ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ ДАННЫХ НА ЯЗЫКЕ PYTHON

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«МИРЭА - Российский технологический университет»
(РТУ МИРЭА)

Демидова Л. А.

Интеллектуальный анализ данных на языке Python

Учебно-методическое пособие

Москва 2021

УДК004
ББК32.97
Д 30

Демидова Л.А. Интеллектуальный анализ данных на языке Python [Электронный ресурс]: Учебно-методическое пособие / Демидова Л.А. — М.: МИРЭА – Российский технологический университет, 2021. — 1 электрон. опт. диск (CD-ROM)

В учебно-методическом пособии рассматриваются аспекты интеллектуального анализа данных средствами языка Python на примере задач, заключающихся в поиске ассоциативных правил в базах данных транзакций. Предлагается перечень заданий, при выполнении которых применяются различные критерии принятия решений. Предназначено для магистрантов по направлению 09.04.04 Программная инженерия.

Учебно-методическое пособие издается в авторской редакции.

Автор: Демидова Лилия Анатольевна.

Рецензенты:

Головин Сергей Анатольевич, д.т.н., профессор, заведующий кафедрой математического обеспечения и стандартизации информационных технологий, Институт информационных технологий, РТУ МИРЭА

Андреева Ольга Николаевна, д.т.н., доцент, начальник отдела научной работы АО «Концерн МОРИНФОРМСИСТЕМА-АГАТ»

Системные требования:
Наличие операционной системы Windows, поддерживаемой производителем.
Наличие свободного места в оперативной памяти не менее 128 Мб.
Наличие свободного места в памяти постоянного хранения (на жестком диске) не менее 30 Мб.
Наличие интерфейса ввода информации.
Дополнительные программные средства: программа для чтения pdf-файлов (Adobe Reader).
Подписано к использованию по решению Редакционно-издательского совета
МИРЭА — Российский технологический университет.
Обьем: 2.82 мб
Тираж: 10

Оглавление

ВВЕДЕНИЕ

Учебно-методическое пособие излагает принципы интеллектуального анализа данных на примере задач, заключающихся в поиске ассоциативных правил в базах данных транзакций. Рассматриваются два алгоритма поиска ассоциативных правил – Apriori и FPGrowth, реализующие различные подходы к формированию частых предметных наборов и характеризующиеся существенно различными требованиями, предъявляемыми к памяти и времени, которые необходимы для работы алгоритмов. Приводятся примеры, демонстрирующие пошагово работу алгоритмов поиска ассоциативных правил. Предлагается выполнять поиск ассоциативных правил в базах данных транзакций средствами языка Python с применением соответствующих библиотек.

Читателю предлагается перечень заданий: каждое из них содержит тестовый вариант и вариант из репозитория данных для машинного обучения. Тестовый вариант позволяет выполнить как подробное ознакомление с алгоритмами поиска ассоциативных правил в ходе ручных расчетов, например, с применением MS Excel, так и эксперименты с применением средств языка Python. Вариант с набором транзакций из репозитория данных для машинного обучения позволяет выполнить эксперименты на основе данных большого объема с применением средств языка Python.

Читатель может получить дополнительные сведения по теоретическим вопросам, а также по аспектам программной реализации для рассматриваемых алгоритмов поиска ассоциативных правил в источниках, указанных в списке литературы.

ПОИСК АССОЦИАТИВНЫХ ПРАВИЛ

1. Основные понятия

Одним из распространенных методов Data Mining является аффинитивный анализ (affinity analysis), целью которого является исследование взаимной связи между событиями, которые происходят совместно.

Анализ рыночной корзины (market basket analysis) является разновидностью аффинитивного анализа.

Целью анализа рыночной корзины является обнаружение ассоциаций между различными событиями посредством поиска ассоциативных правил, позволяющих количественно описать взаимную связь между двумя и более событиями [1, 2].

В контексте задачи поиска ассоциативных правил под транзакцией понимают множество событий, которые произошли одновременно. Таким событием, например, может быть покупка того или иного товара (предмета). Тогда рыночная корзина – это набор товаров (предметов), приобретенных в рамках одной отдельно взятой транзакции.

Транзакционная база данных (база данных транзакций, БД транзакций, transaction database) – двумерная таблица, содержащая номер транзакции и перечень покупок, приобретенных во время этой транзакции.

Пусть число транзакций в базе – N, и каждая транзакция D_i представляет собой некоторый набор предметов ($i = \overline{1, N}$).

В таблице 1 приведен пример набора транзакций (базы данных транзакций) из 10 предметов (товаров). Такой набор может быть сформирован, например, на основе чеков о покупке товаров (предметов) в магазине «Канцтовары». Следует отметить, что этот набор содержит очень мало транзакций (всего 20), что не позволяет считать выводы, сделанные на его основе объективными. Однако этот набор позволяет в полной мере продемонстрировать работу по поиску ассоциативных правил.

Любое ассоциативное правило состоит из двух наборов предметов, называемых условием (antecedent) A и следствием (consequent) B [1, 2].

Ассоциативное правило записывается в виде $A \rightarrow B$, что читается как «из A следует B» или «Если *условие A*, то *следствие B*».

Таблица 1. Пример набора транзакций на основе чеков магазина «Канцтовары»

Номер чека	Список предметов в чеке							
1	Карандаш	Ластик	Альбом	Линейка	Тетрадь			
2	Ручка	Тетрадь	Карандаш	Ластик	Пенал	Линейка		
3	Карандаш	Ластик	Ручка					
4	Краски	Альбом	Палитра	Карандаш	Кисть	Ластик		
5	Карандаш	Ластик	Тетрадь	Ручка				
6	Кисть	Краски	Палитра	Альбом				
7	Краски	Альбом	Карандаш	Ластик	Тетрадь			
8	Карандаш	Ластик	Линейка					
9	Тетрадь	Ручка	Пенал	Ластик	Карандаш			
10	Ластик	Карандаш	Тетрадь	Пенал	Кисть	Альбом	Ручка	Краски
11	Линейка	Карандаш	Ластик	Тетрадь				
12	Пенал	Ластик	Тетрадь	Карандаш				
13	Ручка	Пенал						
14	Ручка	Альбом	Линейка					
15	Альбом	Тетрадь	Карандаш	Ластик				
16	Карандаш	Ластик	Ручка	Тетрадь	Линейка			
17	Краски	Палитра	Альбом	Ластик	Карандаш	Тетрадь		
18	Карандаш	Краски	Альбом	Ластик				
19	Пенал	Линейка						
20	Карандаш	Ручка	Тетрадь	Ластик				

Ассоциативные правила описывают связь между наборами предметов, соответствующими условию *A* и следствию *B*.

Эта связь характеризуется такими показателями, как поддержка (support) *S* и достоверность (confidence) *C* [1, 2].

Поддержка *S* – доля транзакций, содержащих одновременно условие *A* и следствие *B*:

$$S(A \to B) = P(A \cap B) = \frac{\text{число транзакций, содержащих } A \text{ и } B}{\text{общее число транзакций}}.$$

Достоверность *C* – доля транзакций, содержащий и условие *A*, и следствие *B*, к числу транзакций, содержащих только условие *A*.

$$C(A \to B) = P(B \mid A) = P(A \cap B) / P(A) =$$

$$= \frac{\text{число транзакций, содержащих } A \text{ и } B}{\text{число транзакций, содержащих } \textit{только } A}.$$

Рассмотрим ассоциацию *Карандаш* → *Ластик*.

Очевидно, что эти продукты (товары) хорошо совместимы.

Так как число транзакций, содержащих как *Карандаш*, так и *Ластик*, равно 16, а общее число транзакций – 20 (таблица 2), то поддержка этой ассоциации будет:

S(*Карандаш* → *Ластик*) = 16/20 = 4/5=0,8.

