

Настала заря протоколов электронной почты. Как NTP обеспечивает базовый функционал для обмена временем, так и почтовые протоколы позволяют программам обмениваться письмами. Были разработаны различные протоколы, и вскоре каждый почтовый сервер придерживался одного из них. Теперь электронные письма можно было отправлять практически в любое место, куда могли дойти IP-пакеты.

Поскольку любая организация, подключенная к интернету, могла участвовать в таком обмене, электронная почта быстро стала популярной. То, что начиналось как простые записки, оставленные на столе друга, постепенно становилось столь же важным, как и традиционная переписка.

В 1990-х годах интернет-провайдеры начали предоставлять доступ к интернету в домах людей. В качестве одной из основных продающих услуг интернет-провайдеры часто использовали возможность открывать учетные записи электронной почты на своих серверах. В те годы персональные компьютеры начали поставлять с уже предустановленными почтовыми программами. Люди в основном использовали их для подключения к своему интернет-провайдеру или почтовому серверу работодателя.

Электронная почта распространилась подобно лесному пожару. Обычные люди толпами стекались в интернет, иногда исключительно для того, чтобы воспользоваться почтой. И один протокол стал главным фактором этой революции электронной почты. Узнаем же, как передавать по нему сообщения между хостами и почтовыми серверами.

SIMPLE MAIL TRANSFER PROTOCOL

Наиболее широко используемым протоколом является **S**imple **M**ail **T**ransfer **P**rotocol **(SMTP)**. Он задает правила общения между сервером электронной почты и клиентским компьютером. Как мы увидим далее, SMTP более сложен, чем *протоколы запроса-ответа*[1], такие как DNS и NTP.

Во-первых, письма могут содержать сообщения, которые не помещаются в один IP-пакет. Когда электронная почта передается многими IP-

[1] Протоколы запроса-ответа работают согласно своему названию: клиент отправляет запрос на сервер, а затем ожидает ответа.

пакетами, важно, чтобы их полезная нагрузка была собрана в нужном порядке. По этой причине SMTP работает на TCP, а не на UDP.

Почтовые серверы ожидают входящих подключений через TCP-порт 25. Когда соединение установлено, сервер начинает взаимодействие. Он посылает сообщение, чтобы идентифицировать себя:

```
220 mail-server.example.com
```

Все сообщения, отправляемые сервером, начинаются с трехзначного номера, называемого кодом **возврата**. SMTP определяет множество различных кодов, каждый из которых имеет свое собственное значение. В частности, код `220` сообщает, что сервер готов к получению инструкций. После кода сервер указывает собственное имя. В ответ клиент должен отправить команду `HELO`, чтобы идентифицировать себя:

```
HELO client.code.energy
```

На этом этапе сервер решает, хочет ли он продолжить общение, основываясь на том, кем является клиент. Некоторые почтовые серверы используют обратный DNS и сравнивают имя клиента, о котором он сам сообщил, с именем, связанным с его IP-адресом. Если почтовый сервер решает продолжить, он отправляет код ответа `250`, который означает, что действие, запрошенное клиентом, было принято:

```
250 mail-server.example.com
```

При этом первом обмене и клиент, и сервер подтвердили, что они собираются передать письмо. Они также проверили, что каждая сторона общается с предполагаемой другой стороной. Далее клиенту нужно указать обратный адрес электронной почты, которая будет передана. Если сервер не может доставить электронное письмо, он отправит другое письмо на обратный адрес, сообщив о проблеме:

```
MAIL FROM: <ada@code.energy>
```

Опять же у сервера есть возможность принять или отклонить эти данные. Некоторые почтовые серверы настроены на прием только с определенных адресов. Если адрес принят, сервер возвращает код `250`:

```
250 Ok
```

Затем клиент должен сообщить серверу о целевом почтовом ящике:

```
RCPT TO: <charles@example.com>
```

Сервер подтверждает, принимает ли он этот почтовый ящик назначения. Если сервер соглашается принять сообщение, он использует тот же код возврата `250`, сообщая клиенту, что тот может продолжить:

```
250 Ok
```

Теперь клиент может отправить команду `DATA`. Эта команда просит сервер начать передачу сообщения электронной почты:

```
DATA
```

Сервер подтверждает и инструктирует клиента сигнализировать об окончании передачи строкой, содержащей только точку. Код `354` информирует клиента о том, что сервер будет рассматривать следующие символы, которые он получит, как часть сообщения электронной почты[1]:

```
354 End data with <CR><LF>.<CR><LF>
```

Затем клиент передает сообщение. Заголовки `From`, `To` и `Date` являются обязательными и должны присутствовать в каждом электронном письме. Другие заголовки, такие как `Subject`, обычно используются, но не являются обязательными. Тело сообщения также необязательно. Отправка почты без тела сообщения похожа на отправку пустого конверта. Вот пример сообщения, которое может передать клиент:

```
From: <ada@code.energy>
Date: Wed, 27 Nov 2002 15:30:34 +0100
To: <charles@example.com>

That brain of mine is something more than
merely mortal; as time will show.
.
```

После получения завершающих `<CR><LF>.<CR><LF>` сервер проверяет допустимость полученных данных и формально принимает сообщение,

[1] Непечатаемые символы <CR><LF> представляют собой carriage return (возврат каретки), за которым следует line feed (перевод строки). Эта комбинация отмечает конец строки.

если это так. Многие серверы также сообщают клиенту внутренний идентификатор, который они назначают полученному электронному письму:

```
250 Ok: queued as 1079212633C
```

На этом этапе клиент может быть уверен, что сервер либо доставит сообщение, либо **возвратит**[1] его с сообщением об ошибке. Клиент может продолжить и отправить другую команду `MAIL FROM` для отправки новых писем. В противном случае он может вежливо сказать, что закончил, отправив `QUIT`:

```
QUIT
```

Получив это, сервер должен попрощаться и закрыть TCP-соединение:

```
221 Bye
```

Диалог, подобный этому, происходит каждый раз, когда вы отправляете письмо, — ваше почтовое ПО проводит его незаметно от вас. Первоначально в SMTP не было аутентификации: серверы безоговорочно верили, что клиенты отправляли правильные электронные письма. К сожалению, как только электронная почта стала популярной, безрассудные интернет-пользователи принялись отправлять всякий спам на все адреса, которые могли найти. Некоторые даже отправляют мошеннические письма с нелегитимными полями `From`.

ОТПРАВКА ЭЛЕКТРОННЫХ ПИСЕМ

Электронная почта изначально создавалась как открытая система, которую может использовать любой желающий, не спрашивая разрешения у какого-либо центрального органа. Ее пионеры надеялись, что все участники будут действовать добросовестно. К сожалению, как только электронная почта получила широкое распространение, эта надежда была утрачена навсегда. Но инженеры продолжили работать над тем, чтобы

[1] SMTP был разработан ради надежности, поэтому почтовые серверы не будут отбрасывать принятое электронное письмо без предварительного уведомления. Отклоненное сообщение называется возвратом, поскольку его содержимое возвращается отправителю.

сделать электронную почту устойчивой к недобросовестным пользователям, не отказываясь при этом от ее открытой сущности.

Сначала появились публичные **черные списки**, которые достаточно часто дополнялись именами и IP-адресами известных нарушителей. Администраторы настроили серверы на автоматическое отклонение подключений от пользователей, внесенных в черный список. Это уменьшило проблему, но не устранило ее: злоумышленники научились использовать новые IP-адреса и доменные имена для отправки спама.

По сей день большинство спама поступает через интернет-соединения, предоставляемые интернет-провайдерами. Большинство организаций тщательно охраняют и контролируют использование своих сетей в попытках защитить свою репутацию. Но интернет-провайдеры часто обслуживают миллионы пользователей, и индивидуальная проверка каждого из них нецелесообразна.

К счастью, легко проверить, принадлежит ли IP-адрес достойной организации или домашнему соединению: можно обратиться к его обратной DNS-записи. Интернет-провайдеры не позволяют постоянным клиентам устанавливать эти записи, но все надежные почтовые серверы имеют надлежащий обратный DNS для своих IP-адресов. Таким образом, почтовые серверы начали ограничивать домашние IP-адреса, что значительно снижает масштаб злоупотребления.

Для реализации такого поведения SMTP был обновлен до расширенной версии (**e**xtended version, **ESMTP**) с поддержкой аутентификации по имени пользователя и паролю. Для всех домашних пользователей команда `MAIL FROM` принимается только после проверки подлинности пользователя посредством команды `AUTH`.

Вот как работает отправка почты сегодня: вы составляете письмо, и ваш компьютер использует SMTP для отправки его на ваш сервер. Затем ваш сервер подключается к почтовому серверу, связанному с почтовым ящиком назначения, и использует SMTP для ретрансляции вашей почты. Поскольку IP-адрес вашего сервера имеет обратную DNS-запись, он может связываться со всеми другими почтовыми серверами без аутентификации.

Таким образом, почтовый трафик делится на две части: пользователи, отправляющие сообщения на свой собственный почтовый сервер, и почтовые серверы, ретранслирующие сообщения между друг другом. В на-

стоящее время TCP-порт 587 зарезервирован для отправки электронной почты, а порт 25 — для ретрансляции. Домашние пользователи электронной почты перестали использовать TCP-порт 25.

Соответственно, большинство интернет-провайдеров даже отбрасывают домашние IP-пакеты, содержащие сегменты для порта 25, эффективно блокируя домашних пользователей, пытающихся выполнять ретрансляцию почты. Почтовые серверы ожидают входящих соединений на двух TCP-портах:

- порт 25 для получения писем от других почтовых серверов;
- порт 546 для приема писем от аутентифицированных пользователей.

Но всего этого недостаточно, чтобы обуздать мошенников и спамеров. Борьба между инженерами и спамерами все еще продолжается. В некоторых случаях инженеры угадывают, какие электронные письма являются мусором, используя сложные механизмы сортировки. В других случаях используют криптографические подписи для проверки адреса отправителя.

ПОЛУЧЕНИЕ ЭЛЕКТРОННЫХ ПИСЕМ

Сначала люди читали письма, непосредственно открывая файлы почтовых ящиков. В большинстве случаев люди не сидели при этом перед почтовыми серверами, поэтому использовали telnet для удаленного открытия файлов. Инженеры осознали, что людям гораздо проще загружать письма на собственный компьютер и читать их «локально». Это избавило бы их от необходимости использовать telnet каждый раз, когда им нужно прочитать письмо.

Поскольку SMTP не подходит для того, чтобы почтовый сервер доставлял электронную почту конечному пользователю, нужен был другой протокол. В конце концов, большинство пользователей электронной почты не используют свои компьютеры в режиме 24/7, принимая все входящие SMTP-соединения. Следовательно, появились новые протоколы. Они позволили пользователям инициировать подключение к почтовому серверу, получать список электронных писем в почтовом ящике и выборочно загружать электронные письма из списка.

Наиболее часто используемым из таких протоколов является **I**nternet **M**essage **A**процесс **P**rotocol **(IMAP)**. Его принцип работы аналогичен SMTP, сервер также ведет разговор с клиентом, используя командные коды и обычный текст. В нем имеются клиентские команды для перечисления электронных писем в почтовых ящиках и для запроса на передачу определенного электронного письма.

Сегодня примерно половина писем в мире отправляется и принимается через **сервисы электронной почты**, например Gmail. Эти службы предлагают самый беспроблемный доступ к электронной почте, поскольку не требуют установки или настройки клиентского приложения на вашем компьютере. А теперь поговорим о **Сети**: прекрасном наборе механизмов, которые сделали все это — и многое другое — возможным.

2.5. СЕТЬ

Если текстовый документ содержит ссылки, ведущие к другим документам, и эти ссылки позволяют переходить между несколькими документами, текст считается **гипертекстом**: ссылки создают дополнительное измерение в текстовом пространстве. Например, *Википедия* — это гипертекст, а вот бумажная энциклопедия — нет. Гипертекст не обязательно читать линейно, как обычный текст, — его можно изучать, переходя по ссылкам.

По мере распространения графических пользовательских интерфейсов в 1980-х годах работники умственного труда обнаружили, что гипертекст может сделать их работу проще и эффективнее. Когда первые программы для чтения и записи гипертекстовых документов набирали популярность, все еще нельзя было создавать ссылки между гипертекстовыми документами, находящимися на разных компьютерах. Гипертекстовые документы оставались в значительной степени изолированными друг от друга.

С появлением интернета для преодоления этого ограничения была создана новая гипертекстовая система. Документы, созданные и хранящиеся на разных компьютерах, наконец-то получилось объединить в универсальную сеть документов. Система была названа **W**orld **W**ide **W**eb, или **WWW**, а принадлежащие ей гипертекстовые документы назвали **веб-страницами**. Чтобы она работала, различные компьютеры по всему миру должны совместно использовать три основных компонента:

- язык для записи веб-страниц в виде файлов;
- способ связать веб-страницы друг с другом;
- протокол для передачи файлов между компьютерами.

Программа, которая использует эти три компонента, называется **браузером**. Практически каждый веб-браузер сегодня соответствует базовому набору стандартов. Узнаем, как они работают.

ЯЗЫК РАЗМЕТКИ ГИПЕРТЕКСТА

Веб-страницы должны быть простыми в создании и универсально понятными для всех пользователей. Поэтому было решено, что веб-страницы будут состоять из обычного текста, дополненного специальными тегами для задания структуры и внешнего вида (рис. 2.5).

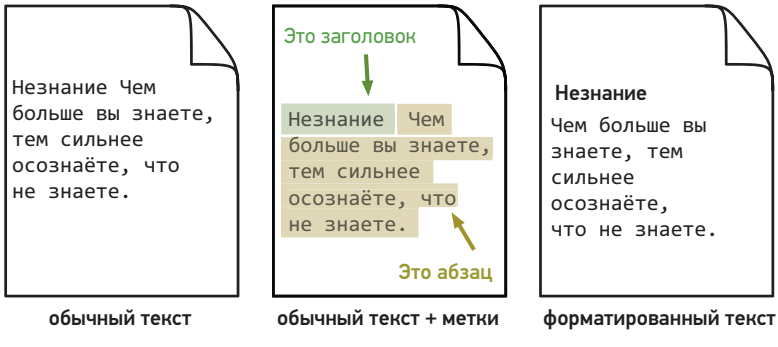


Рис. 2.5. Форматированные документы можно создавать, размечая обычный текст

Теги можно использовать для разметки абзаца, установки фразы в качестве заголовка, подчеркивания слова и многого другого. Набор тегов, которые можно добавлять в текст, чтобы расширить его, образует **язык разметки**. Существует множество различных языков разметки. Тот, который создан для интернета, называется **H**yper**t**ext **M**arkup **L**anguage или HTML. В HTML любой текст, заключенный в символы `<...>`, является тегом. Например, так задается заголовок, за которым следует абзац:

```
<h1>The Unknown</h1>
<p>The more you know, the more you realize you don't know.</p>
```

Этот пример включает в себя две пары HTML-тегов: `<h1>...</h1>` и `<p>...</p>`. Теги `h1` указывают, что `The Unknown` — заголовок. Теги `p` разграничивают абзац. Помимо них, существует множество других тегов, и большинство из них используются для задания *структуры* документа[1].

Как правило, HTML-теги представляют собой пары из открывающих и закрывающих: каждая пара применяется к разделу документа, а закрывающий тег включает в себя косую черту. Однако несколько тегов, называемых **пустыми тегами**, не формируют пары. Например, `<br>` используется для вставки разрыва строки в текст, а `<img>` — для вставки изображения.

Полные веб-страницы HTML должны состоять из двух частей: **заголовочной части** и **тела**. Заголовок содержит информацию о документе, которую браузеры не должны отображать. Тело же, с другой стороны, предназначено для отображения на экране браузерами. Вот пример простой, но законченной веб-страницы HTML:

```
<html>

   <head>
        <title>Daily quote</title>
    </head>
    <body>
        <h1>The Unknown</h1>
        <p>The more you know, the more you realize
        you don't know.</p>
    </body>

</html>
```

Заголовки и тело обернуты в пару тегов с соответствующим именем; и оба они совместно обернуты в пару `<html>`. Создайте текстовый файл с данным содержимым, сохраните его в формате `.html` и откройте его в браузере. Браузер игнорирует интервалы и разрывы строк в HTML-файле. Они используются лишь для того, чтобы людям было удобнее читать файл.

---

[1] Другой язык — Cascading Style Sheets (CSS) — был создан позже для настройки большинства аспектов представления HTML-документа и его структуры: шрифтов, макета, цветов и т. д.

Многие теги могут содержать атрибуты, которые расширяют или дополняют их. Например, тег `<html>` может содержать атрибут, определяющий язык документа: `<html lang="en">`. Некоторые теги содержат обязательный атрибут. Например, в теге `<img>` нужно указывать, какое изображение будет отображаться:

```
<img src="cat.gif">
```

Наиболее важным HTML-тегом является тег **a**nchor — `<a>`. Это делает раздел документа ссылкой на другое место гипертекста. Например:

```
Everybody loves <a href="cats.html">them</a>.
```

Браузеры отобразят эту строку вот так:

Все любят их.

Щелчок на подчеркнутом слове мгновенно перебрасывает в другое место, указанное в атрибуте **h**ypertext **ref**erence (`href`) тега `<a>`. А теперь выясним, как задавать данные местоположения, чтобы веб-браузеры могли их находить.

URL-АДРЕС

Параметр `href` гипертекстовой ссылки — это ссылка на документ, который отображается после нажатия на нее. Он имеет простой формат: две косые черты, за которыми следует имя хоста, на котором находится документ, плюс путь к документу. Например, вот ссылка на документ `/pets/cat.html`, расположенный на узле `zoo.org`:

```
<a href="//zoo.org/pets/cat.html">kitten</a>
```

Браузеры интерпретируют путь к документу как расположение файла в файловой системе узла. Например, предполагается, что документ, на который указывает ссылка выше, является файлом с именем `cat.html` внутри каталога `pets`.

ОТНОСИТЕЛЬНЫЕ ССЫЛКИ Если ссылка не начинается со слеша, она считается относительной ссылкой на папку. Например, предположим, что в документе `//zoo.org/pets/cat.html` есть ссылка:

```
Are <a href="dog.html">they</a> our best friends?
```

Здесь параметр `href` ссылается на файл в том же каталоге, что и текущий файл (`cat.html`). Следовательно, браузер интерпретирует ссылку как `//zoo.org/pets/dog.html`. Есть также ссылки только с одним начальным слешем, которые относятся к узлу:

```
<a href="/bugs/ant.html">Ants</a> are small.
```

Эта ссылка будет интерпретироваться как `//zoo.org/bugs/ant.html`.

ССЫЛКА НА ФРАГМЕНТ Есть способ сослаться на определенную часть или фрагмент HTML-документа. Предположим, что файл dog.html выглядит следующим образом:

```
<h1 id="bulldog">Bulldog</h1>
<p>...</p>

<h1 id="poodle">Poodle</h1>
<p>...</p>

<h1 id="golden">Golden Retriever</h1>
<p>...</p>
```

Мы можем создать ссылку, которая напрямую указывает на заголовок о золотистых ретриверах внутри `dog.html`. Например, так:

```
<a href="dogs.html#golden">These dogs</a> rock!
```

Символ # можно также использовать для доступа к другому месту *внутри* документа. Например, этот трюк используется в поле *Содержание* страниц Википедии, чтобы можно было перейти к интересующему вас разделу.

Аналогичные ссылки используются при обращении к другим типам медиаресурсов, таким как изображения. Например, если узел `zoo.org` содержит фотографию золотистого ретривера, ее можно включить в документ вот так:

```
<h1>Golden Retriever</h1>
<img src="/pics/golden-retriever.jpg">
<p>As you can see in this picture…</p>
```

СХЕМА Есть еще один важный элемент информации, который можно добавить к нашим веб-ссылкам: *способ* доступа к ресурсу, известный как **схема**. Один из них вы, скорее всего, узнаете — это `http`: наиболее широко

используемый протокол передачи файлов веб-страниц между компьютерами. Схема задается в начале ссылки, перед двоеточием:

```
<a href="http://zoo.org/pets/cat.html">kitten</a>
```

Полная ссылка на веб-ресурс, которая включает в себя схему, узел и путь к документу, называется **U**niform **R**esource **L**ocator или **URL**. Люди также называют их **веб-адресами**. Браузеры обычно отображают URL-адрес страницы в строке вверху. Теперь рассмотрим подробнее, как работает схема `http`.

ПРОТОКОЛ ПЕРЕДАЧИ ГИПЕРТЕКСТА

Самое распространенное действие при навигации по интернету — это щелчок на ссылке, позволяющий перейти с просматриваемой веб-страницы на следующую. Каждый раз при нажатии на ссылку браузер должен связаться с компьютером, на котором размещен указанный документ, и получить копию этого документа. **H**yper**t**ext **T**ransfer **P**rotocol (**HTTP**) регулирует эти процессы передачи документов.

Поскольку передаваемые документы могут быть больше MTU, HTTP полагается на TCP, а не на UDP. Протокол имеет дизайн типа «клиент/сервер»: браузер выступает клиентским приложением, а веб-сервер ожидает подключения к порту 80. После установки соединения клиент отправляет запрос. Как только сервер его обработает, он тут же отправит ответ.

В 2015 году HTTP получил серьезное обновление[1], но версия протокола 1990-х годов все еще используется. Запросы на основе более ранней версии HTTP, задаются в виде обычного текста. В первой строке указывается *тип запроса*, *путь к документу* и *версия протокола*. Следующие строки используются для заголовков, аналогично сообщениям электронной почты. Пустая строка указывает на конец запроса:

```
GET /pets/cat.html HTTP/1.0
User-Agent: Mozilla/5.0 (Macintosh)
Accept-Language: en-us
```

[1] HTTP/2 — это последняя версия этого протокола. В ней добавились несколько улучшений, но общая структура передаваемой информации не изменилась.

В приведенном выше примере показан наиболее используемый тип запроса: `GET`. Он указывает, что клиент хочет получить документ. Заголовки необязательны, но почти всегда включаются. В примере показаны два наиболее распространенных заголовка. `Accept-Language` указывает, что предпочтительна английская версия документа, а `User-Agent` показывает, какое ПО и какая операционная система компьютера создали запрос. Типичный ответ сервера на такой запрос выглядит так:

```
HTTP/1.0 200 OK
Server: nginx/1.15.8
Date: Wed, 19 Aug 2020 20:46:53 GMT
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
Content-Length: 49
Content-Type: text/html

<html><body><h1>Cats are cute!</h1></body></html>
```

В первой строке указывается версия протокола, за которой следует код состояния в виде трехзначного кода, аналогичного тем, которые используются в SMTP. В данном случае `200` означает, что запрос прошел без каких-либо проблем.

Существует и много других кодов. Например, `404` указывает, что запрошенный документ не может быть найден. Коды группируются по своей первой цифре. Те, которые начинаются с `2`, сигнализируют об успешных запросах, а те, которые начинаются с `4`, — об ошибках, вызванных неподходящими запросами.

После первой строки следуют необязательные заголовки. В нашем примере есть заголовки, информирующие клиента о программном обеспечении сервера, дате отправки сообщения, времени последнего изменения запрошенного документа, типе документа и его общей длине. Многие другие заголовки широко используются для аутентификации, отслеживания, кэширования и многого другого.

HTTP обрел невероятный успех и стал использоваться за пределами Всемирной сети. Например, приложение в вашем смартфоне может отправить HTTP-запрос для получения необработанных данных о погоде. Полученные данные затем можно использовать для отображения погодных условий с помощью виджета в интерфейсе телефона, а не на веб-странице, предоставленной сервером.

ВЕБ-ПРИЛОЖЕНИЯ

HTTP-запрос может содержать путь, ссылающийся на программу. В таких случаях сервер запускает программу и отвечает ее выводом. Например, запросы `/random`, отправленные `code.energy`, относятся к программе, которая генерирует страницу, содержащую случайное число. Сервер возвращает **динамическую страницу** вместо статической, которая выглядит одинаково при каждом запросе. Зайдите на эту страницу через браузер, обновите ее несколько раз и убедитесь сами:

```
http://code.energy/random
```

Динамические страницы также способны получать входные данные от пользователя в качестве параметров. Например, `google.com` имеет динамическую страницу `/search`, которая принимает параметр с именем `q` в качестве поискового запроса. Параметр добавляется после пути к документу следующим образом:

```
http://google.com/search?q=energy
```

Чтобы загрузить эту веб-страницу, браузеры могут отправить следующий запрос:

```
GET /search?q=energy HTTP/1.0
```

Получив такой запрос, веб-сервер вызывает программу, связанную с `/search`. Любой параметр в форме `?name=значение` после пути будет передан в программу. Такая форма входных параметров называется **строками запроса**.

Поскольку есть символы, которые не могут содержаться в допустимых URL-адресах, есть стандартный способ их кодирования с использованием знака процента. Например, пробел[1] превращается в `%20`, а `!` становится `%2F`. Строки запроса могут содержать несколько параметров, разделенных амперсандами (`&`). При посещении следующего URL-адреса выполняется

[1] В строках запроса пробел может быть закодирован либо как %20, либо просто символом плюс (+). Но это не будет работать в каком-либо другом месте URL-адреса. Символы, имеющие специальные функции (например, +, % ?, = и &), также должны быть закодированы для обхода этих самых функций.

поиск фразы code energy и передается дополнительный параметр num для перечисления только пяти результатов:

```
http://google.com/search?q=code%20energy&num=5
```

Строки запросов отлично интегрированы с HTML. Некоторые теги, включая <form> и <input>, могут быть добавлены на веб-страницы, чтобы пользователь мог легко вводить и отправлять данные с помощью строк запроса.

Такой HTML-код создает поле ввода для ввода ключевых слов и кнопку для отправки запроса:

```
<form action="//google.com/search">
    <input name="q">
    <button>Search Google</button>
</form>
```

После нажатия кнопки создается HTTP-запрос типа GET, передающий введенный текст в виде строки запроса. На стороне сервера программа ищет ссылки, которые считает наиболее подходящими для пользователя, и отправляет их обратно в документе веб-страницы.

Как правило, программы, отвечающие на HTTP-запросы типа GET, не должны выполнять никаких действий, кроме извлечения информации. Например, запрос GET не должен позволять пользователю размещать сообщение о статусе в социальных сетях, удалять фотографию с облачного диска или размещать заказ в интернет-магазине. Это означает, что пользователи могут отправлять запросы GET, не беспокоясь о последствиях, отличающихся от предусмотренных на их веб-странице.

Для выполнения этих и других действий есть несколько других типов HTTP-запросов. Например, если желаемый эффект заключается в создании чего-либо, следует использовать тип POST. HTML-форму можно переключить на отправку POST-запросов, установив параметр метода в теге <form>:

```
<form action="/send-message" method="POST">
    <p>Subject: <input name="subject"></p>
    <p>Message: <input name="message"></p>
    <button>Send message</button>
</form>
```

В POST-запросах входные данные передаются в теле запроса, а не в строке запроса. Однако данные по-прежнему кодируются тем же образом. Вот как выглядит POST-запрос, который мог бы быть сгенерирован после отправки формы в приведенном выше примере:

```
POST /send-message HTTP/1.0

subject=Greetings&message=Hello%20World%2B
```

Можно создавать сложные системы, которыми реально управлять из веб-браузеров. Ярким примером здесь послужат почтовые сервисы — Hotmail и Gmail. Они позволяют людям использовать электронную почту через свой браузер. Веб-серверу доверено общаться с почтовыми серверами, позволяя пользователям отправлять письма без непосредственной установки каких-либо SMTP-соединений с веб-клиентом.

Чтобы облегчить подобные задачи, HTML-страницы могут включать в себя код JavaScript. С его помощью можно программировать веб-страницы, распоряжаясь теми же возможностями, что имеют другие графические приложения. Все чаще веб-страницы превращаются из простых гипертекстовых документов в полноценные приложения.

РЕЗЮМЕ

Мы узнали о некоторых наиболее известных интернет-приложениях и их протоколах, но есть и множество других, до которых мы не добрались. Например, из главы 1 мы узнали, что маршрутизаторы должны регулярно обмениваться информацией о подключении. С этой целью они запускают приложения, которые обмениваются данными с помощью **B-G**ateway **P**rotocol (**BGP**) на TCP-порте 179. Если вы хотите подробно узнать о BGP и других интересных протоколах, обратите внимание на ссылки в конце.

НОМЕРА ПОРТОВ

Мы увидели, что каждый протокол связан с определенным номером порта TCP или UDP. Приложения, использующие один и тот же протокол, почти всегда ожидают подключения к одному и тому же номеру порта. Например, вам не нужно знать, какой номер порта использовать

для получения веб-страницы — все веб-серверы ожидают подключения к TCP-порту 80. IANA — это организация, которая решает, какой номер порта является стандартным для каждого из протоколов. Когда создается новый протокол, его разработчики отправляют в IANA запрос на резервирование номера порта[1]. Вместо того чтобы разрабатывать новый протокол и подавать заявку на номер порта, многие разработчики интернет-приложений используют для связи через интернет общие протоколы. Например, многие приложения становятся доступными через интернет благодаря интеграции с веб-сервером. В таких случаях HTTP используется для передачи необработанных данных (а не веб-страниц) в приложения и из них (вместо браузеров).

ОДНОРАНГОВОЕ СОЕДИНЕНИЕ

Все рассмотренные в этой главе протоколы прикладного уровня следуют архитектуре «клиент — сервер». Но есть альтернативная архитектура, называемая **P**eer-**to**-**P**eer (**P2P**), где каждое приложение действует как сервер и клиент. Известные примеры протоколов P2P включают BitTorrent, протокол обмена файлами, а также Bitcoin и Ethereum — две крупнейшие криптовалютные сети.

Одноранговые (пиринговые) сервисы устраняют посредничество между пользователями и не имеют центральной точки отказа. BitTorrent, Bitcoin и Ethereum нельзя отключить. Вооруженные силы США широко используют технологию P2P для того, чтобы войска могли действовать без централизованной координации, не полагаясь на централизованную инфраструктуру, такую как вышки сотовой связи.

БЕЗОПАСНОСТЬ

Еще один важный аспект интернета, который мы не затронули, — это недостаток безопасности. Тот, кто управляет маршрутизатором, может считывать и изменять данные любого пакета, который до него доходит.

---

[1] Порты 0–49151 должны использоваться только протоколами, зарегистрированными в IANA. Порты 49152–65535 не приписаны к какому-либо протоколу и могут использоваться для любых целей. Можете проверить текущие назначения номеров портов здесь: http://code.energy/ports.

Мы видели, что IP-пакеты обрабатываются множеством неизвестных маршрутизаторов во время их прохождения через интернет. Поэтому мы должны рассматривать все данные, передаваемые в составе IP-пакетов, как потенциально общедоступную информацию. Следовательно, мы не можем быть уверенными, что данные, полученные через интернет, не были изменены по пути. Чтобы сгладить данный недостаток, протоколы прикладного уровня часто включают схемы шифрования, чтобы данные не могли быть прочитаны или незаметно подделаны посредниками. В следующей главе мы увидим, как это работает.

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

- *Куроуз Джеймс, Росс Кит*. Компьютерные сети. Нисходящий подход. — М.: Эксмо, 2016.
- *Таненбаум Э*. Компьютерные сети. — СПб.: Питер, 2019.

Глава 3

БЕЗОПАСНОСТЬ

> Со временем компьютерные системы не становятся более безопасными. Но я надеюсь, что успехи, которые мы делаем в криптографии, исправят плачевную ситуацию в области кибербезопасности.
>
> *Ади Шамир*

Перед программистами стоит задача защиты конфиденциальных данных, таких как личные сообщения, медицинские записи, банковские операции и т. д. И ваша задача — беречь данные, доверенные программам, от хакеров. Нужно убедиться, что лишь те, у кого есть допустимые учетные данные, получают доступ к вашим системам, а конфиденциальные данные должным образом зашифрованы и не могут быть прочитаны посторонними даже в случае утечки.

Способы защиты данных от атак со стороны неавторизованных сторон называются *криптографией*. Алгоритм, который обратимо шифрует открытые данные в непонятную форму, называется *шифром*. Большинство систем могут быть защищены с помощью криптографических библиотек, проверенных экспертами. В этой главе мы рассмотрим основные принципы шифрования и другие инструменты, предоставляемые криптографическими библиотеками. Вот что мы изучим:

устаревшие шифры и почему они небезопасны;

как они эволюционировали, чтобы стать безопасными **симметричными шифрами**;

асимметричные шифры, а также применение их в области шифрования сообщений;

как использовать цифровые отпечатки пальцев, полученные с помощью **хеширования**;

использование безопасных **протоколов** для защищенной работы в интернете;

методы взлома, которые злоумышленники используют для обхода безопасности, для предотвращения проникновений в систему.

Если зашифрованные данные будут перехвачены, есть шанс, что преступники выяснят, как расшифровать их и раскрыть все их тайны. Если такое происходит, мы говорим, что шифр, используемый для шифрования данных, был взломан. Начнем с изучения шифров, которые были популярны в прошлом и которые сегодня достаточно легко взломать. Это поможет понять, почему трудно взломать современные шифры, считающиеся безопасными.

3.1. УСТАРЕВШИЕ ШИФРЫ

Один из самых ранних шифров был использован Юлием Цезарем более двух тысяч лет назад для отправки тайных писем своим генералам. Его зашифрованные сообщения выглядели как-то так[1]:

```
GR QRW EULQJ DQB ERGB RI PHQ DFURVV WKH UKLQH
```

Если секретное письмо Цезаря перехватывали, информация, содержащаяся в нем, так и оставалась конфиденциальной. Враги Цезаря не могли понять смысла зашифрованных сообщений, но генералы Цезаря могли легко их читать. Чтобы зашифровать сообщения, римляне заранее договорились сдвинуть каждую букву исходного сообщения на три позиции вперед по алфавиту, заменив буквы вот так:

[1] В реальности сообщения выглядели немного иначе, потому что Цезарь знал только 23 буквы... и не говорил по-английски.

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
```

Затем сообщения расшифровывались путем восстановления положения букв. Этот шифр известен как **шифр сдвига**. Поскольку враги Цезаря так и не поняли, как он работает, он обеспечивал достаточную безопасность. Но любой, кто знает об этом шифре, может легко взломать его, даже если буквы сдвинуты более чем на три позиции: для латинского алфавита возможны лишь 25 сдвигов. Злоумышленник может взломать шифр, перебирая их один за другим, пока сообщение не обретет смысл.

Давайте введем несколько полезных понятий. Зашифрованное сообщение называется **шифротекстом**. Чтобы расшифровать шифротекст, необходимо знать, какой шифр был использован и какой **ключ шифрования** был применен. Для шифра сдвига ключом выступает число позиций, на которые сдвигаются буквы. Зная шифр и ключ, мы можем обратить шифрование и восстановить исходные данные, называемые **открытым текстом**.

ТАЙНЫЙ КОД 🤔 Листая страницы старой книги из библиотеки вашей бабушки, вы натыкаетесь на такое рукописное примечание:

```
MAXI KBVX HYLX VNKB MRBL XMXK GTEO BZBE TGVX
VTKX EXLL VHFF NGBV TMBH GLVH LMEB OXL
```

Она уже не в силах вспомнить, что означают эти загадочные надписи, но помнит, что, когда она была подростком, ей очень нравился шифр сдвига. Сможете ли вы восстановить открытый текст?

Самый простой способ взломать шифр Цезаря — проверить различные ключи на шифротексте и посмотреть, обретет ли выходной текст смысл. Есть всего 25 возможностей! Решение можно найти в приложении II.

ЗИГЗАГООБРАЗНЫЙ ШИФР

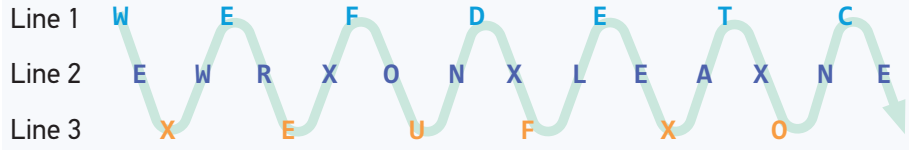
Есть способ зашифровать открытый текст без необходимости замены букв. Вместо этого буквы перемешиваются заранее. Рассмотрим следующее сообщение:

```
We were found. Flee at once.
```

Сначала удалим все ненужное форматирование, так как оно может дать подсказки касательно шифра. Как правило, все буквы заглавные, знаки препинания опущены, а пробелы либо удалены, либо заменены на X:

```
WEXWEREXFOUNDXFLEEXATXONCE
```

Теперь напишем каждую следующую букву на строке, отличной от предыдущей. Например, давайте следовать зигзагообразному узору по трем строкам:



Окончательный шифротекст получается путем соединения трех строк вместе:

```
WEFDETC
EWRXONXLEAXNE          → WEFDETCEWRXONXLEAXNEXEUFXO
XEUFXO
```

Это называется **зигзагообразным шифром**, и его ключом шифрования является количество строк, используемых в шаблоне. Этот шифр страдает от того же недостатка, что и шифр сдвига: доступно лишь ограниченное количество ключей. Можно легко взломать шифр, проведя обратный процесс для каждого возможного количества строк.

ШИФР ПОДСТАНОВКИ

Вернемся к шифрам, которые заменяют буквы в открытом тексте. Шифр, который имеет другое правило для замены каждой буквы, более безопасен, чем шифр сдвига. Например, вы можете договориться со своим другом о следующей карте подстановок букв:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
V H I E R P X N D J F T G L B W O Q K Z M U C S Y A
```

Это **простой шифр подстановки**. Так, слово ENERGY будет зашифровано как RLRQXY. Ключ шифрования — это приведенная выше карта, указывающая, как именно подменяются буквы. Существует 26! способов перетасовки букв алфавита и создания различных ключей[1]. Иными словами, существует больше различных ключей, чем капель в океане. Было бы крайне непрактично перебирать их все по одному.

Несмотря на это, шифр оставляет в шифротексте шаблоны, которые можно проанализировать. С помощью определенного количества проб и ошибок на удивление легко взломать такой шифр, особенно если компьютер поможет нам с подсчетом букв и распознаванием словарных форм. Например, в типичных английских текстах буква E встречается наиболее часто. Если наиболее частая буква в шифротексте — это R, то есть большая вероятность, что подстановка E → R в ключе присутствует. Кроме того, TH — это пара букв, которые чаще всего встречаются рядом в английском языке. Если ZN является наиболее часто встречающейся буквенной парой шифротекста, мы можем предположить, что T → Z и H → N.

Это **частотный анализ** — он служит хорошей отправной точкой для пробы различных ключей шифрования. Попробуйте использовать его для решения следующей задачи.

> **ПУСТАЯ ШКАТУЛКА** Вы нашли старую шкатулку, которая казалась исключительно легкой. Вы уронили ее на пол, она треснула, и из нее выпал крошечный листок бумаги. С помощью увеличительного стекла вы читаете следующее:
>
> ```
> DUA KVYBVHA PVJ OAZQMASAO DW CWES PQLA
> KASJWFVZZC. AMASCDUQFH QJ VZZ SQHUD PQDU
> DUA LVGQZC. PA PQJU CWE JEYYAJJ. HSAADQFHJ
> LSWG DUA YWGSVOAJ.
> ```
>
> Сможете ли вы взломать этот шифр?

Поначалу задача может показаться сложной, но при наличии ручки, бумаги и некоторого терпения она вполне выполнима. Поскольку открытый текст, скорее всего, был на английском, мы начинаем с поиска зашифрованных букв, которые чаще всего означают E и TH. Шаг за шагом

[1] Восклицательным знаком обозначен факториал: 26! = 26 × 25 × ··· × 2 × 1.

мы сравниваем наполовину решенные слова с обычными английскими словами и видим, как различные возможные замены влияют на остальную часть текста. Один из возможных путей решения задачи представлен в приложении III.

ПРОДВИНУТЫЕ ПОДСТАНОВКИ Более сильные шифры подстановки могут преобразовывать букву открытого текста в различные символы. Например, буква открытого текста `E` может быть заменена на `R`, `$` или `*`. При наличии большего числа вариантов замены наиболее частых английских букв частотный анализ усложняется. Но частые слова и пары букв все равно составят едва заметные закономерности в шифротексте. Можно придумать больше символов для замены пар букв и общих слов, что еще больше затруднит частотный анализ, но противостоять ему не получится.

ПРОДУКЦИОННЫЕ ШИФРЫ

Комбинация шифров называется **продукционным шифром**. Такие шифры более эффективны, так как шифры, которые перемешивают открытый текст, объединяются с шифрами, которые делают подстановки. Например, открытый текст может проходить через зигзагообразное шифрование, а затем через простой шифр подстановки. Такой продукционный шифр будет устойчивее любого из своих отдельных компонентов.

Предположим, что `TH` — наиболее часто встречающаяся пара букв в открытом тексте, что ожидаемо для английских текстов. Если `TH` заменен на `ZN`, то `ZN` уже *не будет* наиболее часто встречающейся буквенной парой шифротекста из-за зигзагообразной перетасовки. Тем не менее частотность букв шифротекста будет соответствовать частотности букв открытого текста, поэтому анализ возможен.

Во время Первой мировой войны немцы использовали продукционный шифр для радиосвязи. В шифре был этап подстановки, за которым следовал этап перемешивания. Немцы верили, что взломать такой шифр нельзя. Тем не менее французам это удалось, и они успешно подслушивали переговоры своих врагов. Зная планы Германии заранее, союзники могли предвидеть нападения и лучше управлять своими ресурсами.

ШИФР ВИЖЕНЕРА

До сих пор мы видели шифры, которые выполняют замены согласно одной схеме соответствия букв зашифрованного и открытого текста. Более сильные шифры подстановки используют *несколько* таких схем. Один из способов проделать подобное — проходить по списку шифров сдвига для каждой последующей буквы открытого текста. Например, используя ключ `2-0-1-1-0-6-4`, мы можем заменить буквы вот так[1]:

```
|T H E R E F O|R E T H E Y   L|O A T H E D   T|H E F T
|2-0-1-1-0-6-4|2-0--1-1-0-6--4|2-0-1-1-0-6--4|2-0-1-1-
|     ↓↓↓     |     ↓↓↓      |    ↓↓↓       |    ↓↓↓
|V H F S E L S|T E  U I E E  P|Q A U I E J  X|J E G U
```

Это называется **шифром Виженера**. Обратите внимание, что буквы открытого текста `E` могут быть закодированы как `F`, `E`, `G`, `K` или `I`. При использовании более длинных ключей каждая буква открытого текста будет иметь еще больше возможных замен. В результате шифр Виженера способен уменьшить угрозу частотного анализа. Этот шифр был изобретен в XVI веке и оставался нетронутым более 300 лет. Многие верили, что он несокрушим.

Сегодня шифр Виженера может быть легко взломан любым компьютером[2]. Наблюдая повторяющиеся последовательности букв в шифротексте, можно угадать длину ключа шифрования. Например, обратите внимание, что в примере шаблон `UIE` повторяется. Второе вхождение появляется через семь позиций после первого:

```
V H F S E L S T E U I E E P Q A U I E J X J E G U
                  1 2 3 4 5 6 7
```

Вполне вероятно, что шаблон, который появляется дважды в одном и том же шифрованном шифром Виженера тексте, соответствует шаблону, который также дважды появляется в исходном открытом тексте. Но это про-

---

[1] Ключи Виженера часто выражаются буквами, а не цифрами. Это позволяет использовать более короткие ключи, так как числа 0–25 записываются с использованием одного символа каждый (A–Z). Например, ключ 17-24-15-19-14-2 может быть записан как RYPTOC.

[2] Шифр Виженера был впервые взломан в 1845 году Чарльзом Бэббиджем — тем самым человеком, который разработал первый программируемый компьютер еще в 1837 году.

исходит только в том случае, если расстояние между двумя появлениями кратно длине ключа шифрования. Поскольку две последовательности UIE находятся на расстоянии семи позиций друг от друга, есть большая вероятность, что ключ содержит семь чисел. Если бы, например, они находились на расстоянии 15 позиций друг от друга, это было бы признаком того, что в ключе 3, 5 или 15 цифр.

Как только мы угадали длину ключа, можно использовать частотный анализ. Если длина ключа равна семи, мы извлекаем каждую седьмую букву зашифрованного текста. В этой группе мы находим наиболее часто встречающуюся букву, которая, скорее всего, заменяет `E`.

Достаточно повторить этот процесс еще с шестью начальными буквами, и шифр можно взломать, если текст достаточно длинный. Даже если зашифрованный текст короткий, как в нашем примере, частотный анализ подсказывает нам наиболее вероятные ключи для проверки, и решение может быть найдено с небольшой вычислительной мощностью по современным стандартам[1].

ШИФР ВЕРНАМА

Шифр Виженера обладает наибольшей устойчивостью, если ключ шифрования содержит столько цифр, сколько букв в открытом тексте. Мы называем эту вариацию **шифром Вернама**. Математически доказано, что невозможно взломать шифр Вернама, если ключ выбран случайным образом и используется только *один раз*. В противном случае возникают закономерности.

Мы рассмотрим два шифротекста: `VSXOCQZQCLGHQGBFHTJK` и `NHPJEXGPTDDBFCFOYKRA`. Если они были зашифрованы одним и тем же ключом шифрования Вернама, то у нас в рукаве появляется один козырь. Во-первых, мы выбираем общее слово, которое, скорее всего, будет в открытом тексте и которое мы называем **опорным**. Мы предполагаем, что опорное слово находится в одном из открытых текстов, и проверяем, как это влияет на другой. Используем слово `tomorrow` в качестве опорного (рис. 3.1).

[1] Здесь имеется инструмент для взлома шифра Виженера: http://code.energy/vigenere.

шифротекст 1	V	S	X	O	C	Q	Z	Q	C	L	G	H	Q	G	B	F	H	T	J	K
	T	O	M	O	R	R	O	W												
	2	4	11	0	11	25	11	20												
шифротекст 2	N	H	P	J	E	X	G	P	T	D	D	B	F	C	F	O	Y	K	R	A
	L	D	E	J	T	Y	V	V												

Рис. 3.1. Если первая буква открытого текста 1 — это T, то первое число ключа должно быть 2, так что T → V. В соответствии с этим предположением первая буква открытого текста 2 должна быть L, так что L → N

Если бы слово `tomorrow` было в начале первого открытого текста, второй открытый текст должен был бы начинаться с `ldejtivv`. Если предположить, что ни один из открытых текстов не содержит тарабарщины, слово `tomorrow` не может быть в начале первого открытого текста. Поэтому мы продолжаем пробовать опорное слово в других позициях. И на седьмой позиции — успех (рис. 3.2).

шифротекст 1	V	S	X	O	C	Q	Z	Q	C	L	G	H	Q	G	B	F	H	T	J	K
							T	O	M	O	R	R	O	W						
							6	2	16	23	15	16	2	10						
шифротекст 2	N	H	P	J	E	X	G	P	T	D	D	B	F	C	F	O	Y	K	R	A
							A	N	D	G	O	L	D	S						

Рис. 3.2. Ура! 🎉 Тестируя tomorrow в седьмой позиции первого открытого текста, мы получаем ключевой сегмент, который декодирует другой зашифрованный текст в понятный текст, а не в тарабарщину. Это подтверждает правильность предположения

Этот трюк называется **crib-dragging**. Чтобы сделать шифр Вернама невосприимчивым к нему, мы *никогда* не должны использовать один и тот же ключ шифрования для двух разных открытых текстов. По этой причине мы обычно называем ключ шифрования Вернама **ключом одноразового использования**.