Так как число транзакций, содержащих только *Карандаш* (условие), равно 16 (таблица 2), то достоверность этой ассоциации будет:

C(*Карандаш* → *Ластик*)=16/16=1.

Все наблюдения, содержащие *Карандаш*, содержат и *Ластик*, поэтому такая ассоциация может рассматриваться как ***правило***.

Теперь рассмотрим ассоциацию *Кисть* → *Ластик*.

Очевидно, что у этих продуктов (товаров) совместимость меньше, чем у продуктов (товаров) *Карандаш* и *Ластик*.

Так как число транзакций, содержащих как *Кисть*, так и *Ластик*, равно 2, а общее число транзакций – 20 (таблица 3), то поддержка этой ассоциации будет:

S(*Кисть* → *Ластик*) = 2/20 = 1/10=0,1.

Так как число транзакций, содержащих только *Кисть* (условие), равно 3 (таблица 3), то достоверность этой ассоциации будет:

C(*Кисть* → *Ластик*)=2/3=0,67.

Таблица 2. Пример набора транзакций на основе чеков магазина «Канцтовары» с выделенными предметами Карандаш и Ластик

Номер чека	Список предметов в чеке							
1	Карандаш	Ластик	Альбом	Линейка	Тетрадь			
2	Ручка	Тетрадь	Карандаш	Ластик	Пенал	Линейка		
3	Карандаш	Ластик	Ручка					
4	Краски	Альбом	Палитра	Карандаш	Кисть	Ластик		
5	Карандаш	Ластик	Тетрадь	Ручка				
6	Кисть	Краски	Палитра	Альбом				
7	Краски	Альбом	Карандаш	Ластик	Тетрадь			
8	Карандаш	Ластик	Линейка					
9	Тетрадь	Ручка	Пенал	Ластик	Карандаш			
10	Ластик	Карандаш	Тетрадь	Пенал	Кисть	Альбом	Ручка	Краски
11	Линейка	Карандаш	Ластик	Тетрадь				
12	Пенал	Ластик	Тетрадь	Карандаш				
13	Ручка	Пенал						
14	Ручка	Альбом	Линейка					
15	Альбом	Тетрадь	Карандаш	Ластик				
16	Карандаш	Ластик	Ручка	Тетрадь	Линейка			
17	Краски	Палитра	Альбом	Ластик	Карандаш	Тетрадь		
18	Карандаш	Краски	Альбом	Ластик				
19	Пенал	Линейка						
20	Карандаш	Ручка	Тетрадь	Ластик				

Таблица 3. Пример набора транзакций на основе чеков магазина «Канцтовары» с выделенными предметами Кисть и Ластик

Номер чека	Список предметов в чеке							
1	Карандаш	Ластик	Альбом	Линейка	Тетрадь			
2	Ручка	Тетрадь	Карандаш	Ластик	Пенал	Линейка		
3	Карандаш	Ластик	Ручка					
4	Краски	Альбом	Палитра	Карандаш	Кисть	Ластик		
5	Карандаш	Ластик	Тетрадь	Ручка				
6	Кисть	Краски	Палитра	Альбом				
7	Краски	Альбом	Карандаш	Ластик	Тетрадь			
8	Карандаш	Ластик	Линейка					
9	Тетрадь	Ручка	Пенал	Ластик	Карандаш			
10	Ластик	Карандаш	Тетрадь	Пенал	Кисть	Альбом	Ручка	Краски
11	Линейка	Карандаш	Ластик	Тетрадь				
12	Пенал	Ластик	Тетрадь	Карандаш				
13	Ручка	Пенал						
14	Ручка	Альбом	Линейка					
15	Альбом	Тетрадь	Карандаш	Ластик				
16	Карандаш	Ластик	Ручка	Тетрадь	Линейка			
17	Краски	Палитра	Альбом	Ластик	Карандаш	Тетрадь		
18	Карандаш	Краски	Альбом	Ластик				
19	Пенал	Линейка						
20	Карандаш	Ручка	Тетрадь	Ластик				

Невысокая достоверность этой ассоциации дает повод усомниться в том, что она является правилом. При этом поддержка этой ассоциации является совсем маленькой.

При анализе ассоциаций наиболее часто предпочтение отдается тем, которые имеют высокие значения поддержки и достоверности. Хотя в ряде случаев могут выбираться и ассоциации, имеющие только высокую поддержку или только высокую достоверность.

Правила, значения поддержки (достоверности) которых превышают пороговое значение, заданное пользователем, называются *сильными* правилами (strong rules).

Например, при анализе ассоциаций на основе чеков о покупках в магазине с целью формирования рекомендаций по формированию успешного плана его деятельности могут быть интересны ассоциации, для которых значения поддержки и достоверности не ниже пороговых значений в 25% и 75% соответственно.

При анализе ассоциаций с целью выявления мошеннических действий могут быть интересны ассоциации, значение поддержки которых невелико (например, не больше 1%), так как с такими действиями связано малое число транзакций.

Значимость ассоциации может быть вычислена как разность между поддержкой правила в целом и произведением поддержки *только* условия и поддержки *только* следствия.

Если условие и следствие независимы, то поддержка ассоциации примерно соответствует произведению поддержек условия A и следствия B:

$S_{AB} \approx S_A \cdot S_B$.

В таком случае можно будет сделать следующий вывод: хотя условие A и следствие B часто встречаются вместе, не менее часто они встречаются и по отдельности.

Например, если товар A встречался в 75 транзакциях из 100, а товар B – в 85, и в 64 транзакциях из 100 они встречаются вместе, то, несмотря на высокое значение поддержки ($S_{AB} = 0{,}64$), ассоциация $A \rightarrow B$ не обязательно является правилом: товары A и B покупаются независимо друг от друга, но по причине их популярности часто встречаются в одной транзакции.

Поскольку произведение значений поддержек условия A и следствия B вычисляется как $S_A \cdot S_B = 0{,}75 \cdot 0{,}85 = 0{,}6375$, то есть отличается от $S_{AB} = 0{,}64$ всего на 0,0025, предположение о независимости товаров A и B можно считать достаточно обоснованным.

Если условие A и следствие B независимы, то ассоциация скорее всего не может считаться правилом, даже если она имеет высокие значения поддержки и достоверности.

При оценке **значимости ассоциаций применяют объективные и субъективные меры значимости** [2]

Объективные меры – поддержка и достоверность. Такие меры значимости могут применяться в любой предметной области.

Субъективные меры – лифт (lift) и левередж (leverage, плечо, рычаг). Такие меры значимости связаны напрямую связаны с информацией, определяемой контекстом решаемой задачи анализа данных.

Лифт – отношение достоверности ассоциации $A \to B$ к поддержке следствия B [2]:

$$L(A \to B) = C(A \to B) / S(B).$$

Значения лифта, большие чем 1, показывают, что условие чаще появляется в транзакциях, содержащих следствие B, чем в остальных.

Лифт является обобщенной мерой связи двух предметных наборов, фигурирующих в числителе и знаменателе формулы: при значениях лифта, превосходящих 1, связь положительная, при значении лифта, равном 1, связь отсутствует, а при значениях лифта, меньших 1, связь отрицательная.

Рассмотрим ассоциацию *Ластик* $\to$ *Карандаш* и вычислим для нее лифт.

S(*Карандаш*)=16/20=0,8; C(*Ластик* $\to$ *Карандаш*)=16/16=1.

Следовательно, L(*Ластик* $\to$ *Карандаш*)=1/0,8=1,25.

Рассмотрим ассоциацию *Ластик* $\to$ *Кисть* и вычислим для нее лифт.