В 1940-х годах Красная армия иногда повторно использовала одноразовые ключи при передаче сообщений, зашифрованных шифром Вернама. Американской разведке удалось перехватить их радиопередачи и в конце концов взломать код с помощью приема crib-dragging. Они обнаружили

среди прочего, что советские шпионы уже проникли в программу создания ядерного оружия.

По сей день шифр Вернама является золотым стандартом секретного взаимодействия. Это единственный шифр, который, как оказалось, не поддается взлому при правильном использовании. Но это не очень практично: стороны должны поделиться идентичной большой последовательностью секретных случайных чисел, прежде чем начать обмениваться зашифрованными сообщениями. Безопасный шифр, работающий с более коротким ключом, был бы гораздо более универсальным.

ШИФРОВАЛЬНЫЕ МАШИНЫ

В 1920-х и 1930-х годах военные державы по всему миру нуждались в системах для быстрого шифрования и дешифрования сообщений без необходимости прямого обмена большими ключами шифрования. Поэтому они разработали машины, которые использовали меньшие формы общих секретных данных в качестве суррогатов больших ключей. Новые устройства использовали хитрости, которые позволяли их коротким совместно используемым секретным ключам постоянно расширяться до все больших и больших ключей шифрования, называемых **ключевыми потоками**.

Первые шифровальные машины состояли из последовательности колес, каждое из которых имело собственную сложную внутреннюю проводку (рис. 3.3). Прежде чем зашифровать или расшифровать каждую букву, колеса согласованно перемещались в новое положение. В каждой позиции их провода образовывали совершенно другую схему, расширяя ключевой поток новой, *казалось бы, случайной* схемой.

Сообщения *выглядели* так, как будто они были зашифрованы с помощью бесконечно длинных случайных ключей шифрования. На самом деле поток ключей был не случайным, а **псевдослучайным**: тот же самый поток ключей мог быть воспроизведен в любое время на другой копии машины. Все, что требовалось, — это знание совместно используемого ключа[1]: начального положения колес.

[1] Мы часто используем термин «ключ шифрования» для обозначения короткого совместно используемого ключа, а не ключевого потока, который он создает.

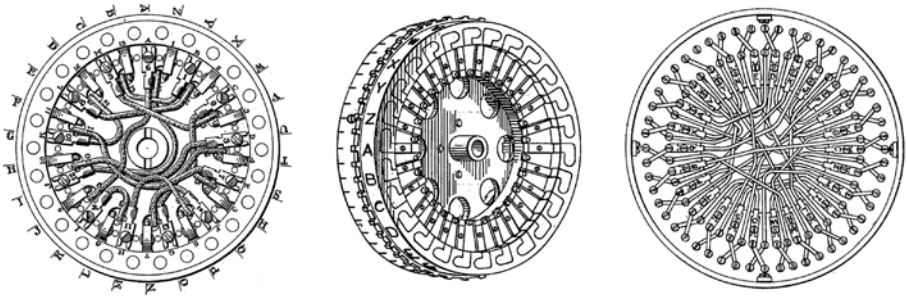


Рис. 3.3. Колеса шифровальной машины. Вращаясь, они образуют кажущиеся хаотичными схемы детерминированным и воспроизводимым образом

Шифровальные машины широко использовались во время Второй мировой войны, позволяя быстро шифровать и расшифровывать сообщения, используя короткие ключи вместо длинных одноразовых. Однако их ключевые потоки порождали едва заметные схемы, которые использовались взломщиками кодов с обеих сторон. Команды математиков и инженеров работали круглосуточно, чтобы взломать перехваченные шифротексты.

Все немецкие шифровальные машины были взломаны до конца войны, даже та, которая использовалась верховной ставкой Гитлера. Между тем самая мощная американская шифровальная машина, называемая SIGABA, так никогда и не была взломана. В последние годы было доказано, что шаблоны всегда будут проявляться при использовании ключевого потока, генерируемого вращающимися колесами, что означает, что даже SIGABA была уязвима.

К счастью, компьютеры позволяют работать с еще более сильными шифрами. А теперь узнаем о следующем шаге, который они позволили сделать от аналоговых шифровальных машин.

3.2. СИММЕТРИЧНЫЕ ШИФРЫ

Сегодня почти все наше шифрование зависит от компьютеров. Поскольку наши компьютеры работают с двоичными данными, мы вынуждены разрабатывать наши шифры для алфавита из двух символов: 0 и 1. Например, шифр Вернама может работать с двоичным алфавитом так же эффективно, как и с 26-буквенным алфавитом (рис. 3.4).

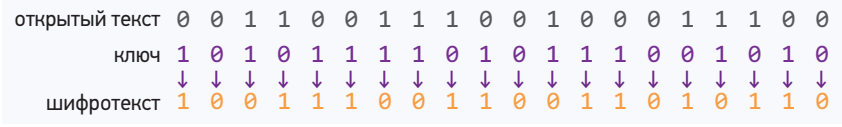


Рис. 3.4. Смещение двоичного открытого текста с помощью двоичного ключа одноразового использования. Когда цифра 1 смещается вперед, она зацикливается и возвращается к 0[1]

С помощью шифра Вернама любая информация, хранящаяся на компьютере, может быть зашифрована с полной секретностью. Но неудобство ключей одноразового использования никуда при этом не девается. Чтобы зашифровать гигабайт видео, необходим гигабайт секретных случайных чисел. И помните, ключи одноразового использования нельзя использовать повторно: crib-dragging в двоичном формате работает так же хорошо!

К счастью, есть безопасные шифры, которые способны шифровать большие объемы данных, используя относительно небольшой ключ совместного использования. Эти шифры делятся на два типа в соответствии с их основным рабочим принципом. Обратимся к самому простому.

ПОТОКОВЫЕ ШИФРЫ

Первый подход заключается в том, чтобы избежать длинных одноразовых ключей, аналогично случаю шифровальных машин с вращающимися колесами. Однако на этот раз мы пишем компьютерную программу, которая генерирует бесконечный поток псевдослучайных чисел. Например, с использованием неквадратичного числа (например, 1337) поток, казалось бы, непредсказуемых цифр появляется, когда мы вычисляем его квадратный корень:

$$\sqrt{1337} = 36{,}56\ 50\ 10\ 59\ 75\ 64\ 44\ 26\ 58\ 66\ 10\ 65\ 18\ 28...$$

Это простая форма генератора псевдослучайных чисел (**P**seudo**r**andom **N**umber **G**enerator, **PRNG**). PRNG требует **начальное значение**, служащее

[1] Этот процесс эквивалентен последовательным логическим операциям XOR. О логических операциях можно почитать в нашей первой книге «Теоретический минимум по Computer Science».

отправной точкой, аналогично совместному ключу, который описывал начальное положение колес старой шифровальной машины. В нашем примере начальное значение — это 1337.

Потоковые шифры полагаются на PRNG для генерации потоков ключей. Возьмем наш пример и сгруппируем цифры после десятичной точки в пары для генерации потока псевдослучайных чисел от 0 до 99 (рис. 3.5).

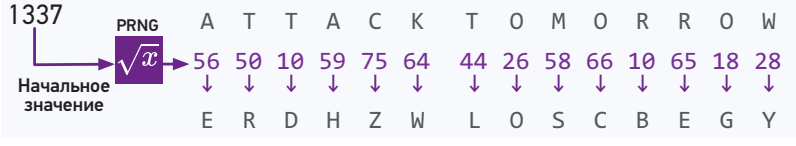


Рис. 3.5. Шифр Вернама, работающий на генераторе псевдослучайных чисел. Общий ключ между сторонами — это всего лишь начальное значение[1]

Сегодня потоковые шифры обычно используют PRNG, которые непосредственно генерируют поток двоичных цифр. Алгоритм PRNG является основой и определяющим элементом потокового шифра. Потоковый шифр безопасен только в том случае, если его PRNG выводит псевдослучайные числа без обнаруживаемых шаблонов. Даже если знать, какой PRNG использован, не обладая общим ключом, невозможно отличить вывод PRNG от последовательности действительно случайных подбрасываний монет.

Мы по сей день не знаем, есть ли в природе неуязвимый PRNG. Уязвимости были обнаружены во многих PRNG, предложенных экспертами в прошлом. Прежде чем выбрать потоковый шифр, узнайте, какие из них проверены экспертами и для каких нет известных уязвимостей[2]. Разработка хорошего PRNG чрезвычайно сложна, поэтому, если вы сами не хотите становиться криптографом, мы рекомендуем вам использовать существующие.

СЛУЧАЙНЫЙ КОД Поскольку потоки ключей имеют то же назначение, что и одноразовые ключи, их никогда не следует использовать более одного раза. Из-за того что PRNG всегда будет генерировать один и тот

[1] Мы также соглашаемся, что 26 соответствует сдвигу на 0 позиций, 27 соответствует сдвигу на 1 и т. д.

[2] Ищите рекомендуемые шифры здесь: http://code.energy/stream.

же поток ключей при задании одного и того же начального значения, начальное значение никогда не должно использоваться более одного раза. Если общий ключ непосредственно используется в качестве начального значения, сообщающиеся стороны должны быть в состоянии договориться о новом ключе перед каждым сообщением. Это непрактично, поэтому криптографы разработали способ, который позволяет нам повторно использовать общий секретный ключ. Хитрость здесь в использовании **случайного кода**: произвольное, однократное, несекретное число, которое может быть объединено с общим ключом для создания начального значения. На рис. 3.6 показан весь этот процесс.

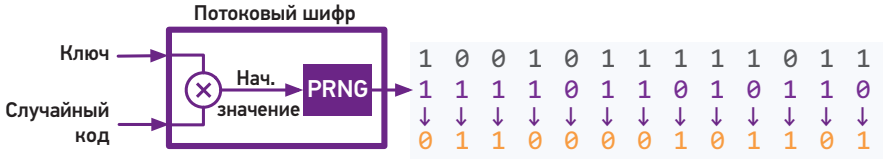


Рис. 3.6. Потоковые шифры обычно требуют общего ключа и случайного кода. В отличие от ключа, случайный код не обязательно должен быть секретным. Если случайный код никогда не используется повторно, ключ может быть использован для шифрования нескольких открытых текстов

Прежде чем шифровать что-либо потоковым шифром, вы должны выбрать случайный код. Если вы выберете случайное число, убедитесь, что его повторный выбор в будущем маловероятен. Например, вы можете сгенерировать случайное 64-битное число: поскольку существует более ста миллионов *триллионов* различных возможностей, вероятность выбора одного и того же случайного кода дважды почти равна нулю.

При передаче сообщения, зашифрованного потоковым шифром, отправьте незашифрованное сообщение, за которым следует зашифрованный текст. Получатель объединит входящий код с ключом, воссоздаст искусственный ключ одноразового использования и с помощью обратного сдвига получит открытый текст. Даже если хакеры перехватят много шифротекстов и узнают каждый из их кодов, они не смогут обнаружить какие-либо закономерности между ними, поскольку каждый из них был зашифрован своим собственным потоком ключей!

ВЫБОР КЛЮЧА Злоумышленник, который перехватывает ваш зашифрованный текст, может попытаться взломать его с помощью грубой

силы (метода полного перебора): перепробовать все возможные ключи, пока не появится понятный открытый текст. Будьте пессимистичны: предположите, что ваш злоумышленник знает, какой шифр вы используете, и обладает в несколько раз большей вычислительной мощностью, чем располагаете вы. Если вы выберете длинный и случайный ключ шифрования, то застрахуете себя от любой возможной атаки грубой силой. Например, 120-битный случайный ключ не оставляет злоумышленнику никаких шансов на грубый взлом ваших шифротекстов[1].

Например, Ада отправляет зашифрованное сообщение в банк с просьбой перевести 100 долларов на счет Эндрю. Предположим, что сообщение следует стандартному формату, где биты от 5 до 9 кодируют номер счета получателя. Чарльз все это знает и отвечает за передачу зашифрованного сообщения от Ады в банк. Если Чарльз в курсе, что номер банковского счета Эндрю `1001`, он может изменить сообщение Ады так, чтобы вклад ушел на его собственный номер счета `11100`.

```
                                  0 1 0 0 1  Номер счета Эндрю
Исходный шифротекст  0 1 1 1 0 1 1 1 1 1 0 0 0 1 1 1 0
                                  0 0 1 1 0  Ключевой сегмент
                                  1 1 1 0 0  Номер счета Чарльза
Поддельный шифротекст 0 1 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0
```

Если злоумышленник способен подделать части зашифрованного текста, только зная соответствующие части открытого текста, мы говорим, что шифр податливый. Как мы увидим далее, можно разработать шифры, которые не являются податливыми.

БЛОЧНЫЕ ШИФРЫ

В потоковых шифрах всегда есть четкая связь между открытым и зашифрованным текстом: если вы обратите десятый бит открытого текста (изменив его на `1`, если это `0`, и на `0`, если это `1`), это приведет к тому, что десятый бит зашифрованного текста тоже изменится. Наличие таких отношений — не очень хорошо. По сути, податливость потоковых шифров

[1] Ищите рекомендуемые шифры здесь: http://code.energy/stream.

является следствием как раз этого факта. Шифр более безопасен, если не создает очевидной связи между зашифрованным и открытым текстом.

Вспомните устаревший продукционный шифр, который объединял зигзагообразный шифр и простую замену. Даже он был сильнее, потому что зигзаг разрушал связь между зашифрованным и открытым текстом. Но зигзагообразного шифра недостаточно: в идеале связь между открытым и зашифрованным текстом должна быть настолько размыта, что изменение одного бита открытого текста приведет к изменению всего шифротекста.

В потоковых шифрах изменение одного бита ключа приводит к совершенно другому шифротексту. Со времен Второй мировой войны криптографы разрабатывали шифры, для которых изменение одного бита *либо* ключа, *либо* открытого текста полностью преобразовывало бы шифротекст.

Этого можно достичь с помощью продукционного шифра, который сочетает в себе простые операции подстановки и перемешивания. Однако избавить его от *всей* очевидной податливости в разных фрагментах шифротекста не так просто: продукционный шифр должен применяться несколько раз, а его операции подстановки и перемешивания должны быть тщательно отобраны. По факту операции для подходящего продукционного шифра могут быть найдены только при условии, что открытый текст всегда имеет один и тот же *фиксированный размер*.

Шифры, работающие по этому принципу, называются **блочными шифрами**, потому что открытый текст должен шифроваться в блоках фиксированного размера. Если размер открытого текста превышает размер блока, его необходимо разделить на несколько блоков. Если открытый текст недостаточно длинный, чтобы заполнить весь блок, он обычно дополняется нулями. В настоящее время наиболее распространены блочные шифры, работающие с блоками по 128 или 256 бит (рис. 3.7).

Блочные шифры уже не являются податливыми. Если один бит изменяется в блоке зашифрованного текста, то все сообщение расшифровывается в искаженном виде. По этой причине, даже если биты открытого текста известны, злоумышленник не может изменить шифротекст и создать поддельное сообщение.

Есть несколько способов разделения и шифрования открытых текстов, которые превышают размер блока. Самый простой — разделить и дополнить

открытый текст блоками нужного размера, а затем непосредственно применить блочный шифр к каждому из них. В таком случае мы говорим, что полученный шифротекст был зашифрован в режиме **E**lectronic **C**ode**b**ook (**ECB**) — электронной книги кодов.

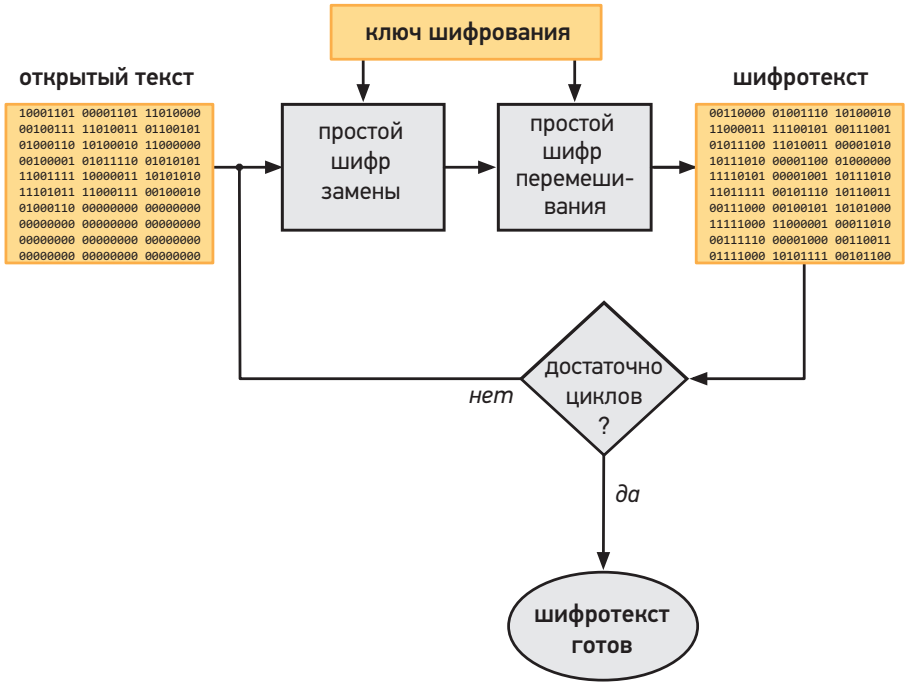


Рис. 3.7. Упрощенная схема блочного шифра. Как правило, блочные шифры должны повторять свой основной продукционный шифр около десяти раз

К сожалению, у ECB есть недостаток: злоумышленник может узнать, зашифрован ли один и тот же блок открытого текста дважды с использованием одного и того же ключа, так как это приведет к одному и тому же блоку зашифрованного текста. Например, если злоумышленник знает, что блок зашифрованного текста говорит банку зачислить 100 долларов на его счет, он потенциально может повторять это сообщение каждый раз, когда ему нужны дополнительные 100 долларов.

И будьте осторожны: эта уязвимость также может позволить злоумышленнику получить информацию о длинном открытом тексте, зашифрованном в режиме ECB. Предположим, что мы хотим зашифровать файл

изображения. Когда мы разделяем разделы файла изображения на блоки открытого текста, разделы одинакового оттенка и цвета дадут одинаковые блоки зашифрованного текста (рис. 3.8).

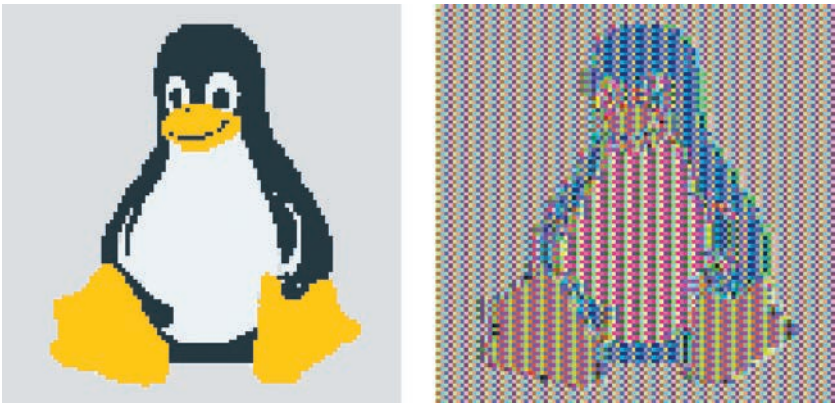


Рис. 3.8. Изображение, зашифрованное с помощью блочного шифра в режиме ECB. Паттерны открытого текста *(слева)* повторяются, и расположение этих повторных блоков в шифротексте *(справа)* раскрывает суть его содержания

Чтобы исправить это, преобразуем открытый текст так, чтобы каждый блок открытого текста имел уникальное значение. Для этого есть множество способов. Один из них — использовать случайный код размером с блок, который называется синхропосылкой (**I**nitialization **V**ector, **IV**).

- Начинаем с преобразования первого блока открытого текста, применяя случайный код, как если бы это был однократно используемый ключ, как было показано в начале этого раздела. Этот преобразованный открытый текст затем должным образом шифруется с помощью ключа блочного шифра.
- Для каждого последующего блока используем предыдущий блок шифротекста в качестве ключа одноразового использования для преобразования открытого текста. Каждый преобразованный открытый текст затем может быть безопасно зашифрован одним и тем же ключом блочного шифра.

Когда блочный шифр используется таким образом, мы говорим, что он работает в режиме сцепления блоков шифротекста (**C**ipher **B**lock

Chaining, **CBC**). Не нужно самостоятельно кодировать какие-либо из этих преобразований открытого текста. Вместо этого вы можете просто настроить свою криптографическую библиотеку так, чтобы ваш блочный шифр работал в нужном режиме. Есть и другие режимы, которые избегают ловушек режима ECB, но, если вы не знакомы с темой достаточно хорошо, рекомендуем придерживаться режима CBC!

Как и в случае с потоковыми шифрами, уязвимости были обнаружены во многих блочных шифрах, предлагавшихся экспертами до сегодняшнего дня. Прежде чем выбрать для использования блочный шифр, узнайте, какие из них в настоящее время рекомендуются криптографами[1].

БЛОЧНЫЙ ШИФР ПРОТИВ ПОТОКОВОГО Блочные шифры требуют большей вычислительной мощности, нежели потоковые. Когда для блочных шифров не нужна дополнительная безопасность, а открытый текст создается динамически (например, живое шифрование телефонного звонка), потоковый шифр может стать наилучшим выбором. Он позволяют шифровать и передавать открытый текст побайтно, в то время как блочные шифры требуют шифрования перед передачей больших фрагментов данных.

Потоковые и блочные шифры входят в более широкую категорию, называемую **симметричными шифрами**, поскольку требуют, чтобы обе стороны имели копию одного и того же ключа. Другими словами, вы можете обеспечить связь с помощью симметричного шифра только в том случае, если предварительно согласовали с другой стороной общий секретный ключ.

3.3. АСИММЕТРИЧНЫЕ ШИФРЫ

Прежде чем мы сможем общаться с помощью симметричного шифра, мы должны найти безопасный способ поделиться секретным ключом шифрования с нашим партнером. В прошлом людям приходилось встречаться лично или отправлять ключи с доверенным лицом. Сегодня есть способы, помогающие двум людям, которые никогда не встречались, поделиться секретным ключом без использования защищенного канала связи.

---

[1] В настоящее время лучшими рекомендуемыми блочными шифрами являются AES и Twofish с размерами блоков 256 бит.

ОБМЕН КЛЮЧАМИ ДИФФИ — ХЕЛЛМАНА

Предположим, Ада хочет отправить Чарльзу секретное сообщение, но они еще не выбрали общий секретный ключ. Есть метод обмена ключами **Диффи — Хеллмана**, при котором Ада и Чарльз совместно выбирают секретное число. Данный процесс требует отправки друг другу сообщения, *которое даже не должно быть секретным*. Из этих публичных сообщений третья сторона не может узнать, какое секретное число выбрали Ада и Чарльз. Разберемся, как это работает.

1. Ада выбирает простое число p и некоторое другое произвольное число g. Затем она выбирает секретное число a, которым не будет делиться.
2. Ада отправляет Чарльзу сообщение, содержащее p, g и третье число A, вычисленное с помощью *операции по модулю*[1]:

$$A = g^a \mod p.$$

3. Чарльз выбирает секретное число c и отправляет Аде сообщение, содержащее число C, вычисленное следующим образом:

$$C = g^c \mod p.$$

4. Ада вычисляет $C^a \bmod p$, а Чарльз вычисляет $A^c \bmod p$. По каким-то удивительным математическим законам оба они получили одно и то же число! Другими словами, их общий секретный ключ таков:

$$C^a \mod p = A^c \mod p,$$

где Ада — единственная, кто знает a, а Чарльз — единственный, кто знает c.

Чтобы такая схема была безопасной, числа должны быть большими: a, c, g и p должны быть выбраны таким образом, чтобы каждое число было длиной несколько сотен цифр. Кроме того, секретные числа a и c

[1] $a \bmod b$ — это остаток от a / b. Например, $27 \bmod 10 = 7$, потому что, когда мы делим 27 на 10, остаток равен 7.

должны выбираться случайным образом. Если требования соблюдены, у злоумышленников не остается разумного метода обнаружения общего секретного ключа без знания *a* или *c*.

Однако все еще существует дыра в безопасности, которая не закрывается обменом ключами Диффи — Хеллмана: как Ада и Чарльз могут быть уверены, что они на самом деле взаимодействуют друг с другом? Возможно, Ада получает *C не* от Чарльза, а от злоумышленника, выдающего себя за него. К счастью, криптографы разработали инструменты, которые способны решить эту проблему.

ШИФРЫ С ОТКРЫТЫМ КЛЮЧОМ

Шифры, которые мы видели до сих пор, используют один и тот же ключ для шифрования и расшифровки. Вдохновленные Диффи и Хеллманом, криптографы изобрели другой тип шифра, в котором второй ключ является производным от основного ключа. Тот, кто знает второй ключ, может выполнить операцию шифрования. Но для расшифровки требуется основной ключ. Шифры, использующие разные ключи для шифрования и дешифрования, называются **асимметричными шифрами**.

С симметричными шифрами есть только один ключ, который известен всем, кто участвует в секретном обмене сообщениями. Ключ полезен только для этой группы людей, и он всегда должен оставаться *совместно используемым секретным ключом*. В асимметричных шифрах имеется два ключа. Главный ключ — это **закрытый ключ**, и он известен только одному лицу (или организации). Второй ключ — **открытый**, и он в идеале общедоступен. Одна пара закрытых и открытых ключей может служить многим различным контекстам связи.

Предположим, Чарльз хочет связаться с Адой. Если Чарльз знает открытый ключ Ады, он может использовать его для шифрования своего текстового сообщения. Затем он передает зашифрованный текст Аде, и Ада может использовать свой закрытый ключ для его расшифровки. Если Эндрю также хочет общаться с Адой, он может следовать тому же процессу, по сути используя ту же пару ключей в другом контексте! Но если Ада захочет ответить им обоим, ей придется использовать открытый ключ Чарльза, чтобы зашифровать сообщение для Чарльза и открытый ключ Эндрю в сообщении для Эндрю.

В использовании шифров с открытым ключом все еще есть свои подводные камни. Например, когда Ада получает зашифрованный текст от Чарльза, как Ада может быть уверена, что сообщение действительно пришло от Чарльза, а не от самозванца? Открытый ключ Ады является *открытым*, поэтому любой может отправить ей зашифрованное сообщение, утверждая, что он Чарльз. Кроме того, если Чарльз не может физически встретиться с Адой, как он может получить копию ее ключа и убедиться, что ключ является достоверным?

ЦИФРОВЫЕ ПОДПИСИ

Помимо шифрования и расшифровки, были созданы еще две операции, основанные на асимметричной криптографии: **подпись** и **проверка**. Операция подписи принимает открытый текст и закрытый ключ в качестве входных данных и выдает большое число, называемое **цифровой подписью**. Операция проверки принимает открытый ключ, открытый текст и цифровую подпись в качестве входных данных и выдает значение `true` или `false` в зависимости от достоверности цифровой подписи.

Единственный способ создать законную цифровую подпись для этого открытого текста и открытого ключа — это выполнить операцию подписи с соответствующим закрытым ключом. При асимметричной криптографии закрытые ключи связаны с конкретным человеком. Использование вашего закрытого ключа для создания цифровой подписи для данного открытого текста является цифровым эквивалентом постановки вашей подписи с открытым текстом. Единственная разница заключается в том, что традиционная подпись всегда выглядит одинаково, а цифровая подпись выглядит *совершенно* по-разному для каждого нового простого текста , даже если в нем изменена всего одна буква.

Вычислите цифровую подпись, опубликуйте ее вместе с сообщением, и любой, кто знает ваш открытый ключ, сможет выполнить операцию проверки и подтвердить, что именно вы создали эту подпись. В то время как мошенники обычно подделывают физические подписи, никто пока не понял, как подделать цифровую подпись, созданную с помощью надежной криптографии.

Цифровые подписи позволяют безопасно общаться с кем угодно, даже если злоумышленники проникают в каналы связи. Вспомните обмен

ключами Диффи — Хеллмана: единственная проблема заключалась в том, что участники не могут быть уверены, что числа, которые они отправляют друг другу, не будут в процессе передачи изменены злоумышленниками. Когда сообщения в обмене ключами Диффи — Хеллмана сопровождаются цифровыми подписями, самозванец не в силах скомпрометировать процесс. Если злоумышленник вмешивается в сообщения, цифровые подписи не совпадают и люди могут понять, что их обманывают, прежде чем начнут обмениваться конфиденциальной информацией.

RSA и ECDSA По состоянию на 2020 год наиболее широко используемыми асимметричными криптографическими схемами выступают RSA и ECDSA[1]. RSA более универсален: его пары ключей можно использовать как для шифрования/дешифрования, так и для подписи/проверки. ECDSA является менее затратным в вычислительном отношении, но может использоваться только для подписи/проверки. Рабочие механизмы обеих схем основаны на более продвинутой математике. Пока вы будете следовать конкретным инструкциям по работе с каждой из этих схем, все будет в порядке. Например, при подписании с ECDSA необходимо указать случайный код. Если вы повторно используете его, безопасность закрытого ключа будет поставлена под угрозу!

ЦИФРОВЫЕ СЕРТИФИКАТЫ

Предположим, что Чарльз хочет отправить секретное сообщение Аде, но не знает ее открытого ключа. Если Ада отправит свой ключ Чарльзу по небезопасному каналу связи, злоумышленники могут перехватить сообщение и изменить его так, чтобы Чарльз получил вместо него их ключ. Если он попадется на такую уловку, его связь с Адой будет поставлена под угрозу перехвата злоумышленниками.

Пусть Чарльз знает открытый ключ своего доверенного друга Луи, а у Луи есть копия открытого ключа Ады. Луи может помочь ему, опубликовав сообщение с ключом Ады, за которым следует заявление: «Я, Луи, подтверждаю, что это открытый ключ Ады». Затем он подписывает это

[1] RSA означает Rivest — Shamir — Adleman (Ривест — Шамир — Адлеман), а ECDSA — Elliptic Curve Digital Signature Algorithm (алгоритм цифровых подписей на основе эллиптических кривых).

сообщение цифровой подписью и делает сообщение и подпись общедоступными. Теперь любой, кто доверяет Луи и имеет копию его открытого ключа, может получить сообщение и проверить подпись. Если подпись действительна, можно быть уверенными, что копия ключа Ады в сообщении Луи является достоверной — и все это без встречи с Адой!

Сообщение о том, что лицо или организация имеет этот открытый ключ, называется **цифровым сертификатом**. Доверенная организация, которая создает и подписывает цифровые сертификаты, называется центром сертификации (**C**ertificate **A**uthority, **CA**). Verisign является одним из наилучших таких аккредитованных центров сертификации. На самом деле, большинство компьютеров изначально продаются с открытым ключом Verisign, записанным вместе с операционной системой. Если Verisign подпишет цифровой сертификат своим открытым ключом, вы можете отправить сертификат и ключ по небезопасной ссылке практически любому человеку и он сможет проверить его подлинность.

Асимметричные шифры в тысячи раз более требовательны к вычислениям, чем симметричные. По этой причине они в основном используются для того, чтобы люди проверяли открытые ключи друг друга с помощью цифровых сертификатов и устанавливали симметричные ключи шифрования с подписанными сообщениями Диффи — Хеллмана. А теперь узнаем, как можно сделать подписание и проверку длинных сообщений менее трудоемкими с вычислительной точки зрения.

3.4. ХЕШИРОВАНИЕ

В первой главе мы узнали про контрольные суммы: функции, которые принимают любые входные данные и на выходе выдают число фиксированной длины. Если даже малая часть входных данных изменяется, контрольная сумма будет вычислена по-другому. С помощью криптографии были созданы специальные функции контрольной суммы, при которых невозможно узнать входные данные, для которых функция вычислила конкретные выходные данные (рис. 3.9). Эти специальные контрольные суммы называются **криптографическими хеш-функциями**, а их вывод — **хешем**.

Хеширование нересурсоемко, поэтому облегчает использование цифровых подписей. Подписывать большой файл с вычислительной точки

зрения непрактично. Вместо этого мы можем быстро вычислить хеш и подписать выходные данные. Подпись защищена, поскольку потенциальные злоумышленники не смогут создать другой файл с таким же хешем. Сегодня почти каждая подпись вычисляется по хешу подписанных данных. Существует несколько других способов использования хеш-функций. Давайте рассмотрим некоторые из них.

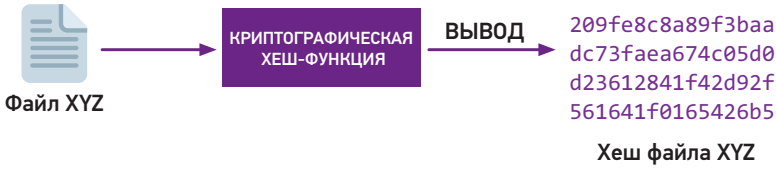


Рис. 3.9. Хеширование простого файла. Эта хеш-функция выдает двузначное число с 256 битами в качестве выходных данных, представленное в шестнадцатеричном формате. Невозможно найти другие входные данные, которые произведут тот же результат

ОБНАРУЖЕНИЕ ЗЛОНАМЕРЕННЫХ ИЗМЕНЕНИЙ

Для определения того, происходили ли случайные изменения в данном блоке данных, используются простые контрольные суммы. Но их нельзя использовать для обнаружения вредоносных изменений: когда злоумышленник пытается изменить файл, не будучи обнаруженным. Ценой некоторых усилий злоумышленник может собрать воедино поддельные данные, которые имеют ту же контрольную суммы, что и исходные данные.

> **АСТРОНОМИЧЕСКАЯ БЕЗОПАСНОСТЬ**  Лабораторный суперкомпьютер хранит данные астрономических наблюдений. Вам кажется, что глубокой ночью кто-то подменяет ваши данные с целью навредить исследованиям. Файлы большие, так что вы не можете сделать копию каждого файла. Как же можно гарантировать, что ваши данные не были злонамеренно изменены?

Ответ прост: выберите хеш-функцию и вычислите хеш каждого файла. Запишите все хеши на флешку, которую после этого заберете домой. Если вы подозреваете, что файлы изменились, снова вычислите их хеши. Если они такие же, как и на вашей флешке, значит, файлы ни на йоту не изменились. Конечно, если только злоумышленнику не удалось взломать

используемую вами хеш-функцию. Вскоре мы объясним, что делает хеш-функцию безопасной.

КОД АУТЕНТИФИКАЦИИ СООБЩЕНИЯ

Код аутентификации сообщения (**M**essage **A**uthentication **C**ode, **MAC**) — это способ доказать, что сообщение исходило от кого-то, кто знает конкретный ключ. В то время как MAC может быть реализован с помощью цифровых подписей, в плане вычислений его быстрее реализовать с помощью хеш-функций.

> **ГЛОБАЛЬНЫЙ ОПРОС**  Вы проводите опрос, охватывающий миллионы респондентов. У вас в планах отправить код опроса каждому из них, который как бы говорит: «Этот человек имеет право ответить на опрос как ■», где ■ заполняется в соответствии с именем или идентификатором каждого респондента. Как разрешить отвечать на ваш опрос только тем, кто представит достоверный код опроса?

Если вы просто отправите сообщения в открытом виде, злоумышленники могут изменить его и вставить собственные данные! Самый простой способ противостоять этим атакам — добавить большое случайное число к каждому коду опроса и вести учет всех созданных вами кодов. Когда незнакомец представляет код опроса с большим случайным числом, происходит обращение к вашей базе данных с целью определить, выдавали ли вы этот код опроса. Решение получается громоздким, потому что вам придется отслеживать миллионы кодов.

При асимметричном шифровании вы можете использовать цифровые подписи и отправлять каждому респонденту подписанный код опроса. Таким образом, вам вообще не нужно вести никаких записей. Когда код представлен вместе с подписью, вы проверяете свою собственную подпись, чтобы определить, создавали ли вы этот код опроса.

Имеется еще один способ достичь того же результата — использовать хеш-функцию. Во-первых, случайно сгенерируйте большое секретное число, известное только вам. Затем рассчитайте следующее для каждого респондента[1]:

[1] Здесь + — это операция конкатенации: code + energy = codeenergy.

```
msg ← "Eligible to answer as " + respondent_name
mac ← hash(secret_key + msg)
```

Код опроса можно создать, объединив `msg` и `mac`. Как только незнакомец представит код, вы можете использовать свое секретное число для пересчета `mac`. Если число `mac`, которое вы вычислили, совпадает с числом, которое представляет незнакомец, можно быть уверенными, что код опроса был выдан вами[1].

MAC пригодятся людям, общающимся с помощью потоковых шифров. Помните, что сообщения, зашифрованные любым потоковым шифром, уязвимы для атаки на податливость. Когда MAC добавляется в конце сообщения, с использованием ключа шифрования в качестве секретного ключа, получатель может пересчитать MAC и убедиться, что сообщение не было подделано в процессе передачи.

ОБРАБОТКА ПАРОЛЕЙ

Большинство компьютерных систем аутентифицируют пользователей, требуя ввести пароль. С этой целью системы должны вести учет всех паролей. Но хранить копии паролей — не лучшая идея. Если система будет взломана, пароли всех пользователей попадут в руки к хакерам. Поскольку многие люди повторно используют свои пароли, нарушение безопасности в одной системе может привести к нарушениям безопасности в других.

Хранение паролей в зашифрованном виде не облегчает проблему. Имея полный доступ к системе, злоумышленники могут выяснить, каким образом система автоматически расшифровывает зашифрованные пароли при входе в систему, и повторить процесс.

Гораздо надежнее вместо пароля хранить его хеш. В отличие от зашифрованного пароля его хеш необратим. Даже без копии пароля вы все равно можете проверить, правильно ли пользователь ввел пароль: вычислите хеш предоставленного пароля и сравните его с хешем, хранящимся в ва-

[1] В зависимости от хеш-функции, MAC, возможно, придется вычислять как hash(key + hash(key + msg)), а не hash(key + msg). Используйте криптографическую библиотеку для создания MAC и убедитесь, что они правильно рассчитаны.

шей базе данных. Если хеши совпадают, то пароль правильный[1]. Но, даже если компьютерная система хранит только хеши паролей, злоумышленники все равно могут узнать, что это за пароли.

АТАКА ПЕРЕБОРОМ ПО СЛОВАРЮ Злоумышленники могут вычислять хеши триллионов возможных паролей, создавая словарь, сопоставляющий хеши с открытыми паролями. Затем хеши паролей, украденные хакерами, можно будет найти в этом словаре. Это так называемый метод грубой силы (брутфорс), но выполняемый однократно. В 2012 году был взломан ресурс LinkedIn и выложены в открытый доступ хеши паролей более шести миллионов человек. Используя атаку перебором по словарю, можно раскрыть большинство этих паролей. Запомните: хранение хешей паролей может быть почти таким же ненадежным методом, как хранение копий самих паролей.

ПОДСАЛИВАНИЕ Есть способ обезопасить себя от атаки перебором по словарю. Вместо сохранения в БД значения `hash(password)` система может сгенерировать случайное число, называемое **солью**, и сохранить эту соль в открытом виде вместе с `hash(password + salt)`. Поскольку у каждого пользователя свое значение соли, это заставит злоумышленников трудиться над каждым хешем пароля индивидуально с помощью грубой силы. При этом, подбирая пароль данного пользователя, злоумышленники даже не знают, использует он простой пароль, вскрываемый перебором за пару минут, или сложный, взлом которого займет столетия.

НЕСКОЛЬКО КРУГОВ ХЕШИРОВАНИЯ Есть способ усложнить задачу злоумышленникам, пытающимся взломать хеши паролей методом грубой силы. Вместо того чтобы хранить `hash(password + salt)`, мы можем вычислить хеш хеша и сохранить `hash(hash(password + salt))`. Это в два раза затруднит злоумышленнику процесс угадывания правильного пароля. Мы можем сделать столько раундов хеширования, сколько захотим. Поскольку вычисление хеша выполняется быстро, некоторые системы выполняют тысячи раундов хеширования. Запомните: если вы храните хеши паролей, убедитесь, что вы используете соль и выполняете несколько раундов хеширования.

---

[1] Именно по этой причине компьютерные системы с надлежащим подходом к безопасности никогда не отправят ваш пароль по электронной почте, когда вы используете функцию «восстановить пароль». Они вообще не знают, какой у вас пароль!

ДОКАЗАТЕЛЬСТВО СУЩЕСТВОВАНИЯ

Хеш-функции могут использоваться для доказательства того, что некоторая информация существует, без раскрытия какой-либо части самой информации.

> **СКРЫТНАЯ РОК-ЗВЕЗДА**  Вы написали потрясающую песню, которая, по вашему мнению, станет следующим хитом типа *Hotel California*. Вы хотите сохранить ее в тайне и выпустить только в своем следующем альбоме, но боитесь, что кто-то может похитить шедевр. Можно заверить песню нотариально, но вы не хотите раскрывать тайну даже нотариусу. Как вы можете доказать, что написали песню, не раскрывая ее?

Ответ прост: запишите песню в файл, вычислите его хеш и попросите нотариуса записать этот хеш-номер. Если авторство песни оспаривается, вы можете представить свой нотариально заверенный хеш-номер вместе с записью песни. Единственный способ для вас предоставить нотариусу на заверение этот конкретный хеш-номер — иметь сам файл записи.

Сайты онлайн-казино также используют хеширование, чтобы их клиенты могли понять, что каждая игра была честной. Давайте проиллюстрируем это простой игрой. Вы можете удвоить свою ставку, если правильно угадаете бросок монетки, и проиграете в случае неудачи. Прежде всего онлайн-казино запрашивает у вас номер и случайным образом генерирует бросок монетки. Предположим, это решка. Затем казино вычисляет:

```
hash("HEADS" + user_salt + house_salt)
```

`house_salt` — это число, выбранное казино, а `user_salt` — ваше. Казино предоставляет хеш и ждет вашей попытки угадать. Если вы проиграете, казино может доказать, что вас не обманывали, раскрыв свой `house_salt`. Вы можете пересчитать хеш и проверить, соответствует ли он хешу, предоставленному казино до того, как сделали свое предположение. Казино никогда не обманывает! Чтобы убедиться, что вы все поняли, задайтесь вопросом, почему казино не может также сообщить свое число `house_salt`, *прежде* чем вы попытаетесь угадать?

ПОДТВЕРЖДЕНИЕ РАБОТЫ

Если вы продолжаете вычислять хеш-числа для разных входных данных, вполне ожидаемо, что в итоге вы столкнетесь с числами, подпадающими под определенные шаблоны. Например, в поисках хеша, который начинается с четырех нулей, вы можете выполнить перебор следующим образом:

```
hash("heads1") → 10111100000101101000011010010000 1…
hash("heads2") → 11011001100111011000011110111000 0…
hash("heads3") → 01110000011010011111110000101111 0…
hash("heads4") → 11000101000110000011010111010000 0…
hash("heads5") → 11101110000011110111101111110001 1…
hash("heads6") → 11001001011010101011010011001001 1…
hash("heads7") → 11000001001100101101101010010010 1…
hash("heads8") → 01001111100001010110100111111000 0…
hash("heads9") → 00111011011110101100010101010111 1…
hash("heads10") → 11101111000011111110100010110100 …
hash("heads11") → 10101011110111001100000111111001 …
hash("heads12") → 11101000110010001110100001001001 …
hash("heads13") → 00100100001011011101101101000100 …
hash("heads14") → 01111100010000100101101011000000 …
hash("heads15") → 00101110000011100011001001100000 …
hash("heads16") → 01010001000001011111011000000000 …
hash("heads17") → 11110011000111011100001001101111 …
hash("heads18") → 00111010001011001001100100110110 …
hash("heads19") → 00100111100101110100011010000110 …
hash("heads20") → 10111101011010100010011111110001 …
hash("heads21") → 10110010100010110001000111000010 …
hash("heads22") → 01111000100001111111010010001000 …
hash("heads23") → 10101111001001011100100001110000 …
hash("heads24") → 01110011110010011101111001101011 …
hash("heads25") → 00100010101101110000111000110110 …
hash("heads26") → 00001110001101011100100110011111 …
```

После 26 различных вычислений хеша мы имеем дело с хешем `heads26`. Это число, записанное в двоичном формате, начинается с четырех нулей! Теперь представьте, что кто-то предоставил вам следующее число:

```
hash("heads750") → 00000000101110000101101110110000 …
```

Здесь хеш начинается с *восьми* нулей! Можно предположить, что предоставивший эту строку должен был приложить некоторые усилия для вычисления множества хешей с целью найти входные данные, которые произвели хеш с таким шаблоном. Мы называем это **подтверждением работы**.

Предположительно, еще больше работы пришлось проделать, чтобы найти следующие входные данные, так как их хеш начинается с 12 нулей:

```
hash("heads11759") → 000000000000011101000001001001…
```

Доказательство работы может быть использовано для усложнения атак. Запрашивая у пользователей подтверждение работы перед отправкой запроса, вы заставляете злоумышленников делать много лишней работы при атаке вас поддельными запросами. Злоумышленникам придется потратить много вычислительных ресурсов, чтобы создать кучу допустимых запросов. Подтверждение работы также используется в некоторых криптовалютах, чтобы позволить людям чеканить виртуальные монеты после того, как представлено достаточное подтверждение работы.

БЛОКЧЕЙН Наряду с цифровыми подписями и хешированием принципы доказательства выполнения работы породили технологию, намеревающуюся изменить мир: блокчейн! На наш взгляд, через несколько лет благодаря блокчейну будет обеспечен экономический суверенитет миллиардов людей, устранены посредники и улучшены аспекты прозрачности, конфиденциальности и подотчетности. Наряду с блокчейном разрабатываются новейшие криптографические инструменты с причудливыми названиями, такие как «доказательство с нулевым разглашением» и «гомоморфное шифрование». Мы можем только мечтать о преимуществах, которые станут доступны в будущем благодаря этим технологиям!

НЕБЕЗОПАСНЫЕ ХЕШ-ФУНКЦИИ

Когда два разных входа приводят к тому, что хеш-функция выдает один и тот же результат, мы называем это **коллизией**. С помощью безопасных хеш-функций единственным методом поиска коллизий является грубая сила: попытка вычислить хеш различных входных данных до тех пор, пока случайно не возникнет коллизия.

Когда хеш-функции выдают выходные данные с более чем 200 битами, становится уже невозможно отыскать коллизии с помощью грубой силы[1].

[1] Если хеш-функция выдает 200-битные хеш-числа, возможны 2^{200} значений хеша. Если взять все песчинки на земле и умножить на 200 триллионов, мы даже не приблизимся к 2^{200}. Представьте, чего будет стоить найти конкретную песчинку в таком пространстве поиска.

Для всех хеш-функций, которые в настоящее время считаются безопасными, не существует известных коллизий. Иногда исследователи находят способы найти коллизии в определенной хеш-функции, не полагаясь при этом на грубую силу. Как только такой обходной путь обнаружен, подверженная ему хеш-функция становится небезопасной.

Так произошло с хеш-функциями MD5 и SHA1, которые считались безопасными в 1990-х годах. Десять лет спустя обе были признаны небезопасными. Сегодня для этих хеш-функций известны коллизии. В настоящее время имеется несколько хеш-функций, которые эксперты по криптографии считают безопасными. Наиболее широко используемой безопасной хеш-функцией является SHA2. По состоянию на 2021 год эта хеш-функция была безопасной в течение уже почти двух десятилетий. Однако в будущем все может измениться. Если вы используете хеш-функции, обязательно держите нос по ветру, чтобы отреагировать как можно раньше, если выбранная вами хеш-функция станет небезопасной.

3.5. ПРОТОКОЛЫ

Как вы помните из предыдущей главы, все данные, передаваемые через IP-пакеты, потенциально являются общедоступной информацией. Хакеры, которые проникают в телекоммуникационные компании, могут изменять IP-пакеты, идущие к их цели и от нее, чтобы нанести ущерб[1]. Мы в силах защитить себя от подобных опасностей с помощью криптографии. Как правило, безопасная связь через интернет включает в себя следующие шаги.

1. Получить достоверный открытый ключ нашего партнера.
2. Использовать алгоритм Диффи — Хеллмана, чтобы создать новый секретный ключ, общий с другой стороной. Сообщения, которыми обмениваются в процессе, проверяются на подлинность с помощью ключа, полученного на шаге 1.
3. Использовать общий секретный ключ с симметричным шифром из шага 2 для шифрования всех последующих сообщений.

[1] В 2013 году Эдвард Сноуден сообщил, что АНБ уже проводило подобные атаки даже против действующих президентов союзных стран.

Когда стороны знают открытый ключ друг друга, можно безопасно общаться уже без алгоритма Диффи — Хеллмана. Однако это создает угрозу: если когда-либо секретные ключи будут скомпрометированы, злоумышленник, записавший прошлые сообщения, сможет расшифровать все. Если каждый раз генерировать новый секретный ключ Диффи — Хеллмана, хакеры не смогут расшифровать прошлые сообщения, даже если получат доступ к компьютерам своих жертв. Говорят, что схема связи с такой особенностью имеет **прямую секретность**.

Чтобы безупречно зашифровать наши сообщения, необходимо учесть ряд особенностей. Например, при непреднамеренном повторном использовании случайного кода безопасность всей связи ставится под угрозу. Для этого было создано несколько протоколов безопасности. Эти протоколы определяют процесс общения людей и выполнения необходимых криптографических шагов в рамках жесткого процесса, что снижает риск ошибок.

БЕЗОПАСНЫЙ ДОСТУП

В первые дни интернета хосты с работающими серверами telnet обычно принимали подключения от чужаков и запрашивали пароль только перед предоставлением удаленного доступа к оболочке. Это почти никак не влияло на безопасность: пароль передавался через интернет в виде открытого текста.

В 1995 году был создан протокол **S**ecure **Sh**ell (**SSH**), который послужил безопасной заменой telnet. Каждый сервер, принимающий SSH-соединения, должен иметь пару закрытых и открытых ключей. Для обеспечения безопасности клиент должен знать правильный открытый ключ, связанный с данным сервером[1].

По умолчанию SSH-серверы ожидают подключения на TCP-порте 22. После того как клиент устанавливает соединение, сервер отправляет свой открытый ключ вместе с криптографическими инструментами

[1] Во многих случаях человек или организация, управляющие клиентским компьютером, также владеют и сервером, поэтому эта задача так же проста, как передача серверного ключа на флеш-накопителе. В других случаях пользователь клиента должен получить открытый ключ из надежного источника, и риски безопасности тут такие же, как и для любого шифра с открытым ключом.

для шифрования, хэширования и аутентификации сообщения (**M**essage **A**uthentication **C**ode, **MAC**). Клиент сравнивает полученный открытый ключ с тем, что связан с данным сервером. Если клиент получает другой открытый ключ, это означает, что либо сервер был перенастроен, либо кто-то подделывает IP-пакеты в то время, как они перемещаются между клиентом и сервером. В этом случае пользователь должен прервать процесс и проверить, был ли сервер перенастроен для использования другого открытого ключа.

Если открытый ключ верен, клиент проверяет список шифров и инструментов, отправленных сервером, и выбирает, какие из них использовать. Клиент отправляет серверу свой выбор вместе с первым сообщением Диффи — Хеллмана. Сервер отправляет клиенту подписанный ответ Диффи — Хеллмана. Если подпись действительна, клиент и сервер вычисляют общий секретный ключ и начинают общаться с использованием симметричного шифра. На этом этапе клиент может безопасно отправить серверу сообщение с паролем. Как только соединение закрыто, клиент и сервер забывают свой общий ключ, чтобы сохранить секретность.

БЕЗОПАСНАЯ ПЕРЕДАЧА

Ранние пользователи интернета не доверяли своим веб-браузерам, когда имели дело с вещами, требующими высокого уровня безопасности, такими как онлайн-банкинг. Их беспокойство было вполне оправданно: с помощью HTTP вся информация передается в виде открытого текста в составе IP-пакетов. Инженеры решили включить шифрование в HTTP, чтобы люди могли безопасно вводить номера банковских карт и просматривать банковские выписки в своих веб-браузерах.

Они создали универсальный протокол под названием **T**ransport **L**ayer **S**ecurity (**TLS**)[1], который способен защитить *любое* TCP-соединение. Его рабочий механизм аналогичен SSH. Прежде всего клиент представляет список криптографических шифров и инструментов, которые он

[1] При своем создании в 1995 году протокол назывался Secure Sockets Layer (SSL). По мере обновления протокола он был переименован в Transport Layer Security (TLS), но многие по-прежнему называют его старым именем или SSL/TLS.

поддерживает, а также случайное число. В ответ сервер отправляет свой открытый ключ, набор используемых шифров и инструментов, а также другое случайное число. Клиент проверяет подлинность открытого ключа. Клиент и сервер вычисляют общий секретный ключ и начинают отправлять друг другу зашифрованные сообщения.

В отличие от SSH, клиент не должен заранее знать открытый ключ сервера. Сервер отправляет свой открытый ключ вместе с цифровым сертификатом. Например, если сервер размещен по адресу `example.com`, он должен отправить открытый ключ, а также цифровой сертификат от центра сертификации (**C**ertificate **A**uthority, **CA**), подтверждающий, что `example.com` использует этот открытый ключ.

После этих шагов зашифрованные HTTP-сообщения могут передаваться между клиентом и сервером. Когда TLS используется с HTTP, мы называем его HTTPS. В то время как порт 80 применяется для HTTP, для HTTPS используется порт 443. Веб-серверы, прослушивающие TCP-порт 443, ожидают, что клиент отправит первое сообщение TLS сразу после установления соединения.

TLS используется в нескольких протоколах, помимо HTTPS, включая SMTP. Внешние библиотеки облегчают программистам использование TCP с TLS. Вместо создания обычного сокета TCP программист создает сокет TLS в согласии с внешней библиотекой. Тогда библиотека может взять на себя всю работу по шифрованию. Программист может управлять сокетом TLS почти так же, как сокетом TCP.

ДРУГИЕ ПРОТОКОЛЫ

Есть несколько других протоколов безопасности. Например, IPSec — это расширение IP, которое обеспечивает безопасность всех коммуникаций на основе IP. В отличие от IP-пакетов пакеты IPSec не могут быть прочитаны или изменены по пути. IPSec часто используется для создания **виртуальных сетей (VPN)**, где интернет-соединение используется для замены физического соединения, так что компьютер может фактически присоединиться к сети, которая находится далеко от него.

DNS также имеет собственное безопасное расширение — DNSSEC. Оно дополняет записи DNS цифровыми подписями. Это позволяет любому пользователю убедиться, что подписанная DNS-запись была

создана законным владельцем домена. Без DNSSEC скомпрометированный DNS-сервер может отправить вам поддельные DNS-записи, чтобы перенаправить вас к вредоносному хосту. По состоянию на 2020 год DNSSEC еще не используется достаточно широко, но его распространение неуклонно растет. Широко используются протоколы, обеспечивающие безопасность беспроводной связи. Благодаря этим протоколам люди, у которых имеются радиосниффферы, не смогут прочитать содержимое IP-пакетов, передаваемых через защищенный Wi-Fi. Вы можете печатать на беспроводной клавиатуре Bluetooth без риска, что несанкционированный компьютер трасшифрует ваши нажатия клавиш. Сейчас электронная почта является большой проблемой в плане безопасности. Письма можно безопасно передавать с помощью SMTP с TLS, но почтовые серверы хранят все электронные письма в виде открытого текста. Крупный провайдер вроде Gmail может читать письма всех своих пользователей! В идеале сообщение должно быть прочитано только его отправителем и получателями. Системы связи, которые придерживаются этого принципа, используют **сквозное** шифрование. Приложения для обмена мгновенными сообщениями, такие как Signal, Threema и WhatsApp, используют сквозное шифрование. Активно разрабатываются новые протоколы для того, чтобы сделать сквозное шифрование доступным для электронной почты.

3.6. ХАКИНГ

Хакер — это человек с глубокими знаниями компьютерных систем, способный достичь определенных целей нестандартными способами. В поп-культуре хакеров часто изображают как странноватых чуваков, которые способны проникнуть даже в военные компьютерные системы всего за несколько нажатий клавиш. Фильмы не всегда реалистичны, но в мысли о том, что хакеры — это своего рода спецназ, есть доля правды. Хакеры часто специализируются на наиболее низкоуровневых аспектах работы компьютеров. Например, многие хакеры в подробностях знают, какие именно шаги делают компьютеры для выполнения того или иного действия.

Большинство хакеров превосходно понимают сетевые протоколы, способны манипулировать отдельными битами в сетевых пакетах и устраиваются на работу в сфере ИТ-безопасности. Обладая такими знаниями,

они обнаруживают лазейки, которые позволяют им делать на компьютере то, на что у них нет разрешения[1].

Это не значит, что все хакеры — злоумышленники. ИТ-специалисты («белые шляпы») усердно работают над тем, чтобы обнаружить уязвимости до того, как их смогут эксплуатировать злоумышленники. Напротив, злонамеренные хакеры («черные шляпы») обычно используют уязвимости для личной выгоды, невзирая на вред, который они наносят в процессе взлома.

Как правило, хакеры находят способы *обойти* (реже взломать) криптографию, которая защищает компьютерную систему, вместо того чтобы взламывать ее. Проектирование компьютерной системы похоже на строительство средневекового замка. Нужно предвидеть, как различные типы злоумышленников будут пытаться проникнуть в ваш замок, и предпринять контрмеры для каждого вектора атаки. Знание того, как работают хакеры, способно помочь. Но поразительно, что наиболее часто используемые механизмы взлома практически не требуют никаких технических навыков.

СОЦИАЛЬНАЯ ИНЖЕНЕРИЯ

Хакерские атаки чаще всего основаны на уязвимостях людей, а не оборудования. Обмануть кого-то, кто имеет доступ к компьютерной системе, часто гораздо проще, чем обойти правила самой системы. Злонамеренные хакеры проделывают это, выдавая себя за других несколькими способами: через электронную почту, телефонные звонки, текстовые сообщения. Изощренные злоумышленники могут создавать целые поддельные вебсайты. Некоторые могут физически явиться в центр обработки данных, выдав себя за сотрудников. Это называется атаками на основе **социальной инженерии**.

Такие хакеры действуют как мошенники: они разрабатывают сложные схемы, чтобы завоевать доверие и получит доступ. Самый распространенный трюк — **фишинг**. Злоумышленник подделывает электронное письмо, которое пришло якобы из надежного источника. Фишинговые

[1] Часто при этом используется Pwn — сленговое выражение, обозначающее акт взлома или проникновения в систему. Например, сайт Have I Been Pwned? (haveibeenpwned.com) проверяет, были ли данные, связанные с этим адресом электронной почты, когда-либо раскрыты при известном взломе.

Рис. 3.10. Любезно предоставлен http://smbc-comics.com

письма обычно содержат ссылки на поддельный веб-сайт, запрашивающий конфиденциальную информацию, такую как логины и пароли или номера банковских карт. Более сложные письма содержат вредоносное программное обеспечение, распространяемое во вложениях.

ФИШИНГ ВО ВРЕМЯ ВЫБОРОВ В США Во время президентских выборов в США в 2016 году хакеры создали фишинговое письмо и направили его члену Национального комитета демократической партии[1]. Письмо предупреждало о подозрительной активности в его учетной

[1] Демократический национальный комитет является руководящим органом Демократической партии, одной из двух основных политических партий США.

записи Google и призывало пользователя изменить пароль. Оно содержало ссылку на фальшивую страницу входа в Google, которая запрашивала пароль учетной записи, прежде чем можно было просмотреть подозрительную активность. Как только пользователь попался на эту уловку и ввел пароль, злоумышленники выгрузили тысячи конфиденциальных писем. Большинство из них попали в открытый доступ, причинив огромный политический ущерб и вызвав отставку нескольких ключевых политиков.

Подобные атаки происходят снова и снова. По оценкам, около 90 % всех атак с целью получения данных начинаются с фишинга. Есть вариация этого направления атаки, называемая **вишингом**. Хакеры выдают себя за важных людей, совершая телефонные звонки, чтобы получить привилегированную информацию или доступ к компьютерным системам.

ВИШИНГ ЦРУ В 2015 году ЦРУ обвинило 15-летнего британского хакера, который позвонил в Verizon, притворившись штатным сотрудником. Ему удалось получить ключевую информацию об особом клиенте Verizon, а именно директоре ЦРУ[1]. Вооружившись полученными данными, хакер затем выдал себя за самого директора во время звонка в техническую поддержку AOL. Он правильно ответил на все вопросы безопасности и изменил пароль электронной почты директора. В конечном счете юный хакер получил доступ к ключевым военным и разведывательным документам, собранным в ходе операций в Ираке и Афганистане.

Социальной инженерии можно противостоять путем обучения пользователей необходимости проверки подлинности электронных писем и веб-страниц перед раскрытием какой-либо личной информации. Важно также обеспечить строгую проверку подлинности пользователей в системе, когда те меняют свой пароль или изменяют какие-либо другие параметры безопасности. Но этих мер предосторожности мало. При более комплексных атаках жертве достаточно нажать на веб-ссылку или открыть вложенный документ, и тогда хакер получит полный контроль над компьютером жертвы.

---

[1] Центральное разведывательное управление (Central Intelligence Agency), или ЦРУ, — это учреждение в США, где хранятся разведданные, добываемые шпионами в других странах, чтобы помочь президенту в вопросах национальной безопасности.

УЯЗВИМОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Программисты знают, что зачастую написанные ими части кода работают не совсем так, как предполагалось. По мере того как софт усложняется, различные ситуации, которые он обрабатывает, умножаются экспоненциально, как и вероятность того, что возникнет ситуация, в которой комбинация входных данных приведет ПО к непредсказуемому, нежелательному поведению.

Это нежелательное поведение может привести к тому, что система обрушится и будет раскрыта секретная информация. В худшем случае хакер сможет выполнять любой код на компьютере, на котором размещена система. Мы называем последовательность входных данных, которая приводит к такому вот нежелательному поведению, **уязвимостью**. Далее кратко рассмотрим некоторые распространенные типы уязвимостей.

НЕДОСТАТОЧНЫЙ КОНТРОЛЬ ДОСТУПА Такое происходит, когда система выполняет потенциально опасное действие, не проверяя, есть ли у пользователя разрешение на его выполнение. Во многих случаях это происходит, когда разработчики забывают добавить соответствующие проверки в свой код. Это также может произойти, когда ПО или внешняя библиотека плохо настроены. В 2016 году техническая компания, работающая с данными избирателей, оставила неверно настроенную базу данных открытой для общественности без правильно настроенного пароля. В результате были обнародованы личные данные 154 миллионов избирателей США. Всегда читайте документацию от веб-серверов, серверов баз данных, серверов электронной почты и другого сложного ПО, которое вы решите использовать, после чего правильно настраивайте их для безопасного доступа.

SQL-ИНЪЕКЦИЯ — это самая распространенная уязвимость. Она позволяет хакеру запускать любой SQL[1]-код в базе данных, что зачастую позволяет злоумышленникам произвольно читать, записывать и удалять данные. Представьте, что вы директор школы и в вашу школу поступил новый ученик. Директор садится за школьный компьютер и вводит имя нового ученика. ПО управления школой использует имя, введенное

[1] SQL — это язык для управления системой баз данных, предназначенный для просмотра, вставки и изменения данных. Основы этого языка можно найти в нашей книге «Теоретический минимум по Computer Science».

директором, для отправки в свою базу данных SQL-запроса, подобного этому:

```
INSERT INTO Students (name)
VALUES ('■');
```

Здесь ■ заменяется именем, которое директор вводит в систему. Что произойдет, если он попытается зарегистрировать имя, представленное ниже?

```
Robert'); DROP TABLE Students;--
```

Если ПО школы уязвимо для внедрения SQL-кода, оно будет бездумно копировать все, что было введено в запрос. В итоге системой будет выполнен такой запрос:

```
INSERT INTO Students (name)
VALUES('Robert'); DROP TABLE Students;--');
```

При добавлении данных об ученике с этим специально созданным именем возникает вот какой побочный эффект: вся таблица удаляется из базы данных. Чтобы защититься от таких атак, проверьте входные данные и замените символы, которые могут вызвать побочные эффекты, такие как символ кавычек[1]. Атаки с внедрением SQL-кода происходят регулярно. В 2012 году хакеры атаковали несколько правительственных сайтов США, включая NASA, ФБР и Пентагон. В результате персональные данные более миллиона сотрудников и подрядчиков стали общедоступными в интернете.

Помимо инъекций, есть другие уязвимости, вызванные плохой обработкой входных данных. Всегда нужно проверять не только содержимое, но и *размеры* данных извне, которые вы пропускаете в свою систему.

ПЕРЕПОЛНЕНИЕ БУФЕРА Прежде чем программы смогут обрабатывать какие-либо входные данные, эти данные должны быть скопированы во внутреннее пространство памяти, называемое **буфером**. Если входные

[1] Помимо инъекций SQL-кода, есть и другие типы внедрений, при которых входные данные копируются в систему, вызывая непредусмотренные последствия. Например, при копировании пользовательских данных на HTML-страницу убедитесь, что символы < и > заменены для предотвращения внедрения тегов.

данные больше, чем буфер, и нет никаких проверок, чтобы остановить обработку входных данных, данные будут продолжать копироваться в память и *после исчерпания буфера*. При этом части входных данных оказываются в не предусмотренных для этого местах памяти. Это называется **переполнением буфера** и способно привести к сбою системы или еще худшим последствиям: если злоумышленнику удастся записать данные в определенное место, он сможет обманом заставить компьютер выполнить части ввода в виде кода. Когда злоумышленнику удается выполнить вредоносный код, как правило, компьютер целиком оказывается под контролем хакера.

Если хакеры обнаруживают на компьютере уязвимость переполнения буфера, они могут выполнять вредоносный код и зачастую получают полный контроль над компьютером жертвы. Хакерам сложно эксплуатировать уязвимости переполнения буфера, хотя они довольно часто встречаются в популярных операционных системах и потребительском ПО. Например, в 2015 году в программе Adobe Reader, используемой для просмотра PDF-файлов, была обнаружена уязвимость переполнения буфера. До того как Adobe выпустила обновление для устранения этой проблемы, хакеры могли создать PDF-файл, содержащий вредоносный код, который вызвал переполнение буфера. Когда пользователи открывали такой PDF-файл, вредоносный код выполнялся напрямую!

УЯЗВИМОСТИ НУЛЕВОГО ДНЯ Многие компании предлагают большие вознаграждения хакерам, которые ответственно сообщат о дырах в безопасности. Однако многие «плохие» хакеры и правительственные разведывательные агентства не раскрывают обнаруженные ими уязвимости, ведь они могут использовать уязвимость на избранных важных целях или продать уязвимость другим хакерам. Уязвимости, о существовании которых публично не известно и которые используются избранной группой хакеров, называются **уязвимостями нулевого дня (или просто 0-day)**. По оценкам, каждое широко используемое ПО или операционная система имеют несколько таких уязвимостей. Если это так, то почти каждый компьютер может быть взломан правительственными учреждениями и хакерскими группировками[1].

[1] Сообщалось, что из-за этой угрозы Федеральная служба охраны Российской Федерации в 2013 году прекратила использовать компьютеры, отдав предпочтение механическим пишущим машинкам.

ЭКСПЛОЙТЫ

Уязвимости обычно обнаруживаются даже в привычных программах, таких как операционные системы. После того как становится известно об очередной уязвимости, хакеры быстро пишут код для автоматизации атаки на уязвимые компьютеры. Такой код называется **эксплойтом**. В зависимости от уязвимости эксплойт может перевести уязвимую машину в автономный режим, раскрыть личную информацию или, что еще хуже, заставить компьютер выполнить код злоумышленника. Если эксплойт обнародован, то любой может использовать его для взлома уязвимых систем с минимальными усилиями[1].

С каждым днем количество известных уязвимостей растет и разрабатываются сложные инструменты взлома, такие как Metasploit, которые осуществляют автоматизированный поиск в системе каких-либо известных уязвимостей и запуск соответствующего эксплойта. Metasploit имеет большую, регулярно обновляемую базу данных уязвимостей и соответствующих эксплойтов. Программой Metasploit может воспользоваться не только хакер, но и ИБ-специалист, чтобы проверить компьютеры легитимной системы на наличие возможных уязвимостей. Компании, которые разрабатывают и поддерживают эти хакерские инструменты, верят в *наступательную безопасность*: облегчают использование известных уязвимостей, чтобы людям было также легко понять, что они уязвимы, и залатать дыры.

РУТКИТЫ И КЕЙЛОГГЕРЫ Злонамеренный код, который хакеры пытаются выполнить на компьютерах своих жертв, называется **полезной нагрузкой**. Существует два основных типа полезных нагрузок. **Руткиты** позволяют хакеру открыто получить доступ к оболочке хост-компьютера. **Кейлоггеры** записывают все, что набирается на клавиатуре хост-компьютера, чтобы впоследствии переслать хакеру. Если хакеру удается заставить своих жертв выполнить эти вредоносные полезные нагрузки, они, как правило, заметают следы и работают в фоновом режиме незамеченными, причиняя вред длительное время.

АНТИВИРУСНОЕ ПО Так называемые **антивирусы** проверяют все файлы и выполняемый код на компьютере. Их задача — обнаружить

[1] Любители, которые мало знают о безопасности и только и делают, что загружают и запускают эксплойты, созданные настоящими хакерами, называются скриптkидди (script kiddies, хакеры-дилетанты).

вредоносную полезную нагрузку и предотвратить ее запуск и дальнейшее распространение. Разработчики антивирусного программного обеспечения постоянно отслеживают и каталогизируют все найденные руткиты и кейлоггеры, а хакеры видоизменяют их, чтобы избежать обнаружения. Всегда существует несколько видов полезной нагрузки, которые еще не детектируются ни одним антивирусом. Антивирусное программное обеспечение лишь частично защищает от сложных атак.

Из-за быстрого распространения эксплойтов от разработчиков ожидают скорейшего выпуска обновления, исправляющего уязвимости в софте. Как только появляется обновление, исправляющее уязвимость, говорят, что она **пропатчена**. Люди, использующие уязвимое ПО, должны как можно быстрее применять обновления безопасности, чтобы избежать **взлома**.

Разработчики должны следить за используемыми библиотеками и за тем, чтобы не опираться на сторонний код с непропатченными уязвимостями. Аналогичным образом системные администраторы должны следить за софтом, который они запускают: ПО веб-сервера, почтового сервера, сервера баз данных и т. д. Важно убедиться, что они не используют ПО с известными уязвимостями.

Есть ресурс, который поможет в этой задаче: общедоступный список **C**ommon **V**ulnerabilities and **E**xposures, **CVE**. Как только уязвимость становится известной, она добавляется в список CVE и ей присваивается номер. Если вы обратитесь к типичным описаниям обновлений ПО, то увидите, что в них часто упоминают уязвимости безопасности, которые исправляются по их номеру CVE. Вы можете ознакомиться со списком CVE[1], чтобы узнать о любых известных уязвимостях в ПО, которое вы используете.

БОТНЕТЫ Некоторым хакерам удалось автоматизировать процесс сканирования интернета на наличие уязвимых компьютеров и запуска эксплойтов для внедрения своих руткитов. Некоторые из них настолько осторожны, что исправляют уязвимость, которую использовали для входа, так что ни один другой хакер не сможет взломать ту же машину. После многих лет автоматизированного использования эксплойтов некоторые хакеры в итоге имеют арсенал из множества подконтрольных компьютеров, называемый **ботнетом**. По оценкам экспертов, сейчас есть ботнеты,

[1] Список опубликован здесь: http://code.energy/cve.

работающие с сотнями тысяч компьютеров. Хакеры часто арендуют эти ботнеты у других, чтобы использовать их в качестве стартовой площадки для собственных атак. Если вы не будете следить за безопасностью своего компьютера, он может незаметно стать частью чьего-то ботнета!

БРАНДМАУЭРЫ Существует еще один тип программного обеспечения, который помогает защищать компьютеры от вредоносных эксплойтов: **брандмауэр (файервол)**. Эта программа блокирует нежелательные IP-пакеты в сети. Например, если к компьютеру не следует подключаться через протокол TCP, можно настроить брандмауэр таким образом, чтобы заблокировать все внешние IP-пакеты, инициирующие TCP-соединение. Это значительно усложнит для злоумышленников задачу проникновения с помощью эксплойта и внедрения вредоносной полезной нагрузки.

БЭКДОРЫ Уязвимости в компьютерных системах не всегда происходят из-за ошибки разработчика или пользователя. Уязвимость, которая была намеренно внедрена в систему программистом или инженером, называется **бэкдором**. Иногда правоохранительные органы или военные организаторы просят разработчиков аппаратного оборудования или программного обеспечения внедрить бэкдоры в свои системы. Хотя цель и может заключаться в добросовестном использовании (например, поимке преступников), злонамеренные хакеры также могут обнаружить и использовать бэкдоры.

Бэкдоры могут внедряться даже во что-то столь же фундаментальное, как алгоритм. Известны случаи, когда АНБ настаивало на использовании целенаправленно измененных с помощью бэкдоров криптографических стандартов. В 2004 году АНБ даже заплатило десять миллионов долларов США ведущему разработчику программного обеспечения с целью саботажа дефолтных алгоритмов. Когда дело доходит до выбора способа шифрования и криптографических инструментов в ваших системах, проведите собственное исследование и найдите те, которые рекомендованы независимыми академическими экспертами, которым вы доверяете.

ЦИФРОВАЯ ВОЙНА

Компьютерные системы нужны для функционирования любого современного государства. Они отвечают за коммуникации как в государственном, так и в частном секторе. Управляют электростанциями и энергетическими

сетями. Служат основой банков и рынков. На них полагаются авиадиспетчеры и радары. Железнодорожные сети и даже сами поезда управляются машинами. Они служат основой банков и рынков.

Мы уже становились свидетелями кибератак военного уровня. В 2010 году американские и израильские хакеры использовали уязвимости нулевого дня для написания вируса, который они назвали **Stuxnet**. Он был запрограммирован на проникновение в компьютеры на ядерных объектах Ирака. Вирус медленно распространялся, пока не заразил компьютеры, управляющие центрифугами по обогащению урана. Он давал команду центрифугам вращаться выше своей рабочей скорости, а затем внезапно останавливаться. Это медленно деформировало центрифуги, пока они не начали выходить из строя одна за другой. Тем временем вирус подделывал показания приборов, чтобы исключить обнаружение. Иракцам удалось обнаружить вирус, но если бы на это ушло еще несколько месяцев, у них бы ничего не вышло: Stuxnet был запрограммирован на удаление самого себя без каких-либо следов.

Stuxnet — самый опасный эпизод цифровой войны, о котором мы знаем, но это не был изолированный акт. Существуют намеки на то, что в настоящее время ведутся операции в рамках цифровой войны между странами и их хакерскими командами. Вооруженные силы многих стран уже вкладывают значительные средства в использование компьютерных сетей (**C**omputer **N**etwork **E**xploitation, **CNE**), то есть в обучение и разработку оружия, способного использовать важные компьютеры цели для разведки и шпионажа. Они также инвестируют в компьютерные сетевые атаки (**C**omputer **N**etwork **A**ttack, **CNA**), создавая цифровое оружие, способное уничтожить информацию и навсегда нарушить работу вражеских сетей.

Любопытная операция с использованием CNA была проведена израильскими военными в 2007 году. Бомбардировщики ВВС Израиля зашли вглубь сирийской территории и уничтожили целевые ядерные объекты, в то время как радары сирийской ПВО даже глазом не моргнули. Позже выяснилось, что израильтяне взломали компьютеры сирийских радаров, фальсифицировав их показания таким образом, чтобы израильские самолеты могли войти в воздушное пространство противника незамеченными, разбомбить цели и безопасно улететь, прежде чем сирийские зенитные подразделения смогут отреагировать.

В этой мрачной цифровой угрозе есть и светлая сторона. Жизни солдат сухопутных войск и шпионов теперь подвергаются гораздо меньшему

риску. Вместо того чтобы внедрять шпиона во вражеский штаб через процесс, который занимает годы, после чего агенты сталкиваются с постоянным риском раскрытия и смерти, теперь можно взломать компьютеры и скопировать документы. Вместо того чтобы запускать ракеты или солдат для уничтожения ядерного объекта, неся при этом большие потери, можно создать компьютерный вирус и уничтожить объект изнутри. Компании также ведут цифровые войны друг с другом, хотя и в меньшем масштабе. Они пытаются добраться до данных конкурентов в поисках секретных технологий или стратегических финансовых отчетов. Вдохновленные военными, компании обучают своих сотрудников, делая их хакерами, с помощью учений, когда синяя команда воюет против красной. В такой виртуальной игре красная команда пытается скомпрометировать или нарушить работу компьютерной сети, в то время как синяя команда пытается отловить атаки и остановить их. Часто компании даже нанимают хакеров извне для выполнения роли красной команды.

КВАНТОВЫЕ ВЫЧИСЛЕНИЯ Криптография и стратегии безопасности непрерывно развиваются. В наши дни военные и разведывательные службы могущественных стран вкладывают значительные средства в исследования и разработки квантовых компьютеров. Они вполне могут стать реальностью уже в этом столетии, и первая нация, которая их получит, потенциально *взломает* мир. Эти компьютеры за доли секунды смогут решать проблемы, которые наши нынешние суперкомпьютеры не способны решить даже за триллион лет. Как только будут созданы полнофункциональные квантовые компьютеры, почти все нынешние криптографические алгоритмы станут небезопасными. Поэтому ярчайшие умы уже сейчас неустанно работают над криптографическими стандартами будущего, которые обеспечат безопасность систем после квантово-вычислительной революции.

ЧЕК-ЛИСТ ЗАЩИТЫ

Вот шаги, которые хакеры с большой вероятностью попытаются предпринять при первом ударе.

1. Создать поддельную версию вашего сервиса и отправить электронные письма его пользователям, пытаясь заставить их ввести свои пароли в поддельной службе.

2. Позвонить вашим пользователям, представившись технической поддержкой, и попытаться заставить их раскрыть учетные данные или конфиденциальную информацию.
3. Поместить явные команды SQL во все входные данные, обрабатываемые вашей системой, которые, по всей вероятности, являются частью запроса к базе данных.
4. Попробовать выполнить привилегированные действия, такие как создание нового пользователя, без соответствующих разрешений.
5. Отправить очень большие строки в качестве всех возможных входных данных, обрабатываемых вашей системой, для проверки переполнения буфера.
6. Создать сценарий для взаимодействия с вашей аутентификацией и попытки взлома паролей с помощью грубой силы.
7. Использовать хакерские инструменты для сканирования всех ваших компьютеров на предмет обнаружения известных уязвимостей.

Подготовка системы к отражению всех этих атак — отличный первый шаг для повышения безопасности.

РЕЗЮМЕ

Криптография обеспечивает безопасность в виртуальном мире. Она позволяет работать с конфиденциальными данными, используя небезопасную инфраструктуру, и при этом сохранять секретность, целостность и подлинность данных. Виртуальная безопасность зависит от трех вещей: использования надежных криптографических инструментов, устранения уязвимостей ПО и обучения пользователей угрозам, связанным с социальной инженерией.

Мы изучили основные криптографические инструменты. Симметричные шифры используются для скрытой передачи и хранения данных. Асимметричные шифры применяются для цифровых подписей, цифровых сертификатов и создания общих секретных ключей по небезопасным каналам. Хеширование помогает обеспечивать целостность данных и работать с секретными ключами.

Подавляющее большинство криптографических инструментов рано или поздно окажутся взломаны, и из истории мы знаем о важности того, чтобы быть в курсе достижений в области криптографии. Мы узнали, что хакеры почти всегда атакуют системы, не взламывая криптографические инструменты, а находя уязвимости в виде человеческого фактора или багов в программном обеспечении. И цифровая война уже стала реальностью. Это означает, что критически важные компьютерные системы могут быть безжалостно атакованы хакерами, спонсируемыми вражескими государствами.

Эта глава была введением в ИТ-безопасность. Здесь лишь основы, которые, по нашему мнению, должен знать каждый программист. Если бы все разработчики ознакомились с концепциями этой главы, многих недавних разрушительных взломов систем безопасности можно было бы избежать.

Защита компьютерных систем от сложных атак — дело непростое, которым должны заниматься высококвалифицированные специалисты по безопасности. В конце концов, команда защиты должна защищаться от *всех* возможных атак. Злоумышленнику же нужно обнаружить только один вектор атаки, чтобы нанести ущерб. Для этого специалисты часто применяют несколько уровней мер безопасности, пытаясь предотвратить полную компрометацию системы в случае обнаружения уязвимости.

Как вы увидели, злонамеренные хакеры способны на многое, чтобы украсть данные. В некоторых случаях крупная утечка данных может даже обанкротить компанию или причинить серьезный ущерб государству. Но почему крупные объемы данных настолько ценны? В следующей главе мы займемся изучением и анализом больших объемов данных, разумеется легально полученных.

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

- *Singh S.* The Code Book. http://code.energy/singh.
- *Aumasson J.-P.* Serious Cryptography. http://code.energy/aumasson.
- *Mitnick K.* Ghost in the Wires. http://code.energy/mitnick.
- *Hickey M., Arcuri J.* Hands on Hacking. http://code.energy/hickey.

Глава 4

АНАЛИЗ ДАННЫХ

> Будущее анализа данных может быть связано с большим прогрессом. Но будет ли? Решение остается за нами, за нашей готовностью идти по нехоженым путям реальных проблем вместо ровной дороги нереалистичных предположений.
>
> *Джон Тьюки*

Данные наделяют нас знаниями. Независимо от того, проводите ли вы опрос для измерения удовлетворенности клиентов или запускаете Большой адронный коллайдер с целью развития физики элементарных частиц, вы хотите что-то получить из собранных вами данных.

Данные с легкостью могут быть неправильно поняты или неверно истолкованы. К счастью, **анализ данных** помогает нам генерировать из них надежные знания. У разных ученых разные подходы к анализу данных. Зачастую это зависит от характера их исследований и объема данных, к которым они имеют доступ. В этой главе предлагается рабочий процесс анализа данных для программистов, разделенный на четыре этапа, таких как: *сбор*, *обработка*, *исследование* и *тестирование*. Мы научимся:

собирать данные надежно и всесторонне;

обрабатывать их, очищая от мусора;

запускать исследования, **обобщая** значения;

более глубоко исследовать с помощью **визуализации данных**;

делать выводы, **проверяя** свои представления.

Даже лучшие из нас, имея дело с данными, часто упускают из виду важнейшие этапы этого непростого процесса. Мы недостаточно точно записываем измерения, игнорируем важную информацию или делаем неправильные выводы. Более того, анализ данных — это итеративный процесс: на любом этапе мы можем вдруг осознать, что более ранний шаг можно было улучшить. Все это может быстро превратиться в запутанный процесс.

Чтобы избежать хаоса, нужно придерживаться методики, хоть это и утомительно. Тщательный анализ данных помог Чарльзу Дарвину открыть происхождение видов, и он же может помочь SpaceX заселить другую планету. Используйте его в своих самых смелых проектах.

СБОР ДАННЫХ

Представьте, что у вас есть маленькая кофейня, переживающая непростые времена. Чтобы разработать прибыльную стратегию, нужно знать свой бизнес. Какие напитки предпочитают ваши клиенты? Сколько кофе продается в день? Сколько в среднем тратит клиент? В какое время клиенты чаще приходят или уходят? У кого из ваших поставщиков лучшие цены? Изменились ли эти показатели за последние несколько месяцев?

Ответы на такого рода вопросы трудно переоценить. Они позволяют успешно планировать и определять ключевые показатели эффективности для оценки и отслеживания прогресса. Чтобы получать подобные важные знания, организации с хорошим управлением анализируют свои данные и создают периодические отчеты. Затем работники и менеджеры могут легко проанализировать текущий прогресс и очертить области с потенциалом к улучшению.

Многие организации не просто выпускают отчеты о своих данных, но и отслеживают их в режиме реального времени. Таким образом, последствия их действий можно наблюдать на панели мониторинга. Подобные процессы поощряют принятие решений посредством обучения и распространения информации, формируя то, что люди называют **бизнес-аналитикой**.

Но бизнес-аналитика не ограничена лишь бизнесом. Рассмотрим отделение неотложной помощи государственной больницы, где можно использовать «бизнес-аналитику» для отслеживания потока пациентов (рис. 4.1). В таком случае, например, если наблюдается необычный всплеск времени ожидания, персонал больницы быстро об этом узнает и может принять меры до того, как ситуация обострится.

Рис. 4.1. Панель мониторинга отделения неотложной помощи больницы, отображающая в режиме реального времени количество пациентов в разных отделениях

В результате такого сбора данных больница оказывается лучше подготовленной к кризисам. Если в этом районе произойдет серьезная авария, больница сможет немедленно отметить сокращение количества доступных карет скорой помощи. Зная, какие ресурсы имеются под рукой, персонал может быстро решить, следует ли вызывать подкрепление еще до возвращения занятых машин.

Бизнес-аналитика и другие мероприятия из области анализа данных требуют от вас сбора, обобщения, визуализации и извлечения пользы из полученных данных. Все эти темы мы и рассмотрим. Если ваша организация не ориентирована на бизнес-аналитику и управление знаниями, подайте пример самостоятельно. Изучите соответствующие данные и поделитесь своими выводами с остальными. Трудно переоценить полезность такого подхода.

4.1. СБОР

Полезные знания возникают только из соответствующих данных. Столкнувшись с колоссальным объемом не относящейся к делу информации, бывает трудно вычленить данные, которые имеют значение. Уточните поиск заранее, определив собственные цели. Какая именно информация связана с этими целями?

Если ваша задача состоит в том, чтобы улучшить меню вашей кофейни, то значение имеют характеристики успешных и неудачных продуктов. Соберите мнения ваших бариста и клиентов. Не менее важны данные об ингредиентах, ценах и продажах. Что делать, если ваша цель — улучшение ухода за пациентами в больнице? Посмотрите, какие факторы учитываются при диагностике пациентов и отслеживании процесса их выздоровления. Соберите данные о пациентах, заболеваниях и методах лечения.

ВИДЫ ДАННЫХ

Подходя к исследованию, постарайтесь получить полную картину. Фиксируйте информацию со всех возможных ракурсов, чтобы у вас не осталось шансов упустить важные детали. Например, если вы измеряете температуру, может оказаться полезным знать время и место проведения измерений, а также было ли при этом солнечно, туманно или дождливо. Как программисты, мы предпочитаем использовать следующую классификацию, чтобы убедиться, что мы не пренебрегли релевантными данными.

ЧИСЛЕННЫЕ	Подсчет баллов, физические измерения...
КАТЕГОРИАЛЬНЫЕ	Олимпийский вид спорта, жанр кино, порода собак...
ВРЕМЕННЫЕ	Дата рождения, время UTC — 08:00...
ГЕОГРАФИЧЕСКИЕ	Место, домашний адрес, граница...
НЕОПРЕДЕЛЕННЫЕ	Аудиозапись, тело электронной почты, кадры с веб-камеры, рецепт приготовления...

Первые четыре типа данных считаются **структурированными**: они организованы определенным заранее образом. Например, электрическая мощность измеряется в ваттах, каждая медаль на Олимпийских играх 1896 года была присуждена за один из девяти возможных видов спорта, даты могут быть организованы в календаре, а международные границы определены по координатам.

Компьютеры неплохо справляются со структурированными данными. С другой стороны, **неструктурированные** данные им переварить уже труднее. Существуют специальные методы извлечения структурированных данных из неструктурированных. Например, софт для распознавания лиц может принимать неструктурированный видеопоток и выводить чью-либо личность в виде категориальных данных.

ПОЛУЧЕНИЕ ДАННЫХ

Многие наши повседневные дела облегчают компьютеры, поэтому наше поведение и действия часто оставляют цифровой след. Изучите компьютерные системы и отследите эти данные. Установите собственные датчики, чтобы собрать еще больше данных, или измените методы работы вашей компании. Кроме того, часто можно получить полезные данные от третьих лиц.

СУЩЕСТВУЮЩИЕ ДАННЫЕ Больничные компьютеры часто отслеживают жизненно важные показатели пациентов, такие как частота сердечных сокращений и температура тела. Компьютеры ресторана записывают, что и за каким столиком было заказано. Какими данными манипулируют ваши компьютеры? Прошерстите *все* системы в поисках копий полезных записей. Например, изучите компьютер, на котором размещен веб-сайт компании, и вы, скорее всего, обнаружите журнал всех его посетителей.

НОВЫЕ ДАННЫЕ Всегда можно найти интересующие нас данные, которые остаются незарегистрированными. Например, большинство ресторанов не записывают, как довольные клиенты чувствуют себя после еды. Чтобы исправить это, каждый счет можно сопроводить квитком, с помощью которого клиент оценит услугу по шкале от одного до десяти. До тех пор пока вы не нарушаете этические границы, всегда полезно искать способы сбора как можно большего количества данных. Если биллинговая система ресторана не записывает, как долго клиенты остаются на своих местах, измените ее.

Рис. 4.2. Будьте эмпатичны при сборе данных

ДАТЧИКИ Мы часто думаем о датчиках для изучения природных явлений. Например, климатологам нужны датчики температуры, давления воздуха, влажности и многого другого. Однако датчики могут пригодиться и в бизнесе. Рестораны могут использовать звуковые сенсоры для отслеживания окружающего шума, который слышат их клиенты, а торговые центры часто используют датчики присутствия для записи ежедневного количества посетителей. Кроме того, веб-приложения часто используют **виртуальные датчики** для отслеживания поведения пользователей, например времени, которое они проводят на страницах.

ВНЕШНИЕ ДАННЫЕ Можно использовать данные, собранные другими пользователями. Например, риелторы будут сверяться со сторонними данными для того, чтобы получить историю цен на недвижимость. Любители спорта могут найти данные о большинстве профессиональных матчей, а любители кино — данные о большинстве фильмов. Правительства обычно предоставляют данные переписи, содержащие важные национальные социально-экономические показатели. Используйте поиск по наборам данных Google[1] для запроса тысяч коллекций данных от компаний, университетов и правительственных учреждений. Вы можете найти и дополнительные данные, которые имеют отношение к вашим задачам.

СКРАПИНГ Часто соответствующие данные из интернета загрузить не получается — они доступны только на веб-страницах. Например, некоторые веб-сайты собирают обзоры баров и ресторанов. Если вам нужны эти данные, напишите скрипт, который заходит на эти страницы и копирует

[1] Поиск по набору данных Google: http://code.energy/google-data.

соответствующие фрагменты вам на компьютер. Это называется **веб-скрапингом**. Это довольно распространенный метод, для которого даже имеется бесплатный софт, автоматизирующий весь процесс.

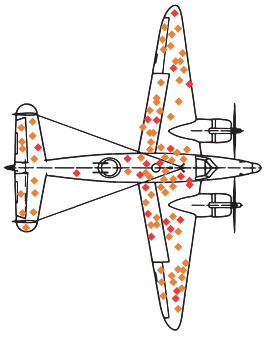
КОНФИДЕНЦИАЛЬНОСТЬ Будьте осторожны и не собирайте личную информацию о людях без их явного согласия. По меньшей мере это неэтично. Кроме того, *не* обманывайте людей, заставляя их принимать навязчивые политики конфиденциальности, как поступают некоторые компании с их бесконечными условиями и положениями, которые никто никогда не читает.

ОШИБКА ВЫБОРКИ

Представьте, что вы владелец ресторана и собираете отзывы об удовлетворенности клиентов. Вы получаете больше заявок, чем способны обработать, поэтому решаете опрашивать только 10 % клиентов. Если выбор осуществляет сам персонал, то он может предпочитать клиентов в хорошем настроении. В результате данные могут указывать на то, что клиенты удовлетворены гораздо больше, чем если бы опрошены были *все*.

Эта проблема называется **ошибкой выборки**. Такое, например, может произойти, если вы даете квитки только клиентам, которые сидят за определенными столиками. Они могут чувствовать себя по-другому, потому что вокруг шумно или им в лицо светит солнце. Все это может повлиять на ответы, и ваши данные все равно не будут отражать реальность. Давайте рассмотрим другой пример.

БРОНЯ ДЛЯ АСА ♠ Идет Вторая мировая война, вас зовут Вальд[1], и вы математик. Вражеская противовоздушная оборона наносит урон авиации, и у вас имеется схема всех попаданий, которые получили вернувшиеся самолеты. Инженеры сообщают, что нужно учесть ограничения по весу, поэтому они могут добавить броню только в одно из трех мест: крылья, фюзеляж или двигатели. Что вы выберете?



[1] Абрахам Вальд — венгерский математик и статистик. В сферу его научных интересов входили теория принятия решений, эконометрика, геометрия, математическая статистика и теория вероятностей. — *Примеч. ред.*

Поначалу кажется, что лучше всего укрепить фюзеляж или крылья, так как именно туда чаще всего попадают. Но эти данные чрезвычайно предвзяты: вы принимаете в расчет только схемы попаданий в *вернувшиеся* самолеты, но вы ничего не знаете о попаданиях в самолеты, которые погибли! Оказывается, многие крушения были вызваны повреждением двигателя. Лучший способ спасения пилотов — добавить броню именно там.

Предубеждения часто трудно обнаружить, потому что они неожиданным образом воздействуют на нашу интуицию, и предвзятость выбора не исключение. Всегда старайтесь убедиться, что ничто не влияет на то, *какие именно* данные записываются, выбирая их *случайным* образом. И имейте в виду, что, как и Вальд, вы иногда просто не в состоянии собрать объективные данные.

4.2. ОБРАБОТКА

Сбор большого количества данных — это как заполнять склад товарами. Может возникнуть соблазн свалить все как попало и сразу же отвлечься на что-то новое. Но чем лучше организован склад, тем легче будет работать. Организуйте свои данные, как только они поступают.

ПЕРВИЧНАЯ ОЧИСТКА ДАННЫХ

Процесс сбора данных обычно приводит к появлению множества файлов: журналов, дампов баз данных SQL, электронных таблиц и т. д. Они называются **необработанными или сырыми источниками**. Соответствующие фрагменты информации в этих файлах должны быть идентифицированы, извлечены и организованы так, чтобы к ним можно было легко получить доступ. Например, извлечение данных из журнала доступа к веб-серверу может выглядеть так, как на рис. 4.3.

Выбирайте только *релевантные* данные из сырых источников. Извлекайте их и разбивайте на новые файлы, используя формат, который компьютеры смогут легко понять. Мы называем такие новые файлы **набором данных**. Хорошо построенные наборы данных могут быть проанализированы непосредственно и без дополнительной организации. Процесс преобразования необработанных источников в набор данных называется **очисткой данных**. Рассмотрим соответствующие шаги.