S(*Кисть*)=3/20=0,15; C(*Ластик* $\to$ *Кисть*)=2/16=0,125.

Следовательно, L(*Ластик* $\to$ *Кисть*)=0,125/0,15=0,833.

Для первой ассоциации лифт больше 1, для второй – меньше 1, значит, ластик больше влияет на покупку карандаша, чем на покупку кисти.

Следует отметить, что лифт не всегда является удачной мерой значимости ассоциации.

Так, например, ассоциация с меньшим значением поддержкой и большим значением лифта может быть менее значимой, чем ассоциация с большим значением поддержки и меньшим значением лифта, так как что вторая ассоциация используется для большего числа транзакций. Следовательно, увеличение числа транзакций приводит к увеличению связи между условием и следствием ассоциации.

Левередж – разность между наблюдаемой частотой, с которой условие и следствие появляются совместно, и произведением частот появления условия и следствия по отдельности, то есть разность между поддержкой ассоциации и произведением поддержек условия и следствия по отдельности [2]:

$Lev(A \to B) = S(A \to B) - S(A) \cdot S(B)$.

Рассмотрим ассоциации *Карандаш* $\to$ *Ластик* и *Тетрадь* $\to$ *Ластик*.

Эти ассоциации имеют одинаковую *достоверность* $C(A \to B) = 1$, т.к. карандаш (*всего 16 раз*) (таблица 2) и тетрадь (*всего 12 раз*) (таблица 4) всегда продаются вместе с ластиком (16 и 12 соответственно).

Эти ассоциации имеют одинаковые *лифты*, так как в обеих ассоциациях поддержка следствия *S*(*Ластик*)=16/20=0,8 (*ластик встретился в транзакциях 16 раз*), и, следовательно,

L(*Карандаш* $\to$ *Ластик*)=*L*(*Тетрадь* $\to$ *Ластик*)=1/0,8=1,25.

Вычислим левереджи для этих ассоциаций.

S(*Карандаш* $\to$ *Ластик*)=16/20=0,8.

S(*Карандаш*)=16/20=0,8. *S*(*Ластик*)=16/20=0,8.

Следовательно, *Lev*(*Карандаш* $\to$ *Ластик*)=0,8 – 0,8·0,8=0,16.

S(*Тетрадь* $\to$ *Ластик*)=12/20=0,6.

S(*Тетрадь*)=12/20=0,6. *S*(*Ластик*)=16/20=0,8.

Следовательно, *Lev*(*Тетрадь* $\to$ *Ластик*)=0,6 – 0,6·0,8=0,12.

Ассоциация *Карандаш* $\to$ *Ластик* представляет больший интерес, так как встречается чаще (то есть применяется в большем числе транзакций, чем ассоциация *Тетрадь* $\to$ *Ластик*).

Таким образом, значимость ассоциации *Карандаш* $\to$ *Ластик* более высокая.

Иногда вместо лифта используют еще одну субъективную меру – улучшение (improvement), которую вычисляют подобно левереджу, но берут не разность, а отношение между наблюдаемой частотой, с которой условие и следствие появляются совместно, и произведением частот появления условия и следствия по отдельности [2]:

$$I(A \to B) = \frac{S(A \to B)}{S(A) \cdot S(B)}.$$

Таблица 4. Пример набора транзакций на основе чеков магазина «Канцтовары»

Номер чека	Список предметов в чеке							
1	Карандаш	Ластик	Альбом	Линейка	Тетрадь			
2	Ручка	Тетрадь	Карандаш	Ластик	Пенал	Линейка		
3	Карандаш	Ластик	Ручка					
4	Краски	Альбом	Палитра	Карандаш	Кисть	Ластик		
5	Карандаш	Ластик	Тетрадь	Ручка				
6	Кисть	Краски	Палитра	Альбом				
7	Краски	Альбом	Карандаш	Ластик	Тетрадь			
8	Карандаш	Ластик	Линейка					
9	Тетрадь	Ручка	Пенал	Ластик	Карандаш			
10	Ластик	Карандаш	Тетрадь	Пенал	Кисть	Альбом	Ручка	Краски
11	Линейка	Карандаш	Ластик	Тетрадь				
12	Пенал	Ластик	Тетрадь	Карандаш				
13	Ручка	Пенал						
14	Ручка	Альбом	Линейка					
15	Альбом	Тетрадь	Карандаш	Ластик				
16	Карандаш	Ластик	Ручка	Тетрадь	Линейка			
17	Краски	Палитра	Альбом	Ластик	Карандаш	Тетрадь		
18	Карандаш	Краски	Альбом	Ластик				
19	Пенал	Линейка						
20	Карандаш	Ручка	Тетрадь	Ластик				

Улучшение $I(A \to B)$ показывает, полезнее ли ассоциация случайного угадывания. Если улучшение $I(A \to B)$ больше 1, то то означает, что вероятнее предсказать наличие набора *B* в следствии с помощью ассоциации, чем угадать случайно.

Такие меры, как лифт $L(A \to B)$, левередж $Lev(A \to B)$ и улучшение $I(A \to B)$, могут использоваться для последующего сокращения числа рассматриваемых ассоциаций посредством установки порогового значения для используемой меры значимости: ассоциации, у которых значение меры значимости ниже порогового, исключаются из дальнейшего рассмотрения.

2. Алгоритм Apriori

Число возможных ассоциаций растет экспоненциально с увеличением числа предметов, на основе которых могут формироваться ассоциации.

Пусть общее число предметов в базе данных транзакций равно *k*.

Пусть все ассоциации содержат по одному предмету в условии и в следствии.

В этом случае требуется проанализировать $2 \cdot C_k^2$ ассоциаций.

Пусть, например, *k*=100.

Тогда число возможных ассоциаций равно $2 \cdot C_{100}^2$=9900.

Очевидно, что в условии и следствии может содержаться большее число предметов. Следовательно, число возможных ассоциаций будет еще бóльшим.

Исходя из вышесказанного, можно сделать следующий вывод: поиск ассоциативных правил посредством вычисления поддержки и достоверности для всех возможных ассоциаций и сравнения их с заданным пороговым значением будет малоэффективным из-за больших вычислительных затрат.

Обычно при поиске ассоциативных правил используется методика, предполагающая выявление частых предметных наборов. При реализации такой методики анализируются только те ассоциации, которые встречаются часто. При этом задается некоторое пороговое значение, позволяющее отнести предметных набор к частым или к нечастым.

Одним из алгоритмов, основанных на применении методики выявления частых предметных наборов, является алгоритм Apriori [1 – 4].

Этот алгоритм предложили ученые Ракеш Агравал (Rakesh Agrawal) и Рамакришнан Шрикант (Ramakrishnan Srikant) в 1994 году.

Цель алгоритма Apriori – сузить пространство поиска до размеров, обеспечивающих приемлемые временные затраты на поиск ассоциативных правил.

2.1. Выявление частых предметных наборов

В основе алгоритма Apriori лежит понятие частого набора (frequent itemset), который также можно назвать частым предметным набором, для которого определяется частота его появления в базе транзакций.

Под частотой понимается простое число транзакций, в которых содержится рассматриваемый предметный набор.

Частым предметным набором будет тот, который встречается чаще, чем в числе транзакций, описываемом некоторой пороговой частотой Δ.

Кроме того, можно считать, что частый предметный набор – предметный набор с поддержкой, большей заданного порогового значения либо равной этому значению.

Это пороговое значение называется минимальной поддержкой.

Методика поиска ассоциативных правил с использованием частых наборов содержит следующую последовательность шагов.

Шаг 1. Поиск частых предметных наборов.

Шаг 2. Генерация на основе частых предметных наборов ассоциативных правил, удовлетворяющих условиям минимальной поддержки и достоверности.