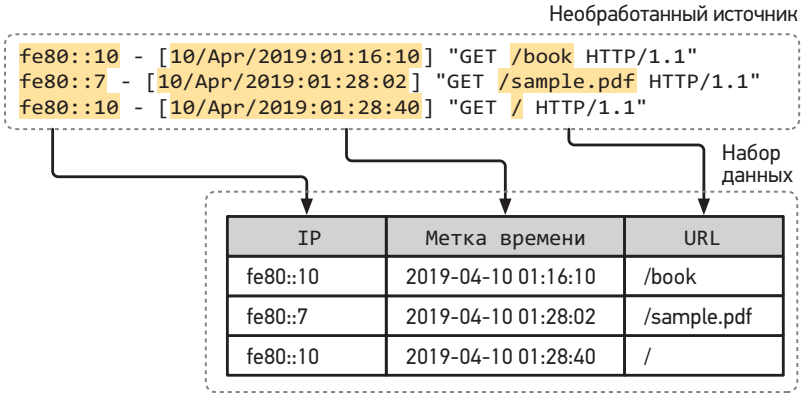


Рис. 4.3. Извлечение данных из журнала доступа к веб-серверу

СВЕДЕНИЕ В ТАБЛИЦЫ Таблицы служат наиболее распространенным способом структурирования данных. Они являются структурой по умолчанию для обработки данных: большинство алгоритмов лучше всего работают именно с табличными данными. В каждой таблице хранятся записи событий или объектов одного и того же типа. Первым шагом в очистке данных является разделение всех соответствующих данных на таблицы.

В таблицах каждая запись становится строкой. То, что мы знаем об этих записях, становятся столбцами. В контексте анализа данных столбцы также называются *переменными*. На рис. 5.3 показан исходный текст, собранный в таблицу с тремя переменными: IP, метка времени и URL.

Обычно табличные данные сохраняются в формате CSV[1]. CSV-файлы можно с легкостью импортировать практически в любую среду программирования. Если ваши табличные данные сохранены в реляционных базах данных[2], стоит потратить усилия на их преобразование в CSV-файлы. Как только появляется одна большая таблица на объект данных и исчезают связи SQL, становится гораздо проще анализировать данные, так как

[1] CSV (значения, разделенные запятыми, Comma-Separated Values) — самый простой формат для хранения табличных данных. Это обычный текст, где запятые и новые строки разделяют ячейки и строки.

[2] SQL — это распространенный формат для кодирования табличных данных, который препятствует появлению повторов. Соотнесение с базой данных устраняет повторяющиеся фрагменты информации.

пропадает необходимость разрешать зависимости между различными таблицами для интерпретации данных.

НОРМАЛИЗАЦИЯ Всегда следите, чтобы ячейки в одном столбце были однородными. Когда есть несколько способов выразить одно и то же, вы должны **нормализовать** данные. Например, если столбец содержит температуру, выберите градусы Фаренгейта или Цельсия[1] и преобразуйте остальные значения. Три примера нормализации проиллюстрированы на рис. 4.4–4.6.

Температура
98,6 °F
36,11 C
99,3778 f
едва теплый
30C

Нормализация →

Температура (°C)
37,0
36,1
37,4
NULL
30,0

Рис. 4.4. Нормализация температуры. Будьте последовательны в количестве десятичных знаков и использовании запятых и точек. Удалите лишние символы F, C или ° и отбросьте либо повторно выразите значения, которые не являются числами

Гражданство
Британское
GB
🇬🇧
Соединенное Королевство
UK
Great Britain

Нормализация →

Гражданство
GB
GB
GB
GB
GB
GB

Рис. 4.5. Нормализация записей о гражданстве. Будьте готовы к неожиданностям, особенно работая со старыми записями. Должны ли народы СССР считаться русскими? А как насчет югославов?

[1] Ученые обычно нормализуют данные, приводя их к метрической системе так, чтобы числами было легче манипулировать и сравнивать с данными со всего мира.

КОНВЕРТАЦИЯ ВАЛЮТЫ → УЧЕТ ИНФЛЯЦИИ

Год	Деньги
1970	$500
1977	£440 000
1988	¥65 000
1999	$500

Год	USD
1970	500,00
1977	502,42
1988	508,40
1999	500,00

Год	USD/2019
1970	6514,85
1977	2095,70
1988	1086,32
1999	758,63

Рис. 4.6. Нормализация показателей по различным валютам и годам. Из-за инфляции один доллар в 1970 году имел ту же покупательную способность, что и шесть долларов в 1988 году. Кроме того, некоторые валюты больше не существуют, например итальянская лира! Нам понадобятся исторические обменные курсы и темпы инфляции, чтобы нормализовать подобные показатели

Нормализация также может включать разделение столбца на два или более. Представьте, что в базе данных картин музея есть столбец «размеры», содержащий текст, описывающий размеры холста. Было бы разумно разделить эту информацию на столбцы по высоте и ширине, каждый из которых содержит хорошо отформатированные числа в единой системе измерения (рис. 4.7).

Название	Размеры
№ 5, 1948 г.	8# x 4#
Лас-Менинас	318 x 276 см
Крик	91 см, 73,5 см
Монализа	30¼ на 21⅞ дюйма

→

Название	В (см)	Ш (см)
№ 5, 1948 г.	244	122
Лас-Менинас	318	276
Крик	91	74
Монализа	77	53

Рис. 4.7. Нормализация размеров картин

УДАЛЕНИЕ МУСОРА Всегда проверяйте, соответствуют ли ваши значения ожидаемому типу и являются ли они разумными. Очищайте ячейки, содержащие абсурдные данные. Это называется **удалением мусора**.

Частота сердечных сокращений никогда не может быть отрицательной. «Гугл» сообщает нам, что самая высокая частота сердечных сокращений, когда-либо зафиксированная, составляет около 600 ударов в минуту. Записи за пределами диапазона (0, 660) можно без колебаний

отбрасывать[1]. Ячейка, содержащая 819, должна быть установлена в NULL или полностью удалена.

ПРОПУСК УДАРА Как менеджер систем бизнес-аналитики больницы, вы составляете отчет с измерениями частоты сердечных сокращений множества пациентов. Просматривая данные, вы сталкиваетесь с рядом удивительных значений, таких как 819 ударов в минуту. Что с ними делать?

Время	Пациент	Частота (уд./мин.)
07:00	472	61
07:01	677	78
07:03	780	819
07:04	472	180

Тщательно следите, чтобы нули *никогда* не использовались вместо NULL, если в ячейке отсутствуют данные. В примере с температурными записями (см. рис. 4.4) было бы ошибкой нормализовать «едва теплый» до `0,0` градусов по Цельсию! К сожалению, подобная небрежность встречается не так уж редко. Например, многие программы выводят координаты (0°, 0°), когда не могут получить местоположение[2]. Независимо от типа данных, которые вы собрали, проверьте свои нули — являются ли они точными числовыми записями, или на их месте должны быть NULL?

ДУБЛИКАТЫ Убедитесь, что в ваших таблицах нет дубликатов: каждая строка должна хранить нечто уникальное. Допустим, в таблице хранятся названия, годы выпуска и жанры различных фильмов. Такая таблица должна содержать только одну строку для каждого фильма. Если в двух строках записано `Метрополис, 1927, Научная фантастика`, удалите одну из них.

Дубликаты труднее обнаружить, когда данные еще не нормализованы. Если запись имеет жанр НФ, а ее дубликат — `Научная фантастика`, простой поиск не обнаружит проблемы. Дубликаты имен с орфографическими ошибками или альтернативным написанием часто проходят

[1] У большинства людей частота сердечных сокращений в состоянии покоя составляет 40–100 ударов в минуту. Тем не менее в зависимости от того, для чего мы анализируем данные, иногда полезно оставить место для невероятных, но достоверных данных. Здесь превышение на 10 % выше рекордного максимума было выбрано произвольно.

[2] Согласно многим неочищенным источникам данных, это пустое место посреди Атлантического океана является одним из самых оживленных мест на Земле. Айтишники даже дали ему особое имя: «Нулевой остров».

незамеченными. Всегда нормализуйте и очищайте свои данные, а также тщательно проверяйте похожие строки, чтобы исключить дублирование. Если вы используете систему управления базами данных, разберитесь, какие встроенные инструменты и функции способны вам в этом помочь.

АНОНИМИЗАЦИЯ ДАННЫХ

Проявляйте предельное уважение к данным, которые относятся к подробностям жизни людей. Финансовые отчеты, данные о медицинском обслуживании и личные сообщения — это лишь немногие примеры чрезвычайно конфиденциальной информации, которая никогда не должна становиться достоянием общественности. Во многих организациях каждый хранит копию конфиденциальных данных, с которыми работает. Катастрофическая утечка данных — это всего лишь вопрос единственного взлома! Обрабатывайте и храните личные данные с особой осторожностью.

Чтобы снизить риски, данные должны быть преобразованы так, чтобы уменьшить объем личной информации, позволяющей идентифицировать личность. Это называется **анонимизацией данных**. Во многих странах уже действуют законы, предписывающие анонимизацию данных в ряде случаев[1, 2].

ОТБРАСЫВАНИЕ Опускайте данные, которые служат лишь для идентификации людей. Например, имена и фамилии, идентификационные номера социального страхования и налоговых органов, номера телефонов и адреса электронной почты почти никогда не имеют статистического значения. Данные, не связанные с демографией, целями организации и поведенческими схемами, могут быть сразу отброшены.

РАЗМЫТИЕ Некоторые личные данные отбросить не получится. Например, возраст — это важная демографическая информация, которая часто объясняет поведение, так что сохраняйте ее. Однако *точные* даты рождения могут быть использованы для поиска людей и создания

[1] Например, см. Генеральный регламент ЕС о защите персональных данных (General Data Protection Regulation, GDPR).

[2] В Российской Федерации это регулирует Федеральный закон «О персональных данных» от 27 июля 2006 г. № 152-ФЗ. — *Примеч. ред.*

угрозы их анонимности. Храните только год рождения: это достаточное указание на возраст и по нему нельзя идентифицировать личность. Процесс снижения точности в пользу анонимности называется **размытием данных**.

Подумайте о самых простых способах снизить точность конфиденциальных данных. Предположим, вы работаете в компании, которая отслеживает расходы клиентов. Поможет ли ведение записей вплоть до копейки понять их поведение? Отбросьте копейки, и вы сделаете записи более анонимными. Это также относится и к домашнему адресу: почтового индекса, скорее всего, будет достаточно, чтобы дать информацию о целевой аудитории в определенном районе. Размывайте ненужную точность.

ПОВТОРНАЯ ИДЕНТИФИКАЦИЯ Даже тщательно анонимизированные данные часто могут быть «деанонимизированы» через процесс, называемый **повторной идентификацией данных**. Если вы достаточно знаете о ком-то, вы можете отфильтровать анонимизированные записи и определить, какая из них принадлежит этому человеку.

Рассмотрим анонимизированный набор данных из больницы. Если вы знаете чей-то возраст, пол, продолжительность пребывания и тип болезни, то, вероятно, можете отыскать его или ее точную запись в анонимизированном наборе данных. Анонимизация данных является лишь сдерживающим фактором, а не решением. Предоставьте анонимизированным данным тот же уровень защиты и секретности, что и необработанным источникам.

ВОСПРОИЗВОДИМОСТЬ

Может возникнуть соблазн завершить этап обработки данных как можно скорее, без написания сценариев и документирования каждого шага. Говорят, что такие быстрые, грязные и плохо документированные работы проделаны **без подготовки (ad hoc)**. В целом, обработка данных в стиле ad hoc — это очень плохая практика.

Часто появляются новые версии необработанных данных. Иногда мы просто получаем свежий срез данных с дополнительными записями. Если преобразования были сделаны без подготовки, повторить их на новых данных будет сложно, и это займет много времени. Кроме того, трудно

проверить, не возникло ли ошибок во время таких преобразований: на любом из этапов мы не в силах сравнить состояние набора данных с предыдущими этапами. Кроме того, выявленную ошибку может быть очень трудно исправить.

Чтобы избежать таких проблем, опытные специалисты по обработке данных стараются убедиться, что каждое преобразование может быть легко повторено. Это называется **воспроизводимостью**. Вы можете достичь базового уровня воспроизводимости, документируя все специальные преобразования, которые вы выполняете, но все действия все равно придется выполнять вручную.

Лучше всего написать программу, которая выполняет все этапы преобразования данных в режиме конвейера. И при изменении источников данных можно воспроизвести преобразования, просто запустив программу на новых сырых источниках. При работе в команде каждый участник может проверить эту программу, что облегчает поиск ошибок. Затем, если обнаружена ошибка в программе, ее можно исправить, и новый исправленный набор данных может быть сгенерирован путем повторного запуска обновленной программы. Самое главное, никогда не изменяйте свои необработанные источники напрямую: новые очищенные и анонимизированные данные должны записываться в новые файлы. Ваши исходные источники по-прежнему будут содержать истинные данные, если вы когда-нибудь заподозрите ошибку и нужно будет откатить операции. Если исходные источники содержат конфиденциальную информацию, не забудьте убедиться, что доступ к ним ограничен.

Теперь, когда наш набор данных абсолютно чист, приступаем к следующему этапу: *исследованию*. Вот тут-то и начинается настоящее веселье. В следующих разделах вы познакомитесь с основами электронного анализа (**E**xploratory **D**ata **A**nalysis, **EDA**). Вы научитесь *обобщать* и *визуализировать* свои данные. По мере того как вы начнете *видеть* свои данные, будет развиваться интуиция. Интуиция порождает вопросы о глубинных явлениях, сформировавших данные. Эти вопросы, в свою очередь, помогут еще глубже погрузиться в анализ: ведь легче найти что-то, если знать, что искать.

4.3. ОБОБЩЕНИЕ

Можно обобщить многие важные характеристики набора данных в нескольких ключевых числах. Это даст краткое представление о том, что говорят данные, без необходимости проверять отдельные записи. Рассмотрим некоторые из наиболее полезных обобщающих чисел.

КОЛИЧЕСТВО

Его называют n, и это количество отдельных записей, которые у вас есть. Например, если таблица здравоохранения описывает госпитализацию двух десятков пациентов, то $n = 24$.

СРЕДНИЕ ЗНАЧЕНИЯ

Среднее значение — это «срединное», или «типичное», значение группы. Существует несколько типов средних значений. Наиболее распространенным является **среднее арифметическое**: просуммируйте все значения и разделите на количество. Среднее арифметическое — это хорошее, но далеко не идеальное среднее значение. Почему? Давайте разбираться.

БИТКОИН-ПУЗЫРЬ Пятеро друзей утверждают, что они эксперты по блокчейну, и убеждают вас купить 100 долларов в криптовалюте. Каждый сообщает, насколько, по его мнению, изменится стоимость вашего кошелька в течение недели в зависимости от монеты, которую вы купите. Речь идет о биткоине (BTC), эфире (ETH) и доджкоине (Dogecoin). Какие же из них вам купить?

Эксперт	BTC	ETH	DOGE
Адам	8	NULL	7
Гэвин	6	2	7
Роджер	NULL	4	7
Сергей	NULL	3	–1
Виталик	7	21	–2

Что лучше, биткоин (7, 8, 6) или эфир (2, 4, 3, 21)? Трудно сравнивать группы чисел. Чтобы упростить задачу, можно обобщить каждую группу в одном числе. Давайте попробуем среднее арифметическое.

Прогноз прибыли ($)						
Монета	**Адам**	**Кевин**	**Роджер**	**Сергей**	**Виталик**	**Среднее**
BTC	8	6	–	–	7	7,0
ETH	–	2	4	3	21	7,5
DOGE	7	7	7	–1	–2	3,6

Более высокое среднее значение монеты эфира намекает на то, что у нее наибольший потенциал для прибыли. Однако обратите внимание, что среднее арифметическое эфира сильно зависит от одного чрезвычайно высокого прогноза. Возможно, биткоин — лучший выбор, несмотря на его более низкое среднее арифметическое.

С другой стороны, **медиана** — это среднее значение, на которое не влияют экстремумы группы. Чтобы получить медиану, отсортируйте значения в числовом порядке и выберите значение из середины. Для доджкоина: (–2, –1, 7, 7, 7). Если число элементов четное, то срединного значения не существует, как это видно на примере эфира: (2, 3, 4, 21). Медиана в таком случае равна среднему арифметическому двух срединных элементов. В обоих случаях медиана всегда будет указывать на середину: она больше одной половины значений и меньше другой.

Прогноз прибыли ($)						
Монета	**Адам**	**Кевин**	**Роджер**	**Сергей**	**Виталик**	**Среднее**
BTC	8	6	–	–	7	7,0
ETH	–	2	4	3	21	3,5
DOGE	7	7	7	–1	–2	7,0

Медиана тоже неидеальна. Обратите внимание, как биткоин и доджкоин демонстрируют *одинаковую* медиану семь долларов, хотя эксперты дают очень разные прогнозы: 100 % согласны с тем, что биткоин заработает около семи долларов, в то время как 40 % говорят, что доджкоин зарегистрирует убытки!

В то время как среднее арифметическое учитывает все числа группы, оно очень чувствительно к экстремальным значениям. На медиану не влияют экстремумы, но она игнорирует все значения, кроме центрального. Среднее арифметическое и медиана — это взаимодополняющие способы усреднения чисел[1].

ИЗМЕНЧИВОСТЬ

Независимо от того, используем ли мы среднее арифметическое или медиану, мы не способны узнать, насколько точно они представляют свою группу чисел. Например, (9, 10, 11) и (0, 10, 20) в среднем равны 10, хотя значения первых гораздо ближе к 10.

Стандартное отклонение группы значений показывает, насколько они отстоят от среднего значения. Чем больше стандартное отклонение, тем больше разброс чисел. Большинство языков программирования имеют встроенные функции, которые вычисляют стандартное отклонение[2]. Подумаем, как его можно использовать.

ИНВЕСТИЦИИ В ДАННЫЕ Ваша бабушка захотела инвестировать в технологии! Поскольку она планирует оплачивать членские взносы своего бридж-клуба с помощью прибыли, ей нужно, чтобы вы определили, какой из трех пакетов акций может обеспечить наиболее стабильный годовой доход. Как вы можете сравнить волатильность этих трех видов акций, основываясь на их пятилетней истории прибыльности (в %)?

Год	AAPL	GOOG	MSFT
2014	40,0	-2,4	27,2
2015	-2,8	46,6	22,2
2016	12,2	1,7	14,7
2017	48,2	32,9	40,2
2018	-5,1	-0,8	20,2

Для каждой акции вы можете рассчитать стандартное отклонение ее годовой доходности, и это хороший показатель волатильности.

---

[1] Есть и другие типы средних значений, такие как среднее гармоническое, которые будут рассматриваться в следующей главе. Некоторые средние значения идеально подходят для темпов роста, другие — для скорости перемещения. Еще больше информации см. здесь: http://code.energy/average.

[2] Дисперсия — еще одна распространенная мера изменчивости. Дисперсия — это квадрат стандартного отклонения.

Общая доходность (%)						
Акция	**2014**	**2015**	**2016**	**2017**	**2018**	**Стандартное отклонение**
AAPL	40,0	–2,8	12,2	48,2	–5,1	21,8
GOOG	–2,4	46,6	1,7	32,9	–0,8	20,2
MSFT	27,2	22,2	14,7	40,2	20,2	8,6

Из данных акций Apple и Google показали похожую волатильность, в то время как Microsoft приносила более стабильную прибыль из года в год.

Можно использовать стандартное отклонение каждый раз, когда вычисляете среднее значение: это могут быть мера меткости лучника, колебания температуры в городе или неравенства доходов населения. Не забудьте использовать его для описания того, насколько хорошо среднее суммирует группу значений.

СВОДКА ПЯТИ ЧИСЕЛ

Предположим, вы рассматриваете несколько городов, в которых можно провести лето, и для каждого города нашли базу данных с пиковыми показателями температуры для каждого дня 2018 года. Как оценить количество дней с приятной погодой в каждом городе? Прежде всего следует отметить, что медиана суточных пиковых температур вполне информативна: она делит дни таким образом, что половина из них холоднее медианы, а половина — теплее. Мы можем повторить этот процесс и разделить половинки на четверти (рис. 4.8).

После того как медиана делит упорядоченные единицы данных на половины, **верхний квартиль** делит на четверти *верхнюю* половину, а **нижний квартиль** — *нижнюю*. Верхний и нижний квартили — вместе со средним, минимальным и максимальным значениями — составляют **сводку пяти чисел**. Многие современные языки программирования имеют библиотеки для простого вывода этих чисел. Вот сводные данные по суточным пиковым температурам нескольких городов за весь 2018 год (n = 365).

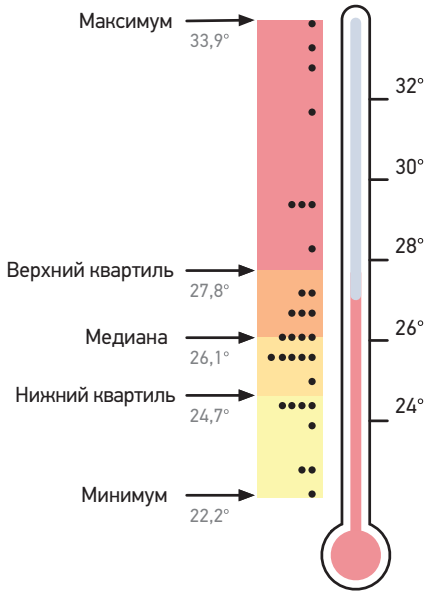


Рис. 4.8. Суточные пиковые температуры (°C) в Лос-Анджелесе (июль 2018 года). Каждая точка представляет один день. Поскольку $n = 31$, каждая четверть данных состоит из восьми пунктов

Пятизначные сводки пиковых температур					
Город	**Min**	**Нижний квартиль**	**Медиана**	**Верхний квартиль**	**Max**
Рио-де-Жанейро	20,1	26,8	29,9	32,8	39,1
Лос-Анджелес	13,9	19,4	22,2	25,0	34,4
Гонолулу	23,9	27,8	29,4	30,6	33,3
Миннеаполис	–17,1	0,9	10,6	26,7	37,8

Эти обобщения иллюстрируют интересные факторы, которые следует учитывать при планировании отпуска: даже если в Миннеаполисе четверть года стоит мороз, каждый из его 91 наиболее теплого дня теплее, чем в Лос-Анджелесе!

ВЫБРОСЫ Иногда мы натыкаемся на аномально высокие или низкие значения, которые не выглядят адекватными рядом с другими показа-

телями в столбце. Эти значения обычно указывают на экстремальные события, которые заметно отличаются по своей природе. Стоит выделить эти значения и тщательно изучить их. Чтобы систематически идентифицировать подобные значения, ученые определяют границы нормальности. Значения за пределами границ, если таковые имеются, помечаются как **выбросы**.

Существует множество методов определения границы нормальности. Например, мы часто начинаем с вычитания нижнего квартиля из верхнего квартиля. Эта разница называется **межквартильным диапазоном** или **IQR**, и значения, превышающие 1,5 × IQR от его ближайшего квартиля, считаются выбросами.

В измерениях температуры июля 2018 года в Лос-Анджелесе нет более низких выбросов: все значения находятся выше нижней границы (рис. 4.9). С другой стороны, существует группа из четырех теплых дней, три из которых являются выбросами с максимумами 32,4, 33,3 и 33,9°.

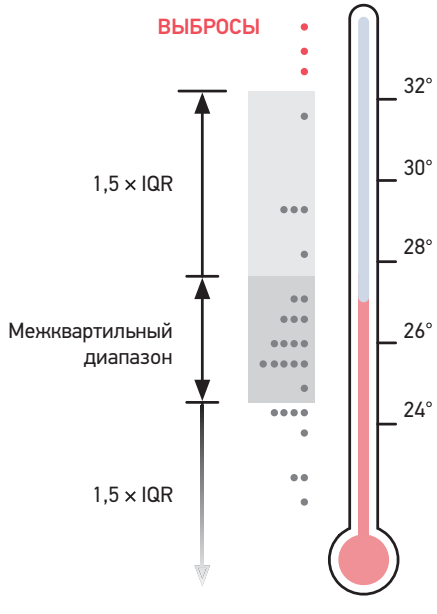


Рис. 4.9. Определение выбросов на рис. 4.8. Межквартильный диапазон равен 3,1°, поэтому 1,5 × IQR ≈ 4,5°. Таким образом, верхняя граница составляет 32,3°, а нижняя — 20,2°

Они действительно отличаются по своей природе: с 6 по 9 июля 2018 года на Южную Калифорнию обрушилась рекордная жара. Седьмого числа высокие температуры привели к тому, что огромный лесной пожар поглотил многие улицы Санта-Барбары. Более двух тысяч человек пришлось эвакуировать из этого района, а термометры в кампусе Калифорнийского университета в Лос-Анджелесе зафиксировали самые высокие температуры за всю историю.

КАТЕГОРИАЛЬНОЕ ОБОБЩЕНИЕ

Записи категориальных данных содержат метки, а не числа, поэтому напрямую суммировать их с помощью среднего значения, стандартного отклонения или сводки пяти чисел нельзя. Но можно их пересчитать.

Обычно мы суммируем категориальные данные, подсчитывая, сколько раз встречается каждая категория. Когда количество категорий выражается в процентах от общего количества n, мы получаем частоту, с которой была записана каждая категория. Например, таблицу с каждой олимпийской медалью и ее страной-обладательницей на первых летних играх 1896 года можно обобщить, подсчитав наиболее часто встречающиеся страны.

Страна	Количество	Частота
Греция	46	38 %
США	20	16 %
Германия	13	11 %
Франция	11	9 %
Великобритания	7	6 %
Другие	25	20 %

КОРРЕЛЯЦИОННАЯ МАТРИЦА

Когда идет дождь, дороги становятся более скользкими и видимость уменьшается, поэтому страховые компании ожидают большего количества происшествий на дорогах. Представьте, что вы такая страховая

компания и вы собрали набор данных из трех столбцов таким образом, что каждая строка содержит дату, количество осадков на эту дату и соответствующее количество дорожно-транспортных происшествий. Вы ожидаете, что значения в двух последних столбцах увеличиваются и уменьшаются одновременно: чем больше дождей, тем больше несчастных случаев происходит.

Когда два таких столбца изменяются совместно, мы говорим, что они коррелируют. Можно выразить силу этого явления с помощью числа, называемого коэффициентом **корреляции** или сокращенно корреляцией[1]. Коэффициент корреляции, равный 0, означает, что значения *вообще* не зависят друг от друга. Коэффициент корреляции, равный 1, указывает на то, что эти два столбца идеально сочетаются друг с другом.

На практике коэффициент корреляции редко равен ровно 0 или ровно 1. Чтобы быстро найти связи между столбцами наших таблиц, мы вычисляем значения корреляции для каждой пары столбцов в таблице. Результаты могут быть собраны в **корреляционную матрицу**, которая обобщает то, насколько сильно столбцы меняются вместе друг с другом.

Корреляции в ежедневных происшествиях (Лос-Анджелес, 2018, *n* = 365)

Ветер		1,0	0,21	–0,18	0,05	–0,03
Ливень		0,21	1,0	–0,19	0,19	–0,11
Температура		–0,18	–0,19	1,0	0,18	0,28
Происшествия		0,05	0,19	0,18	1,0	0,07
Нападения		–0,03	–0,11	0,28	0,07	1,0

[1] Есть несколько методов измерения корреляции. Каждый из них дает немного другой коэффициент. Мы используем коэффициент корреляции Пирсона, он же *r* Пирсона.

Предположим, что вы работаете в Лос-Анджелесе и построили таблицу, в которой каждая строка — это день, а столбцы представляют дневную пиковую температуру, среднюю скорость ветра за день, объем дождя, количество ДТП и количество нападений, зарегистрированных полицейским управлением. Построение корреляционной матрицы позволяет выявить взаимосвязи между этими переменными.

От левого верхнего угла до нижнего правого идет диагональ единиц: столбцы прекрасно коррелируют сами с собой. Кроме того, верхняя правая половина матрицы симметрична нижней левой половине: корреляция между 🍃💨 и 💥 равна корреляции между 💥 и 🍃💨. Все значения, выделенные серым цветом, не содержат дополнительной информации и, как правило, удаляются из корреляционной матрицы для ясности и краткости.

Обратите внимание, что некоторые столбцы имеют отрицательные коэффициенты корреляции, такие как 🌡️ и 🌧️. Отрицательные значения корреляции указывают на то, что столбцы изменяются совместно в *противоположных* направлениях: по мере увеличения количества осадков регистрируются более низкие пиковые температуры.

Когда значения корреляции близки к нулю, мы говорим, что они практически не коррелируют. Например, на несчастные случаи, похоже, не влияет скорость ветра. И наоборот, самая сильная корреляция наблюдается между 🔪 и 🌡️. Данные, по-видимому, свидетельствуют о том, что преступники, как правило, более агрессивны, когда погода теплее.

Такая корреляционная матрица помогает нам исследовать то, как столбцы связаны друг с другом. Когда две переменные коррелируют, исследуйте причины в реальном мире — и вас ждут интересные открытия.

ПРИЧИННОСТЬ Остерегайтесь ловушки. Когда мы находим корреляцию между двумя явлениями, то часто поспешно предполагаем, что одно явление вызвало другое. Это далеко не так. Например, торговые центры обычно регистрируют высокую корреляцию между продажами солнцезащитных очков и мороженого. Означает ли это, что ношение солнцезащитных очков заставляет вас хотеть мороженого? Или что поедание мороженого делает ваши глаза более чувствительными? Конечно же,

Рис. 4.10. «Корреляция». Любезно предоставлено http://xkcd.com

нет. Всегда имейте в виду, что *корреляция не подразумевает причинно-следственной связи*.

ПРЕОБРАЗОВАНИЯ Столбцы часто приходится преобразовывать, чтобы отыскать корреляции. Например, несчастные случаи из-за заносов в вашем городе могут плохо коррелировать с осадками или скоростью движения транспортных средств, но гораздо сильнее — с количеством осадков, *умноженным* на ограничение скорости *в квадрате*. Может оказаться сложной задачей обнаружить преобразования, которые выявляют такие корреляции. Если коэффициент корреляции двух столбцов или их преобразований близок к нулю, это не обязательно означает, что эти две переменные не связаны друг с другом.

Обобщение чисел упрощает и сокращает объем информации, которую мы должны учесть. Однако большая часть нюансов остается вне нашей досягаемости, и может быть трудным понять, каким должен быть следующий шаг. Захват большего количества данных с помощью форм и цветов — это следующий шаг в нашем исследовании. Картинка стоит тысячи чисел!

4.4. ВИЗУАЛИЗАЦИЯ

Среднее значение, стандартное отклонение и сводка пяти чисел дают полезный, но упрощенный обзор истории набора данных. Чтобы найти

еще больше подсказок, придется прибегнуть к нашим инстинктам визуального распознавания образов. Построение графиков и рисунков позволяет нам видеть данные, исследовать их закономерности и нюансы, а также выявлять аномалии, вызванные либо экстраординарными событиями, либо — что происходит чаще — простыми ошибками обработки.

ЯЩИК С УСАМИ

Есть графическое представление сводки пяти чисел, и оно называется **диаграммой «ящик с усами»**. Верхний и нижний квартили рисуются как верхняя и нижняя стороны ящика, а горизонтальная линия разделяет его по медиане. Усы выступают над и под ящиком, достигая максимальных и минимальных значений, *не являющихся* выбросами, а выбросы часто добавляются в виде отдельных точек. Построим график температур июля 2018 года в Лос-Анджелесе (рис. 4.11).

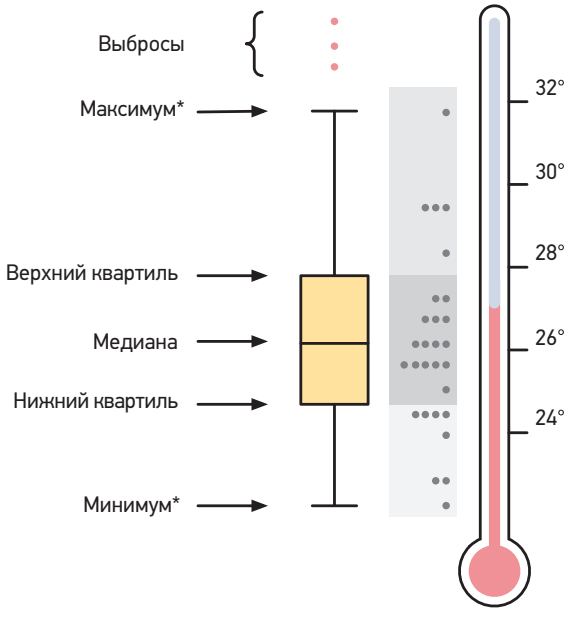


Рис. 4.11. Построение по рис. 4.8 диаграммы «ящик с усами» суточных пиковых температур в Лос-Анджелесе в июле 2018 года (n = 31 день) (выбросы игнорируем)

Обратите внимание, что увеличение объема данных за весь год влияет на диаграмму (рис. 4.12). Теперь посмотрим, как можно использовать такие диаграммы для сравнения разных городов на одном графике (рис. 4.13).

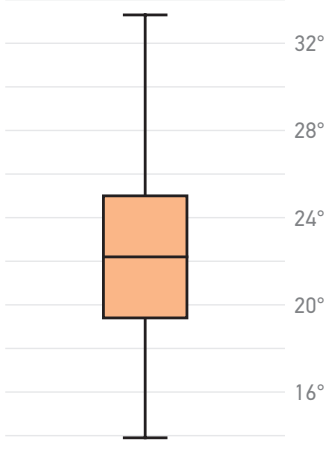


Рис. 4.12. Диаграмма «ящик с усами» суточных пиковых температур в Лос-Анджелесе за весь 2018 год (n = 365). Остаются только два выброса, поскольку весь набор из 365 единиц данных имеет другой межквартильный диапазон и границы нормальности, чем набор только для июля

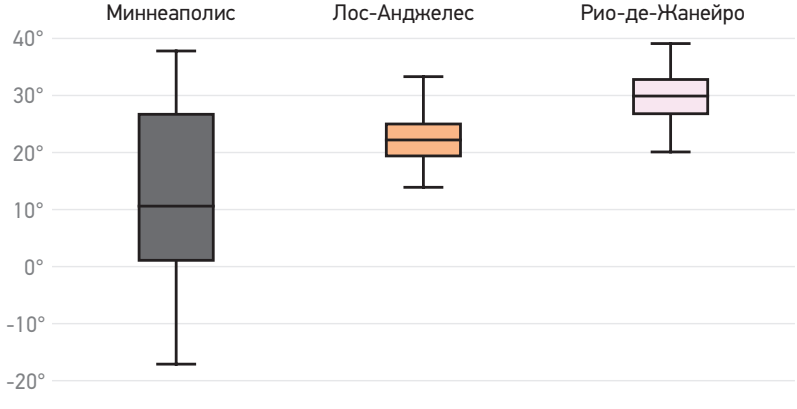


Рис. 4.13. Диаграммы суточных пиковых температур в 2018 году. Миннеаполис и Рио-де-Жанейро не показывают выбросов

В то время как таблица облегчает чтение значений сводки пяти чисел, диаграмма «ящик с усами» значительно облегчает сравнение распределений, описываемых различными сводками из пяти чисел.

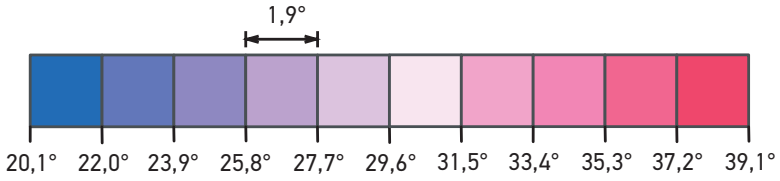
ГИСТОГРАММЫ

Сводка пяти чисел описывает поддиапазоны, охватываемые группами чисел, и вы научились получать ее в два шага: а) поделить единицы данных на группы одинакового размера и б) пронаблюдать, какие диапазоны они охватывают. С другой стороны, **гистограмма** точно показывает, *где в пределах диапазона* сосредоточены единицы данных, и создается она с помощью обратного процесса:

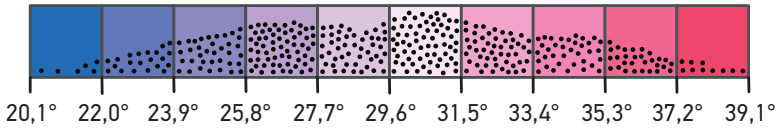
1) разделите диапазон на интервалы одинакового размера, называемые бинами (столбиками);
2) обратите внимание, сколько единиц данных содержит каждый из них;
3) наконец, мы можем построить каждый бин в виде столбика, высота которого представляет количество единиц данных, которые он содержит. Далее показано построение гистограммы с десятью бинами, но мы могли бы использовать любое число. На рис. 4.14–4.16 мы видим, как изменение количества бинов изменяет данный график и как гистограммы могут помочь нам сравнить температуру в разных городах.

КУМУЛЯТИВНЫЕ ГИСТОГРАММЫ Есть и другой способ построения гистограмм. Вместо того чтобы выводить высоту каждого столбика только из количества единиц данных, которые он содержит, мы также можем добавить число из предыдущих бинов. Таким образом, бины будут становиться выше и выше по мере движения вправо. Вот как можно получить такую **кумулятивную гистограмму** из обычной (рис. 4.17).

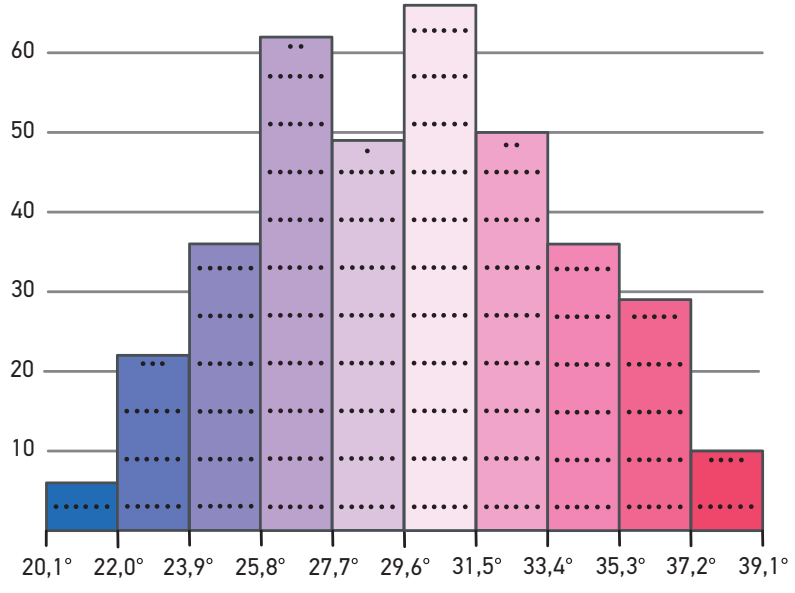
Обратите внимание, что бин, заканчивающийся на температуре 33,4 °C, имеет высоту 292. Это говорит нам о том, что 292 дня, или 80 % года, были холоднее 33,4 °C. Теперь сравним наши четыре города с десятибинными кумулятивными гистограммами (рис. 4.18).



А. Разделите диапазон температур на 10 бинов



Б. Поместите 365 ежедневных пиковых температур в соответствующие бины



В. Растяните каждый бин так, чтобы его высота соответствовала количеству точек, которые он содержит

Рис. 4.14. Создание гистограммы суточных пиковых температур в Рио-де-Жанейро, которые варьировались от 21,1 до 39,1 °C в 2018 году. Обратите внимание, что на типичных гистограммах все бины имеют один и тот же цвет

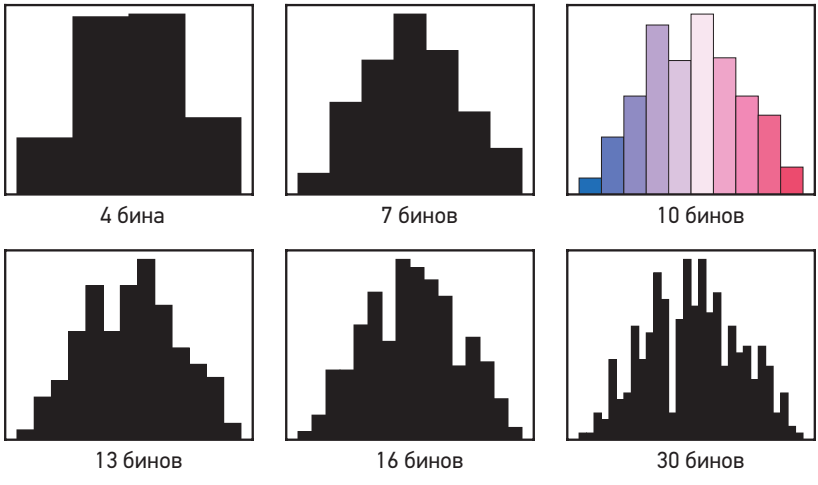


Рис. 4.15. Шесть гистограмм из одного и того же набора данных. Большее количество бинов дает более точное описание распределения. Но слишком много бинов сделают вашу гистограмму неровной и неопределенной

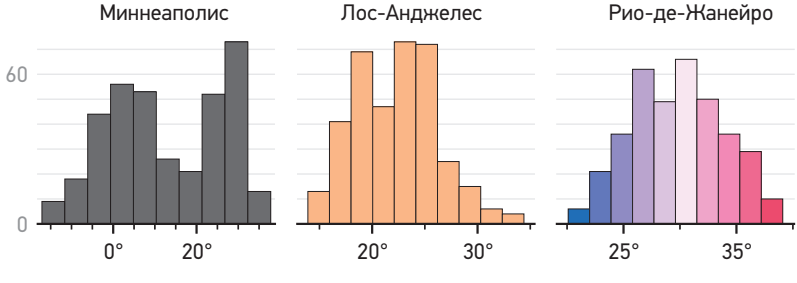
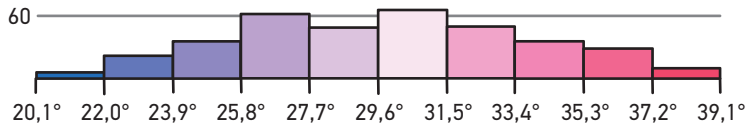
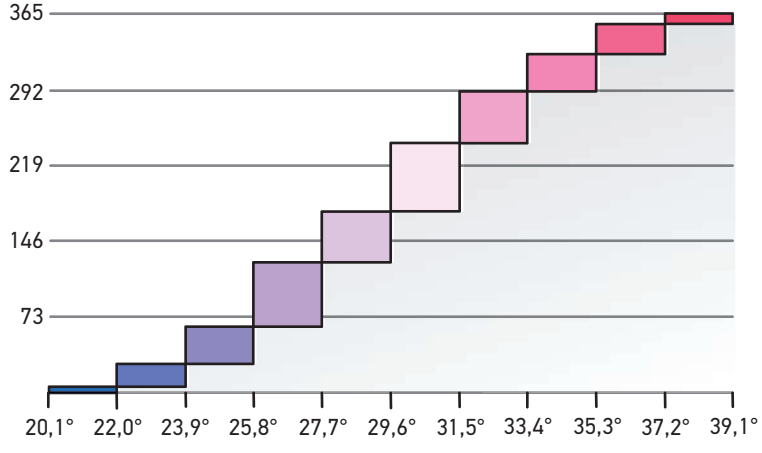


Рис. 4.16. Гистограммы пиковых температур для разных городов (2018 год). Сразу видно, что Миннеаполис — идеальный город, если вы не любите умеренные температуры и обожаете экстремальные — будь то жара или холод!

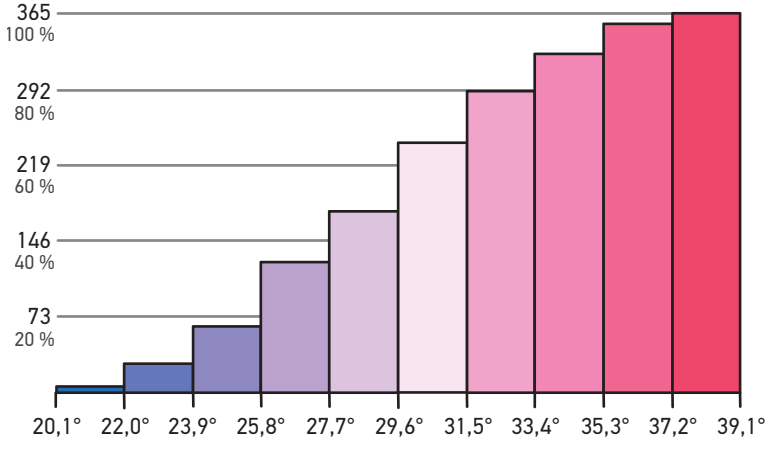
При наблюдении за любым графиком всегда проверяйте его масштаб. Температуры в Миннеаполисе и Рио-де-Жанейро на рис. 4.18 могут показаться довольно схожими, но эти графики имеют очень разные масштабы по горизонтальной оси. Если построить одну гистограмму поверх другой, различия между городами станут очевидными (рис. 4.19).



А. Уменьшите масштаб рис. 4.14 с



Б. Сложите каждый бин с предыдущим



В. Цвет в бинах кумулятивной гистограммы

Рис. 4.17. Создание кумулятивной гистограммы

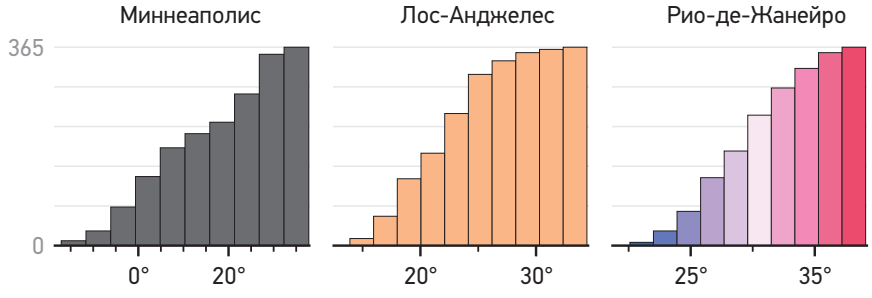


Рис. 4.18. Кумулятивные гистограммы пиковых температур (2018 год)

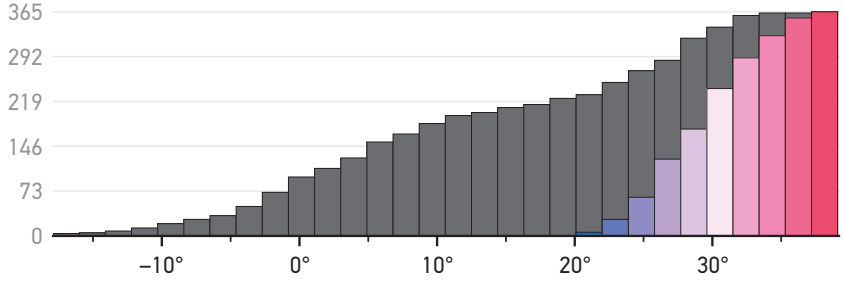


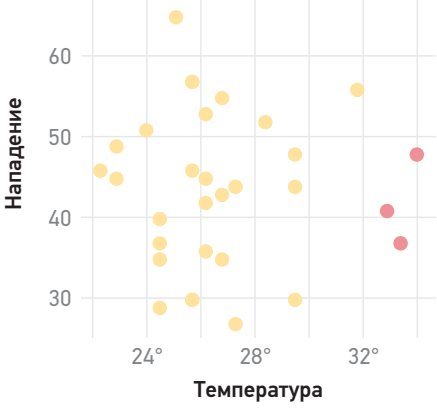
Рис. 4.19. Сравнение кумулятивных гистограмм пиковой температуры в Миннеаполисе (серый) и Рио-де-Жанейро (цветной). Бины были тщательно установлены на одних и тех же температурных интервалах для обоих городов. Поскольку диапазоны не совпадают, Рио-де-Жанейро получает несколько пустых бинов!

ТОЧЕЧНЫЕ ДИАГРАММЫ

До сих пор мы встречали графики, которые исследуют *одну* переменную или столбец данных. **Точечная диаграмма** исследует то, как *две* переменные связаны друг с другом. На этом графике каждая запись данных представляет собой одну точку. Положение точки на вертикальной оси представляет значение записи в одном столбце. Положение точки на горизонтальной оси представляет значение записи в другом столбце. На рис. 4.20 показано, как построить точечную диаграмму из единиц данных рис. 4.11.

Из таблицы корреляционной матрицы на с. 177 мы знаем, что нападения и ветер не коррелируют, но есть корреляция 0,28 между нападениями

и температурой. Можете ли вы увидеть на данном графике, что в теплые дни, *как правило*, случается больше нападений?



Дата	Температура	Нападения
5 июля	26,7°	55
6 июля	33,3°	37
7 июля	33,9°	48
8 июля	31,7°	56
9 июля	32,8°	41
⋮		

Рис. 4.20. Число зарегистрированных нападений в Лос-Анджелесе в сравнении с дневными пиковыми температурами в июле 2018 года. Выбросы температуры — красные

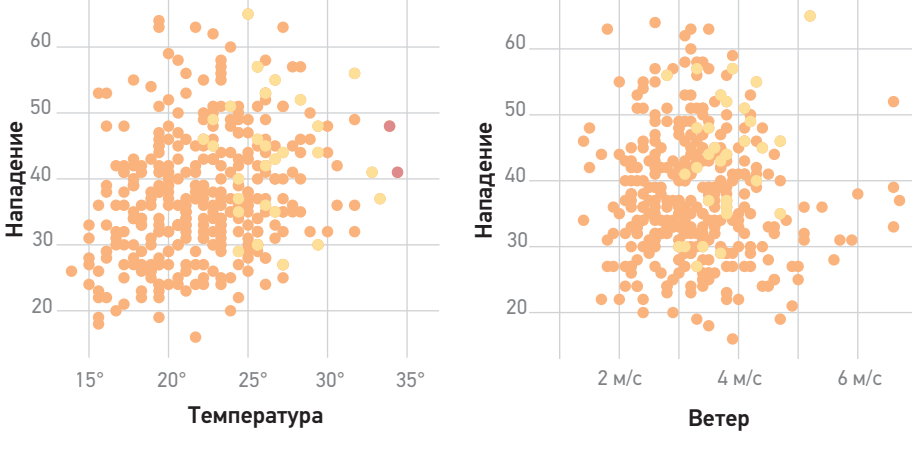


Рис. 4.21. Количество нападений в зависимости от суточных пиковых температур, а затем средняя скорость ветра за оставшуюся часть 2018 года

Будет легче обнаружить корреляцию с температурой, если данные охватывают больший диапазон температур. Зарегистрируем ежедневное количество уголовных преступлений в Миннеаполисе (рис. 4.22–4.25).

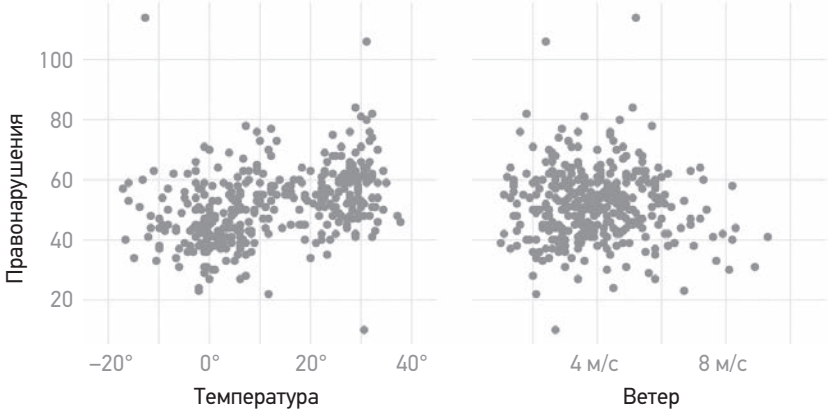


Рис. 4.22. Миннеаполис в зависимости от пиковой температуры и средней скорости ветра в 2018 году

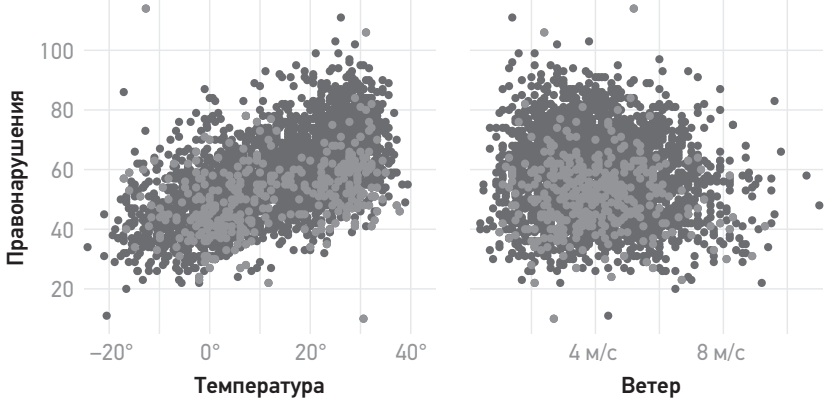


Рис. 4.23. Число зарегистрированных правонарушений в Миннеаполисе с 2010 по 2018 год. Каждый участок имеет более 3000 точек, поэтому многие перекрываются и визуализацию трудно интерпретировать

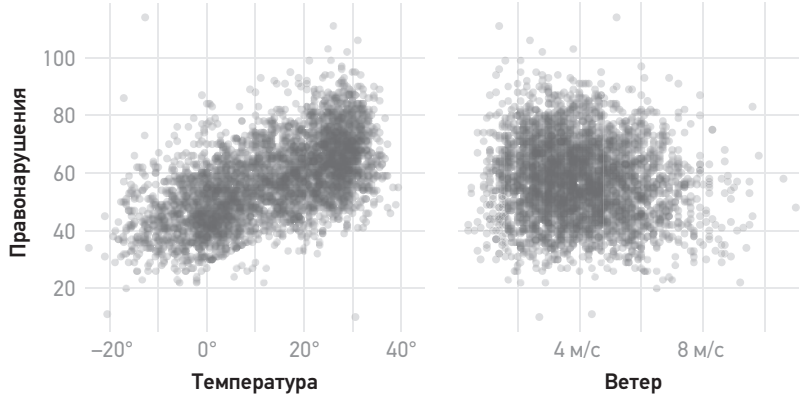
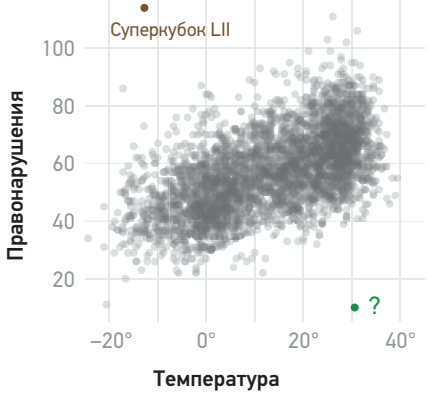


Рис. 4.24. Число зарегистрированных правонарушений в Миннеаполисе с 2010 по 2018 год с непрозрачностью 20 %. Более темные оттенки серого образуются в областях с более высокой концентрацией точек. Левый участок имеет коэффициент корреляции 0,58: более теплые дни, как правило, приносят большее количество преступлений. Нет никакой видимой закономерности между ветром и преступностью, и единицы данных, показанные справа, дают корреляцию –0,11



Дата	Температура	Правонарушения
4 февр. 2018 г.	–12,7°	114
⋮		
4 июня 2018 г.	30,6°	10
⋮		

Рис. 4.25. Обратите пристальное внимание на выбросы: одинокие точки вдали от остальных. Выясните, каким строкам соответствуют эти точки. Холодный день со множеством правонарушений — 4 февраля 2018 года, когда в Миннеаполисе состоялся Суперкубок! Теплый день с небольшим количеством правонарушений не имеет очевидного объяснения, поэтому вполне может быть связан с ошибкой системы сбора данных департаментом полиции

ВРЕМЕННЫЕ РЯДЫ

Если все ваши единицы данных имеют соответствующую метку времени, вы можете выстроить их в хронологическом порядке и провести линию, соединяющую последовательные точки. Такой тип графика называется **временным рядом**. Он может выявить интересные взаимосвязи и закономерности между вашими данными и временем. На рис. 4.26 показан временной ряд суточных пиковых температур в Лос-Анджелесе в июле 2018 года, использующий те же данные, что и на рис. 4.11.

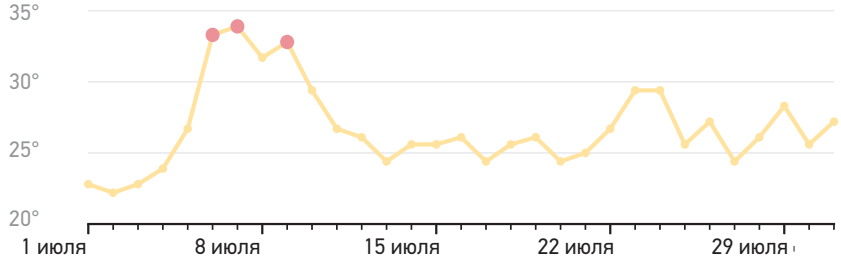


Рис. 4.26. Пиковые температуры в Лос-Анджелесе в июле 2018 года

Когда временной ряд имеет много точек, расположенных слишком близко друг к другу, вариации между последовательными точками могут добавить графику заметную флуктуацию. Посмотрите, что произойдет, если один и тот же график будет составлен для каждого дня года с использованием тех же данных, что и на рис. 4.12 (рис. 4.27).

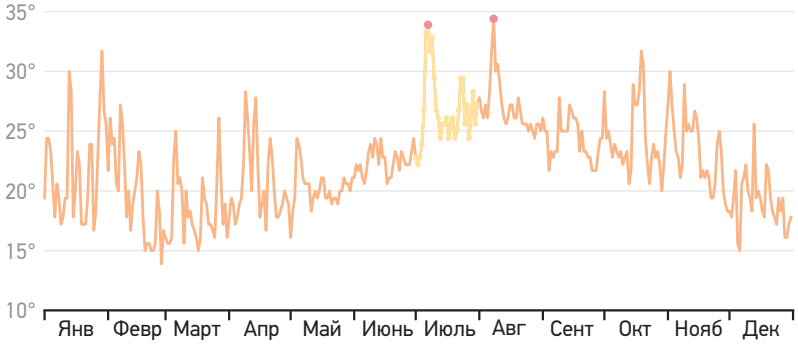


Рис. 4.27. Пиковые температуры в Лос-Анджелесе за все месяцы 2018 года

В таких случаях можно сгладить график, чтобы его было легче читать. Один из распространенных методов называется **скользящей средней**: каждая точка данных заменяется средним арифметическим *между ней самой и несколькими предыдущими*. Количество использованных предыдущих точек называется **размером окна**. Посмотрим, как это выглядит (рис. 4.28, 4.29).

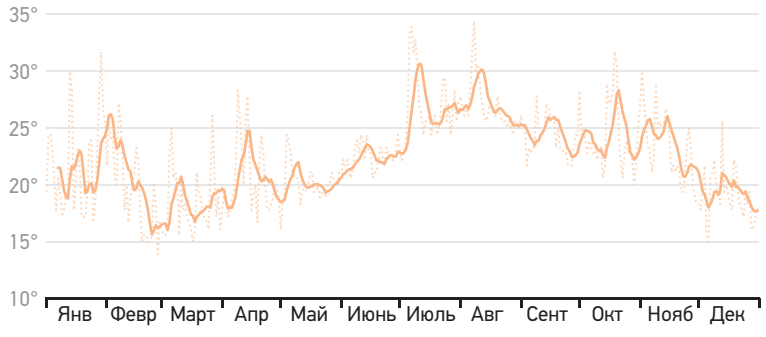
Мы уже знали из графиков (см. рис. 4.13) и гистограмм (см. рис. 4.19), что пиковые температуры в Рио-де-Жанейро и Миннеаполисе имеют очень разные распределения. Теперь же мы ясно видим, что они почти одинаковы с июня по август, когда в Миннеаполисе лето, а в Рио зима!

На многие другие события также влияет время года. К примеру, продажи мороженого выше летом, а случаи респираторных заболеваний — зимой. Как только вы выстроите свои данные в виде временного ряда, сезонные закономерности станет очень легко обнаружить.

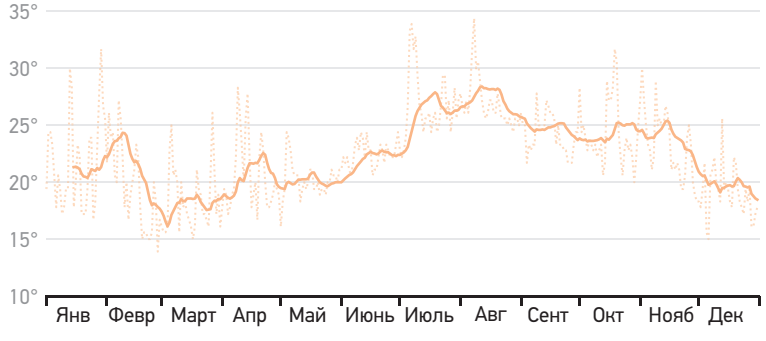
Временные графики полезны, когда нужно понять, как переменная растет или уменьшается с течением времени. Например, цена хранения цифровых данных снизилась, что подпитывает революцию в науке о данных. Чтобы полностью осознать, насколько кардинальными были изменения, построим график стоимости хранения данных с течением времени (рис. 4.30).

В 1985 году хранение одного гигабайта данных стоило более 100 000 долларов (в долларах 2019 года). В 2018 году это стоило менее трех центов. Поскольку масштабы этих цен разнятся на несколько порядков, линия на графике выравнивается после 1995 года и уже трудно понять, что происходит с этого момента.

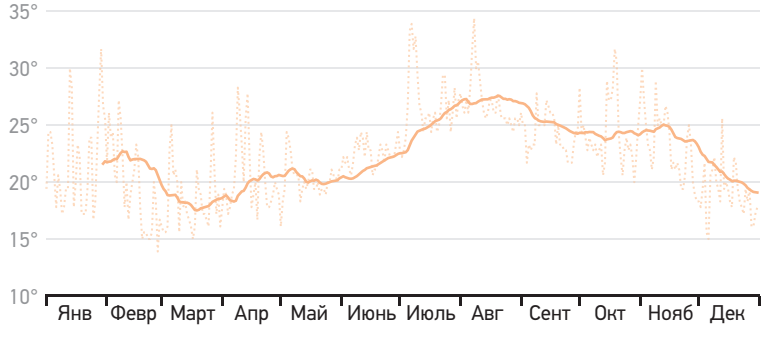
Есть полезный трюк для просмотра данных при таких порядках величин: замените обычную **линейную шкалу** на **логарифмическую**. Линейная шкала задается так, что каждая отсечка равна предыдущей *плюс* некоторое число. Например, каждая отсечка на вертикальной оси рис. 4.30 равна предыдущей *плюс* 20 000 долларов. С другой стороны, логарифмическая шкала определяется таким образом, что каждая отсечка равна предыдущей, но *умноженной* на некоторое число. Число 10 выбирается наиболее часто, так как оно дает отсечки в виде красивых круглых чисел (рис. 4.31).



А. Размер окна — семь дней



Б. Размер окна — 15 дней



В. Размер окна — 30 дней

Рис. 4.28. Скользящие средние суточных пиковых температур за 2018 год в Лос-Анджелесе при различных размерах окна. Увеличение размера окна делает скользящую среднюю нашего временного ряда с каждым разом все более гладкой

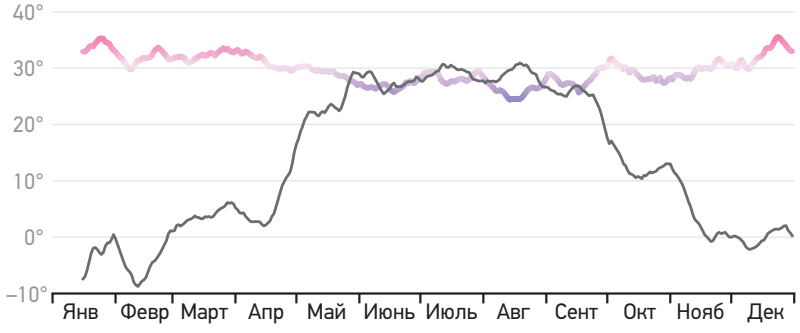


Рис. 4.29. Сравнение 15-дневных скользящих средних временных рядов пиковой температуры в Миннеаполисе (серый) и Рио-де-Жанейро (цветной)

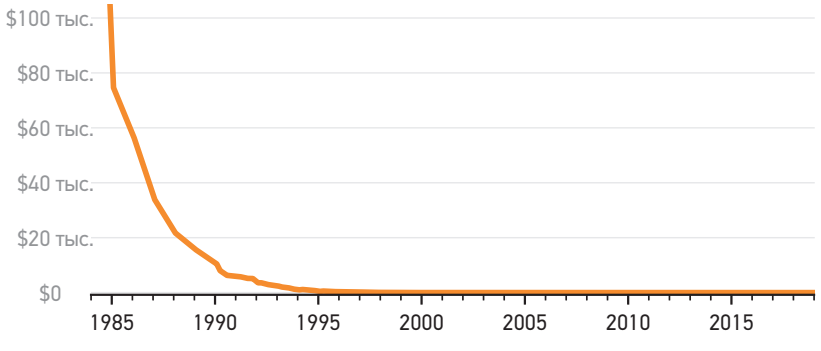


Рис. 4.30. Скорректированная на инфляцию стоимость хранения 1 Гбайт

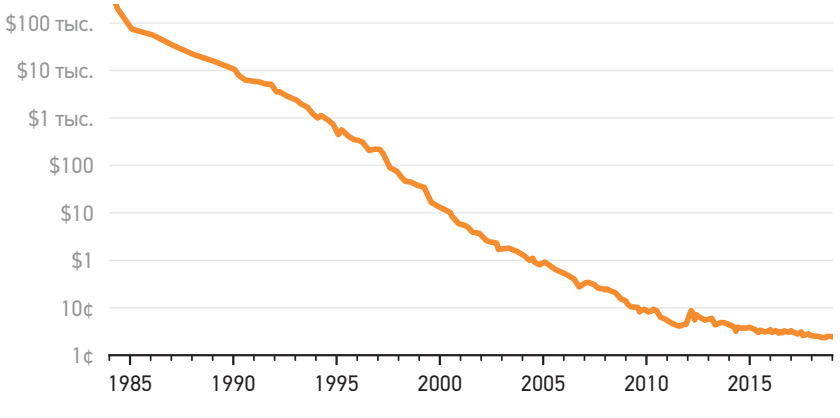


Рис. 4.31. Скорректированная на инфляцию стоимость хранения 1 Гбайт (логарифмическая шкала)

Обратите внимание, что цены с 1995 по 2018 год больше не напоминают линию. Теперь можно точно просчитать цены на хранение для каждого года. С 1985 по 2010 год цена за гигабайт становилась примерно в десять раз дешевле каждые пять лет, за исключением периода с 1995 по 2000 год, когда цены снизились почти в 100 раз!

В 2011 году в Таиланде произошли ужасные наводнения, унесшие жизни 815 человек и повлиявшие на миллионы людей. Многие заводы были сильно повреждены, из-за чего в течение 2012 года нарушились глобальные цепочки поставок жестких дисков. Это трагическое событие объясняет очень заметный скачок цен на рис. 4.31.

КАРТЫ

Когда имеются географические данные, отразите их на карте. Если, например, вы знаете дневные пиковые температуры для каждого округа в США, то можете построить тепловую карту (рис. 4.32).

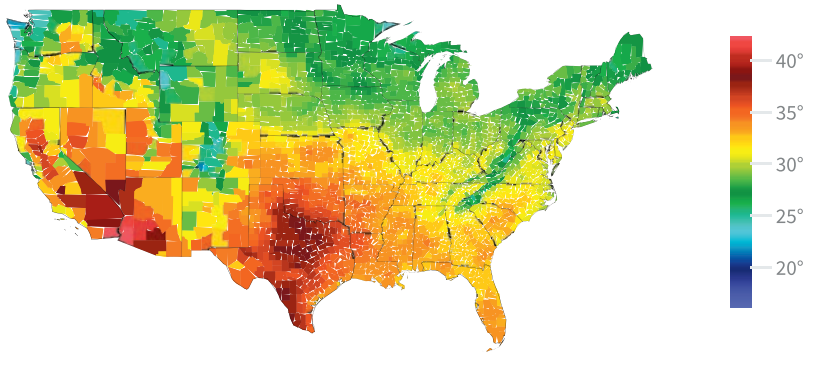


Рис. 4.32. Средняя максимальная дневная температура (июль 2018 года), показывающая прохладные районы вокруг Скалистых гор и Аппалачей

Картографирование не ограничивается географическими данными. Например, когда специалисты по аэродинамике получают значения температуры для каждой точки пространства воздушного потока, они отображают его в трех измерениях (рис. 4.33).

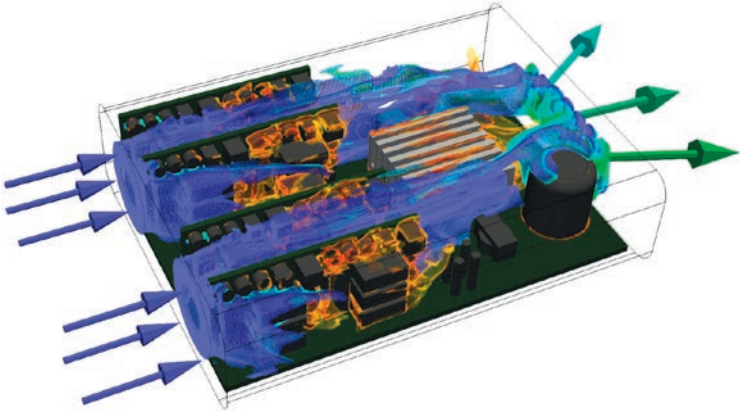


Рис. 4.33. Ветер, генерируемый вентилятором, охлаждает электронную печатную плату. Воздух в вентиляционном отверстии *(справа)* немного теплее, чем у вентиляторов *(слева)*

4.5. ТЕСТИРОВАНИЕ

Исследуя данные и выявляя закономерности, мы часто разрабатываем теории, объясняющие, что именно наблюдаем, и многие из них оказываются ошибочными или уводящими в сторону. Как сказал космолог Карл Саган в своем последнем интервью, *«наука — это нечто большее, чем совокупность знаний. Это способ мышления, способ скептически исследовать вселенную с ясным пониманием того, что человек склонен ошибаться»*.

Может показаться неестественным подвергать сомнению собственные предположения и, как следствие, трудно убедительно объяснять закономерности. К счастью, ученые ценят скептицизм и разработали ряд полезных инструментов.

ГИПОТЕЗЫ

Чтобы ответить на вопрос или оценить интуитивное знание, не делая при этом неточных выводов, начните с правильной формулировки. Выразите это как **гипотезу**: утверждение, которое может быть поддержано или опровергнуто данными, которые вы можете собрать. Придется проверять гипотезу, поэтому пока *не* предполагайте, что она верна или ложна. Сделайте свою гипотезу простой и объективной, чтобы ее было легче проверить.

ДЕЛО ВКУСА Ваша кофейня в Миннеаполисе теряет своих клиентов-веганов. Десять дней назад вы решили обновить меню и предложили зеленый чай и мороженое с кокосовым молоком. Вы сделали это без каких-либо исследований, основываясь на следующих интуитивных предпосылках.

Клиенты-веганы жаждали зеленого чая в дни занятия йогой.

Клиентам не нравился старый выбор мороженого.

Теперь, когда новое меню готово, вы хотите провести исследование, чтобы определить, было ли такое изменение хорошим решением. Можете ли вы выразить каждое из своих интуитивных представлений в виде гипотезы? Попробуйте сформулировать такую гипотезу, которая относится к вашим целям и которую вы можете легко проверить.

Если говорить об интуиции , то трудно найти объективную меру для понятия «жаждать». Однако одной из целей вашего бизнеса должна быть продажа вашей продукции, поэтому вы можете выразить гипотезу следующим образом: *клиенты-веганы* **покупают** *зеленый чай в дни йоги*. Проверка этой гипотезы может помочь вам понять, сможет ли ваше решение вернуть лояльность клиентов-веганов. Однако это потребует определенного труда: вы должны провести опрос, чтобы определить, которые из ваших клиентов являются йогами-веганами.

«Интуицию » сложно проверять, так как опрос давних клиентов может привести к ошибке выборки: вы можете опросить только тех, кто вернулся и кого, скорее всего, вполне устраивало старое меню. Но здесь имеется тесная связь с вашей бизнес-целью в виде продажи продуктов, поэтому гипотеза может быть такой.

Продажи мороженого с новым меню стали выше.

Эта гипотеза не дает особого представления о том, *почему* вы достигаете или не достигаете своих целей в вопросе мороженого, но, по крайней мере, ее легко проверить: ваша система учета, вероятно, собирала данные о продажах как до, так и после изменения меню. У вас уже есть все необходимые данные! Используя среднее значение, подведем итоги и сравним последние 100 дней старого меню с первыми десятью днями нового (рис. 4.34).

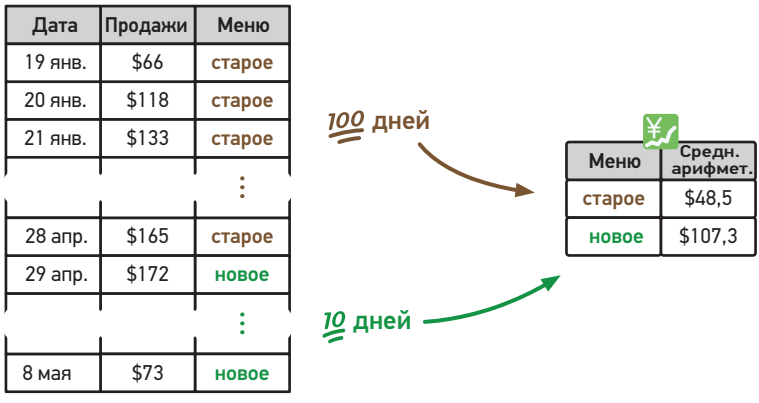


Рис. 4.34. Обобщение ежедневных продаж для проверки гипотезы

Эти данные согласуются с гипотезой: продажи стали выше при использовании нового меню. Достаточно ли этих доказательств, чтобы подтвердить гипотезу о том, что новое меню увеличит продажи мороженого?

Конечно же нет. *Невозможно* прийти к заключению, сравнивая средние значения по этому небольшому набору данных. На данные могли повлиять многие другие факторы. Изменения в поведении клиентов могут быть вызваны другими изменениями, привнесенными в вашу кофейню, такими как, например, расположение столов. Они могут быть вызваны факторами, находящимися вне вашего контроля, такими как погода (рис. 4.35).

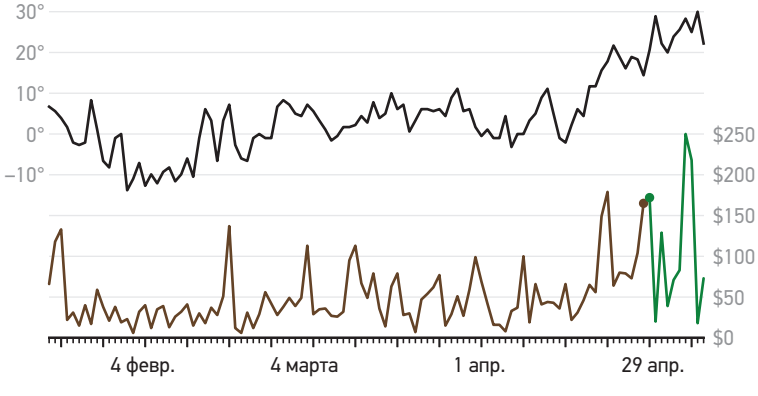


Рис. 4.35. Ежедневные пиковые температуры и продажи мороженого в вашей кофейне в Миннеаполисе. Температура показана сверху черным цветом (левая шкала), продажи мороженого показаны снизу цветом (правая шкала)

Независимо от вашего меню, клиенты наверняка съедят больше мороженого в жаркий день, чем во время снежного бурана. Возможно, именно более высокие температуры вызвали рост продаж, а вовсе не новое меню. Может быть, вам просто повезло. И если вы не можете быть уверены в том, что именно вызвало изменение продаж, то как можете узнать, отражает ли результат проверки гипотезы истину? Далее подробно рассмотрим методы, разработанные учеными для тщательного решения этой проблемы.

ЭКСПЕРИМЕНТЫ

Хорошие исследователи всегда разрабатывают процедуры, позволяющие уменьшить неопределенность вокруг их результатов. Научная процедура проверки гипотезы называется **экспериментом**. Гипотезы могут быть должным образом проверены только с помощью хорошо продуманных экспериментов.

Первым шагом в разработке эксперимента является определение переменных, которые необходимо измерить, а также данных, которые требуется собрать. Для гипотезы  этими переменными являются *продажи* и *меню*. Они уже представлены в виде столбцов на рис. 4.34.

Всегда есть и другие переменные, которые влияют на тех, кого вы хотите изучить. Мы называем их **внешними переменными**, и будьте внимательны: они вполне могут привести к неправильным выводам. Попробуйте собрать данные касательно внешних переменных, если они еще не включены в ваши записи. Если вы внимательно изучите рис. 4.35, то заметите, что пики продаж кажутся выше, когда погода теплее. Продажи довольно нерегулярны, но мы наблюдаем пик продаж каждые семь дней. Может быть, люди едят больше мороженого по выходным. Мы уже определили две внешние переменные: *температуру* и *день недели*.

Когда вы изучаете данные, продолжайте искать подсказки, чтобы идентифицировать различные внешние переменные. Когда вы их найдете, убедитесь, что записи соответствующим образом обновлены (рис. 4.36).

В хорошо продуманном эксперименте гипотеза проверяется с использованием записей, в которых дисперсия внешних переменных минимизируется. Но это может серьезно ограничить количество используемых

записей. Например, если мы рассмотрим только один день недели с аналогичной пиковой температурой, останутся только две записи данных (рис. 4.37).

Дата	Продажи	Меню	День	Темп.
19 янв.	$66	старое	Пт	6,7°
20 янв.	$118	старое	Сб	5,6°
21 янв.	$133	старое	Вс	3,9°
⋮				
28 апр.	$165	старое	Сб	14,4°
29 апр.	$172	новое	Вс	20,6°
⋮				
8 мая	$73	новое	Вт	22,2°

Рис. 4.36. Сбор данных о факторах, которые могут повлиять на продажи

Дата	Продажи	Меню	День	Темп.
22 апр.	$179	старое	Вс	17,8°
29 апр.	$172	новое	Вс	20,6°

Рис. 4.37. Сравнение продаж в аналогичных условиях

В этом эксперименте продажи были ниже с новым меню. Данные не подтверждают гипотезу  , поэтому мы на шаг приблизились к ее отклонению. Но достаточно ли этого, чтобы ее отвергнуть?

Конечно же нет. Даже при разработке эксперимента в контролируемой среде мы никогда не в силах контролировать *всё*. Всегда есть факторы, влияющие на данные, некоторые из которых трудно идентифицировать. Могли ли вы знать, что ваши продажи увеличила группа евших мороженое на спор, а также непредвиденное число поломок морозильников? И наоборот, могли ли вы знать, что неожиданный всплеск аллергии на пыльцу удерживал людей дома, снизив потенциальные продажи? Мир

слишком сложен, чтобы даже при всем старании можно было вычленить *все* внешние переменные.

ШУМ В некотором масштабе сложность нашей Вселенной всегда будет давать случайные флуктуации, независимые от любых внешних переменных, которые мы способны обнаружить. Такие случайные необъяснимые колебания в просторечии называются **шумом**. Когда доступно слишком мало точек данных, трудно понять, какие колебания являются шумом, а какие — нет, и истина становится размытой. В конечном счете чем больше у нас данных, подтверждающих гипотезу, тем ближе мы подходим к ее подтверждению. И наоборот, чем больше данных, не согласующихся с гипотезой, тем ближе мы подходим к ее отклонению. К сожалению, на рис. 4.37 есть только две точки данных, и их недостаточно, чтобы подтвердить или отвергнуть гипотезу. Нужно больше данных.

Поскольку это уменьшает неопределенность, может возникнуть соблазн продолжать бесконечный сбор данных. Давайте попробуем определить, *сколько* данных будет достаточно, чтобы удостовериться в том, что результат проверки гипотезы отражает истину.

P-ЗНАЧЕНИЯ

Поскольку всегда имеется риск невезения и получения данных, подтверждающих ложную гипотезу, специалисты по статистике разработали методы оценки того, *насколько маловероятно*, что это произошло. Такой метод называется **статистической проверкой гипотезы** (рис. 4.38).

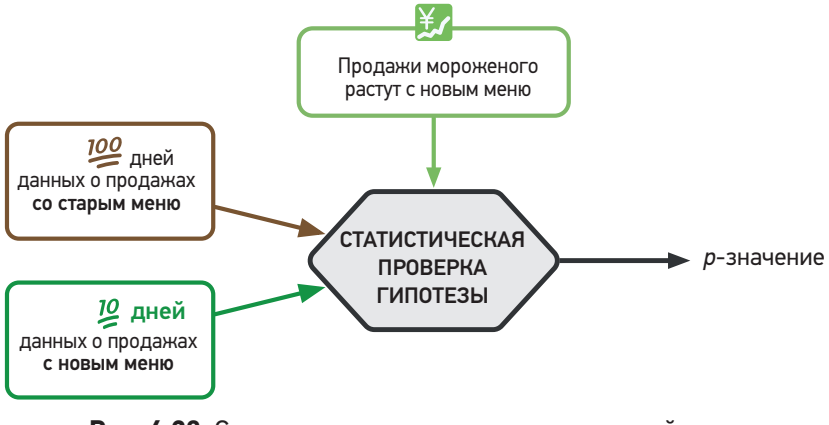


Рис. 4.38. Статистическая проверка гипотезы в действии

Вероятность того, что нам не повезло, называется ***p*-значением**. Чем меньше *p*-значение, тем ближе мы к подтверждению гипотезы. Если гипотеза верна, то сбор большего количества данных будет иметь тенденцию к снижению *p*-значения.

Существует множество различных типов проверки гипотез, каждый из которых лучше всего подходит для конкретного сценария. Чтобы *p*-значение было значимым, проведите исследование и убедитесь, что вы выбрали именно тот тип, который соответствует вашим данным и гипотезе[1]. Все они работают одинаково: вы вводите экспериментальные данные, а они выводят *p*-значение для проверяемой гипотезы.

Вероятность — это всегда число от 0 до 1, и *p*-значение не является исключением. Получение $p = 0{,}2$ означает, что вероятность того, что ваши данные подтверждают неверную гипотезу из-за вашего невезения, составляет 20 %. Если вас не устраивает такой уровень риска, соберите больше данных. Запустите проверку еще раз, и если ваша гипотеза верна, *p*-значение должно уменьшиться. Будет ли этого наконец достаточно, чтобы подтвердить гипотезу с абсолютной уверенностью?

Неа. Значение *p* никогда не бывает равным 0 или 1. Другими словами, вы никогда не получите 0 или 100 % вероятности того, что гипотеза ложная. А еще будьте очень внимательны при интерпретации этих вероятностей, потому что результат проверки гипотезы по надежности не может превышать сами данные. Если данные страдают от ошибки выборки, то ваше *p*-значение не имеет смысла. Проверка гипотезы не заменяет контролируемые эксперименты, а лишь дополняет их.

СТАТИСТИЧЕСКАЯ ЗНАЧИМОСТЬ Даже притом, что *p*-значение никогда не может быть равным нулю, наступит момент, когда мы будем уже в состоянии сделать выводы. Обычно мы определяем максимально допустимое *p*-значение, которое должна дать наша проверка гипотезы, чтобы ее можно было бы считать *скорее всего* верной. Этот порог называется **уровнем значимости**. Когда *p*-значение ниже этого порога, можно утверждать, что имеются **статистически значимые** доказательства, подтверждающие нашу гипотезу. Ученые обычно работают с уровнями значимости от 0,05 до 5 %.

---

[1] Обзор распространенных проверок гипотез и способов использования каждой проверки см. в разделе http://code.energy/hypothesis-test.

Рис. 4.39. «Бойфренд», любезно предоставлен http://xkcd.com

Делать выводы из данных без анализа статистической значимости — распространенная ошибка. Во многих случаях нам не хватает данных, чтобы получить статистически значимые доказательства, подтверждающие наши гипотезы, и поэтому приходится основывать наши решения на определенных предположениях. Важно признавать ограниченность знаний и быть готовыми к тому, что предположения окажутся ошибочными.

ДОВЕРИТЕЛЬНЫЕ ИНТЕРВАЛЫ

Если бы мы могли опробовать новое меню в течение неограниченного количества дней, средние ежедневные продажи представляли бы его истинные показатели. Можем ли мы быть уверены, что средние ежедневные продажи в размере 107,3 доллара, которые мы получили в ходе нашего эксперимента, эквивалентны этой истине? Другими словами, какое *p*-значение мы получим, проверив такую гипотезу?

NEW *Средний показатель ежедневных продаж мороженого составляет 107,3 доллара.*

Почти невозможно получить *ровно* 107,3 доллара из среднего арифметического продаж. Поскольку мы продолжаем собирать данные, вероятность того, что мы *не* получим 107,3 доллара, составляет почти 100 %. Другими словами, *p*-значение будет равно почти 1, что далеко от почти нулевых уровней значимости, которые мы хотим увидеть. Чтобы получить более низкое *p*-значение, нужно допускать значения, близкие к $107,3. Например, можно проверить, находится ли среднее

арифметическое между $A = 97{,}3$ и $B = 117{,}3$. Тогда гипотеза становится такой.

NEW *Средний показатель ежедневных продаж мороженого составляет от 97,3 до 117,3 доллара.*

По мере того как мы увеличиваем разрыв между A и B, p-значение становится все меньше. Когда разрыв задан таким образом, что p-значение равно выбранному нами уровню значимости, мы называем (A, B) **интервалом доверия**. Как правило, мы вводим наши данные и ориентировочный уровень значимости в функцию, называемую калькулятором доверительного интервала, которая затем выводит (A, B). Есть большой выбор калькуляторов доверительного интервала в зависимости от характера ваших данных и гипотезы. Но все они работают одинаково (рис. 4.40).

Рис. 4.40. Нахождение доверительного интервала

Калькуляторы доверительного интервала часто запрашивают **степень достоверности** вместо уровня значимости. Он равен единице *минус* уровень значимости. Другими словами, степень достоверности 95 % совпадает с уровнем значимости 5 %, а степень достоверности 99 % эквивалентна уровню значимости 1 %.

РЕЗЮМЕ

Анализ данных — это процесс, которому нужно тщательно следовать, чтобы добиться убедительных выводов. Чтобы обеспечить целостность своих результатов, проводите свои данные через каждый шаг конвейера.

Избавьте себя от лишней головной боли и *ни в коем случае не пропускайте* шаги и не меняйте их порядок. Если на этапе исследования обнаружится, что у вас недостаточно данных для проверки предположений, убедитесь, что все этапы выполняются со всем тщанием и следуют по порядку: сбор, обработка, исследование и только затем тестирование.

СОБЕРИТЕ На старте все ваше внимание должно быть сосредоточено на сборе и накоплении данных всех возможных типов и из всех возможных источников. Когда вы очищаете имеющиеся данные или создаете механизмы для сбора новых, ваша главная забота — избежать любых ошибок выборки.

ОБРАБОТАЙТЕ После того как данные собраны, облегчите компьютеру их понимание. На этом этапе проявите перфекционизм в вопросе *очистки*: убедитесь, что каждый аспект набора данных организован и что в итоге остались чистые, достоверные и непротиворечивые данные. Кроме того, вы несете ответственность за *анонимизацию* всех конфиденциальных данных до последнего бита.

ИССЛЕДУЙТЕ Как только нужная чистота достигнута, *обобщите* и *визуализируйте* свой набор данных. Сравните различные группы значений, понаблюдайте, как они распределены по своим диапазонам, и постройте график с течением времени. При обнаружении аномальных свойств исследуйте причины. Изучите, как данные отражают особенности реального мира.

ПРОВЕРЬТЕ И наконец, как выводы, полученные в результате исследования, соотносятся с вашими целями? Изучите таблицы и графики, которые могут повлиять на принятие решений. Сформулируйте гипотезы и проверьте, подтверждают ли данные их статистическую значимость.

Соблюдение этих принципов позволит принимать решения, основанные на фактах, а не на интуиции. И все же самое интересное еще впереди. Еще немного преобразовав данные, вы сможете передать их алгоритмам, которые дадут такую комплексную информацию о будущем, какую человеческий разум никогда не сможет получить в одиночку. Готовы?

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

- *Skiena S.* The Data Science Design Manual. http://code.energy/skiena.
- *Watts D. J.* Everything is Obvious. http://code.energy/watts.
- *Wheelan C.* Naked Statistics. http://code.energy/wheelan.

Глава 5

МАШИННОЕ ОБУЧЕНИЕ

> Вопрос о том, могут ли компьютеры думать, похож на вопрос о том, могут ли подводные лодки плавать.
>
> *Эдсгер Дейкстра*

Предсказания ведут к принятию решений. Столетия назад древние майя наблюдали за урожаями и движениями звезд, чтобы предсказать урожайность и решить, в какой момент лучше сажать кукурузу.

Сегодня фермеры могут принимать более взвешенные решения благодаря достижениям в прогнозировании погоды.

Один из способов предсказать будущее — обратиться к прошлому. Паттерны в прошлых событиях иногда раскрывают исход аналогичных будущих событий. Вполне очевидно, что, если вы замечаете, как бар заполняется почти каждую ночь, когда идет футбол, вы знаете, что он заполнится и в следующий вечер матча. **Машинное обучение (МО)** — это использование компьютеров для обработки данных о прошлых событиях, поиска закономерностей и раскрытия их предсказательных способностей. Эта технология повсюду. Когда вы оформляете автомобильную страховку или получаете кредит, компании записывают ваши данные. Со временем банки учатся предсказывать, кто может обанкротиться, а страховщики догадываются, кто может разбить свой автомобиль. В этой главе показано, как именно они это делают. Вы научитесь:

превращать собранные данные в **признаки**;

обучать машину и **оценивать** признаки;

- методически **сверять** ее прогнозы;
- **тонко настраивать** ее работу для достижения лучших результатов.

Делая прогнозы без компьютеров, пришлось бы тратить много времени и энергии, а также консультироваться с ведущими экспертами. С компьютерами можно автоматизировать это в нужном масштабе. Например, банки могут автоматически одобрять кредиты, экономя на расходах на персонал и быстрее обслуживая клиентов. В здравоохранении машины могут автоматически проверять пациентов с высоким риском, чтобы обеспечить профилактическую помощь и спасти жизнь.

МОДЕЛИ

Алгоритм МО, который делает прогнозы, называется **моделью**. Есть различные типы моделей, каждая из которых использует различные математические приемы. Некоторые из этих трюков не самые простые, но не волнуйтесь: мы не будем углубляться в то, *как именно* работают модели. Мы научимся *использовать* модели.

Представьте, что вы риелтор в городе Свитуотер и вам нужно предсказать, по какой цене будут проданы три квартиры. Сначала вы собираете подсказки — расстояние до центра города, возраст и район. Вы можете организовать их в таблице X, где каждый столбец является типом подсказки, известным как **признак** (feature), и каждая строка соответствует одной продаваемой квартире. Затем вы можете добавить столбец y_{pred} для ваших прогнозируемых цен (рис. 5.1).

X			y_{pred}
Расстояние	**Возраст**	**До центра**	**Предполагаемая цена**
0,4 км	4 года	114 м^2	?
2,4 км	10 лет	68 м^2	?
3,4 км	16 лет	40 м^2	?

Рис. 5.1. Квартиры на продажу в Свитуотере

Заполнение столбца y_{pred} не такая простая задача. Сама по себе таблица признаков X не содержит никаких базовых ценовых данных. Если бы эти

квартиры были в Токио или Тимбукту, цены, безусловно, были бы другими. Чтобы делать достоверные прогнозы, нужны примеры, на которых можно учиться. Добавим в таблицу квартиры, которые уже были проданы в Свитуотере, помеченные их истинной ценой продажи y, и назовем эти записи **размеченным набором** данных (рис. 5.2).

	X			y_{pred}	y
	Расстояние	Возраст	До центра	Предполагаемая цена	Истинная цена
	0,4 км	4 года	114 м^2	?	
РАЗМЕЧЕННЫЙ НАБОР ДАННЫХ	2,1 км	8 лет	120 м^2		$840 000
	10,6 км	9 лет	91 м^2		$540 000
	0,5 км	3 года	90 м^2		$745 000
	3,5 км	26 лет	35 м^2		$185 000
	2,4 км	10 лет	68 м^2	?	
	3,4 км	16 лет	40 м^2	?	

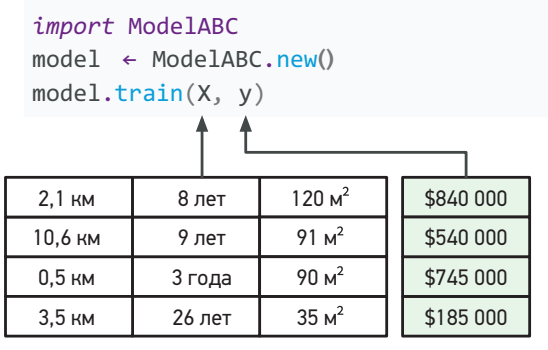
Рис. 5.2. Квартиры на продажу и уже проданные квартиры в Свитуотере

Взглянув на эту таблицу, можно уже что-то заподозрить. Цена квартиры площадью 114 м^2, вероятно, ближе к 1 000 000 долларов, чем к 100 000. Если у вас набралось гораздо больше размеченных примеров, то себе в помощь вы можете использовать прогнозирующую модель.

Не нужно самостоятельно писать код какой-либо модели, чтобы начать работу, ведь для большинства языков программирования доступны опенсорсные библиотеки МО. После установки библиотеки ее модели можно импортировать в свой код и тут же начать использовать.

ОБУЧЕНИЕ Прежде чем вы сможете использовать модель для прогнозирования, нужно заставить ее обучиться. Конкретный синтаксис зависит от используемого языка программирования и библиотеки, но он всегда сводится к вызову *обучающей* функции, которая принимает в качестве аргументов таблицы X и y размеченного набора данных. Поскольку они описывают один и тот же список событий, таблицы должны содержать одинаковое количество строк. По мере того как модель учится, она подкручивает свои внутренние шестеренки, чтобы

найти математическую формулу, которая аппроксимирует значения y из строк X. Как правило, X и y требуют огромного количества примеров, чтобы модель начала делать хорошие аппроксимации. В зависимости от приложения их количество может исчисляться тысячами, миллионами или миллиардами.



ПРОГНОЗИРОВАНИЕ Как только модель обучится, мы можем начинать прогнозировать. Если теперь мы возьмем строки X, для которых y неизвестен, мы можем использовать функцию *прогнозирования*, чтобы оценить y_{pred}. Этот вывод является лучшей аппроксимацией значений y, которые позже будут наблюдаться в реальной жизни. В данном случае прогнозы, вероятно, крайне ненадежны. Для их улучшения нам нужно обучить модель на более чем четырех размеченных примерах. Кроме того, эта модель учитывает лишь три признака. Посмотрим, как можно собрать больше данных для обучения более мощных моделей (рис. 5.3).