Алгоритм Apriori использует свойство антимонотонности, чтобы сузить пространство поиска.

Свойство антимонотонности: если предметный набор Z не является частым, то добавление некоторого нового предмета A к набору Z не делает его более частым.

Если набор Z не является частым, то и набор $Z \cup A$ также не будет являться частым.

Использование свойства антимонотонности позволяет значительно уменьшить пространство поиска ассоциативных правил.

Рассмотрим набор транзакций, приведенный в таблице 1, и применим к нему алгоритм Apriori.

Таблица 5. Представление данных из набора транзакций в нормализованном виде

№ чека	Альбом	Карандаш	Кисть	Краски	Ластик	Линейка	Палитра	Пенал	Ручка	Тетрадь
1	1	1	0	0	1	1	0	0	0	1
2	0	1	0	0	1	1	0	1	1	1
3	0	1	0	0	1	0	0	0	1	0
4	1	1	1	1	1	0	1	0	0	0
5	0	1	0	0	1	0	0	0	1	1
6	1	0	1	1	0	0	1	0	0	0
7	1	1	0	1	1	0	0	0	0	1
8	0	1	0	0	1	1	0	0	0	0
9	0	1	0	0	1	0	0	1	1	1
10	1	1	1	1	1	0	0	1	1	1
11	0	1	0	0	1	1	0	0	0	1
12	0	1	0	0	1	0	0	1	0	1
13	0	0	0	0	0	0	0	1	1	0
14	1	0	0	0	0	1	0	0	1	0
15	1	1	0	0	1	0	0	0	0	1
16	0	1	0	0	1	1	0	0	1	1
17	1	1	0	1	1	0	1	0	0	1
18	1	1	0	1	1	0	0	0	0	0
18	0	0	0	0	0	1	0	1	0	0
20	0	1	0	0	1	0	0	0	1	1
Сумма по столбцу	9	16	3	6	16	7	3	6	9	12

Будем считать частыми такие предметные наборы, которые встречаются в в базе транзакций более 5 раз, то есть определим пороговое значение частоты как $\Delta = 5$.

Выполним поиск ассоциативных правил, реализуя следующие шаги.

Шаг 1. Выполним поиск частых однопредметных наборов, представив набор транзакций в нормализованном виде (таблица 5).

На пересечении строки транзакции (строки с номером чека) и столбца с названием предмета поставим число «1», если предмет присутствует в транзакции, и «0» – в противном случае.

Просуммируем значения в каждом столбце для вычисления частоты появления каждого предмета в наборе транзакций.

Запишем частоту появления каждого предмета в наборе транзакций в соответствующую ячейку последней строки таблицы 5.

Как видно из таблицы 5, сумма в каждом столбце, за исключением столбцов с названиями предметов Кисть и Палитра больше порогового значения частоты $\Delta = 5$.

Все предметы, для которых частота появления в наборе транзакций больше или равна $\Delta = 5$, будем считать частыми однопредметными наборами.

Сформируем множество F_1 частых однопредметных наборов:

F_1={Альбом, Карандаш, Краски, Ластик, Линейка, Пенал, Ручка, Тетрадь}.

Шаг 2. Выполним поиск частых двухпредметных наборов (таблица 6).

В общем случае для формирования множества F_k частых *k*-предметных наборов реализуют следующий алгоритм:

– создать множество наборов-кандидатов в частные *k*-предметные наборы посредством связывания множества F_{k-1} с самим собой;

– выполнить редукцию множества F_k, используя свойство антимонотонности.

Оставшиеся предметные наборы формируют искомое множество F_k частых *k*-предметных наборов.

Так как пороговое значение частоты $\Delta = 5$, то в множество F_2 войдут только те двухпредметные наборы, которые встречаются 5 и более раз (таблица 6).

Информация по найденным частным двухпредметным наборам выделена в таблице 6 жирным шрифтом.

Таблица 6. Двухпредметные наборы-кандидаты в F_2

Предметный набор		Частота появления в наборе транзакций
Предмет 1	Предмет 2	
Альбом	**Карандаш**	**7**
Альбом	**Краски**	**6**
Альбом	**Ластик**	**7**
Альбом	Линейка	2
Альбом	Пенал	1
Альбом	Ручка	2
Альбом	**Тетрадь**	**5**
Карандаш	**Краски**	**5**
Карандаш	**Ластик**	**16**
Карандаш	**Линейка**	**5**
Карандаш	Пенал	1
Карандаш	**Ручка**	**7**
Карандаш	**Тетрадь**	**12**
Краски	**Ластик**	**5**
Краски	Линейка	0
Краски	Пенал	1
Краски	Ручка	1
Краски	Тетрадь	3

Окончание таблицы 6

Предметный набор		Частота появления в наборе транзакций
Предмет 1	Предмет 2	
Ластик	**Линейка**	**5**
Ластик	Пенал	4
Ластик	**Ручка**	**7**
Ластик	**Тетрадь**	**12**
Линейка	Пенал	2
Линейка	Ручка	3
Линейка	Тетрадь	4
Пенал	Ручка	4
Пенал	Тетрадь	4
Ручка	**Тетрадь**	**6**

Сформируем множество F_2 частых двухпредметных наборов:

F_2 {{Альбом, Карандаш}, {Альбом, Краски}, {Альбом, Ластик},
{Альбом, Тетрадь}, {Карандаш, Краски}, {Карандаш, Ластик},
{Карандаш, Линейка}, {Карандаш, Ручка}, { Карандаш, Тетрадь},
{Краски, Ластик}, {Ластик, Линейка}, {Ластик, Ручка},
{Ластик, Тетрадь}, {Ручка, Тетрадь}}.

Шаг 3. Выполним поиск частых трехпредметных наборов (таблица 7).

Для получения множества F_3 свяжем множество F_2 с самим собой.

В общем случае *k*-предметные наборы являются связываемыми, если у них первые $k-1$ предметов, следующих в алфавитном порядке, общие.

Так, наборы {Альбом, Карандаш} и {Альбом, Краски}, для которых $k=2$, чтобы быть связываемыми, должны иметь один ($k-1=1$) общий первый предмет, которым и является Альбом.

В результате связывания получим:

{Альбом, Карандаш} + {Альбом, Краски}={Альбом, Карандаш, Краски}.

Так как пороговое значение частоты $\Delta = 5$, то в множество F_3 войдут только те трехпредметные наборы, которые встречаются 5 и более раз (таблица 7). Информация по найденным частным двухпредметным наборам выделена в таблице 7 жирным шрифтом.

Множество трехпредметных наборов-кандидатов в F_3 сокращается с помощью свойства антимонотонности: поддержка любого набора предметов не может превышать минимальной поддержки любого из его поднаборов.

С увеличением числа предметов в наборе поддержка уменьшается или остается такой же.

В общем случае любой *k*-предметный набор будет часто встречающимся тогда и только тогда, когда все его $(k-1)$-предметные поднаборы будут часто встречающимися.

Рассмотрим набор {Альбом, Карандаш, Краски}.

В нем есть двухпредметные поднаборы: {Альбом, Карандаш}, {Альбом, Краски}, {Карандаш, Краски}, частота появления которых в наборе тракзакций соответственно равна 7, 6 и 7. Следовательно, эти поднаборы частые и трехпредметный набор {Альбом, Карандаш, Краски} не подлежит изъятию из дальнейшего рассмотрения.