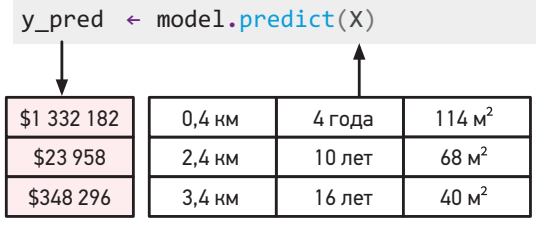


Рис. 5.3. «Благодаря алгоритмам машинного обучения восстание машин длилось недолго». Любезно предоставлено http://smbc-comics.com

5.1. ПРИЗНАКИ

Сбор и обработка данных для машинного обучения должны выполняться максимально тщательно. Прежде чем перейти к обучению модели, необходимо разобраться, как должны выглядеть данные, которые мы планируем передать модели. Представьте, к примеру, что вы работаете в больнице и в логистических целях хотите предсказать, как долго будет оставаться каждый поступающий пациент. Если вы начнете искать данные, то обнаружите сотни различных таблиц в разных форматах (рис. 5.4).

Машины пока еще не разумны, поэтому мы не можем просто бросить им данные и ждать, пока они обучатся и начнут что-то предсказывать. Нужно переработать данные в хорошо структурированные *таблицы* X и y, с которыми могут работать модели. Посмотрим, как это делается.

Пациенты

ID	Пол	Кровь	Широта	Долгота	Дата рожд.
57	F	O	36°53'22" N	27°17'05" E	30-Jan-1957
58	F	B	36°53'24" N	27°17'20" E	29-May-1999
59	M	A	36°45'10" N	26°59'21" E	28-Apr-1956
60	F	O	NULL	NULL	18-Feb-2006
61	NULL	NULL	36°53'36" N	27°17'23" E	04-Nov-[illegible]
62	M	AB	36°53'27" N	27°17'05" E	12-Oct-[illegible]

Поступления

Пациент	Прибытие	Боль	Убытие
57	28 мая	😟	NULL
58	28 мая	NULL	29 мая
59	28 мая	😫	7 июня
60	28 мая	😖	1 июня
61	28 мая	😖	29 мая
62	29 мая	😐	5 июня
58	30 мая	😟	30 мая

Пациент 58

Time	T
9:00 pm	37,0
10:00 pm	37,2
11:00 pm	37,0

Пациент 62

Время	T
12:00 am	38,1 °C
1:00 am	38,9 °C
2:00 am	40,3 °C
3:00 am	39,1 °C
4:00 am	38,5 °C

Рис. 5.4. Необработанные данные больницы

АДАПТАЦИЯ ДАННЫХ

Первый шаг в построении X и y из нашего набора данных — проверить, что у нас правильный тип данных. Большинство алгоритмов прогнозирования работают только с 1 2 3 4 **ЧИСЛОВЫМИ ДАННЫМИ**. Посмотрим, как обрабатывать другие виды данных, чтобы их тоже можно было включить в качестве признаков.

КАТЕГОРИАЛЬНЫЕ ДАННЫЕ Мы говорим, что категориальные данные **кодируются** в числовой форме. Метод кодирования будет зависеть от того, являются ли ваши данные *двоичными*, *порядковыми* или *номинальными*. Рассмотрим, что все это значит:

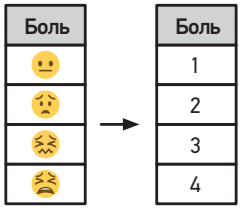
Пол
F
M

→

Пол
0
1

Кодирование *двоичных* данных. Они делятся на две категории, поэтому одна категория выражается как 0, а другая — как 1. В нашем случае данные указывают на биологический пол[1] пациента.

Кодирование *порядковых* данных. Категории упорядочены, поэтому мы присваиваем им соответствующие номера. Например, если в приемном покое неотложной помощи пациентам предлагается выбрать один из четырех смайликов для описания их боли, он может быть выражен числом от 1 до 4.



Кодирование *номинальных* данных. Когда категории нашего столбца не имеют внутреннего порядка, мы обычно кодируем столбец в несколько двоичных признаков: по одному для каждой категории. Этот метод называется **прямым унитарным** кодированием. Данный пример показывает, как он используется для выражения группы крови пациента в четырех признаках. Поскольку у пациента может быть только *одна* группа крови, закодированная запись должна иметь одну клетку, содержащую 1, а все остальные клетки должны содержать 0.

ВРЕМЕННЫЕ ДАННЫЕ Самым простым способом является включение записей даты или времени в качестве элементов числовых данных: дата может быть разделена на три признака для года, месяца и дня, а время может быть разделено на часы, минуты и секунды.

Это редко оказывается полезно для прогнозирующего алгоритма. Например, рождение пациента в апреле или ноябре не должно влиять на время его выздоровления. С другой стороны, имеет значение, 20 ему лет или 60. Следовательно, если у вас есть дата рождения и дата поступления, вы

[1] До 2 % из нас рождаются интерсекс-людьми, например, с хромосомами XXY. Двоичное кодирование может использоваться, если прогнозы непосредственно не влияют на отдельных пациентов, например, если время госпитализации оценивается для внутренних логистических целей.

можете **создать** новый признак, рассчитав возраст в годах (рис. 5.5). Если вы пытаетесь предсказать, как долго они пробудут в больнице, вы делаете то же самое для y: продолжительность пребывания — это количество дней между поступлением и выпиской (рис. 5.6).

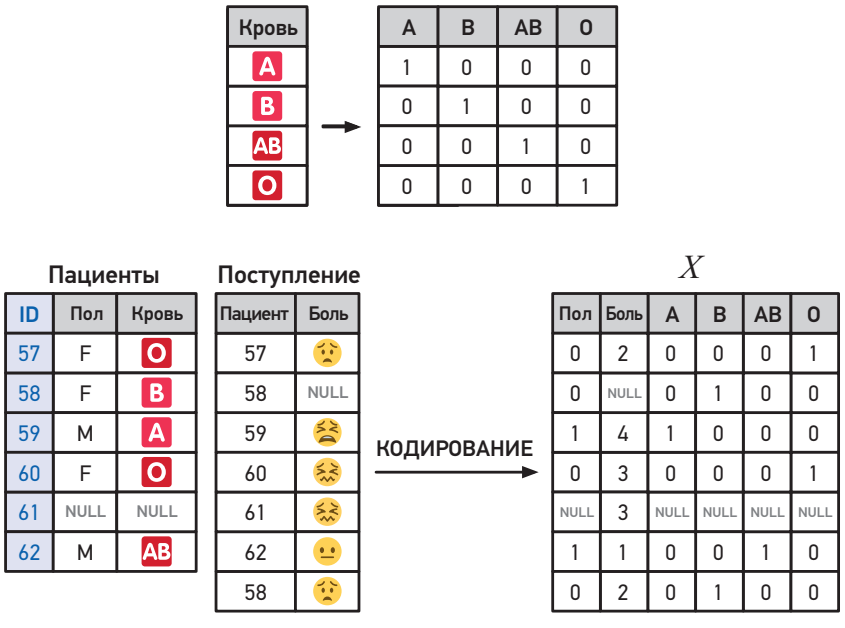


Рис. 5.5. Получение признаков из различных типов категориальных данных

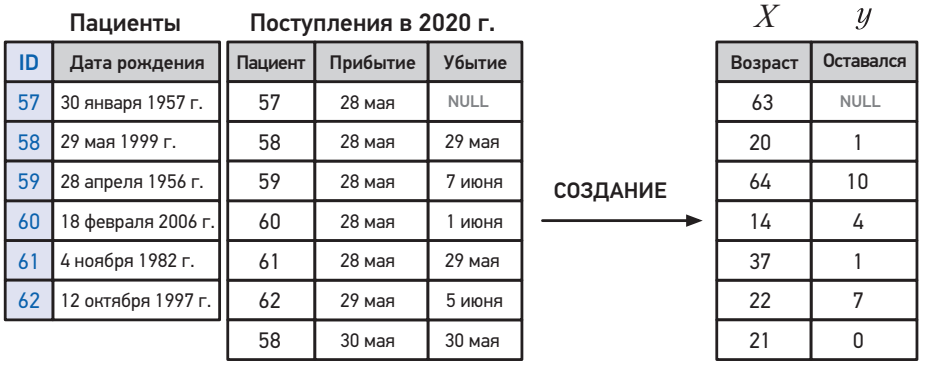


Рис. 5.6. Получение признаков из временных данных

Еще один способ создания признака — преобразование и кодирование временных данных в категориальные данные. Например, если госпитализации из-за интоксикации чаще происходили ночью или в выходные дни, это может повлиять на y. Можно сделать эту информацию доступной в X в виде двоичных данных. Так можно выявить очень много скрытой информации, например, о времени года, наступлении праздников или наличии открытых аптек. Создайте признаки для хронологических и сезонных паттернов, которые, как вам кажется, связаны с тем, что вы хотите предсказать.

Дата	Время
28 мая	2:56 am
28 мая	4:32 pm
29 мая	11:05 pm
30 мая	8:40 pm

→

Выходные	Ночь
0	1
0	0
0	1
1	1

ГЕОГРАФИЧЕСКИЕ ДАННЫЕ Географические данные часто представлены числами в координатах широты и долготы. Сами по себе эти цифры указывают только на то, насколько далеко на север и восток находится то или иное место. Числа могут быть разделены на различные числовые признаки, но, как и для временных данных, эти функции, как правило, не очень полезны. Чтобы получить значимые признаки на основе географических данных, подумайте о том, как знание местоположений соотносится с тем, что вы хотите предсказать. Часто бывает полезно рассчитать расстояния: если вы хотите спрогнозировать цены на квартиры, создайте признаки для их расстояний до центра города, до ближайшего пляжа или до ближайшей школы, супермаркета и больницы (рис. 5.7).

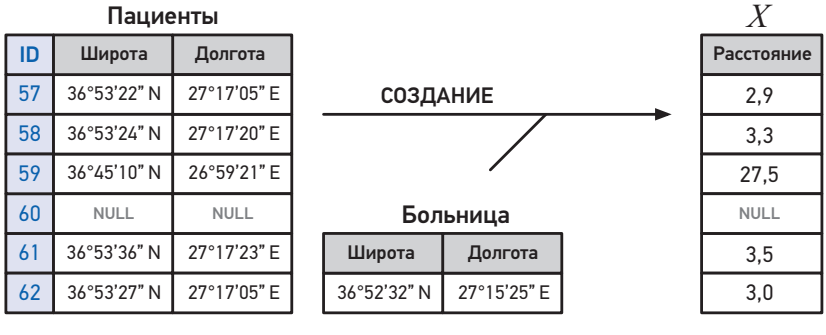


Рис. 5.7. Путем преобразования адресов пациентов в географические координаты можно получить расстояние от них до больницы

Числовые координаты также могут быть преобразованы в категориальные данные, такие как город, район или почтовый индекс. Некоторые из этих категориальных данных могут даже позволить вам внести дополнительную информацию в качестве признаков, например сопоставить индекс доходов и уровень преступности.

НЕСТРУКТУРИРОВАННЫЕ ДАННЫЕ  По оценкам, около 80 % цифровых данных в мире не структурированы и, следовательно, не могут быть закодированы в виде строк и столбцов, а скорее в виде отдельных файлов. Построение признаков из неструктурированных данных затруднено. Чтобы получить численные данные из неструктурированных данных, мы должны выяснить, какие поддающиеся количественной оценке аспекты имеют отношение к нашему проекту.

Представьте, что набор записей пациентов содержит компьютерную томографию легких. Полезное число, которое врачи могли бы получить из этих изображений, — это площадь легких каждого пациента.

Наиболее распространенной формой неструктурированных данных выступает текст. Какой количественный аспект текстового файла может нас заинтересовать? Количество слов — простое и популярное значение. Другим распространенным признаком служит появление определенных слов. Например, тот факт, что медицинское заключение содержит слова «опухоль» или «рак», может быть закодирован как двоичные категориальные данные. Или, если мы подсчитаем количество вхождений ключевого слова и разделим его на общее количество слов, мы получим *частоту*, с которой оно встречается[1].

Понять, какие полезные признаки могут прятаться в неструктурированных данных, будет проще, если вы проконсультируетесь с экспертами, которые регулярно их собирают и анализируют. Если вы спросите врачей, что именно они ищут при проверке медицинских заключений или компьютерной томографии, они могут подкинуть интересные идеи для создания соответствующих признаков.

[1] Мешок слов — это наиболее часто используемый метод для создания признаков из частоты появления слов. Для справки см. http://code.energy/bag-of-words.

ОБЪЕДИНЕНИЕ ДАННЫХ

Необработанные данные часто будут иметь различные формы и размеры, и иногда признаки не получится закодировать без предварительной обработки. Кроме того, некоторые единицы данных сами по себе не имеют большого значения, но мы можем объединять их с другими. Посмотрим, как это сделать.

СОВОКУПНОСТИ Если у нас имеется некоторая информация, поступающая в виде набора чисел для каждой записи в X или y, мы зачастую не можем кодировать каждое число как признак. Например, если мы отслеживаем температуру тела пациента несколько раз в день, в нашем наборе данных будет много замеров температуры каждого пациента. Для преобразования этих данных в признаки способны помочь инструменты обобщения из предыдущей главы (см. раздел 4.3) (рис. 5.8).

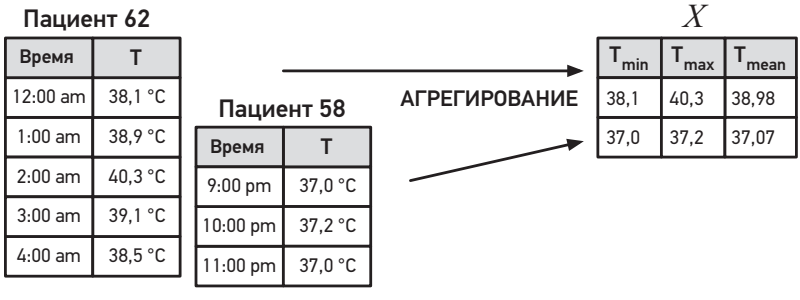


Рис. 5.8. Построение признаков путем агрегирования данных

При попытке спрогнозировать будущие показатели обычно используются агрегированные данные на основе исторических данных — прошлые показатели часто являются хорошим индикатором будущих. Например, если вы прогнозируете, выиграет ли футбольная команда, постройте признак на основе сводки пяти чисел по результатам команды за последние годы.

ОЦЕНКИ Мы можем выразить интересующий нас атрибут в одном числе, даже если информация, связанная с этим атрибутом, поступает из разных переменных. **Оценка** — это число, сгенерированное математической формулой для представления определенного атрибута. Например, *индекс массы тела (ИМТ)* часто используется в здравоохранении и рассчитывается исходя из роста пациента в метрах и веса в килограммах (рис. 5.9).

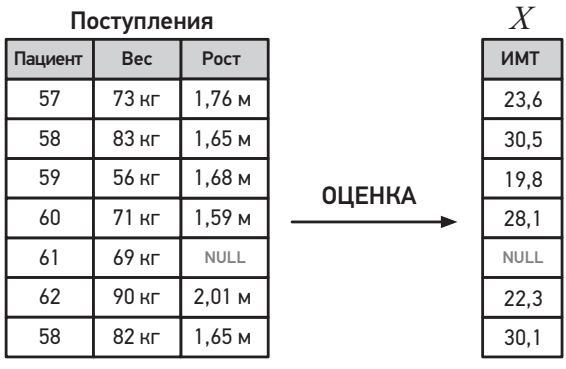


Рис. 5.9. ИМТ, рассчитанный на основе роста и веса пациента

Хорошо продуманные оценки инкапсулируют знания о том, как именно несколько факторов взаимодействуют и вносят вклад в значимый атрибут. Вычисление оценки и добавление ее к X делает эти знания доступными для модели. Например, если вы пытаетесь спрогнозировать, разовьется ли у человека диабет, и у вас есть его рост и вес, то включите и показатель ИМТ в качестве признака. Ожирение является фактором риска развития диабета, и его лучшим способом измерения часто служит ИМТ.

В социальных науках *индекс гуманитарного развития* (human development index, HDI) оценивает развитие страны по уровню грамотности, средней продолжительности жизни и валовому внутреннему продукту (ВВП) на душу населения. В финансовой сфере кредитный рейтинг FICO описывает кредитоспособность человека, учитывая его чистый долг и историю платежей.

Если для данного показателя нет четко установленной оценки, вы можете создать свою собственную. Например, если вы пытаетесь предсказать будущую цену продажи квартир и считаете, что предполагаемая безопасность здания является хорошим фактором, вы можете построить свой собственный «показатель безопасности», который сочетает в себе множество факторов: количество преступлений, зарегистрированных по соседству, расстояние от здания до полицейского участка, имеется ли в нем швейцар или видеонаблюдение.

После создания новых признаков вернитесь к методам исследования данных из главы 4. Создавайте сводки и графики для новых признаков. Это поможет проверить, были ли они созданы корректно.

Кроме того, коэффициент корреляции признака с y иногда может обеспечить предварительными данными касательно ее предсказательной способности.

ПРОПУЩЕННЫЕ ЗНАЧЕНИЯ

Как правило, требование, чтобы данные были *числовыми* в X, является строгим: большинство алгоритмов прогнозирования не способны обрабатывать ячейки, содержащие `NULL`. Когда значение признака недоступно для каких-то записей, необходимо очистить столбец от пустых ячеек. Есть несколько способов проделать это.

СБРОС Самый простой метод состоит в том, чтобы удалить из X все строки, содержащие одну или несколько ячеек с `NULL` (рис. 5.10).

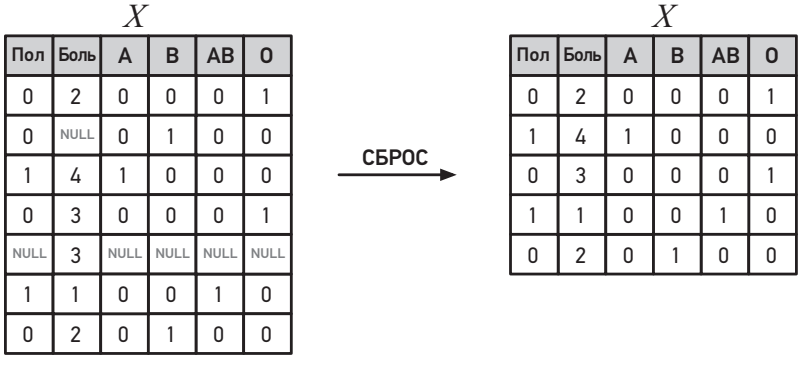


Рис. 5.10. Сброс неполных строк из X

Сброс — хороший подход, когда дело касается нескольких строк. Однако уменьшение объема данных X в процессе обучения, как правило, снижает производительность прогнозирования модели, а удаление слишком большого количества строк ограничит ее способность к прогнозу. Более того, удаление строк не только означает, что у вас теперь меньше данных для обучения: это также означает, что вы утрачиваете возможность прогноза любого важного события с отсутствующими данными.

Иногда у нас есть *столбец*, в котором большинство значений отсутствуют. В таких случаях удаление данного признака из X часто является лучшим вариантом, поскольку позволяет сохранить много строк.

Есть еще один способ справиться с недостающими данными: заполнить ячейки с NULL теми значениями, которые вы считаете подходящим. Этот процесс называется **подстановкой**, и тремя наиболее используемыми значениями выступают *наиболее частые*, *средние* и *новые метки*.

НАИБОЛЕЕ ЧАСТЫЕ Если в столбце имеется много повторных значений, то наиболее часто такое значение используется во всех ячейках данного столбца, содержащих NULL. Это особенно полезно для устранения пропусков в категориальных данных (рис. 5.11, 5.12).

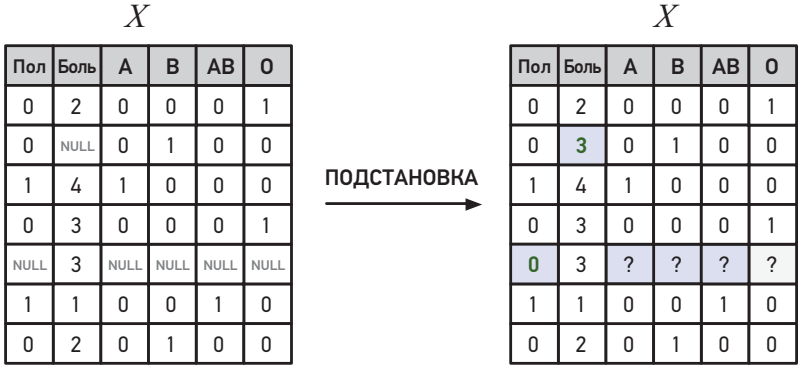


Рис. 5.11. Подстановки для двоичных и порядковых кодировок, таких как биологический пол и уровень боли, могут быть сделаны непосредственно в X. Однако унитарное кодирование для группы крови исправить таким образом не получится, поскольку все столбцы данного свойства содержат в основном нули

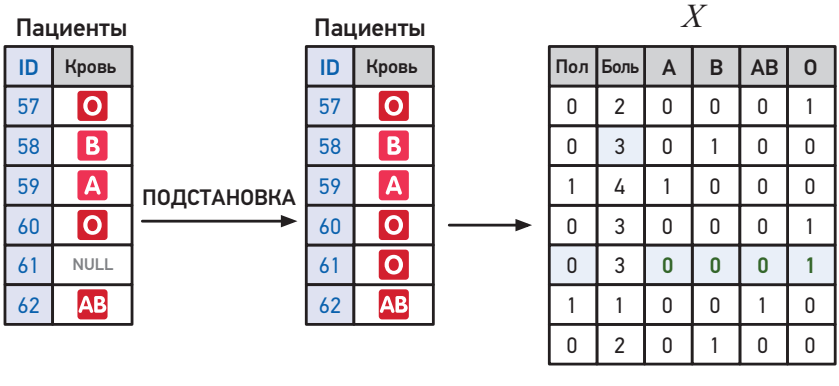


Рис. 5.12. Номинальные данные должны быть подставлены до прямого унитарного кодирования. Таким образом, одна из вмененных ячеек будет равна 1

Обратите внимание, что эта операция должна быть выполнена *перед* унитарным кодированием, чтобы убедиться, что каждая строка с унитарным кодированием содержит `1`. Именно так мы избежали заполнения заполнения ячеек с вопросительными знаками на рис. 5.11 одними нулями.

УСРЕДНЕНИЕ В результате работы такого подхода ячейки с `NULL` заполняются средними значениями остальных ячеек столбца (рис. 5.13).

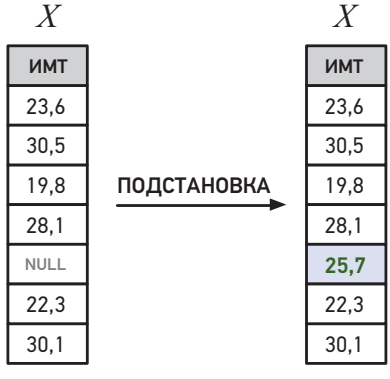


Рис. 5.13. Подстановка в ячейки среднего значения столбца

Обычно для этого используется среднее арифметическое или медиана. Будьте осторожны с этим подходом: он не будет работать с категориальными признаками. Например, если вы попытаетесь сделать такое с двоичными данными, пустые ячейки могут оказаться заполнены десятичными значениями вместо единиц и нулей.

НОВАЯ МЕТКА Эта стратегия работает только для категориальных переменных. Когда ячейки с `NULL` часто встречаются в категориальном столбце, мы можем создать новую метку и назначить ее всем ячейкам с пропущенными значениями.

> **Тест для лечения**  Больница борется с пандемией коронавируса, и вы должны оценить, насколько вероятно, что поступающие пациенты нуждаются в аппаратах ИВЛ. Вы тестируете пациентов на наличие вируса и создаете категориальный признак, в котором каждый из них помечается как «отрицательный» или «положительный». Тестов для всех не хватает, поэтому приоритет отдается пациентам, у которых проявляются

такие симптомы, как повышенная температура и кашель. В итоге лишь половина пациентов была протестирована, и 60 % из тестов оказались положительными. Как заполнить ячейки с NULL непроверенных пациентов?

Сброс строк для непроверенных пациентов не подходит, так как он исключил бы половину записей, сделав модель бесполезной для прогнозирования, скажем, общего количества аппаратов ИВЛ, необходимых в будущем. Использование *наиболее частого* значения чревато проблемами: большинство тестов оказались положительными, но тесты не проводились случайным образом, поэтому результаты не обязательно совпадут с состоянием бессимптомных пациентов. Это одна из форм ошибки выборки (см. раздел 4.1): пациенты без симптомов с меньшей вероятностью заражены вирусом, нежели пациенты с симптомами, поэтому учет их как «положительных» по умолчанию был бы ошибкой!

Таким образом, у вас остается два варианта: попросить больницу протестировать небольшую выборку бессимптомных пациентов и обобщить этот результат или же просто создать *новую метку* для непроверенных пациентов. Первый подход позволит вам сохранить двоичное кодирование признака, в то время как второй потребует унитарного кодирования для трех конечных меток («отрицательный», «положительный» и «непроверенный»).

Удаление записей, выбор наиболее частых значений, вычисление средних значений, а также создание новых меток — это самые простые и наиболее распространенные способы обработки пропущенных значений. Ученые активно ищут новые и более эффективные способы условной подстановки в большие наборы данных. Как правило, подстановка эффективна, если отсутствующие данные появляются случайным образом. Если есть некая глубинная причина, по которой ряд ячеек должен содержать `NULL`, эти стратегии могут скорее навредить.

УТЕЧКА ДАННЫХ

Когда мы строим X и y исходя из прошлых событий, мы можем случайно включить в X информацию, которая существует только после наблюдения y. Это называется **утечкой данных**. Если такое происходит, модель обучается прогнозировать будущее по будущим данным, что бессмысленно.

Когда вы добавляете признаки в X, подумайте об информации, которая будет доступна, когда вы сделаете прогнозы. Проверьте X и убедитесь, что он выглядит точно так же, как данные, которые модель должна будет оценить y_{pred}. Давайте посмотрим.

> **ЗАНЯТЫЕ КОЙКИ** Вы работаете менеджером системы бизнес-аналитики больницы. На следующий день после поступления каждого пациента вас просят предсказать, как долго тот будет в больнице. Вам предоставлен доступ к следующей информации о предыдущих пациентах: возраст, пол, ИМТ, группа крови, жизненно важные показатели и общая стоимость пребывания. Как вам выбрать, какие данные включать в качестве столбцов набора данных?

Первый шаг — проверить, когда и как была собрана эта информация. Например, предположим, что возраст ( ) и пол ( ) собираются во время поступления, а группа крови ( ) — вскоре после этого. Однако рост ( ) и вес ( ), необходимые для получения ИМТ, обычно поступают через пару дней. Измерения жизненно важных показателей, таких как частота сердечных сокращений ( ), кровяное давление ( ) и температура тела ( ), производятся на протяжении всего пребывания пациента. Наконец, стоимость пребывания ( ) может быть рассчитана только после выписки пациента (рис. 5.14).

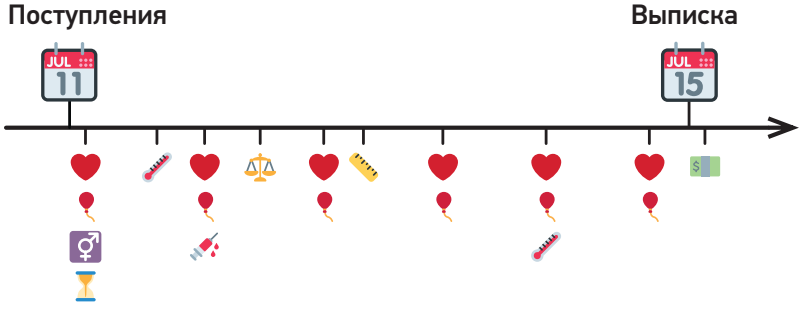


Рис. 5.14. Временной отрезок наблюдений для типичного пациента

Продолжительность пребывания y рассчитывается при выписке. Мы уже видим, что включение стоимости пребывания в таблицу признаков X размеченных примеров было бы бессмысленным, поскольку это означало бы предсказание прошлого! Чтобы точно определить, какие данные

не могут повлиять на X, давайте добавим на временной отрезок момент наблюдения y и момент, когда мы хотели бы, чтобы модель оценила y_{pred} (рис. 5.15).

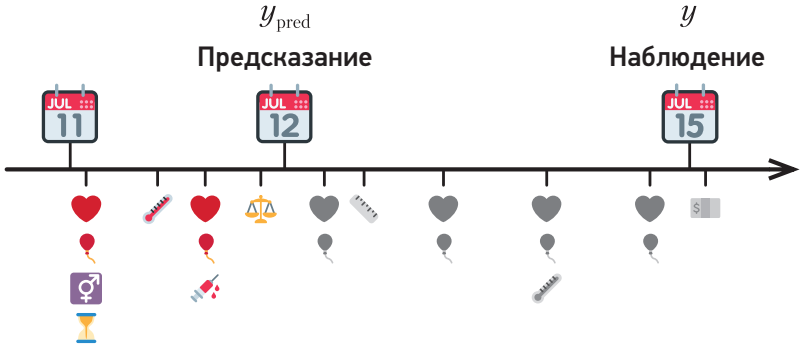


Рис. 5.15. Наблюдения, выделенные серым цветом, никогда не должны влиять на X

Любые значения, измеренные или наблюдаемые после момента оценки y_{pred}, не должны включаться в X, даже во время фазы обучения. В нашем примере ИМТ не должен включаться в признаки, поскольку он зависит от роста, который обычно измеряется *после* того, как сделан прогноз. Жизненные показатели могут быть включены в X, если они суммируют только значения, измеренные в первый день.

Пока вы уверены, что утечки данных нет, извлекайте как можно больше признаков. Это повысит вероятность того, что модель выявит хорошие шаблоны для надежных прогнозов. Теперь посмотрим, как выбрать модель и оценить, насколько точны ее прогнозы.

5.2. ОЦЕНКА

После построения X и y следующим шагом является выбор модели для обучения. Первое, что вы заметите, — это то, что модели являются либо **регрессорами**, либо **классификаторами**. Регрессоры предсказывают величины, в то время как классификаторы — метки. Чтобы дать прогноз о национальности людей, вы будете использовать классификатор. Но чтобы предсказать возраст, лучше использовать регрессор.

Выберите одну из моделей из вашей библиотеки и приступайте к работе. Не беспокойтесь о выборе конкретной модели, пока можете выбрать любую. На этом этапе просто убедитесь, что вы выбираете регрессор, если y — это числовые данные, или классификатор, если категориальные.

Большинство моделей используют метод, называемый **контролируемым обучением** (или обучением с учителем). Суть состоит в том, чтобы модель делала серию догадок на основе обучающих примеров. Каждый раз модель будет использовать свою внутреннюю функцию прогнозирования (`predict`), а затем оценивать, насколько хорошо она справилась, сравнивая выходной y_{pred} с истинным y. Используя эту оценку и ряд математических трюков, она пытается улучшить себя и снова угадать. Как только прогнозы перестанут улучшаться, модель прекратит обучение. Следующая блок-схема обобщает процесс (рис. 5.16).

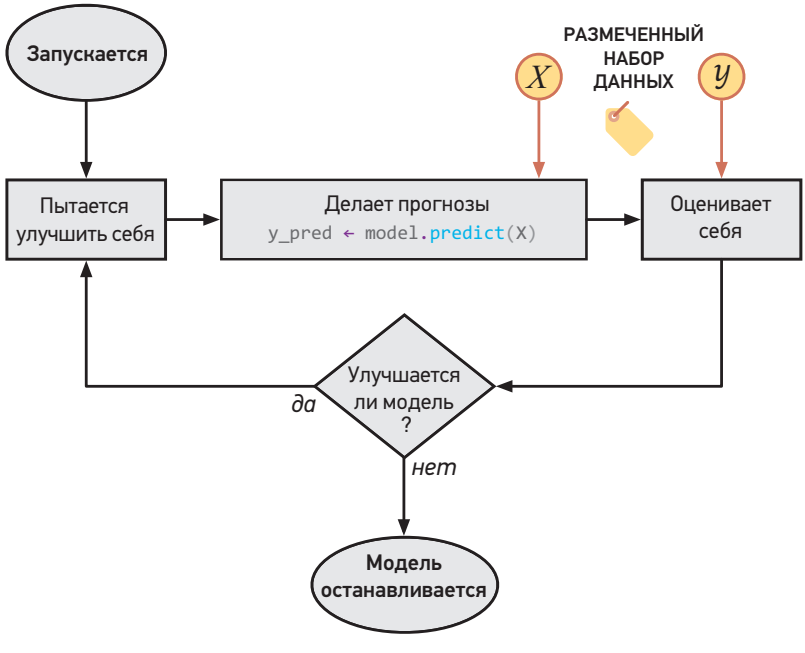


Рис. 5.16. Контролируемое обучение в действии

Теперь у нас есть обученная модель, которую можно использовать для прогнозирования будущего. Чудненько. Но насколько хорошо эта модель

научилась предсказывать? Как можно это оценить? Есть несколько различных оценочных метрик, которые работают либо для регрессоров, либо для классификаторов. Начнем с первых.

ОЦЕНКА РЕГРЕССОРОВ

Поскольку регрессоры предсказывают величины, естественным способом оценить, насколько близко данное предсказание y_{pred} к истинному y, является вычисление **ошибки** между ними:

$$\text{error} = y_{\text{pred}} - y.$$

Вычисление дает одно значение ошибки на строку X и y. Как правило, мы обобщаем все эти ошибки, используя какой-либо тип среднего значения. Вот несколько распространенных способов сделать это.

СРЕДНИЙ МОДУЛЬ РАЗНОСТИ (ОТКЛОНЕНИЯ) (**M**ean **A**bsolute **E**rror, **MAE**). Среднее арифметическое — это наиболее часто используемое среднее значение. Однако при вычислении среднего арифметического положительные и отрицательные ошибки компенсируют друг друга, создавая ложное впечатление точности. Избежать этого можно с помощью MAE, среднего арифметического всех *абсолютных* значений ошибок, то есть среднего арифметического всех значений ошибок *по модулю*. Например, предположим, что мы обучаем модель для прогнозирования цен на квартиры в тысячах долларов на основе размеченных примеров из рис. 5.2 (рис. 5.17).

y_{pred} Предполагаемая цена	y Истинная цена	Ошибка	Модуль разности
650	840	–190	190
662	540	122	122
835	745	90	90
179	185	–6	6

$$\text{MAE} = \frac{190 + 122 + 90 + 6}{4} = 102.$$

Рис. 5.17. Расчет MAE для четырех прогнозов

Согласно этому методу средняя ошибка составляет 102 000 долларов. Если бы мы рассчитали среднее арифметическое *ошибок* вместо *абсолютных значений ошибок*, мы получили бы вводящее в заблуждение среднее значение 4000 долларов! Всегда учитывайте, что при расчете MAE вы суммируете положительные значения.

СРЕДНЕКВАДРАТИЧНАЯ ОШИБКА (КОРЕНЬ) (**R**oot **M**ean **S**quare **E**rror, **RMSE**). В некоторых случаях одно крайне неточное предсказание способно привести к катастрофе. Если мы чувствительны к экстремальным ошибкам, мы можем заставить нашу оценку браковать их более явно, вычисляя среднее арифметическое *квадратов* ошибок (рис. 5.18).

y_{pred} Предполагаемая цена	y Истинная цена	Ошибка	(Ошибка)^2
650	840	–190	36 100
662	540	122	14 884
835	745	90	8100
179	185	–6	36

$$\text{RMSE} = \sqrt{\frac{36\,100 + 14\,884 + 8100 + 36}{4}} \approx 122.$$

Рис. 5.18. Расчет RMSE для четырех прогнозов

На этот раз прогнозы дают среднюю ошибку около 121 600 долларов. Здесь нет никаких проблем с отрицательными числами, так как *квадрат* отрицательного числа — это всегда положительное число. Кроме того, **RMSE** — это *квадратный корень* из среднего арифметического, который возвращает значение к исходному масштабу ошибок.

Посмотрим, как выбор между MAE и RMSE влияет на обучение. Представьте, что модель попыталась улучшить себя и теперь дает следующие прогнозы (рис. 5.19).

MAE увеличился с 102 000 до 110 000 долларов, в то время как RMSE снизился с примерно 122 000 до 112 000 долларов. Это показывает, что на вопрос «Улучшается ли модель?» от одной итерации обучения к другой (см. рис. 5.16) можно ответить по-разному в зависимости от того, какой метод оценки используется.

y_{pred} Предполагаемая цена	y Истинная цена
748	840
650	540
887	745
281	185

Ошибка	Модуль разности	(Ошибка)^2
–92	92	8464
110	110	12 100
142	142	20 164
96	96	9216

MAE = 110, RMSE ≈ 112.

Рис. 5.19. Оценка новых прогнозов с использованием MAE и RMSE

Будьте внимательны и не сравнивайте ошибки модели, когда они предсказывают результаты разных масштабов. Например, если бы одна и та же модель была обучена предсказывать *ежемесячную арендную плату* одних и тех же квартир, MAE и RMSE, вероятно, составили бы около 500 долларов. Эти меньшие оценки *не* будут указывать на то, что модель стала лучше: они просто отразят тот факт, что весь столбец меток y был примерно в 200 раз меньше.

MAE и RMSE могут дать некоторое представление о способности модели предсказывать будущее. Чем лучше модель обобщается на новые данные, тем ближе эти оценки будут от ошибок будущих прогнозов.

А ЧТО ТАМ С КЛАССИФИКАТОРАМИ? Если модель является классификатором, то, кроме прогнозирования, она будет выдавать метки, а не числа. Это означает, что методы оценки, которые мы разбирали до этого, к ней применить уже не получится. Оценка классификатора может оказаться *намного* сложнее, чем оценка регрессора. Чтобы не отвлекать вас от основной темы построения системы прогнозирования, мы поместили исчерпывающее объяснение того, как оценивать классификаторы, в приложение IV. Не забудьте его прочитать!

5.3. ПРОВЕРКА РАБОТОСПОСОБНОСТИ

Иногда модель обучается так, что оказывается слишком привязана к конкретным событиям, на которых она училась. Когда это происходит, модель аппроксимирует значения y для записей в размеченном наборе данных намного лучше, чем для записей, которые она никогда не видела. Это называется **переобучением**.

Когда мы оцениваем переобученную модель для событий, на которых она училась, результирующая оценка вводит в заблуждение. Поскольку этот риск всегда сохраняется, мы **проверяем** наши модели: оцениваем их прогнозы для событий, *не* использовавшихся в обучении. Это единственный способ убедиться, что оценка модели не получилась обманчиво хорошей из-за переобучения. Таким образом, оценка отразит *истинные показатели работы модели*: те, которых мы будем ожидать от нее при составлении реальных прогнозов в реальном мире. Чтобы иметь возможность проверить модель, мы должны отделить ряд размеченных примеров *прежде*, чем приступать к обучению. С этой целью мы разделяем строки X и y на 🎓 **обучающий набор** и ✅ **проверочный набор**.

Модель обучается с использованием данных из обучающего набора и оценивается по тому, как она предсказывает события в проверочном (рис. 5.20).

Обратите внимание, что строки размеченного набора данных должны быть разделены *случайным* образом. Разделение набора данных без перетасовки его строк часто привносит в процесс ошибку выборки.

Нет никаких строгих правил для того, насколько большим должен быть каждый набор. Как правило, от 10 до 20 % данных резервируется для проверки. Чем больше обучающий набор, тем успешнее модель может научиться работать. Чем больше проверочный набор, тем больше наша уверенность, что итоговая оценка отражает истинные показатели модели. Но если проверочный набор не является чрезвычайно большим с сотнями тысяч записей, специалисты по обработке данных едва ли полагаются исключительно на единственную оценку, полученную с помощью этого простого разделения набора данных.

К сожалению, записи, которые мы выбираем для включения в обучающие или проверочные наборы, влияют на наш окончательный результат. Часто различные выборки размеченного набора данных не одинаково сложны для изучения моделью. Если вы оцениваете модель только один раз, есть риск, что вы смешали строки в нетипичный обучающий или проверочный набор, в результате чего оценка не отражает истинные показатели модели.

Перекрестная проверка снижает этот риск, повторяя весь процесс несколько раз. Каждый размеченный набор данных разбивается на

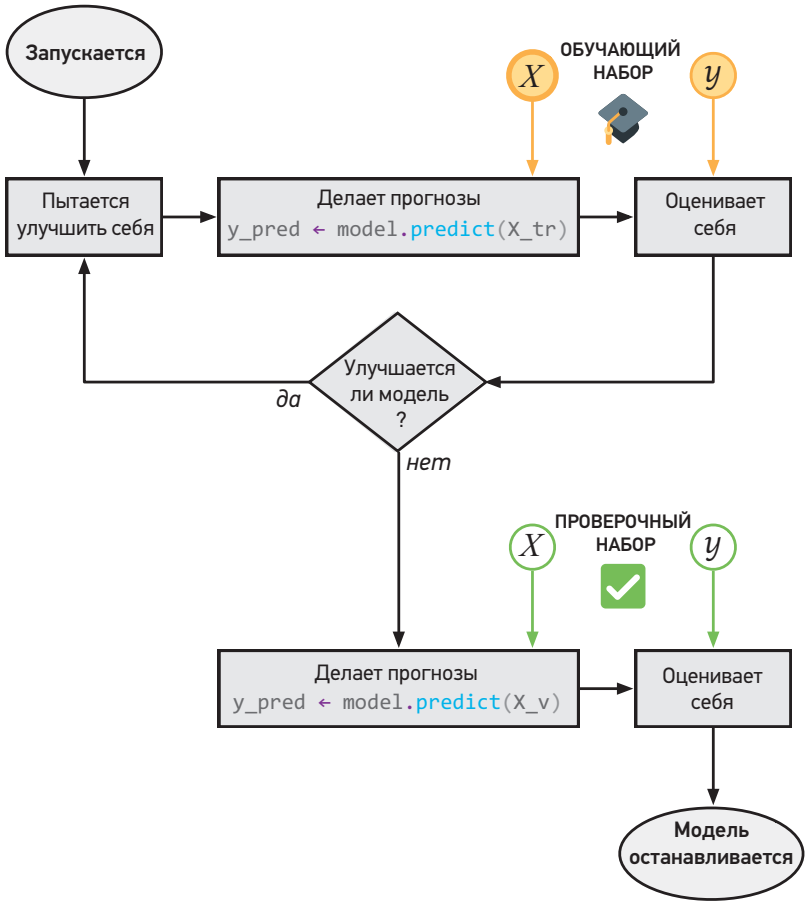


Рис. 5.20. Проверочный набор отделен для итоговой независимой оценки модели

различные наборы для обучения и проверки. Это дает множество оценок, которые в совокупности описывают производительность модели. Есть несколько различных способов выполнения перекрестной проверки. Изучим три наиболее распространенных из них.

K-FOLDS

Вместо того чтобы разбивать перемешанный набор данных на две части, как мы делали при создании пары обучающих и проверочных наборов,

разделим его на десять различных групп, или **блоков (folds)**, (почти) одинакового размера (рис. 5.21).

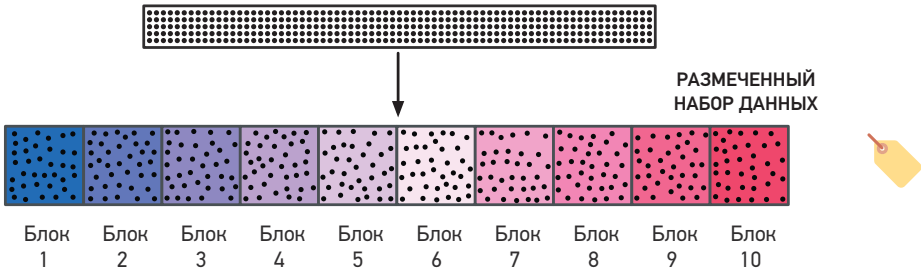


Рис. 5.21. Перемешивание 365 записей в блоки по 36 или 37 штук каждый

Используя эти блоки, можно построить обучающий набор, объединив записи в первых девяти блоках и оставив последнюю для проверки (рис. 5.22).

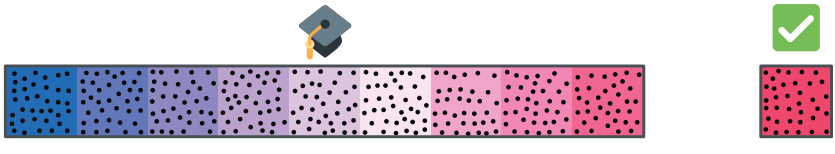


Рис. 5.22. Блоки 1–9 — это обучающий набор, блок 10 — проверочный набор

Зарезервировав другой блок для проверки, мы можем построить десять различных наборов обучения и проверки. Это позволяет оценить модель десять раз, получив десять разных оценок (рис. 5.23).

Этот метод называется ***k*-fold**, так как можно использовать любое количество блоков. С большим количеством блоков вы оцениваете свою модель большее число раз и количество записей в тренировочных наборах увеличивается. При этом большее количество блоков требует и большего количества вычислений. Кроме того, обучающие наборы становятся более похожими друг на друга, что сводит на нет некоторые преимущества многократной оценки модели. Как правило, лучше придерживаться конфигурации по умолчанию, используемой большинством ученых: либо пять, либо десять блоков.

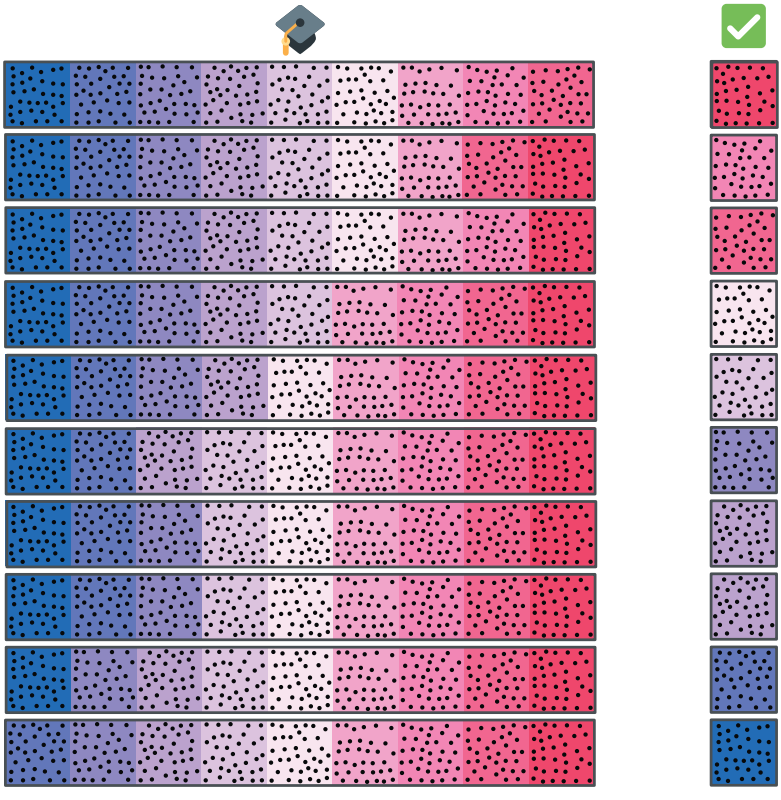


Рис. 5.23. Десять различных наборов для обучения и проверки

МОНТЕ-КАРЛО

При подходе *k*-folds каждая запись попадает в проверочный набор лишь единожды. Это означает, что каждая запись используется для проверки модели только один раз. Это может стать проблемой: если бы модель была обучена с другим набором обучающих данных, она по-другому предсказывала бы одно и то же событие. Этого никак не проверить с помощью *k*-folds.

С другой стороны, **метод Монте-Карло** допускает, чтобы одна запись появлялась в нескольких проверочных наборах с целью более тщательной проверки модели. Вместо использования фиксированного набора блоков метод использует полное перемешивание размеченного набора данных перед каждым разделением. Таким образом, выбирается множество

случайных наборов для обучения и проверки, а затем модель обучается и оценивается для каждого случайного выбора.

Монте-Карло используется, когда имеется достаточно вычислительных мощностей и модель может быть повторно обучена и оценена с небольшими затратами. Если бы мы хотели обучаться на 90 % наших 365 записей и использовать оставшиеся 10 % для проверки, метод Монте-Карло мог бы оценивать модель практически неограниченное количество раз[1] с различными случайными обучающими и проверочными наборами. После нескольких миллионов оценок мы будем гораздо более уверены, что различные оценки в совокупности отражают истинные показатели модели.

ИСКЛЮЧЕНИЕ ПО ОДНОМУ (LEAVE-ONE-OUT)

Иногда приходится работать с очень мелким размеченным набором данных, состоящим всего из нескольких десятков записей. В таких случаях имеет смысл сохранять обучающий набор как можно большим. С этой целью мы обучаем модель на всех записях, *кроме одной*, и проводим оценку на этой оставшейся записи.

Эта стратегия называется **leave-one-out**. Ее можно рассматривать как частный случай *k*-fold: число блоков равно количеству записей, поэтому каждый блок содержит только одну запись.

Но есть загвоздка: все тренировочные наборы чрезвычайно похожи друг на друга. Несмотря на то что модель будет оцениваться многократно, оценки будут очень похожи, сводя на нет некоторые преимущества нескольких оценок. Используйте leave-one-out только в крайнем случае, если не в силах увеличить размер своего крошечного набора данных.

ИНТЕРПРЕТАЦИЯ

Перекрестная проверка дает множество оценок, которые в совокупности описывают успешность вашей модели. Обычно мы суммируем эти зна-

[1] Теоретически есть триллионы триллионов триллионов триллионов различных возможных наборов проверки, которые используют 10 % наших 365 записей. В книге «Теоретический минимум по Computer Science» объясняется, как посчитать такие комбинации.

чения, чтобы лучше интерпретировать их, например, с помощью сводки пяти чисел. Интерпретация может зависеть от того, для чего именно предназначена ваша модель.

Например, если вы не против плохих результатов, если модель *иногда* срабатывает очень хорошо, сосредоточьтесь на верхнем квартиле и максимальной оценке. Если, с другой стороны, вы не можете позволить себе даже редкие случаи плохого результата работы модели, используйте пессимистический подход и сосредоточьтесь на нижнем квартиле и минимуме.

В общем случае среднее значение берется как наиболее релевантный итог того, как работает модель. Но среднее значение — это лишь приблизительное представление о работе модели. Рекомендуется учитывать его вместе со стандартным отклонением: степень, в которой производительность модели обычно отличается от среднего значения.

Помимо оценки ожидаемой производительности модели, результаты перекрестной проверки также позволяют нам сравнивать различные модели. Есть множество типов моделей с различным внутренним устройством. Чтобы сравнить две модели и решить, какая из них лучше, не стоит смотреть на средние значения. Выполните проверку гипотезы на перекрестно проверенных оценках обеих моделей, чтобы проверить, есть ли статистическая значимость в утверждении, что одна модель имеет более высокую среднюю оценку, чем другая.

Часто более тяжеловесная в вычислительном плане модель будет иметь несколько более высокую среднюю оценку, чем более простая. Тем не менее проверка гипотезы покажет, что разница не является статистически значимой. В подобных случаях выбирайте более простую модель.

5.4. ПОДСТРОЙКА

Мы научились подготавливать данные, обучать универсальную модель МО и оценивать ее работу. Если вы остановитесь на этом и не внесете коррективы в свою систему, то добьетесь лишь малой доли доступной предсказательной способности.

Систему прогнозирования можно подстроить по-разному. Например, корректировка признаков может значительно повысить точность

прогнозирования. Выбор наиболее подходящего типа модели и настройка ее конкретных параметров под вашу задачу может оказаться не менее полезным делом.

Специалисты по обработке данных начинают, как правило, с обучения и перекрестной проверки простой модели прогнозирования. Затем вносят небольшие постепенные изменения в свою систему, каждый раз проводя новую перекрестную проверку. Результаты до и после каждого изменения сравниваются, после чего изменения, которые приносят статистически значимые улучшения, сохраняются. Это продолжается до тех пор, пока перестанут проявляться видимые улучшения в системе.

Начнем с изменений, которые можно внести в признаки. Признаки наиболее эффективны, когда дают модели четкие подсказки. Ваши столбцы *X* должны представлять собой набор подсказок, которые как можно более прямо связаны с тем, что вы предсказываете. Иногда в достижении этого способно помочь преобразование значений в *X*.

Рис. 5.24. «Машинное обучение», любезно предоставлено http://xkcd.com

Преобразование признаков с целью их лучшей совместимости с моделью называется **конструированием признаков**. Из всех корректировок, которые могут быть внесены в систему прогнозирования, эта зачастую имеет наибольший потенциал для улучшения. Рассмотрим некоторые из наиболее распространенных методов конструирования признаков.

ПОДСТАНОВКА

В разделе 5.1 мы сказали о том, что X не может содержать ячейки с `NULL`, и обсудили стратегии их обработки: удаление строк с отсутствующими данными, удаление столбцов с отсутствующими данными, заполнение пустых ячеек средним значением столбца или значением по умолчанию. Чтобы узнать, какие стратегии лучше всего подходят для вашей задачи прогнозирования и ваших данных, протестируйте их и оцените их эффективность.

Например, если вы заполняете ячейки медианой столбца, также попробуйте среднее значение и константу по умолчанию. Для каждого столбца, содержащего более 1 % ячеек `NULL`, проверка различных стратегий подстановки, как правило, будет стоить усилий. Если ни одна стратегия не является видимо лучшей, придерживайтесь самой простой: заполните все пустые ячейки постоянным значением по умолчанию.

ВЫБРОСЫ

Наличие выбросов в X может сбивать с толку некоторые модели. Как и в случае с подстановкой пропущенных значений, разработаны различные способы борьбы с выбросами. Например, если столбец содержит лишь несколько выбросов, часто лучше удалить эти строки из X. Если выбросы являются частыми (например, более 1 % строк), можно попытаться заменить их менее экстремальными значениями. Этот метод известен как **отсечение** (clipping) (рис. 5.25).

Рис. 5.25. Отсечение трех выбросов

НОРМАЛИЗАЦИЯ

Некоторые модели сбивают с толку признаки, которые отображают совершенно разные диапазоны. Признак, значения которого варьируются от нуля до нескольких тысяч, может получить смещенный вес, если другие столбцы варьируются от нуля до нескольких десятков. Чтобы предотвратить эту проблему, столбцы можно масштабировать для отображения одинакового диапазона. Это называется **нормализацией**. Самый простой метод нормализации называется **«минимакс» (MinMax)**. Минимальное значение в столбце становится нулем, а максимальное — единицей (рис. 5.26). Все остальные значения размещаются между ними.

$$x_{new} = \frac{x - \min}{\max - \min}$$

2	5522		0,20	0,53
90	7840		1,00	0,83
72	9138	MinMax →	0,84	1,00
34	8431		0,49	0,91
27	1388		0,43	0,00
−20	6983		0,00	0,72

Рис. 5.26. Минимакс сжимает все значения до диапазона 0–1

Можно нормализовать значения в соответствии с их индивидуальным расстоянием до среднего и стандартным отклонением столбца: значение, равное среднему, становится нулем; значение, равное стандартному отклонению, становится единицей. Этот метод нормализации называется **z-score** (z-оценка) (рис. 5.27).

$$x_{new} = \frac{x - \text{mean}}{\text{std.dev.}}$$

2	5522		−0,85	−0,20
90	7840		1,48	0,50
72	9138	Z-SCORE →	1,00	1,00
34	8431		0,00	0,73
27	1388		−0,19	−2,00
−20	6983		−1,43	0,17

Рис. 5.27. При нормализации z-score диапазоны не фиксируются

При использовании минимакса все столбцы имеют *один и тот же* диапазон. Более того, наличие выбросов заставит не-выбросы прижиматься ближе друг к другу, создавая более мелкие области в диапазоне 0–1. При использовании z-score выбросы оказывают меньшее влияние, так как не вызывают этого сжатия, и диапазоны не-выбросов имеют меньше различий между столбцами. Чтобы узнать, каким методом нормализации воспользоваться, попробуйте оба и сравните результаты перекрестной проверки.

ЛОГАРИФМИЧЕСКОЕ ПРЕОБРАЗОВАНИЕ

В разделе 4.4 мы построили график снижения затрат на хранение данных за последние десятилетия и обнаружили, что данные, охватывающие величины, различающиеся на несколько порядков, могут нуждаться в специальной обработке. То же самое верно и для передачи данных такого типа в модели прогнозирования. Признаки с огромной дисперсией часто ослабляют производительность модели.

Если в таблице X содержатся такие столбцы, попробуйте преобразовать их значения, используя логарифмическую функцию (`log`). Если столбец содержит значения от нуля до единицы, добавьте единицу перед вычислением логарифма, чтобы избежать больших отрицательных результатов.

Если столбец содержит еще и отрицательные числа, преобразование становится сложнее, так как логарифмы для таких чисел не определены. Один из способов обойти это — удалить знак минус, добавить единицу, применить логарифм и вернуть знак минус:

$$x_{\log} = \begin{cases} \log(x+1) & \text{если } x \geq 0, \\ -1 \times \log((-1 \times x)+1 & \text{в противном случае.} \end{cases}$$

Кроме того, можно нормализовать столбец сразу после выполнения логарифмического преобразования. Рассмотрим пример на рис. 5.28.

Прямая нормализация данных с экстремальной дисперсией приводит к тому, что многие значения становятся крайне близкими. Если выполнить перед нормализацией логарифмическое преобразование, числа лучше распределяются по диапазону и различия между значениями легче воспринимаются моделью.

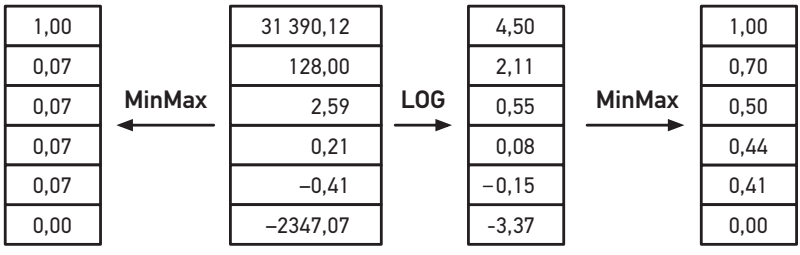


Рис. 5.28. Минимакс с предварительным логарифмическим преобразованием и без него

БИННИНГ

Создание признаков, несущих *меньше* информации, иногда полезно для модели. Предположим, ваша больница пытается предсказать продолжительность пребывания поступающих пациентов и у вас есть столбец с их возрастом. Вы замечаете, что небольшие возрастные различия мало влияют на продолжительность госпитализации, при этом между детьми и взрослыми наблюдается значительная разница. В этом случае попробуйте помочь модели дифференцировать эти группы. Например, можно преобразовать возраст в категориальный признак с метками «ребенок», «подросток» и «взрослый».

Этот метод называется **биннингом**. Он заключается в организации исходных данных в «бины», которые более релевантны, чем исходные значения. Это не менее полезно, когда у вас есть категориальные функции, где некоторые метки появляются редко. Изучите их и объедините метки, связанные между собой.

Например, категориальный признак, указывающий на болезнь пациента, может иметь тысячи меток. Объединение их в бины связанных заболеваний может помочь модели понять смысл данных. Если нет времени разрабатывать сложные бины, попробуйте просто заменить все низкочастотные метки одной меткой «другие». Одно лишь это потенциально может очень помочь модели.

КЛАСТЕРИЗАЦИЯ

В больницах пациенты часто распределяются по группам в соответствии с их симптомами или болезнью. Это помогает врачам и медсестрам: они

могут проверить, к какой группе относится пациент, и тут же получить ценную информацию о пациенте.

Если вы менеджер ресторана, то и вам будет полезно разбить посетителей на группы. Можно выделить любителей фастфуда, любителей здоровой пищи, скряг и т. д. Знание этих групп позволяет ориентировать различные рекламные акции на тех клиентов, которые с наибольшей вероятностью на них откликнутся. Кроме того, ресторан может адаптировать предложения для того, чтобы угодить наиболее денежным клиентам.

Будьте осторожны: важно разумно группировать элементы, чтобы это помогло модели. Большинство наборов данных будут содержать наборы строк с аналогичными характеристиками. Если вам удастся сгруппировать записи в соответствии с этими сходствами, добавьте столбец в X с указанием, в какую группу входит каждая запись. Эта новая категориальная функция зачастую повышает производительность прогностической модели.

Процесс поиска сходных характеристик и приписывание каждой записи к группе называется **кластеризацией**. Есть алгоритмы, которые могут автоматизировать кластеризацию. Наиболее широко используемый алгоритм кластеризации называется **методом k-средних (k-means)**. Установите библиотеку, содержащую этот алгоритм, и запустите ее на своем наборе данных. В дополнение к X алгоритм требует, чтобы вы ввели k, количество различных групп, которые нужно создать. Для начала запустите его несколько раз, установив k от 2 до 10. Каждый раз вы должны получать другой дополнительный столбец для X. Проведите перекрестную проверку модели для каждой группы и сохраните ту, которая больше всего помогает вашей модели.

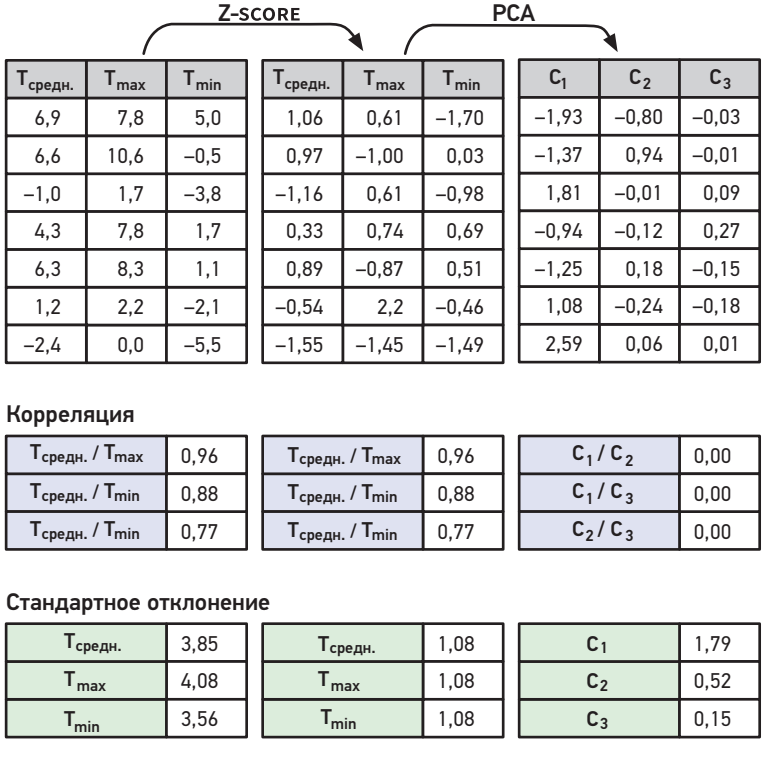
Если используете автоматически созданную группировку строк в качестве новой функции, то можете выполнить интересное упражнение по исследованию данных. Просмотрите записи, приписанные к каждой группе, и попытайтесь придумать имя, которое лучше всего ее представляет. Это зачастую приводит к интересным открытиям.

ИЗВЛЕЧЕНИЕ ПРИЗНАКОВ

По мере того как мы собираем признаки, X может в итоге приобрести столбцы, которые сильно коррелируют друг с другом. Когда это происходит, в модель передается избыточная информация, которая в большинстве

случаев не очень-то полезна. Большинство моделей работают лучше, когда столбцы в X содержат различающуюся информацию. К счастью, специальные математические приемы помогают сжать информацию из группы столбцов в меньший, но более репрезентативный объем данных. Этот процесс называется **извлечением признаков**.

Хорошо известный метод извлечения признаков — **метод главных компонент (Principal Component Analysis, PCA)**[1]. Он требует, чтобы столбцы в X были предварительно нормализованы с помощью z-score. PCA преобразует некоторые столбцы в менее коррелированные с сохранением исходной информации. Рассмотрим пример, в котором столбцы хранят среднюю, минимальную и максимальную дневную температуру в городе (рис. 5.29).



$T_{средн.}$	T_{max}	T_{min}
6,9	7,8	5,0
6,6	10,6	–0,5
–1,0	1,7	–3,8
4,3	7,8	1,7
6,3	8,3	1,1
1,2	2,2	–2,1
–2,4	0,0	–5,5

$T_{средн.}$	T_{max}	T_{min}
1,06	0,61	–1,70
0,97	–1,00	0,03
–1,16	0,61	–0,98
0,33	0,74	0,69
0,89	–0,87	0,51
–0,54	2,2	–0,46
–1,55	–1,45	–1,49

C_1	C_2	C_3
–1,93	–0,80	–0,03
–1,37	0,94	–0,01
1,81	–0,01	0,09
–0,94	–0,12	0,27
–1,25	0,18	–0,15
1,08	–0,24	–0,18
2,59	0,06	0,01

Корреляция

$T_{средн.}$ / T_{max}	0,96
$T_{средн.}$ / T_{min}	0,88
$T_{средн.}$ / T_{min}	0,77

$T_{средн.}$ / T_{max}	0,96
$T_{средн.}$ / T_{min}	0,88
$T_{средн.}$ / T_{min}	0,77

C_1 / C_2	0,00
C_1 / C_3	0,00
C_2 / C_3	0,00

Стандартное отклонение

$T_{средн.}$	3,85
T_{max}	4,08
T_{min}	3,56

$T_{средн.}$	1,08
T_{max}	1,08
T_{min}	1,08

C_1	1,79
C_2	0,52
C_3	0,15

Рис. 5.29. Признаки, преобразованные с помощью z-score и PCA. На каждом шаге показан коэффициент корреляции между столбцами, а также стандартное отклонение каждого столбца

[1] Визуальное объяснение PCA см. здесь: http://code.energy/pca.

PCA возвращает новый X, где столбцы отображают небольшую корреляцию друг с другом. Кроме того, PCA имеет тенденцию сжимать больше информации в столбцах слева и упорядочивать их, уменьшая уровни дисперсии. После этого можно попробовать отбросить столбцы справа, которые отображают небольшую дисперсию, и посмотреть, будет ли модель работать лучше. И она часто работает!

Еще один трюк с извлечением функций — это **неотрицательное матричное разложение** (**N**on-negative **M**atrix **F**actorization, **NMF**). Как следует из его названия, он работает только в том случае, если X не содержит отрицательных значений. Он позволяет сжимать информацию X в количество столбцов по нашему выбору. Используем NMF для преобразования данных из нашего предыдущего примера в два столбца (рис. 5.30).

MinMax

$T_{средн.}$	T_{max}	T_{min}
6,9	7,8	5,0
6,6	10,6	−0,5
−1,0	1,7	−3,8
4,3	7,8	1,7
6,3	8,3	1,1
1,2	2,2	−2,1
−2,4	0,0	−5,5

$T_{средн.}$	T_{max}	T_{min}
1,00	0,74	1,00
0,97	1,00	0,48
0,15	0,16	0,16
0,72	0,74	0,69
0,93	0,78	0,63
0,39	0,21	0,32
0,00	0,00	0,00

NMF(2)

C_1	C_2
0,67	0,67
0,91	0,00
0,13	0,09
0,62	0,34
0,74	0,27
0,22	0,24
0,00	0,00

Корреляция

$T_{средн.}$ / T_{max}	0,96
$T_{средн.}$ / T_{min}	0,88
$T_{средн.}$ / T_{min}	0,77

$T_{средн.}$ / T_{max}	0,96
$T_{средн.}$ / T_{min}	0,88
$T_{средн.}$ / T_{min}	0,77

C_1/C_2	0,34

Стандартное отклонение

$T_{средн.}$	3,85
T_{max}	4,08
T_{min}	3,56

$T_{средн.}$	0,41
T_{max}	0,48
T_{min}	0,34

C_1	0,35
C_2	0,24

Рис. 5.30. Признаки, преобразованные с помощью минимакса и NMF. На каждом шаге показан коэффициент корреляции между столбцами, а также стандартное отклонение каждого столбца

В отличие от PCA, NMF генерирует неупорядоченные столбцы, которые одинаково релевантны. В то время как PCA выводит ряд столбцов с уменьшающейся релевантностью, NMF пытается разделить информацию на столбцы, представляющие различные модели поведения. Как обычно, полезно будет поэкспериментировать с разным количеством выходных столбцов. Протестируйте свою модель с таблицами NMF разных размеров и выберите число, которое работает лучше всего.

Перед PCA или NMF столбцы X обычно представляют собой определенное измерение или атрибут. После любого преобразования столбцы больше не имеют прямой связи с атрибутом из реального мира, даже если они в совокупности и сохраняют исходную информацию. Это может помочь машине, но делает данные менее прозрачными для людей. По этой причине извлечение признаков обычно выступает последним этапом процесса конструирования признаков.

ОТБОР ПРИЗНАКОВ

Некоторые признаки могут скорее навредить производительности модели, нежели помочь. Это часто происходит с избыточными признаками, признаками, содержащими данные низкого качества, или же признаками, которые на деле вообще не связаны с y. Удаление бесполезных столбцов из X называется **отбором признаков**.

Производительность модели будет отличаться, когда мы используем различные подмножества признаков и хотим выбрать подмножество, которое дает максимальный эффект. Единственный способ добиться этого — оценить все возможные варианты столбцов и использовать лучший из найденных вариантов. Это называется **исчерпывающим поиском**, и он выполним только в случае, если X содержит несколько столбцов. При наличии множества признаков количество подмножеств для оценки возрастает[1].

В таких случаях мы отбираем признаки с помощью *эвристических методов*: методов, которые приводят к решению, не гарантируя, что оно лучшее или оптимальное. Самый простой такой метод называется **обратным исключением**. Мы постоянно проверяем, улучшает ли исключение одного

[1] Количество подмножеств набора с n элементами равно 2^n.

из признаков работу модели. Если подобное происходит, признак удаляется. После того как мы попытались удалить все признаки, не влияющие на успешность, останавливаемся и сохраняем оставшиеся:

```
keep_changing ← True
while(keep_changing)
    keep_changing ← False
    for each column in X
        X_new ← X.remove(column)
        if is_better(X_new, X)
                X ← X_new
                keep_changing ← True
                break
```

Обратный процесс называется **прямым отбором**: начните с пустого набора признаков и продолжайте по одному добавлять новые, пока они улучшают производительность. Когда стоит выбор между большей или меньшей группой признаков, выбирайте меньшую[1].

Есть еще множество стратегий отбора признаков. Они, как правило, делают модель проще и быстрее в дополнение к улучшению ее производительности.

И СНОВА УТЕЧКА ДАННЫХ

Мы уже видели, как размеченный набор данных должен быть разделен на обучающие и проверочные наборы с целью оценки модели. Поскольку проверочный набор используется для оценки истинных показателей работы модели за пределами области обучения, он никогда не должен влиять на обучение модели.

Это также относится к конструированию свойств: все преобразования будут влиять на обучение модели и, следовательно, должны калиброваться *только* с использованием обучающего набора. Невыполнение этого требования равносильно *утечке данных*, описанной в разделе 5.1. Рассмотрим несколько примеров, чтобы вы точно уловили мысль.

[1] Обратное исключение и прямой отбор — это формы жадных эвристических методов, которые мы описали в нашей первой книге «Теоретический минимум по Computer Science. Все, что нужно программисту и разработчику».

Допустим, вы выполняете подстановку среднего в один из своих столбцов. Средние значения вычисляются только по строкам обучающего набора. Пустые ячейки обучающего и проверочного наборов будут заполнены средним значением *обучающего набора*. Таким образом, никакая информация из проверочного набора не просочится в обучающий.

Аналогично всегда нужно определять выбросы, основываясь только на обучающем наборе. После принятия решения примените одни и те же числовые операции как к наборам обучения, так и к наборам проверки.

При нормализации с помощью минимакса используйте минимальные и максимальные значения обучающего набора в качестве эталона для преобразования как обучающего, так и проверочного наборов. Если вы нормализуетесь с помощью z-score, считайте среднее значение и стандартное отклонение обучающего набора. При разработке стратегий биннинга учитывайте наличие меток в обучающем наборе только для того, чтобы решить, как следует классифицировать ваши данные.

Те же правила применяются и для извлечения признаков: алгоритм PCA или NMF должен быть откалиброван с использованием обучающего набора. Затем точно такие же математические операции нужно выполнить как на обучающих, так и на проверочных наборах.

Мы знаем, что обучающий набор изменяется для каждой независимой оценки во время перекрестной проверки. Это означает, что ваши инженерные преобразования признаков должны быть повторно откалиброваны и применены к каждой новой паре наборов обучения и проверки. Будьте осторожны: не калибруйте и не применяйте преобразования признаков только один раз и до перекрестной проверки. Сделайте это много раз и *во время* перекрестной проверки. Это распространенная ошибка, которая приводит к неточным прогнозам.

ВЫБОР МОДЕЛИ

Опенсорсные библиотеки машинного обучения доступны для большинства современных языков программирования. Как правило, эти библиотеки обычно поставляются с десятками различных предустановленных моделей, которые можно сразу же использовать.

Каждая отдельная модель использует математические приемы для обучения и прогнозирования. Став опытным специалистом по МО, вы

узнаете, как работают модели и как они ведут себя в различных ситуациях. Этот опыт поможет более эффективно их использовать.

Если вы только начинаете работать с МО, можно попробовать все доступные модели из вашей библиотеки, даже если вы понятия не имеете, что происходит у них под капотом. Когда их все опробуете, то эмпирически обнаружите, какие из них лучше всего подходят для задач прогнозирования, над которыми вы работаете.

Есть лишь одна вещь, которую важно понять относительно каждой используемой модели: ее параметры конфигурации, также известные как **гиперпараметры**. Они должны быть адаптированы к вашей задаче прогнозирования. Использование настроек по умолчанию часто приводит к плохой производительности и мешает как следует оценить потенциал модели.

Чтобы правильно настроить гиперпараметры, начните с документации к модели. Как только вы узнаете, какие бывают варианты, начните экспериментировать. Например, можно попробовать удвоить или уменьшить вдвое значения по умолчанию для каждого гиперпараметра и посмотреть, как это повлияет на производительность. Повозитесь с гиперпараметрами, пока не удостоверитесь, что обнаружили правильные настройки для каждого из них[1]. После настройки гиперпараметров каждой модели, которую собираетесь оценивать, можно сравнить их работу и наконец выбрать лучшую.

ЗАКЛЮЧИТЕЛЬНЫЕ ШАГИ

После настройки признаков и модели пришло время подготовить систему к развертыванию в реальном мире. Поскольку вы уже провели все нужные корректировки, проверочный набор больше не нужен. Возьмите все подстановки, обработки выбросов, нормализацию и правила биннинга, а также преобразования извлечения признаков и откалибруйте их, используя *весь* размеченный набор данных.

[1] С опытом вы найдете методы, которые автоматически выполняют поиск по многим настройкам гиперпараметров для данной модели и автоматически выбирают лучшую.

Далее возьмите выбранную модель с ее идеальными гиперпараметрами и обучите, используя весь размеченный набор данных. Ваши правила преобразования признаков вместе с обученной моделью — это и есть итоговая система прогнозирования, готовая к запуску.

Как только произойдет новое, невиданное ранее событие, передайте его признаки в систему. Если все было сделано правильно, это позволит получить представление о будущем этого события. Этот проблеск будет настолько же точным, насколько верно вы оценили будущую производительность модели с помощью перекрестной проверки... Верно?

К сожалению, поскольку весь набор данных для настройки системы используется из раза в раз, есть вероятность того, что на этапе тонкой настройки в систему вмешается переобучение. Это означает, что система адаптирована к вашему набору данных лучше, чем к каким-либо другим событиям. Она будет лучше работать с записями вашего размеченного набора данных, нежели с неизвестными ранее событиями.

ТЕСТОВЫЙ НАБОР Единственный способ обеспечить абсолютно беспристрастную оценку системы прогнозирования — оценить ее с помощью данных, с которыми она никогда не контактировала. С этой целью можно выделить небольшую часть нашего размеченного набора данных в **тестовый набор** еще до того, как начнем перекрестную проверку. Тестовый набор *никогда* не используется для обучения, проверки или окончательной подготовки модели. Он используется только для однократной окончательной оценки модели перед запуском в эксплуатацию. Если размеченные данные объективны, производительность модели на тестовом наборе должна быть отражением ее производительности в реальном мире.

РЕЗЮМЕ

Создание системы машинного обучения — непростой процесс. Есть множество тонкостей, которые важно понять, чтобы получить полезные результаты прогнозирования. Поэтому пользуйтесь преимуществами имеющихся библиотек: они помогут с конструированием свойств, перекрестной проверкой и оценкой модели. Многие из них умеют делать почти все, что мы видели в этой главе.

При использовании таких библиотек преобразования данных и этапы перекрестной проверки занимают всего несколько строк кода. Програм-

мисты любят приложения для совместного использования кода по типу блокнотов. Блокноты кода содержат текст, код и графики в одном файле, и их можно легко просматривать, повторно выполнять и экспериментировать с ними.

Принцип воспроизводимости, который подробно обсуждался в предыдущей главе, также применим и к МО. Убедитесь, что все преобразования ваших данных воспроизводимы и структурированы в виде конвейера, который легко изменить. То же самое относится и к процессам тонкой настройки модели и ее гиперпараметров.

Имейте в виду, что эта глава была лишь введением в практику МО. За бортом осталось множество приемов. Например, есть методы объединения результатов прогнозирования различных моделей в так называемый *ансамбль*, который часто превосходит любую отдельную модель.

МО идет вперед семимильными шагами. Каждый год публикуются все новые методы конструирования свойств и выбора моделей. Крупные технологические компании в настоящее время предлагают услуги облачных вычислений с автоматизированными платформами для создания моделей прогнозирования. Используя идеи, представленные в этой главе, вы сможете наилучшим образом использовать эти ресурсы... и осмысливать революцию в области искусственного интеллекта по мере ее развития.

Рис. 5.31. Машинное обучение решает не все[1]

[1] Игра слов, patient — «терпеливый». — *Примеч. пер.*

Мысль о том, что машины влияют на наши решения — или что они попросту принимают их за нас, — может напугать до чертиков. Так что всякий раз, когда предсказание может повлиять на чью-то жизнь, думайте о возможных последствиях. Машинное обучение — это ни добро, ни зло, а ваша задача — использовать его этично и ответственно.

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

- *Theobald O.* Machine Learning for Absolute Beginners. https://code.energy/theobald.
- *Skiena S.* The Data Science Design Manual. https://code.energy/skiena.
- *Goodfellow J.* Deep Learning. https://code.energy/goodfellow.
- *Tibshirani R.* The Elements of Statistical Learning. https://code.energy/tibshirani.
- *Raschka S.* Model Evaluation and Selection. https://code.energy/raschka.

ЗАКЛЮЧЕНИЕ

> В действительности мы преуспели в том, чтобы сделать нашу дисциплину наукой, причем на удивление простым способом: просто решив назвать ее computer science.
>
> *Дональд Кнут*

В книге описаны средства, позволившие computer science высвободить силы для построения цифрового мира, в котором мы теперь и живем. Эти концепции помогут вам познакомиться ближе с сетями и наукой о данных. И так вы внесете свой вклад в построение еще более изобретательного социума.

Мы попытались ограничить глубину своего исследования начальным уровнем. Задача состояла в том, чтобы научить основам, которые должны знать все, кто работает с кодом. Мы надеемся, что пробудили ваше любопытство и вы изучите книги, упомянутые в конце каждой главы. Далее вы найдете дополнительную главу о регулярных выражениях: о методе поиска по шаблонам, который поможет вам на этапах анализа данных и решения задачи машинного обучения.

Мы хотели включить в эту книгу больше тем, но она стала бы слишком большой. Например, данные, которые вы отправляете через интернет, могут быть прочитаны любым маршрутизатором, передающим пакеты, и мы хотели бы обсудить технологию, которая позволяет безопасно использовать интернет. Говоря о машинном обучении, мы пропустили алгоритмы, которые позволяют компьютерам собирать знания без участия человека. И не говорили об облачных вычислениях, которые позволяют творить новые миры намного быстрее. Следите за обновлениями, если вам интересны эти темы, — возможно, мы расскажем о них в следующей книге! Вы можете подписаться на наши новости по адресу http://code.energy/list.

Надеемся, эта книга обогатила ваши теоретические знания. А теперь к делу! Начните новый проект и попробуйте применить новые знания в своем коде. Как сказал один мудрый программист:

> *«Если вы обнаружите, что тратите почти все свое время на теорию, начните уделять время и практике; это улучшит ваши теории. Если вы обнаружите, что тратите почти все свое время на практику, начните уделять некоторое внимание теории; это улучшит вашу практику».*

И не стесняйтесь присылать нам свой фидбэк по адресу hi@code.energy. Обратная связь, которую мы получили по нашей первой книге «Теоретический минимум по Computer Science», побудила нас написать новую. А еще мы постарались улучшить свой стиль письма. Спасибо!

Бонусная глава 6

ШАБЛОНЫ

> Регулярные выражения позволяют управлять данными. Контролируйте их. Пусть они работают на вас. Овладеть регулярными выражениями — значит овладеть своими данными.
>
> *Джеффри Фридл*

Программисты часто работают с данными, которые соответствуют заданному шаблону. Представим документ, содержащий даты, записанные в виде «27 февраля 2013 года», «27/2/13» и «2013-02-27». Как найти все даты в подобном файле? Написание кода для обнаружения шаблонов занимает много времени и довольно утомительно.

Рис. 6.1. Лучший дейтинг, по мнению босса

К счастью, можно задать шаблоны, такие как форматы дат, с помощью **регулярных выражений**. Затем эти выражения могут быть интерпретированы программным обеспечением, поэтому не нужно самим писать код соответствия шаблону. В этой главе мы научимся:

- **сопоставлять** основные шаблоны;
- точно **количественно** определять повторяющиеся шаблоны;
- **привязывать** шаблоны к местоположениям;
- захватывать **группы** элементов в шаблоне.

Регулярные выражения, также называемые **regex** или **regexp**, — это обожаемое программистами средство экономии времени. Они легко интегрируются в код благодаря множеству легкодоступных библиотек и инструментов. На самом деле они уже встроены в большинство языков программирования, редакторы исходного кода, инструменты командной строки и т. д.

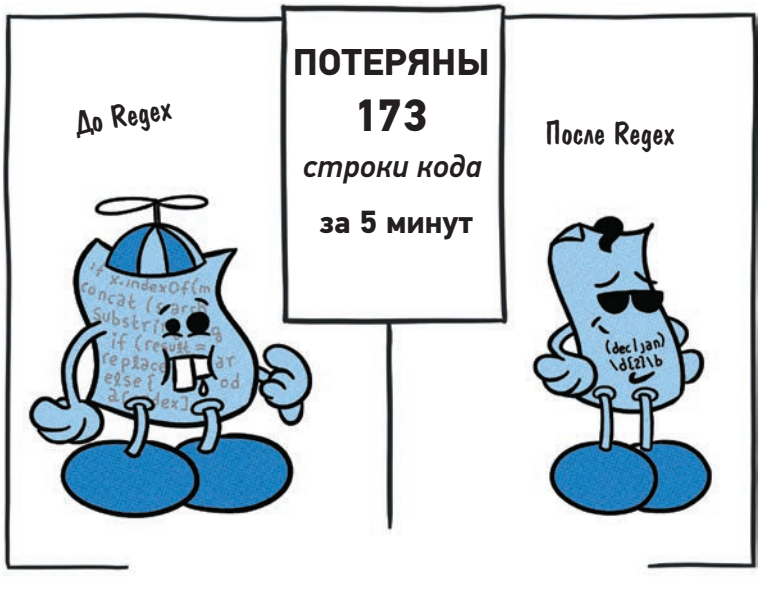


Рис. 6.2. «Раньше я был тормознутым толстячком кода. А теперь я строен и точен и легко нахожу даты!»

6.1. СООТВЕТСТВИЕ

Регулярное выражение — это своего рода поисковый запрос на стероидах. Представим следующее:

```
District 1, Paris
```

Это простое регулярное выражение, и оно работает как обычный поиск. Движок **регулярных выражений** — это программа, которая интерпретирует это выражение, применяет его к какому-то тексту и сообщает о совпадении, если строка «Район 1, Париж» найдена. Есть небольшие различия в синтаксисе, функциях и поведении различных движков. Суровые айтишники называют их **разновидностями** (flavors).

Большинство разновидностей по умолчанию чувствительны к регистру, поэтому «район 1, Париж» не приведет к совпадению. К счастью, они также позволяют настроить способ обработки данных с помощью **флагов**. Например, флаг без учета регистра (обычно обозначаемый `i`) позволит нам найти «район 1, Париж» в нашем выражении из заглавных букв[1].

Настоятельно рекомендуем попробовать все регулярные выражения из этой главы. Откройте http://code.energy/regex и увидите поле для регулярного выражения и текстовое поле для тестового ввода. Оно покажет, где выражение соответствует входным данным. Вы также можете установить флаги. Попробуйте и посмотрите, как флаг «без учета регистра» влияет на совпадения!

ТОЧКА

Предположим, надо найти любую из этих фраз:

- «Район 1, Париж»;
- «Район 2, Париж»;
- «Район 3, Париж»;
- «Район 4, Париж».

[1] Способ установки флагов варьируется от разновидности к разновидности. Иногда флаги добавляются после слеша, например: район 1, Париж/i.

Чтобы не выполнять четыре поиска, можно позволить десятому символу одного поиска соответствовать чему угодно:

```
District ., Paris
```

В регулярных выражениях **точка (.)** имеет особое значение: она соответствует любому отдельному символу. Выражение будет даже соответствовать фразам, которые нас не интересуют, таким как «Район 0, Париж» и «Район !, Париж». Тем не менее точка обеспечивает быстрый способ найти текст, который слабо соответствует вашему шаблону. Рассмотрим такое выражение:

```
к.......я
```

Он соответствует словам «кинология», «катавасия» и «конверсия», а поскольку точка также соответствует символу пробела, то даже «кот иудея» приведет к совпадению. Обратите внимание, что границы между словами здесь не рассматриваются, поэтому «станция союз» будет найдена в «Инстанция союза художников». Давайте посмотрим, как все это позволяет нам делать гораздо больше, чем просто статический поиск.

> **ДОМЕННОЕ ИМЯ** Вы помните электронное письмо с упоминанием сайта, который заканчивается на `mil` и код страны. Больше вы ничего не помните, так что это может быть `mil.br/`, `eng.mil.ru/`, `mil.be/frite` и т. д. Как же найти такой домен?

Все домены верхнего уровня с кодом страны имеют два символа. Точка соответствует любому символу, даже фактической точке. Следующее выражение найдет все возможные совпадения:

```
mil../
```

Точка в основном используется для быстрой грязной работы, и обычно с ее помощью находится гораздо больше, чем нужно. Теперь посмотрим, как *очень* точно использовать регулярные выражения.

МНОЖЕСТВО

Член, записанный в скобках `[•••]`, называется **множеством**. Множество работает как точка, хотя он будет соответствовать только одному из сим-

волов, содержащихся в скобках. Замените точку множеством, и поиск парижских районов станет точным:

```
District [1234], Paris
```

Это будет соответствовать только четырем случаям, в которых мы и заинтересованы. Такие диапазоны символов часто используются во множествах, и поэтому существует сокращенная запись, делающая их компактными и удобными для чтения:

- `[1-4]` эквивалентно `[1234]`;
- `[h-p]` эквивалентно `[hijklmnop]`.

Это улучшает решение проблемы с доменным именем  :

```
mil.[a-z][a-z]/
```

Множество может содержать несколько диапазонов. Например, следующее регулярное выражение соответствует символу, который находится либо в диапазоне 0–9, либо в диапазоне A–F, то есть он соответствует шестнадцатеричной цифре[1]:

```
[A-F0-9]
```

Добавив к `#` шесть из этих множеств, мы можем сопоставлять шестнадцатеричные цветовые коды[2], такие как **#E67F31**:

```
#[A-F0-9][A-F0-9][A-F0-9][A-F0-9][A-F0-9][A-F0-9]
```

Некоторые множества очень распространены в регулярных выражениях. Ленивые программисты задали **сокращения** для сопоставления с цифрой и с буквенно-цифровым символом:

---

[1] В шестнадцатеричном формате одна цифра — это значение от 0 до 15. Поскольку у нас закончились арабские цифры, буквы a–f используются в качестве цифр для значений от 10 до 15. Дополнительные сведения о шестнадцатеричных цифрах см. в приложении I.

[2] Символ #, за которым следуют три пары шестнадцатеричных цифр, является распространенным способом кодирования цветов. Каждая пара указывает количество красного, зеленого или синего цвета в составе итогового. Например, #FF0000 указывает на максимально красный цвет, без зеленого или синего компонентов. Поиск в Google такой строки немедленно вернет соответствующий цвет!

- `\d` эквивалентно `[0-9]`;
- `\w` эквивалентно `[A-Za-z0-9_]`[1].

Чтобы задать сопоставление с десятизначным номером телефона (например, 307-555-0177), нам пришлось бы повторить `[0-9]` десять раз. Повторение сокращения более компактно:

```
\d\d\d-\d\d\d-\d\d\d\d
```

Вскоре мы узнаем о еще лучших способах повторения множеств.

ОБРАТНОЕ МНОЖЕСТВО

Иногда хочется найти соответствие *почти* любому символу. Вместо того чтобы перечислять все допустимые символы множества, мы перечисляем символы, которые *не* принимаются в **обратном множестве**. Это противоположность множеству в том смысле, что оно находит соответствие любому символу, который *не* указан в скобках. Выглядит как обычное множество, но с кареткой после первой скобки: `[^•••]`.

> **ОБМЕН ВАЛЮТ**  У вас есть список валют, каждая из которых записана в виде трехбуквенного кода, например JPY (¥), USD ($) и GPB (£). Некоторые из них являются специальными валютами, такими как XBT (биткоин) и XAU (золото). Специальные всегда начинаются с X. Перечислите все неспециальные коды валют.

Используя множества, мы должны были бы перечислить каждую букву:

```
[A-WY-Z][A-Z][A-Z]
```

Сопоставление всего, что не содержит *X* в качестве первого символа, проще:

```
[^X][A-Z][A-Z]
```

---

[1] Примечательно, что \w включает в себя подчеркивание. Подчеркивание разрешено в именах переменных исходного кода, поэтому программисты могут задавать такие переменные, как beer_count. Добавление подчеркивания в \w облегчает сопоставление имен переменных в исходных файлах.

Обратите внимание, что выражение также будет соответствовать `0AA`, `.AA` или `AA`, так как обратный список дает соответствие с *любым символом*, который не находится в его скобках. Будет ли `[^X][A-Z][A-Z]` соответствовать только фактическим кодам валют, зависит от наших входных данных. Обратные множества также полезны для отлавливания того, что не нужно.

> **КОРРЕКТУРА** 🔎 При вычитке этой книги мы должны были убедиться, что за каждым знаком препинания следует пробел. Как это отследить?

Если есть что-то, кроме пробела после знака препинания, можно найти соответствие и проверить:

```
[.,:;?!][^ ]
```

Вы заметили, что мы используем символ «точка» внутри множества и что он соответствует фактической точке? Это потому, что символы внутри множества теряют свои сверхспособности. А теперь научимся проделывать подобное за пределами множеств.

СПЕЦИАЛЬНЫЕ СИМВОЛЫ

Чтобы специальный символ[1] утратил свою функцию в регулярном выражении, мы используем `\`, **escape-символ**. Например, `\.` соответствует фактическому символу «точка» — это то же самое, что и `[.]`. Проблема доменных имен 🌐 теперь имеет еще более точное решение:

```
mil\.[a-z][a-z]/
```

Другие символы также могут быть выражены с помощью escape-последовательности. Это удобно при сопоставлении непечатаемых символов:

- `\t` соответствует вкладке;
- `\n` соответствует новой строке;
- `\r` соответствует возврату каретки (полезно в Windows);
- `\f` соответствует разрыву страницы;
- `\s` эквивалентно `[ \t\r\n\f]`.

[1] Вот эти ребята: . ? * + ^ $ | [ ] { } ( ) \.

Сокращение `\s` называется **пробелом**. Обратите внимание, что он включает в себя обычный символ пробела. Теперь можно улучшить корректуру 🔎, чтобы не тратить время на возврат знаков препинания, за которыми следуют новые строки и разрывы страниц:

```
[.,:;?!][^\s]
```

С помощью точек, множеств, обратных множеств, специальных символов и сокращений теперь можно с большой гибкостью сопоставлять отдельные символы. Но что, если мы не знаем точно, сколько символов мы ищем?

6.2. КВАНТИФИКАТОРЫ

В выражении сопоставления на с. 256, соответствующем десятизначным телефонным номерам, повторяющиеся `\d` выглядят громоздко и трудночитаемы. Для упрощения используем **квантификаторы**, которые изменяют количество символов, которым будет соответствовать `\d`.

ФИГУРНЫЕ СКОБКИ

Можно точно указать, сколько раз символ или набор должны совпасть, используя **фигурные скобки**. Например, выражение, соответствующее десятизначным телефонным номерам, можно переписать так:

```
\d{3}-\d{3}-\d{4}
```

Это выражение эквивалентно `\d\d\d-\d\d\d-\d\d\d\d`, только выглядит гораздо лучше. Оно гласит: «Сопоставьте три цифры, затем тире, затем три цифры, затем тире, затем четыре цифры». Квантификатор также можно использовать для улучшения выражения, которое ищет шестнадцатеричные цветовые коды:

```
#[A-F0-9]{6}
```

Компоненты выражения по умолчанию совпадают ровно один раз, поэтому `{1}` не имеет никакого эффекта: `B{1}e{2}r{1}` — это то же самое, что `Beer`. Кроме того, помимо данных *фиксированных* требований к соответствию, мы также можем указывать и *диапазон*. Можно указать минимальное или максимальное количество совпадений:

- точно *n* раз: `{n}`;
- не менее *n* и не более *m* раз: `{n,m}`;
- не менее *n* раз, без верхнего предела: `{n,}`.

Обратите внимание, что *n* может быть равно нулю, что фактически делает пиво (beer) необязательным.

> **БЕСПОРЯДОЧНЫЕ НОМЕРА** Бразильские телефонные номера имеют различную длину. Код города состоит из двух цифр, а местный номер может состоять из восьми или девяти цифр. Кроме того, дефис перед последними четырьмя цифрами иногда опускается. Как же нам отыскать эти номера в документе?

Можно использовать один квантификатор диапазона, чтобы указать количество допустимых цифр, а другой — чтобы сделать второй символ тире необязательным:

```
\d{2}-\d{4,5}-{0,1}\d{4}
```

Это выражение может показаться пугающим, поэтому расшифруем его шаг за шагом:

- `\d{2}` соответствует двум цифрам;
- `-` соответствует одному дефису;
- `\d{4,5}` соответствует четырем или пяти цифрам;
- `-{0,1}` соответствует либо отсутствию дефиса, либо одному дефису;
- `\d{4}` соответствует четырем цифрам.