Таблица 7. Трехпредметные наборы-кандидаты в F_3

Предметный набор			Частота появления в наборе транзакций
Предмет 1	Предмет 2	Предмет 3	
Альбом	**Карандаш**	**Краски**	**5**
Альбом	**Карандаш**	**Ластик**	**7**
Альбом	**Карандаш**	**Тетрадь**	**5**
Альбом	**Краски**	**Ластик**	**5**
Альбом	Краски	Тетрадь	3
Альбом	**Ластик**	**Тетрадь**	**5**
Карандаш	**Краски**	**Ластик**	**5**
Карандаш	Краски	Линейка	0
Карандаш	Краски	Ручка	1
Карандаш	Краски	Тетрадь	3
Карандаш	**Ластик**	**Линейка**	**5**
Карандаш	**Ластик**	**Ручка**	**7**
Карандаш	**Ластик**	**Тетрадь**	**12**
Карандаш	Линейка	Ручка	2
Карандаш	Линейка	Тетрадь	4
Карандаш	**Ручка**	**Тетрадь**	**6**
Ластик	Линейка	Ручка	2
Ластик	Линейка	Тетрадь	4
Ластик	**Ручка**	**Тетрадь**	**6**

Рассмотрим набор {Альбом, Краски, Тетрадь}.

В нем есть двухпредметные поднаборы: {Альбом, Краски}, {Альбом, Тетрадь}, {Краски, Тетрадь}, частота появления которых в наборе тракзакций соответственно равна 6, 5 и 3.

Первые два поднабора – частые, а третий – нечастый. Следовательно, и трехпредметный набор {Альбом, Краски, Тетрадь} не является частым и подлежит изъятию из дальнейшего рассмотрения.

Сформируем множество F_3 частых трехпредметных наборов:

F_3{{Альбом, Карандаш, Краски}, {Альбом, Карандаш, Ластик},
{Альбом, Карандаш, Тетрадь}, {Альбом, Краски, Ластик},
{Альбом, Ластик, Тетрадь}, {Карандаш, Краски, Ластик},
{Карандаш, Ластик, Линейка}, {Карандаш, Ластик, Ручка},
{Карандаш, Ластик, Тетрадь}, {Карандаш, Ручка, Тетрадь},
{Ластик, Ручка, Тетрадь}}.

Шаг 4. Выполним поиск частых четыреххпредметных наборов (таблица 8).

Для получения множества F_4 свяжем множество F_3 с самим собой и сформируем множество F_4 частых четырехпредметных наборов:

F_4{{Альбом, Карандаш, Краски, Ластик},
{Альбом, Карандаш, Ластик, Тетрадь},
{Карандаш, Ластик, Ручка, Тетрадь}}.

Эти четырехпредметные наборы уже нельзя связать. Поэтому задача поиска частых предметных наборов на исходном множестве транзакций решена.

2.2. Генерация ассоциативных правил

Для генерации ассоциативных правил на основе частых предметных наборов set_j ($j=\overline{1,m}$, m – число выявленных частных предметных наборов) к каждому частому набору set_j необходимо применить следующую процедуру.

1. Сгенерировать все возможные поднаборы $subset_{j,l_j}$ ($j=\overline{1,m}$; $l_j=\overline{1,m_j}$, m_j – число поднаборов на основе набора set_j).

2. Рассмотреть ассоциацию *R*: $subset_{j,l_j} \rightarrow (set_j - subset_{j,l_j})$, где $subset_{j,l_j}$ –непустой поднабор набора set_j; $(set_j - subset_{j,l_j})$ –набор set_j без поднабора $subset_{j,l_j}$. Ассоциация *R* считается ассоциативным правилом, если удовлетворяет условию заданного минимума для поддержки и достоверности.

Таблица 8. Четырехпредметные наборы-кандидаты в F_4

Предметный набор				Частота появления в наборе транзакций
Предмет 1	Предмет 2	Предмет 3	Предмет 4	
Альбом	**Карандаш**	**Краски**	**Ластик**	**5**
Альбом	Карандаш	Краски	Тетрадь	3
Альбом	**Карандаш**	**Ластик**	**Тетрадь**	**5**
Карандаш	Ластик	Линейка	Ручка	2
Карандаш	Ластик	Линейка	Тетрадь	4
Карандаш	**Ластик**	**Ручка**	**Тетрадь**	**6**

Рассмотрим предметный набор-кандидат в ассоциативные правила.

set={Альбом, Карандаш, Краски}.

В него входят поднаборы:

{Альбом}, {Карандаш}, {Краски}, {Альбом, Карандаш}, {Альбом, Краски}, {Карандаш, Краски}.

Построим ассоциативные правила, в которых условия содержат два предмета или один предмет.

Для ассоциативного правила с двумя предметами в условии предположим, что

subset ={Альбом, Карандаш}.

Тогда ($set - subset$)={Краски}.

Рассмотрим правило:

R: {Альбом, Карандаш}$\rightarrow${Краски}.

Поддержка, показывающая долю транзакций, которые содержат как условие {Альбом, Карандаш}, так и следствие {Краски}, в общем наборе из 20 транзакций составляет 25 % (5 из 20 транзакций).

Чтобы найти достоверность, нужно учесть, что набор {Альбом, Карандаш} появляется в 7 из 20 транзакций, 5 из которых также содержат {Краски}. Тогда достоверность будет равна 5/7 (71,4 %).

В таблице 9 приведены ассоциации с двумя предметами в условии для трехпредметного набора {Альбом, Карандаш, Краски}.

Если предположить, что минимальная достоверность для ассоциативного правила составляет 60%, то все ассоциации в таблице 9 будут правилами.

Если предположить, что минимальная достоверность для ассоциативного правила составляет 80%, то только вторая и третья ассоциации будут правилами.

Таблица 9. Ассоциации с двумя предметами в условии для трехпредметного набора {Альбом, Карандаш, Краски}

Если условие, то следствие	Поддержка	Достоверность
Если {Альбом и Карандаш}, то {Краски}	5/20 = 25 %	5/7 = 71,4 %
Если {Альбом и Краски}, то {Карандаш}	5/20 = 25 %	5/6 = 80 %
Если {Карандаш и Краски}, то {Альбом}	5/20 = 25 %	5/5 = 100 %

Таблица 10. Ассоциации с одним предметом в условии

Если условие, то следствие	Поддержка	Достоверность
{Альбом}→{Карандаш}	0,35	0,778
{Карандаш}→{Альбом}	0,35	0,438
{Альбом}→{Краски}	0,3	0,667
{Краски}→{Альбом}	0,3	1
{Альбом}→{Ластик}	0,35	0,778
{Ластик}→{Альбом}	0,35	0,438
{Альбом}→{Тетрадь}	0,25	0,556
{Тетрадь}→{Альбом}	0,25	0,417
{Карандаш}→{Краски}	0,25	0,313
{Краски}→{Карандаш}	0,25	0,833
{Карандаш}→{Ластик}	0,8	1
{Ластик}→{Карандаш}	0,8	1
{Карандаш}→{Линейка}	0,25	0,313
{Линейка}→{Карандаш}	0,25	0,714
{Карандаш→{Ручка}	0,35	0,438
{Ручка}→{Карандаш}	0,35	0,778
{Карандаш}→{Тетрадь}	0,6	0,75
{Тетрадь}→{Карандаш}	0,6	1
{Краски}→{Ластик}	0,25	0,833
{Ластик}→{Краски}	0,25	0,313
{Ластик}→{Линейка}	0,25	0,313
{Линейка}→{Ластик}	0,25	0,714
{Ластик}→{Ручка}	0,35	0,438
{Ручка}→{Ластик}	0,35	0,778
{Ластик}→{Тетрадь}	0,6	0,75
{Тетрадь}→{Ластик}	0,6	1
{Ручка}→{Тетрадь}	0,3	0,667
{Тетрадь}→{Ручка}	0,3	0,5