ВОПРОС

Квантификатор, который мы использовали, чтобы сделать дефис необязательным в нашем последнем выражении, настолько распространен, что существует его короткая запись. Можно заменить `{0,1}` на `?`. Выражение поиска номера телефона теперь выглядит так:

```
\d{2}-\d{4,5}-?\d{4}
```

Знак `?` называется **вопросом**. Не имеет значения, встречается ли символ, который непосредственно предшествует `?`, так как совпадение происходит независимо от этого. Этот параметр полезен при поиске адресов, которые могут использовать или не использовать шифрование[1]:

```
https?://code\.energy
```

В данном случае найдется как http://code.energy, так и https://code.energy.

ПЛЮС

Плюс (`+`) — еще одно полезное сокращение. Как и `{1,}`, он указывает, что что-то должно встретиться хотя бы один раз, без верхнего предела. Например:

```
du+de
```

Выражение будет соответствовать *dude*, *duude*, *duuude* и т. д. Если нужно отловить ссылки на любой домен .com[2]:

```
https?://[a-z0-9-]+\.com
```

ЗВЕЗДОЧКА

Самым мощным квантификатором является **звездочка** (`*`). Она эквивалентна `{0,}` и сочетает в себе эффект вопроса и плюса. Она указывает на необязательный термин, который может встречаться неограниченное количество раз. Например:

```
yea*h
```

Это будет соответствовать *yeh*, *yeah*, *yeaah*, *yeaaah* и т. д. Точка и звездочка (`.*`) будут соответствовать чему угодно: приниматься будет лю-

---

[1] Помните, что https используется в URL-адресе вместо http для указания на то, что к нему следует обращаться с помощью шифрования SSL.

[2] Это выражение также будет соответствовать некоторым недопустимым доменам, поскольку дефис не допускается в начале или в конце имени. Чтобы разобраться с этим, нужна продвинутая функция регулярных выражений под названием lookarrounds, но здесь мы не будем ее рассматривать.

бой символ любое количество раз. Если вы считаете, что ввод содержит слово *restaurant*, какой-то случайный текст, а затем слово *London*, можно попытаться найти:

```
restaurant.*London
```

`.*` позаботится о любом тексте между двумя словами. Вы также можете использовать этот трюк, чтобы найти любую закавыченную цитату:

```
".*"
```

ЖАДНОСТЬ

Последнее регулярное выражение (`".*"`) содержит проблему. Протестируйте его со следующими входными данными:

```
Avaritia porro "hominem" ad quod "vis maleficium" impellit.
```

Квантификаторы **жадные**: они стараются потреблять как можно больше символов[1]. Точка и звездочка будут проходить от первой кавычки нашего ввода до последней, возвращая:

```
"hominem" ad quod "vis maleficium"
```

Чтобы соответствовать только *hominem*, замените точку обратным множеством:

```
"[^"]*"
```

По умолчанию движки регулярных выражений вернут их первое совпадение. Для того чтобы получить несколько совпадений, имеется **глобальный флаг**, обычно обозначаемый **g**. Он сделает наш последний поиск совпадающим как с *hominem*, так и с *vis maleficium*. Обратите внимание, что символ не может быть частью двух совпадений, поэтому выражение *не* вернет *ad quod*.

[1] Многие разновидности также содержат ленивые квантификаторы в качестве альтернативы. Ленивые квантификаторы будут потреблять как можно меньше символов.

6.3. ПРИВЯЗКИ

До сих пор наши регулярные выражения могли совпадать с символами и фразами в любом месте ввода. **Привязки** точно контролируют, *где именно* разрешены совпадения. Привязки не потребляют символы, скорее они ограничивают возможные позиции для совпадений.

КАРЕТКА

При использовании за пределами скобок **каретка** (^) ограничивает совпадения началом строки. Рассмотрим такое выражение:

```
^Once upon a time
```

И будет соответствовать только фрагменту строки, с которой начинается сказка. Многие программисты используют последовательность дефисов (`----`) для создания горизонтального разделителя в текстовых файлах. Их можно найти с помощью такого выражения:

```
^-+
```

Оно соответствует последовательности дефисов, но только в том случае, если с нее начинается строка.

ДОЛЛАР

Доллар привязывает совпадение к концу строки. Возьмем такое выражение:

```
happily ever after\.$
```

Оно будет соответствовать только строке, которая заканчивается на `happily`. Вы можете использовать обе привязки в одном и том же выражении. Это выражение ищет сказку[1]:

```
^Once upon a time.*happily ever after\.$
```

[1] В большинстве разновидностей точка будет соответствовать символу новой строки только в том случае, если применяется флаг s. Используйте этот флаг, чтобы найти сказку, состоящую из нескольких абзацев.

Мы можем обновить наше выражение, которое находит разделители, составленные из тире, чтобы находились строки, которые содержат только дефис:

```
^-+$
```

Догадаетесь, как найти пустую строку? Подсказка: вам просто нужны две привязки.

> **Чистый код** 🧹 В большинстве языков программирования окончание строки кода пробелом не одобряется, поскольку оно не служит никакой цели. Как потом отыскать эти «завершающие пробелы»?

Найти их просто — нужно искать один или несколько пробелов в конце строки:

```
+$
```

ФЛАГ МНОГОСТРОЧНОСТИ В некоторых разновидностях каретка и доллар относятся к началу и концу всего ввода: они игнорируют разрывы строк. Чтобы изменить такое поведение, существует флаг **многострочности**, обычно обозначаемый `m`. Он заставляет каретки и доллары распознавать символы новых строк.

ГРАНИЦА

Некоторые слова встроены в другие. Например, слово «здоровый» найдется в слове «нездоровый», а поиск «опера» вернет «телеоператор». Это так называемые **коала-слова**[1]. Выполните следующий поиск:

```
art             Martial arts impart artists as art.
```

Но что, если мы просто ищем слово *art*? Привязка **границы** (`\b`) решает эту проблему, ограничивая совпадения началом или концом слова:

```
\bart       Martial arts impart artists as art.

art\b       Martial arts impart artists as art.
```

[1] Мы выдумали этот термин. 

```
\bart\b    Martial arts impart artists as art.

\bart?\b   Martial arts impart artists as art.
```

В последнем выражении форма множественного числа является общепринятым коала-словом. Если нас интересуют только слова, которые начинаются с `a` и заканчиваются на `s`, то:

```
\ba[a-z]+s\b  Martial arts impart artists as art.

\ba[a-z]*s\b  Martial arts impart artists as art.
```

Не позволяйте коалам сбить вас с толку. Привязывайте свои выражения.

6.4. ГРУППЫ

До сих пор мы видели, как квантификаторы работают только с одним предшествующим символом. **Группы** позволяют квантификторам работать с последовательностью символов или даже с целым выражением. Они создаются с помощью круглых скобок: `(•••)`. Например:

```
(meta-)*analysis
```

Здесь часть выражения `meta-` может находиться бесконечное число раз. Выражение находит «analysis», « meta-analysis», «meta-meta-analysis» и т. д.

> **БОЛЬШИЕ ЧИСЛА** 1 2 3 4 Вам нужно найти большие числа в длинном отчете. Запятые разделяют тысячи, а некоторые числа имеют дробные части. Как получить все числа?

Мы могли бы поддаться лени и попробовать:

```
\b[0-9,.]+\b
```

Однако будут найдены соответствия и с другими элементами, такими как 10.12.1815. Для правильного извлечения чисел, в которых используются точки и запятые, мы можем применять группы с квантификаторами:

```
\b\d{1,3}(,\d{3})*(\.\d+)?\b
```

Разберем это выражение слева направо:

```
\b
```

Совпадение должно начинаться с границы слова.

```
\d{1,3}
```

Максимум три крайние левые цифры.

```
(,\d{3})*
```

Это может быть запятая, за которой следуют три цифры[1].

```
(\.\d+)?
```

Это может быть точка, за которой следует хотя бы одна цифра.

```
\b
```

Совпадение должно заканчиваться на границе слова.

ЗАХВАТ ГРУПП

Группы также называются **захватом групп**: регулярное выражение сохраняет текст, которому они соответствуют, в качестве внутренних переменных. Эти переменные часто используются для создания сложных схем поиска и замены.

> **Кавычки-елочки**  Переводчик передал вам текст, который включает цитаты, обрамленные прямыми кавычками, "вот так". Он просит вас заменить их *кавычками-елочками*, чтобы цитаты выглядели «вот так».

Вы не можете напрямую найти и заменить символ ". Он должен стать либо «, либо », в зависимости от его положения. Если мы добавим группу в наше предыдущее регулярное выражение, которое находит кавычки, текст внутри кавычек будет сохранен во внутренней переменной, называемой `\1`. Мы можем приказать движку регулярных выражений

[1] Может повторяться неограниченное количество раз, чтобы соответствовать тысячам, миллионам, миллиардам и т. д.

заменить соответствующую строку новым текстом, использующим эту переменную.

Найти:

```
"([^"]*)"
```

Заменить:

```
"  \1  "
```

Если регулярное выражение имеет несколько групп, создается больше переменных захвата. Чтобы переформатировать телефонные номера США, разделенные пробелами, используя скобки и дефисы, можно сделать следующее.

Найти:

```
(\d{3}) (\d{3}) (\d{4})
```

Заменить:

```
(\1) \2-\3
```

В этом случае `\1` обозначает текст, который был найден внутри первой группы, `\2` — текст из второй группы и т. д. Всякий раз, когда нужно выполнить операцию поиска и замены, которая не является статической, не забывайте использовать группы!

ЧЕРЕДОВАНИЕ

Когда мы хотим найти соответствие одному из нескольких выражений, то используем **чередование**, обозначаемое `|`. Оно работает аналогично логическому ИЛИ. Например, для поиска телефонных номеров только из Пуэрто-Рико нужен один из этих двух кодов города:

```
(787|939)-\d{3}-\d{4}
```

Обратите внимание, что мы ограничили чередование внутри группы, потому что оно имеет наименьший приоритет. Другими словами, удаление скобок:

```
787|939-\d{3}-\d{4}
```

эквивалентно:

```
(787)|(939-\d{3}-\d{4})
```

Это неправильно: машина поймет это как «совпадение с 787 или десятизначным номером телефона, начинающимся с 939». Всегда будьте осторожны при использовании чередования вне группы. Вот другой пример.

> **СУМЧАТЫЕ ПАРЫ**  Рейнджер в австралийской глубинке изучает поведение кенгуру при размножении. Ее агентство следит за их популяцией и датирует свою статистику в формате Mon YY. Она знает, что пик рождаемости у кенгуру приходится на летние месяцы. Как ей искать данные для такой летней статистики?

Поиск должен быть ограничен летними месяцами Австралии: декабрем, январем и февралем:

```
(Dec|Jan|Feb) \d\d
```

Если наша рейнджер исследует одно конкретное лето, ей придется опустить скобки и указать годы:

```
Dec 92|Jan 93|Feb 93
```

Если использовать две группы `(Dec|Jan|Feb) (92|93)`, поиск будет соответствовать месяцам, охватывающим три лета: ~~Jan 92~~, ~~Feb 92~~, Dec 92, Jan 93, Feb 93, ~~Dec 93~~.

РЕЗЮМЕ

В этой главе вы узнали, как лаконично отыскивать сложные шаблоны. Узнали, что регулярные выражения строятся из комбинации символов, некоторые из которых представляют сами себя, а некоторые — специальные функции. Эти инструменты нужно использовать, когда:

- требуется убедиться, что ввод соответствует ожидаемому формату (например, номер телефона, дата, IP-адрес, номер банковской карты);

- вы не знаете точной последовательности символов, которые ищете, но знаете, как она должна выглядеть;
- нужно выполнить сложные операции поиска и замены.

Если вы работаете с общим шаблоном, таким как IP-адрес или номер телефона, то можете найти готовые к использованию регулярные выражения в Сети. Есть куча библиотек регулярных выражений и туториалов, которые помогут найти распространенные шаблоны.

При разработке собственного выражения начните с чего-то простого, что в целом соответствует тому, что вы ищете. После этого затем уточняйте свое выражение, чтобы оно становилось все более и более конкретным.

Почти все языки программирования и инструменты поддерживают регулярные выражения, но будьте осторожны: есть множество тонкостей в конкретных разновидностях. Некоторые интерпретируют круглые скобки как литеральные символы и, следовательно, требуют, чтобы вы набрали `\(•••\)` вместо `(•••)` для создания группы. Некоторые старые движки не поддерживают определенные привязки, такие как граница слова. И как же искать японские иероглифы? Общие концепции, которые вы усвоили из этой главы, помогут сориентироваться в справочных материалах по регулярным выражениям.

Установите расширение для регулярных выражений для браузера, чтобы протестировать его на веб-страницах. Посмотрите, как использовать регулярные выражения в редакторе исходного кода. Эту книгу мы писали в редакторе Vim, который выполняет *все* поиски с помощью регулярных выражений. Если есть способ сделать регулярные выражения режимом поиска по умолчанию в редакторе, сделайте это. Так вы сможете практиковаться и сэкономите время в долгосрочной перспективе.

Даже если вы не расследуете особо важные преступления (рис. 6.3), быстрый доступ к конкретной информации важен для эффективной работы. Чем лучше вы владеете регулярными выражениями, тем быстрее сможете получить нужные данные, изучить их и сэкономить часы ценного времени на написание кода.

Теперь, когда вы научились находить точно определенные шаблоны текста, давайте выясним, как изучать и исследовать основные идеи сложных шаблонов, которые могут скрываться в больших коллекциях данных.

Рис. 6.3. «Регулярные выражения», любезно предоставленные http://xkcd.com

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

- *Фридл Д.* Регулярные выражения. — СПб.: Питер, 2018.
- *Goyvaerts J., Levithan S.* Regular Expressions Cookbook. https://code.energy/goyvaerts.

ПРИЛОЖЕНИЯ

I. ОСНОВАНИЯ СИСТЕМ СЧИСЛЕНИЯ

Вычисления могут быть сведены к работе с числами, так как информация выражается именно в них. Буквы могут быть сопоставлены с числами, поэтому текст может быть записан численно. Цвета — это сочетание интенсивности красного, синего и зеленого, которые тоже могут быть заданы в виде чисел. Изображения могут быть составлены из мозаики цветных квадратов, поэтому их можно выразить в виде чисел.

Архаичные системы счисления (например, римские цифры: I, II, III...) составляют числа из сумм цифр. Используемая сегодня система счисления также основана на суммах цифр, но значение каждой цифры в позиции i умножается на d в степени i, где d — число отдельных цифр. Мы называем d **основанием**. Обычно мы используем $d = 10$, потому что у нас десять пальцев, но система работает для любого основания d (рис. П.1).

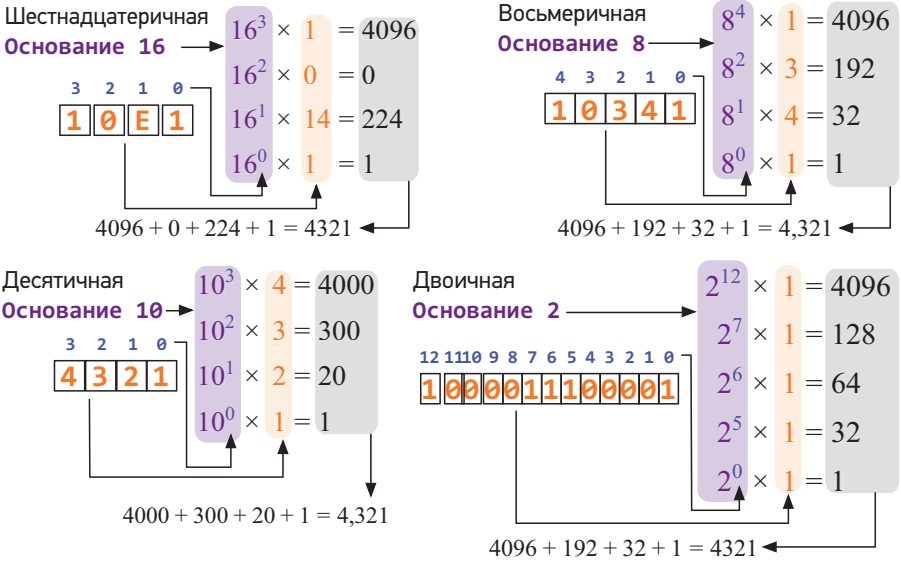


Рис. П.1. Число 4321 при различных основаниях

II. ВЗЛОМ ШИФРА СДВИГА

В задаче «Тайный код 🤔» из раздела 3.1 мы попросили вас взломать шифр подстановки, которым зашифровано следующее сообщение:

```
MAXI KBVX HYLX VNKB MRBL XMXK GTEO BZBE TGVX VTKX
EXLL VHFF NGBV TMBH GLVH LMEB OXL
```

Простой подход состоит в том, чтобы перепробовать все возможные сдвиги на первых нескольких буквах зашифрованного текста:

```
LZWH JAUW    KYVG IZTV   JXUF HYSU         NBYJ LCWY
  ↑↑↑          ↑↑↑         ↑↑↑      ...      ↑↑↑
   1            2           3                25
MAXI KBVX    MAXI KBVX   MAXI KBVX         MAXI KBVX
```

Это вполне достойный способ найти решение, но с ручкой и бумагой мы можем проделать это немного быстрее. Прежде всего возьмите первую букву зашифрованного текста и запишите остальную часть алфавита в обратном порядке, вот так:

```
  1 2 3                                           25
M L K J I H G F E D C B A Z Y X W V U T S R Q P O N
```

Обратите внимание, что, дойдя до буквы `A`, мы продолжаем отсчет от последней буквы алфавита: `Z, Y, Z`. Давайте добавим дополнительные строки перевернутого алфавита, каждый раз начиная со следующей буквы зашифрованного текста:

```
  1 2 3                                           25
M L K J I H G F E D C B A Z Y X W V U T S R Q P O N
A Z Y X W V U T S R Q P O N M L K J I H G F E D C B

  1 2 3                                           25
M L K J I H G F E D C B A Z Y X W V U T S R Q P O N
A Z Y X W V U T S R Q P O N M L K J I H G F E D C B
X W V U T S R Q P O N M L K J I H G F E D C B A Z Y

  1 2 3                                           25
M L K J I H G F E D C B A Z Y X W V U T S R Q P O N
A Z Y X W V U T S R Q P O N M L K J I H G F E D C B
X W V U T S R Q P O N M L K J I H G F E D C B A Z Y
I H G F E D C B A Z Y X W V U T S R Q P O N M L K J
```

Видите — в столбце проявляется определенный шаблон? Давайте продолжим еще несколькими строками:

```
  1 2 3                             19          25
M L K J I H G F E D C B A Z Y X W V U T S R Q P O N
A Z Y X W V U T S R Q P O N M L K J I H G F E D C B
X W V U T S R Q P O N M L K J I H G F E D C B A Z Y
I H G F E D C B A Z Y X W V U T S R Q P O N M L K J
K J I H G F E D C B A Z Y X W V U T S R Q P O N M L
B A Z Y X W V U T S R Q P O N M L K J I H G F E D C
V U T S R Q P O N M L K J I H G F E D C B A Z Y X W
X W V U T S R Q P O N M L K J I H G F E D C B A Z Y
```

В 19-й колонке написано THE PRICE! Поскольку все остальные столбцы — это тарабарщина, мы понимаем, что нашли наш ключ:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
                      19
                      ↓↓↓
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
```

Теперь мы можем расшифровать весь текст:

```
THEP RICE OFSE CURI TYIS ETER NALV IGIL ANCE CARE
LESS COMM UNIC ATIO NSCO STLI VES
                      ↑↑↑
                      19
MAXI KBVX HYLX VNKB MRBL XMXK GTEO BZBE TGVX VTKX
EXLL VHFF NGBV TMBH GLVH LMEB OXL
```

Наконец, переставив пробелы и добавив знаки препинания, мы получаем:

```
THE PRICE OF SECURITY IS ETERNAL VIGILANCE.
CARELESS COMMUNICATIONS COST LIVES.
```

III. ВЗЛОМ ШИФРА ПОДСТАНОВКИ

В задаче «Пустая шкатулка  » из раздела 3.1 мы бросили вам вызов, попросив взломать шифр подстановки, которым зашифровано следующее сообщение:

```
DUA KVYBVHA PVJ OAZQMASAO DW CWES PQLA KASJWFVZZC.
AMASCDUQFH QJ VZZ SQHUD PQDU DUA LVGQZC. PA PQJU
CWE JEYYAJJ. HSAADQFHJ LSWG DUA YWGSVOAJ.
```

Начнем с самого базового наблюдения, а именно частотного анализа: в английском языке наиболее распространенной буквой является E, а наиболее распространенной парой последовательных букв — TH. Поскольку наиболее распространенной буквой зашифрованного текста является A, а наиболее распространенной парой букв является DU, мы можем попробовать сопоставить E → A и T,H → D,U:

```
THE KVYBVHE PVJ OEZQMESEO TW CWES PQLE KESJWFVZZC.
EMESCTHQFH QJ VZZ SQHHT PQTH THE LVGQZC. PE PQJH
CWE JEYYEJJ. HSEETQFHJ LSWG THE YWGSVOEJ.
```

Наша работа облегчается тем фактом, что зашифрованный текст сохранил пробелы и знаки препинания открытого текста. Мы сразу видим, что наше предварительное отображение T,H,E → D,U,A имеет хорошие шансы быть правильным, так как распространенное слово THE появляется целых три раза!

Данное предположение дает предложение, которое начинается с PE, и это может быть be, he или we. У нас уже есть отображение для H, поэтому мы пробуем B → P и W → P. Первое заводит нас в тупик: предложение тогда начиналось бы с фразы BE BQJH, которая могла бы иметь смысл только в случае, когда она означала be both, тем самым создавая конфликт с нашим предыдущим отображением T → D. Поэтому мы оставляем W → P:

```
THE KVYBVHE WVJ OEZQMESEO TW CWES WQLE KESJWFVZZC.
EMESCTHQFH QJ VZZ SQHHT WQTH THE LVGQZC. WE WQJH
CWE JEYYEJJ. HSEETQFHJ LSWG THE YWGSVOEJ.
```

Посмотрим на слова с наименьшим количеством пропущенных букв. Например, WQTH сразу дает нам I → Q. С помощью этого нового отображения WQJH превращается в WIJH, что, в свою очередь, дает нам S → J:

```
THE KVYBVHE WVS OEZIMESEO TW CWES WILE KESSWFVZZC.
EMESCTHIFH IS VZZ SIHHT WITH THE LVGIZC. WE WISH
CWE SEYYESS. HSEETIFHS LSWG THE YWGSVOES.
```

И снова мы ищем слова с наименьшим количеством пропущенных букв: WVS дает нам A → V, а TW дает нам O → W. Кроме того, тот факт, что неизвестный символ появляется дважды в одном и том же слове, также может быть ключом: SEYYESS дает нам U,C → E,Y, а HSEETIFHS дает нам G,R,N → H,S,F:

```
THE KACBAGE WAS OEZIMEREO TO COUR WILE KERSWNAZZC.
EMERCTHING IS AZZ RIGHT WITH THE LAGIZC. WE WISH
COU SUCCESS. GREETINGS LROG THE COGRAOES.
```

Чем ближе мы подходим к решению, тем быстрее продвигаемся вперед! `COUR` и `COU` дают нам `Y → C`, `EMERYTHING` дает `V → M` и `EVERYTHING IS AZZ RIGHT` дает нам `L → Z`:

```
THE KACBAGE WAS OELIVEREO TO YOUR WILE KERSWNALLY.
EVERYTHING IS ALL RIGHT WITH THE LAGILY. WE WISH
YOU SUCCESS. GREETINGS LROG THE COGRAOES.
```

Мы почти закончили! Отсутствующие сопоставления могут быть быстро разрешены в виде `P,K,D,F,O,M → K,B,O,L,W,G`, и открытый текст больше не тайна:

```
THE PACKAGE WAS DELIVERED TO YOUR WIFE PERSONALLY.
EVERYTHING IS ALL RIGHT WITH THE FAMILY. WE WISH
YOU SUCCESS. GREETINGS FROM THE COMRADES.
```

Наконец, можно записать все, что нам известно о ключе шифрования, на случай, если мы найдем новый зашифрованный текст, который тоже его использует. Недостающие буквы — это те, которые отсутствуют в сообщении:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
E K Y T U N M G _ S P F V _ D W I _ R _ H A O _ C L
```

Такой простой шифр подстановки не очень безопасен — мы смогли взломать его за несколько простых шагов. Хотя эти шифры более сложны, когда положение пробелов неизвестно, компьютеры могут быстро работать с ними независимо от этого факта. Чтобы попробовать, переходите на http://code.energy/substitution.

IV. ОЦЕНКА КЛАССИФИКАТОРОВ

В главе 6 мы узнали о моделях прогнозирования: алгоритмах, умеющих создавать серию предсказаний, которые мы называем y, и данных, связанных с каждым предсказанием, которые мы называем X. Существует два типа моделей. Регрессоры делают численные прогнозы, в то время как классификаторы предсказывают результат, разделенный на раз-

меченные группы. Мы уже научились оценивать, насколько хорошо регрессор делает прогнозы. Теперь узнаем, как оценивать классификаторы.

Поскольку классификаторы предсказывают метки, начнем с простого: предположим, что y состоит из двоичных данных. Поскольку он имеет только *две метки*, все, что нам нужно, — это ответ «да» или «нет» для каждой строки. Это называется **бинарной классификацией**. Примерами бинарной классификации являются предсказания того, разовьется ли у пациента диабет или является ли покупка с помощью кредитной карты мошеннической.

Наивный подход заключается в том, чтобы оценить процент правильных прогнозов, называемый **точностью**. Однако это часто самый ужасный способ оценивать, насколько хорошо работает модель. Посмотрим, как обстоят дела.

> **ПОИСК МОШЕННИКА** 💳 Вы программист в банке и пытаетесь предотвратить мошенничество с кредитными картами. Вам нужно создать модель, которая предсказывает, является ли платеж мошенническим, основываясь на деталях транзакции. Вы обучили две модели — *спокойную* 😎 и *параноидальную* 😱 — предсказывать мошенничество с использованием данных о тысяче транзакций, из которых о десяти известно, что это случаи мошенничества. Теперь вы хотите сравнить их работу с неподготовленной *ленивой* моделью 😴, которая случайным образом помечает 1 % транзакций как мошеннические. Вот как сработала каждая из трех моделей.
>
> 😴 Не нашел мошенничества, неправильно назвал 10.
>
> 😎 Нашел 8 случаев мошенничества, неправильно назвал еще 19.
>
> 😱 Нашел все случаи мошенничества, неправильно назвал еще 87.
>
> Какая из моделей наиболее точна?

Параноидальная модель не пропустила ни одного случая мошенничества, поэтому сделала 0 + 87 = 87 ошибок и дала 913 правильных прогнозов, что дало точность 91,3 %.

Спокойная модель пропустила два случая мошенничества, поэтому сделала 2 + 19 = 21 ошибку и дала 979 правильных прогнозов, что обеспечило

точность 97,9 %. С другой стороны, ленивая модель пропустила 10 случаев мошенничества, поэтому сделала 10 + 10 = 20 ошибок и дала 980 правильных прогнозов, что дало точность 98 %. Ленивая модель оказалась самой точной, хотя и бесполезной!

Чтобы лучше оценить классификаторы, мы должны увидеть, *как именно* они делают верные и неверные прогнозы. С этой целью мы создадим **таблицу ошибок**, которая подсчитывает верные и неверные прогнозы для каждой метки (рис. П.2).

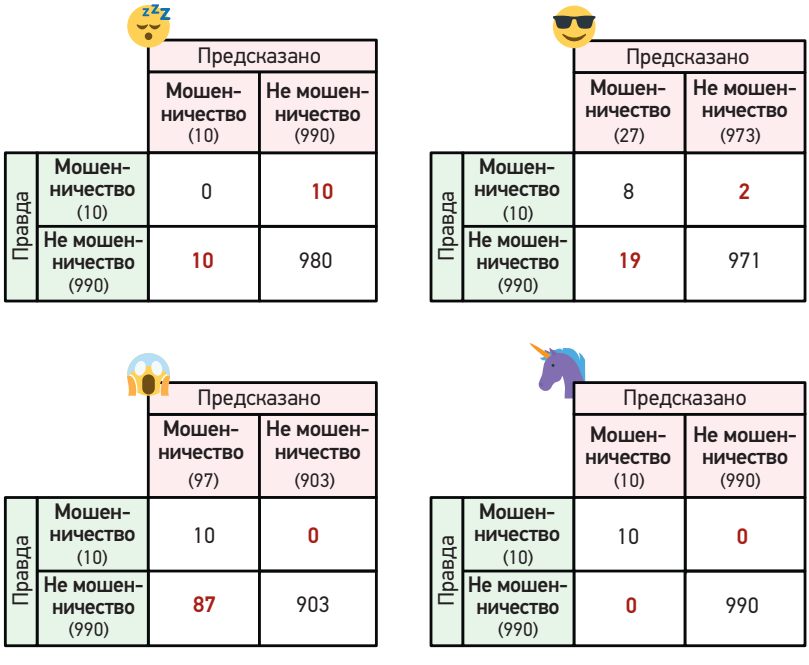


Рис. П.2. Таблицы ошибок для ленивого, спокойного и параноидального классификаторов. Классификатор-единорог со 100%-ной точностью приведен для сравнения

Предсказание неверно, если предсказанная и истинная метки отличаются, как это показано красным цветом на рис. П.2. Обратите внимание, что ранее упомянутая точность может быть получена для каждой таблицы путем деления суммы черных ячеек на 1000. Теперь посмотрим, какие еще показатели мы можем использовать, чтобы таблицы ошибок обрели смысл.

ЧУВСТВИТЕЛЬНОСТЬ Процент мошенничества, который каждая модель может идентифицировать, является показателем **чувствительности**. Например:

- найдено 0 из 10 случаев мошенничества, чувствительность = 0 %;
- найдено 8 из 10 случаев мошенничества, чувствительность = 80 %;
- найдено 10 из 10 случаев мошенничества, чувствительность = 100 %.

Здесь наилучший показатель чувствительности получает параноидальная модель, поскольку она смогла идентифицировать полный список мошенников. С другой стороны, спокойная модель обнаружила большинство случаев мошенничества, но список был неполным. Ленивая модель была наименее чувствительной, так как не нашла вообще ничего достойного внимания!

ТОЧНОСТЬ Процент правильных прогнозов модели касательно мошенничества — **это показатель точности**. Например:

- права в 0 из 10 случаев предсказанных мошенничеств, точность = 0 %;
- права в 8 из 27 случаев предсказанных мошенничеств, точность ≈ 29,6 %;
- права в 10 из 97 случаев предсказанных мошенничеств, точность ≈ 10,3 %.

Наилучшая оценка точности была получена с помощью спокойной модели, поскольку почти треть прогнозируемых мошенничеств оказались фактическими мошенничествами. Параноидальная модель менее точна: только 10 % предсказаний полезны. Наконец, ленивая модель оказалась наименее точной, поскольку все ее предсказания о мошенничестве бесполезны!

Теперь стало окончательно ясно, что ленивая модель ужасна. Но может оказаться трудным выбор между спокойной и параноидальной моделями, поскольку одна из них обладает лучшей чувствительностью, а другая — большей точностью. К счастью, есть широко используемая метрика, которая решает эту проблему.

ОЦЕНКА F_1 Хорошая оценка прогностической эффективности модели учитывает как чувствительность, так и точность. Если любая из них равна нулю, модель не обладает предсказательной силой, поэтому ее итоговая оценка должна быть равна нулю. Чтобы достичь этого, ученые рассчитывают специальный вид среднего значения (называемый *гармоническим средним*) по показателям чувствительности и точности. Они называют это оценкой $\mathbf{F}_1$:

$$F_1 = 2 \times \frac{\text{точность} \times \text{чувствительность}}{\text{точность} + \text{чувствительность}}.$$

Оценка F_1 колеблется от 0 до 1 (или 100 %). Использовав оценки точности и чувствительности в формуле для каждой модели, мы находим:

- $F_1 = 0$ %;
- $F_1 \approx 0{,}43 = 43$ %;
- $F_1 \approx 0{,}19 = 19$ %.

Согласно этой метрике, спокойная модель обладает более чем в два раза большей предсказательной силой, чем параноидальная. Это связано главным образом с неприемлемой точностью параноидальной модели.

ЛОЖНАЯ ТРЕВОГА С точки зрения клиента, есть еще один важный показатель: как часто его честные транзакции будут отклоняться? Процент действительно хороших транзакций, которые были неправильно классифицированы как мошенничество, называется коэффициентом **ложной тревоги**, также известным как коэффициент **ложного срабатывания**. Например:

- не права насчет 10 из 990 честных транзакций, FAR ≈ 1,0 %;
- не права насчет 19 из 990 честных транзакций, FAR ≈ 1,9 %;
- не права насчет 87 из 990 честных транзакций, FAR ≈ 8,8 %.

Здесь параноидальная модель отличается от других тем, что порождает множество случаев ложной тревоги: почти каждая десятая честная сделка классифицируется как мошенничество. В зависимости от приложения

для количественной оценки полезности модели вместо точности используется частота ложных срабатываний.

Оценивая свою модель, постарайтесь сформулировать то, что вы вычисляете словами, а не только с помощью математики. Для нашего примера мы могли бы сказать так: *«Параноидальная модель заблокировала 8,8 % транзакций честных клиентов»*. Это поможет выбрать, какие показатели имеют наибольшее значение.

КОМПРОМИСС В КЛАССИФИКАЦИИ

Функция прогнозирования (`predict`) классификатора выдает значение, которое, по его мнению, с наибольшей вероятностью примет каждая строка. Кроме того, большинство классификаторов могут предоставить свою оценку *вероятности* того, что строка принадлежит метке. Таким образом, можно изменить логику назначения меток модели, чтобы чаще предсказывать метку. Этот метод позволяет получить две, казалось бы, очень разные модели из одного источника (рис. П.3).

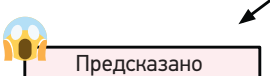
С установкой более низкого порога модель становится более чувствительной к признакам мошенничества, поэтому чувствительность повышается. С другой стороны, все больше покупок ошибочно классифицируются как мошеннические, что препятствует точности.

Поскольку точность, чувствительность и оценка F_1 изменяются в зависимости от порога, одного измерения этих оценок недостаточно для всесторонней оценки классификатора. В нашем небольшом примере более высокий порог дает $F_1 \approx 0{,}67$, в то время как более низкий порог — $F_1 \approx 0{,}57$. Но это не обязательно означает, что более высокий порог предпочтительнее.

Выберите метод оценки и пороговое значение классификатора в соответствии с реальными потребностями. Если польза от выявления мошенничества высока, а недостаток неправильной классификации валидных покупок невелик, более высокий порог может оказаться лучшим выбором.

Мы стараемся заранее оценить минимально приемлемые показатели чувствительности и точности. Если другие люди собираются использовать наш классификатор, они должны помочь определить границы компромисса для корректировки порога.

y	y_{pred}	
Правда Метка	Предполагаемое мошенничество Вероятность	0,50 порог	0,30 порог
Нет мошенничества	0,05	Нет мошенничества	Нет мошенничества
Мошенничество	0,55	Мошенничество	Мошенничество
Нет мошенничества	0,39	Нет мошенничества	**Мошенничество**
Нет мошенничества	0,48	Нет мошенничества	**Мошенничество**
Нет мошенничества	0,17	Нет мошенничества	Нет мошенничества
Нет мошенничества	0,33	Нет мошенничества	**Мошенничество**
Мошенничество	0,31	**Нет мошенничества**	Мошенничество



Правда	Предсказано: Мошенничество (1)	Предсказано: Нет мошенничества (6)
Мошенничество (2)	1	**1**
Нет мошенничества (5)	**0**	5

Правда	Предсказано: Мошенничество (3)	Предсказано: Нет мошенничества (4)
Мошенничество (10)	2	**0**
Нет мошенничества (990)	**3**	2

Рис. П.3. Классификация с использованием двух различных пороговых значений

КРИВЫЕ ROC

Радар раннего предупреждения был совершенно новой технологией в начале Второй мировой войны. Иногда радары улавливали сигналы, которые озадачивали их операторов. Они не всегда были уверены, исходят ли эти слабые сигналы от вражеских самолетов или же от паразитных радиопомех, но им нужно было решать, поднимать ли тревогу. Ложная тревога растрачивает драгоценные ресурсы, но игнорирование реальной угрозы может иметь разрушительные последствия.

Пытаясь определить, какие радиосигналы несут истинную угрозу, оператор радара работал аналогично классификатору. Метрики, используемые

для оценки классификаторов, также работают для оценки эффективности радаров: чувствительность — это процент вражеских самолетов, которые вызвали тревогу, а частота ложных тревог — это процент неугрожающих радиосигналов, которые вызвали тревогу.

Операторы радаров быстро поняли, что существует компромисс между чувствительностью и частотой ложных тревог. Чувствительность можно было бы повысить, если бы радиоприемники регистрировали более слабые сигналы, но это бы также приводило к большему количеству ложных тревог. Чтобы решить, насколько чувствительными должны быть их приемники, они решили собрать данные при различных настройках и построить результирующие оценки (рис. П.4).

Это назвали кривой соотношений правильного и ложного обнаружения сигналов (**R**eceiver **O**perating **C**haracteristic, **ROC**). Со временем усовершенствованные радары обеспечили высокую чувствительность при той же частоте ложных тревог, и кривые ROC сдвинулись вверх. В целом, чем лучше радиолокационная технология, тем больше заштрихованная область под кривой (**A**rea **U**nder the **C**urve, оценка **AUC**), показанная на рис. П.4.

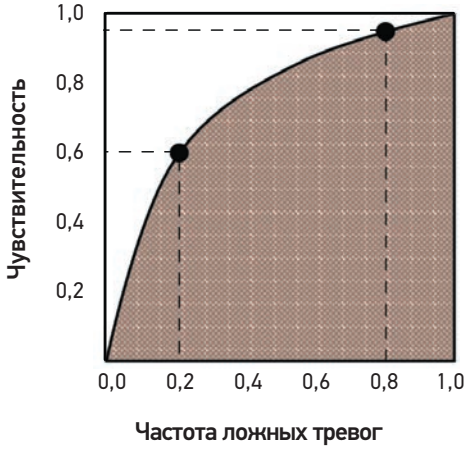


Рис. П.4. Если этот старый радар был настроен на обнаружение 95 % самолетов, то около 80 % неопасных радиосигналов приводили бы к ложным тревогам. Снижение чувствительности приемника снизило частоту ложных тревог до 20 %, но при этом удалось обнаружить только 60 % входящих самолетов

Вы можете построить кривую ROC для собственного классификатора, рассчитав чувствительность и частоту ложноположительных результатов для всех возможных пороговых значений классификации. Эта кривая поможет определиться, какой порог использовать. Кроме того, оценка AUC является хорошим показателем общей производительности классификатора. Эта оценка полезна в качестве общей оценки предсказательной силы, особенно если вы еще не выбрали, какой порог вы будете использовать с классификатором.

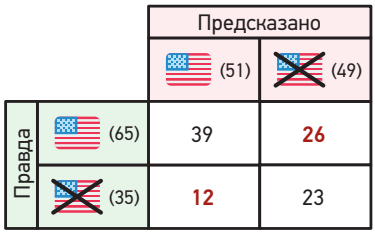
МНОГОКЛАССОВАЯ КЛАССИФИКАЦИЯ

Оценки для бинарных классификаторов также могут быть адаптированы для классификации более чем двух меток. Например, предположим, вы пытаетесь угадать, родился ли человек в США, Мексике или Канаде. Вы обучаете классификатор, а затем пробуете его со 100 людьми: 65 американцами, 25 мексиканцами и 10 канадцами. Результаты прогнозирования могут быть сведены в таблицу ошибок размером 3 на 3, чтобы мы могли видеть, *насколько* прогнозы были верными или неверными для каждой метки (рис. П.5).

		Предсказано		
		(51)	(37)	(12)
Правда	(65)	39	24	2
	(25)	11	13	1
	(10)	1	0	9

Рис. П.5. Таблица ошибок многоклассового классификатора

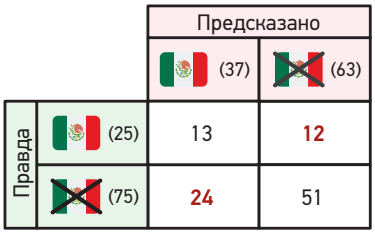
В таблице показано, где модель ошибается. Ей оказалось трудно дифференцировать американцев и мексиканцев: американцы были неправильно классифицированы как мексиканцы 24 раза, а мексиканцы были неправильно классифицированы как американцы 11 раз. Это говорит нам о том, что следует добавить больше функций с указаниями, как отличить американцев от мексиканцев.



чувствительность = 60 %,

точность = 76 %,

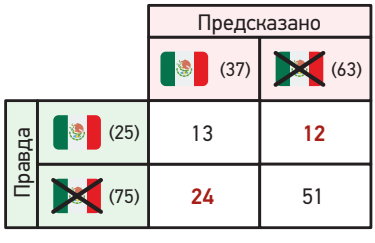
F_1 = 67 %.



чувствительность = 52 %,

точность = 35 %,

F_1 = 42 %.



чувствительность = 90 %,

точность = 75 %,

F_1 = 82 %.

Рис. П.6. Многоклассовый классификатор, рассматриваемый как три двоичных классификатора. Для каждой метки строится таблица ошибок, в которой сгруппированы все остальные метки. Например, первая таблица ошибок показывает, как именно модель классифицировала людей как американцев или неамериканцев[1]. Такой трюк поможет рассчитать точность, чувствительность и F_1 для каждой метки

Диагональ черных чисел содержит правильные догадки. Модель оказалась права 39 + 13 + 9 = 61 раз из 100, так что ее эффективность составляет 61 %. Как мы уже видели, точность может вводить в заблуждение, и мы должны использовать такие показатели, как чувствительность и точность, чтобы узнать истинную предсказательную силу модели. Поскольку эти

[1] Если выходные данные y_{pred} закодированы с одним горячим состоянием, каждая из этих таблиц ошибок соответствует одному из столбцов с одним горячим состоянием.

метрики определены для двоичной классификации, следует повторно выразить многоклассовый классификатор как комбинацию двоичных классификаторов (рис. П.6).

Чтобы получить одну оценку для модели, мы вычисляем среднее значение трех отдельных оценок F_1, которое составляет 0,64. Это называется **макрооценкой F_1**. Однако эта оценка может быть несправедливой: метка 🇨🇦 вносит такой же вклад в оценку макроса, как и метка 🇺🇸, хотя меток 🇺🇸 в шесть раз больше. При усреднении показателей метки лучше всего рассчитать *средневзвешенное* значение, где каждый вес — это количество меток в y. Мы называем это **микрооценкой F_1**:

$$\frac{(65\times 0{,}67)+(25\times 0{,}42)+(10\times 0{,}82)}{65+25+10}\approx 0{,}62=62\,\%.$$

Убедитесь, что понимаете, откуда на рис. П.6 взялись числа 65, 25 и 10. Кроме того, помните, что эти оценки представляют только производительность модели для конкретных пороговых значений меток, которые вы используете, как было показано в случае бинарных классификаторов. Если отрегулировать пороговые значения, одна и та же модель даст разные оценки.

Владстон Феррейра Фило, Мото Пиктет

Теоретический минимум по Computer Science. Сети, криптография и data science

Перевел с английского *Р. Чикин*

Руководитель дивизиона	*Ю. Сергиенко*
Ведущий редактор	*Н. Гринчик*
Литературный редактор	*К. Тульцева*
Художественный редактор	*В. Мостипан*
Корректоры	*С. Беляева, Е. Павлович*
Верстка	*Л. Егорова*

Изготовлено в России. Изготовитель: ООО «Прогресс книга». Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург, Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 06.2022. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 21.04.22. Формат 70×100/16. Бумага офсетная. Усл. п. л. 23,220. Тираж 2500. Заказ 0000.

Владстон Феррейра Фило

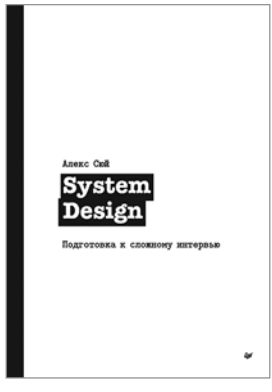
ТЕОРЕТИЧЕСКИЙ МИНИМУМ ПО COMPUTER SCIENCE. ВСЕ, ЧТО НУЖНО ПРОГРАММИСТУ И РАЗРАБОТЧИКУ

Хватит тратить время на скучные академические фолианты! Изучение Computer Science может быть веселым и увлекательным занятием. Владстон Феррейра Фило знакомит нас с вычислительным мышлением, позволяющим решать любые сложные задачи. Научиться писать код — просто: пара недель на курсах, и вы «программист», но чтобы стать профи, который будет востребован всегда и везде, нужны фундаментальные знания. Здесь вы найдете только самую важную информацию, которая необходима каждому разработчику и программисту каждый день.

«Эта книга пригодится и для решения повседневных задач. Упреждающая выборка и кэширование помогут сложить рюкзак, параллелизм облегчит готовку на кухне. Ну и, разумеется, ваш программный код будет просто потрясающим». — Владстон Феррейра Фило

Алекс Сюй

SYSTEM DESIGN. ПОДГОТОВКА К СЛОЖНОМУ ИНТЕРВЬЮ



Интервью по System Design (проектированию ИТ-систем) очень популярны у работодателей, на них легко проверить ваши навыки общения и оценить умение решать реальные задачи.

Пройти такое собеседование непросто, поскольку в проектировании ИТ-систем не существует единственно правильных решений. Речь идет о самых разнообразных реальных системах, обладающих множеством особенностей. Вам могут предложить выбрать общую архитектуру, а потом пройтись по всем компонентам или, наоборот, сосредоточиться на каком-то одном аспекте. Но в любом случае вы должны продемонстрировать понимание и знание системных требований, ограничений и узких мест.

Правильная стратегия и знания являются ключевыми факторами успешного прохождения интервью!

КУПИТЬ

Гэйл Лакман Макдауэлл

КАРЬЕРА ПРОГРАММИСТА

6-е издание

Очередное собеседование обернулось разочарованием… в очередной раз. Никто из десяти кандидатов не получил работу. Может быть, «экзаменаторы» были слишком строги?

Увы, для поступления на работу в ведущую IT-компанию академического образования недостаточно. Учебники — это замечательно, но они не помогут вам пройти собеседование, для этого нужно готовиться на реальных вопросах. Нужно решать реальные задачи и изучать встречающиеся закономерности. Главное — разработка новых алгоритмов, а не запоминание существующих задач.

«Карьера программиста» основана на опыте практического участия автора во множестве собеседований, проводимых лучшими компаниями. Это квинтэссенция сотен интервью со множеством кандидатов, результат ответов на тысячи вопросов, задаваемых кандидатами и интервьюерами в ведущих мировых корпорациях. Из тысяч возможных задач и вопросов в книгу были отобраны 189 наиболее интересных и значимых.

Шестое издание этого мирового бестселлера поможет вам наилучшим образом подготовиться к собеседованию при приеме на работу программистом или руководителем в крупную IT-организацию или перспективный стартап. Основную часть книги составляют ответы на технические вопросы и задания, которые обычно получают соискатели на собеседовании в таких компаниях, как Google, Microsoft, Apple, Amazon и других. Рассмотрены типичные ошибки, а также эффективные методики поготовки к собеседованию. Используя материал этой книги, вы с легкостью подготовитесь к устройству на работу в Google, Microsoft или любую другую ведущую IT-компанию.