Таблица 11. Ассоциации с двумя предметами **в условии**

Если условие, то следствие	Поддержка	Достоверность
{Альбом, Карандаш}→{Краски}	0,25	0,714
{Альбом, Краски}→{Карандаш}	0,25	0,833
{Карандаш, Краски}→{Альбом}	0,25	1
{Альбом, Карандаш}→{Ластик}	0,35	1
{Альбом, Ластик}→{Карандаш}	0,35	1
{Карандаш, Ластик}→{Альбом}	0,35	0,438
{Альбом, Карандаш}→{Тетрадь}	0,25	0,714
{Альбом, Тетрадь}→{Карандаш}	0,25	1
{Карандаш, Тетрадь}→{Альбом}	0,25	0,417
{Альбом, Краски}→{Ластик}	0,25	0,833
{Альбом, Ластик}→{Краски}	0,25	0,714
{Краски, Ластик}→{Альбом}	0,25	1
{Альбом, Ластик}→{Тетрадь}	0,25	0,714
{Альбом, Тетрадь}→{Ластик}	0,25	1
{Ластик, Тетрадь}→{Альбом}	0,25	0,417
{Карандаш, Краски}→{Ластик}	0,25	1
{Карандаш, Ластик}→{Краски}	0,25	0,313
{Краски, Ластик}→{Карандаш}	0,25	1
{Карандаш, Ластик}→{Линейка}	0,25	0,313
{Карандаш, Линейка}→{Ластик}	0,25	1
{Ластик, Линейка}→{Карандаш}	0,25	1
{Карандаш, Ластик}→{Ручка}	0,35	0,438
{Карандаш, Ручка}→{Ластик}	0,35	1
{Ластик, Ручка}→{Карандаш}	0,35	1
{Карандаш, Ластик}→{Тетрадь}	0,6	0,75
{Карандаш, Тетрадь}→{Ластик}	0,6	1
{Ластик, Тетрадь}→{Карандаш}	0,6	1
{Карандаш, Ручка}→{Тетрадь}	0,3	0,857
{Карандаш, Тетрадь}→{Ручка}	0,3	0,5
{Ручка, Тетрадь}→{Карандаш}	0,3	1
{Ластик, Ручка}→{Тетрадь}	0,3	0,857
{Ластик, Тетрадь}→{Ручка}	0,3	0,5
{Ручка, Тетрадь}→{Ластик}	0,3	1

Таблица 12. Ассоциации с тремя предметами **в условии**

Если условие, то следствие	Поддержка	Достоверность
{Карандаш, Ластик,Ручка}→{Тетрадь}	0,3	0,857
{Ластик, Ручка,Тетрадь}→{Карандаш}	0,3	1
{Ручка, Тетрадь,Карандаш}→{Ластик}	0,3	1
{Тетрадь, Карандаш,Ластик}→{Ручка}	0,3	0,5
{Альбом, Карандаш,Краски}→{Ластик}	0,25	1
{Карандаш, Краски,Ластик}→{Альбом}	0,25	1
{Краски, Ластик,Альбом}→{Карандаш}	0,25	1
{Ластик, Альбом,Карандаш}→{Краски}	0,25	0,714
{Альбом, Карандаш,Ластик}→{Тетрадь}	0,25	0,714
{Карандаш, Ластик,Тетрадь}→{Альбом}	0,25	0,417
{Ластик, Тетрадь,Альбом}→{Карандаш}	0,25	1
{Тетрадь, Альбом,Карандаш}→{Ластик}	0,25	1

Получим все ассоциативные правила на основе выявленных частых предметных наборов *с одним предметом в следствии*.

Ассоциативные правила с двумя и более предметами в следствии могут быть построены аналогичным образом.

В таблице 10 приведены все ассоциации с одним предметом в условии и одним предметом (всего – 28 ассоциаций).

В таблице 11 приведены все ассоциации с двумя предметами в условии (всего – 33 ассоциации).

В таблице 12 приведены все ассоциации с тремя предметами в условии (всего – 12 ассоциаций).

Чтобы оценить значимость сгенерированных ассоциаций необходимо перемножить соответствующие им значения поддержки и достоверности.

Выберем в качестве ассоциативных правил те ассоциации из таблиц 10 – 12, у которых достоверность не ниже 80% (таблица 13).

В результате применения алгоритма Apriori удалось найти 34 ассоциативных правил, которые с достоверностью не ниже 80% показывают, какие продукты из исходного набора предметов чаще всего продаются вместе.

Если понизить пороговое значение достоверности до 60%, то число ассоциативных правил увеличиться на 17 штук и будет равно 51.

Таблица 13. Ассоциативные правила при минимальной достоверности 80%

Если условие, то следствие	Поддержка S	Достоверность C	$S \cdot C \cdot$
{Карандаш, Ластик, Ручка}→{Тетрадь}	0,3	0,857	0,257
{Ластик, Ручка ,Тетрадь}→{Карандаш}	0,3	1	0,3
{Ручка, Тетрадь, Карандаш}→{Ластик}	0,3	1	0,3
{Альбом, Карандаш, Краски}→{Ластик}	0,25	1	0,25
{Карандаш, Краски, Ластик}→{Альбом}	0,25	1	0,25
{Краски, Ластик, Альбом}→{Карандаш}	0,25	1	0,25
{Ластик, Тетрадь, Альбом}→{Карандаш}	0,25	1	0,25
{Тетрадь, Альбом, Карандаш}→{Ластик}	0,25	1	0,25
{Альбом, Краски}→{Карандаш}	0,25	0,833	0,208
{Карандаш, Краски}→{Альбом}	0,25	1	0,25
{Альбом, Карандаш}→{Ластик}	0,35	1	0,35
{Альбом, Ластик}→{Карандаш}	0,35	1	0,35
{Альбом, Краски}→{Ластик}	0,25	0,833	0,208
{Краски, Ластик}→{Альбом}	0,25	1	0,25
{Альбом, Тетрадь}→{Ластик}	0,25	1	0,25
{Карандаш, Краски}→{Ластик}	0,25	1	0,25
{Краски, Ластик}→{Карандаш}	0,25	1	0,25
{Карандаш, Линейка}→{Ластик}	0,25	1	0,25
{Ластик, Линейка}→{Карандаш}	0,25	1	0,25
{Карандаш, Ручка}→{Ластик}	0,35	1	0,35
{Ластик, Ручка}→{Карандаш}	0,35	1	0,35
{Карандаш, Тетрадь}→{Ластик}	0,6	1	0,6
{Ластик, Тетрадь}→{Карандаш}	0,6	1	0,6
{Карандаш, Ручка}→{Тетрадь}	0,3	0,857	0,257
{Ручка, Тетрадь}→{Карандаш}	0,3	1	0,3
{Ластик, Ручка}→{Тетрадь}	0,3	0,857	0,257
{Ручка, Тетрадь}→{Ластик}	0,3	1	0,3
{Краски}→{Альбом}	0,3	1	0,3
{Краски}→{Карандаш}	0,25	0,833	0,208
{Карандаш}→{Ластик}	0,8	1	0,8
{Ластик}→{Карандаш}	0,8	1	0,8
{Тетрадь}→{Карандаш}	0,6	1	0,6
{Краски}→{Ластик}	0,25	0,833	0,208
{Тетрадь}→{Ластик}	0,6	1	0,6

Очевидно, что по результатам выявления ассоциативных правил можно разработать адекватную маркетинговую стратегию, оптимизировать закупки и размещение товаров (продуктов) на прилавках и витринах.

3. Алгоритм FPGrowth

Как уже было ранее отмечено, алгоритм Apriori реализует генерацию кандидатов в частые предметные наборы, а это приводит к тому, что вычислительные и временные затраты, которые необходимы для обработки (анализа) кандидатов, могут быть неприемлемыми.

Кроме того, алгоритм Apriori требует многократного сканирования базы данных транзакций: необходимо выполнить столько сканирований, сколько предметов содержит самый длинный предметный набор.

В связи с этим были предложены алгоритмы, позволяющие отказаться от генерации кандидатов и сократить требуемое число сканирований.

Одной из наиболее эффективных альтернатив алгоритму Apriori является алгоритм Frequent Pattern-Growth (FPGrowth), название которого можно интерпретировать как «выращивание часто встречающихся предметных наборов». Алгоритм FPGrowth позволяет избежать затратной процедуры генерации кандидатов, а также – уменьшить необходимое число проходов по базе данных транзакций до двух [4 – 7].

В основе алгоритма FPGrowth лежит предобработка БД транзакций, в процессе которой на основе БД транзакций формируется компактная древовидная структура, а именно – дерево частых предметных наборов (Frequent-Pattern Tree, FP-дерево).

Сжатие БД транзакций в компактную структуру обеспечивает эффективное и полное извлечение частых предметных наборов. При этом при построении FP-дерева применяется парадигма «разделай и влавствуй» (divide and conquer), позволяющий выполнить декомпозицию сложной задачи на некоторое число более простых.

Сравнивая алгоритмы Apriori и FPGrowth, можно отметить следующее.

Алгоритм Apriori использует подход, реализующий генерацию кандидатов в частые предметные наборы и тестирование кандидатов на предмет, являются ли они частыми.

Генерация кандидатов требует больших временных затрат, а также больших затрат памяти для их хранения.

Расчет значений поддержки также требует больших временных затрат, так как сопровождается анализом поднаборов и множественными проходами по БД транзакций.

Алгоритм FPGrowth позволяет выявлять частые предметные наборы без генерации кандидатов в них. При этом алгоритм FPGrowth требует выполнения 2 нижеуказанных этапов [4 – 7].

Этап 1. Построение компактной структуры данных в виде FP-дерева с реализацией 2 проходов по БД транзакций.

Этап 2. Извлечение частых предметных наборов непосредственно из FP-дерева.

При реализации алгоритма FPGrowth предполагается следующее.

1. Узлы FP-дерева соответствуют предметам в БД транзакций и имеют счётчики поддержки.

2. При обращении к БД транзакций считывается одна транзакция, в соответствие которой ставится путь в FP-дереве.

3. Осуществляется упорядочение предметов в каждой транзакции по убыванию частоты появления этих предметов в БД транзакций. Такое упорядочение предметов позволяет сделать так, что пути в дереве перекрываются (накладываются), если имеют общие предметы (т.е. имеют одинаковые префиксы). При наличии наложения путей для транзакций счётчики поддержки соответствующих узлов увеличиваются.

4. Создаются односвязные списки, указатели которых используются перемещения между узлами, соответствующими одному и тому же предмету (на изображении FP-дерево связи между такими узлами помечаются пунктирными линиями со стрелками).

5. Чем больше наложение путей, тем больше сжатие, что позволяет в итоге без больших проблем разместить FP-дерево в памяти.

6. Частые предметные наборы извлекаются из FP-дерева.

Рассмотрим работу алгоритма FPGrowth на примере той же БД транзакций (таблица 1), которая была использована для алгоритма Apriori.

Будем считать (как и в алгоритме Apriori) частыми такие предметные наборы, которые встречаются в базе транзакций более 5 раз, то есть определим пороговое значение частоты как $\Delta = 5$.

В дальнейшем в FPGrowth алгоритме значение частоты будем называть поддержкой.

Таблица 14. Пример набора транзакций на основе чеков магазина «Канцтовары» с упорядочением предметов в транзакциях по убыванию значений их частоты (поддержки)

Номер чека	Список предметов в чеке							
1	Карандаш	Ластик	Тетрадь	Альбом	Линейка			
2	Карандаш	Ластик	Тетрадь	Ручка	Линейка	Пенал		
3	Карандаш	Ластик	Ручка					
4	Карандаш	Ластик	Альбом	Краски	~~Кисть~~	~~Палитра~~		
5	Карандаш	Ластик	Тетрадь	Ручка				
6	Альбом	Краски	~~Кисть~~	~~Палитра~~				
7	Карандаш	Ластик	Тетрадь	Альбом	Краски			
8	Карандаш	Ластик	Линейка					
9	Карандаш	Ластик	Тетрадь	Ручка	Пенал			
10	Карандаш	Ластик	Тетрадь	Альбом	Ручка	Краски	Пенал	~~Кисть~~
11	Карандаш	Ластик	Тетрадь	Линейка				
12	Карандаш	Ластик	Тетрадь	Пенал				
13	Ручка	Пенал						
14	Альбом	Ручка	Линейка					
15	Карандаш	Ластик	Тетрадь	Альбом				
16	Карандаш	Ластик	Тетрадь	Ручка	Линейка			
17	Карандаш	Ластик	Тетрадь	Альбом	Краски	~~Палитра~~		
18	Карандаш	Ластик	Альбом	Краски				
19	Линейка	Пенал						
20	Карандаш	Ластик	Тетрадь	Ручка				

Этап 1. Построение компактной структуры данных в виде FP-дерева.

Число сканирований, т.е. число проходов по БД транзакций, равно 2.

1. Первое сканирование базы данных транзакций.

Произведем сканирование БД транзакций и сформируем множество частых предметов, т.е. предметов, которые встречаются Δ (например, $\Delta = 5$) или более раз. Исключим из дальнейшего рассмотрения нечастые предметы.

Упорядочим выявленные частые предметы в порядке убывания частот их появления в БД транзакций (таблица 5) и получим следующий набор:

(Карандаш, 16), (Ластик, 16), (Тетрадь, 12), (Альбом, 9), (Ручка, 9), (Линейка, 7), (Краски, 6), (Пенал, 6).

Этот порядок будем использовать при построении FP-дерева. В результате общие для транзакций префиксы (пути) в FP-дереве будут накладываться.

Разместим эти наборы в указанном порядке в столбик, отделив название предмета от его частоты (счетчика поддержки) двоеточием (рисунок 1).

В дальнейшем *будем размещать узлы дерева, соответствующие одному и тому же предмету, на одном уровне*, делая ветви дерева при необходимости более длинными.

Например, все узлы, соответствующие предмету Пенал, должны быть размещены на самом нижнем уровне, т.к. предмет Пенал имеет одну из самых низких частот (наряду с предметом Краски) в списке упорядочения частых предметов.

2. Второе сканирование базы данных транзакций.

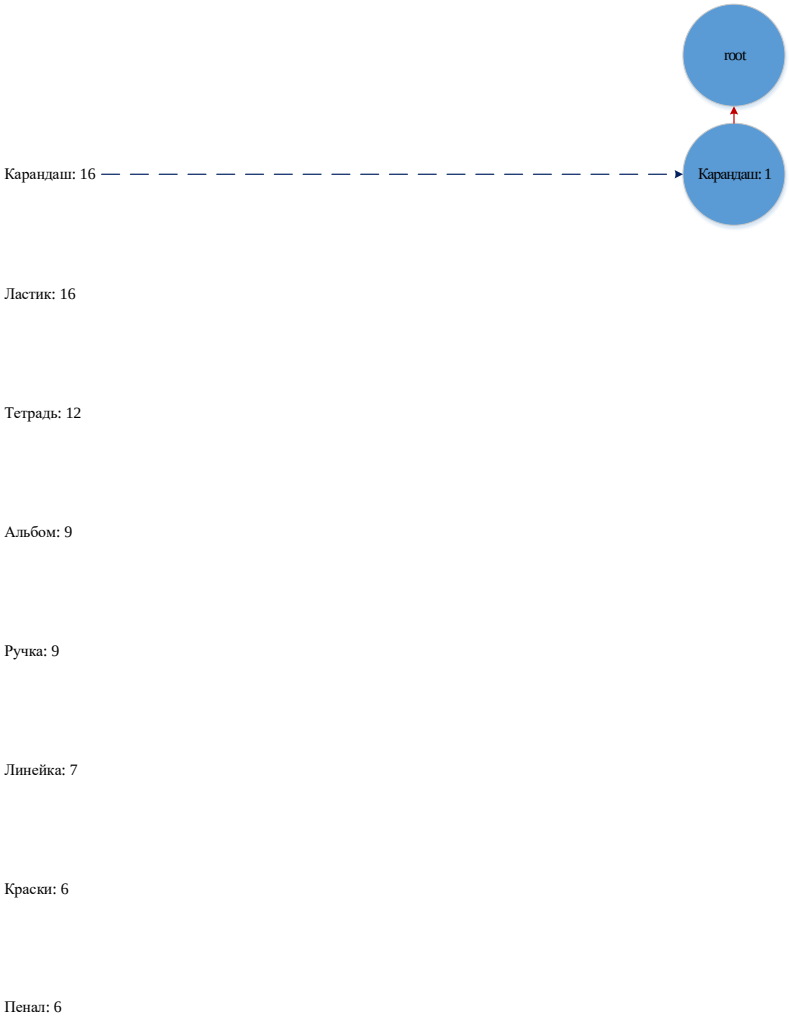
Построим FP-дерево, предварительно упорядочив предметы в транзакциях по убыванию значений их частот, то есть по убыванию значений поддержки (таблица 14), удалив из транзакций те предметы, у которых значение поддержки меньше, чем Δ (например, $\Delta = 5$). В таблице 14 нечастые предметы помечены как «вычеркнутые».

Создадим начальный (корневой, root) узел FP-дерева.

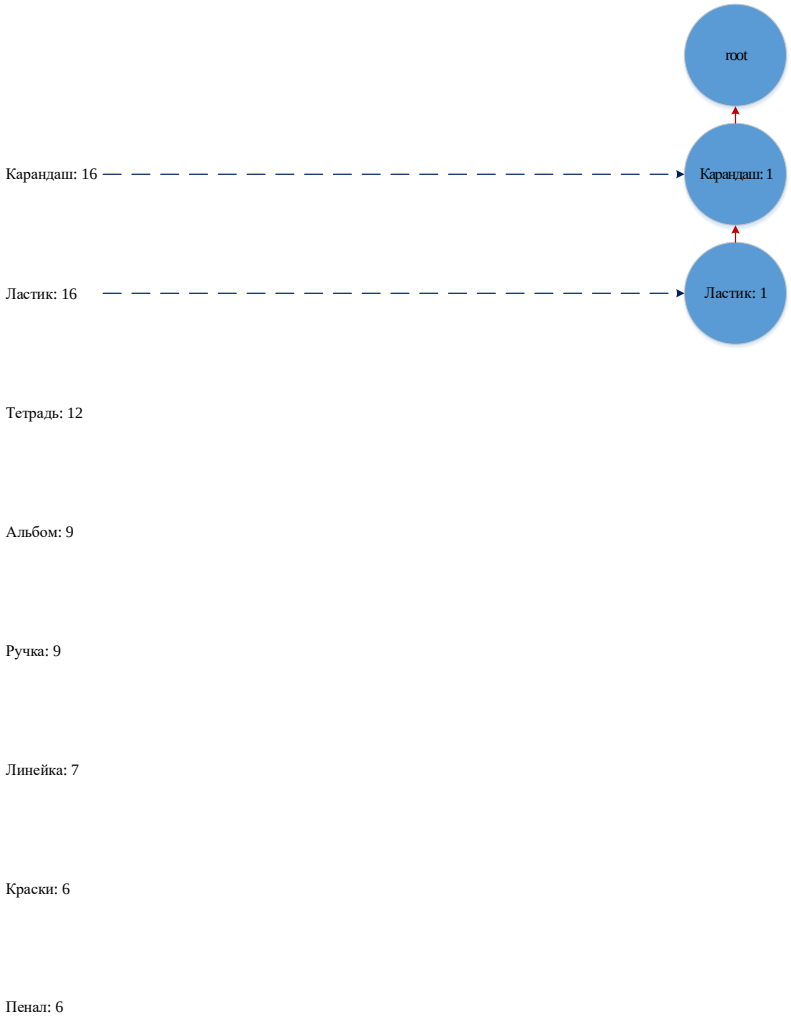
Осуществим построение дерева, начиная с транзакции №1 для упорядоченных предметных наборов, т.е. рассмотрим сначала предметный набор (Карандаш, Ластик, Тетрадь, Альбом, Линейка).

При построении дерева будем использовать следующее правило.

Правило. *Если для очередного предмета в FP-дереве уже содержится узел, то для предмета не создается новый узел, а счетчик поддержки существующего узла увеличивается на 1. В противном случае для этого предмета создается новый узел и его счётчику поддержки присваивается значение, равное 1.*

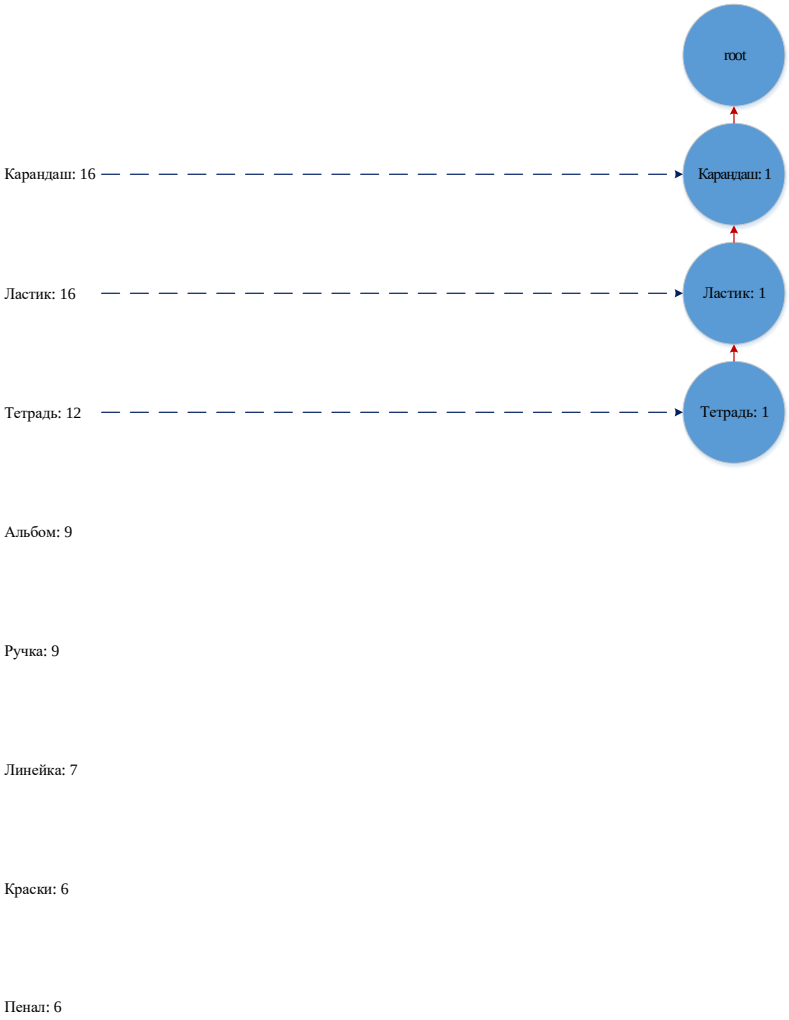


а

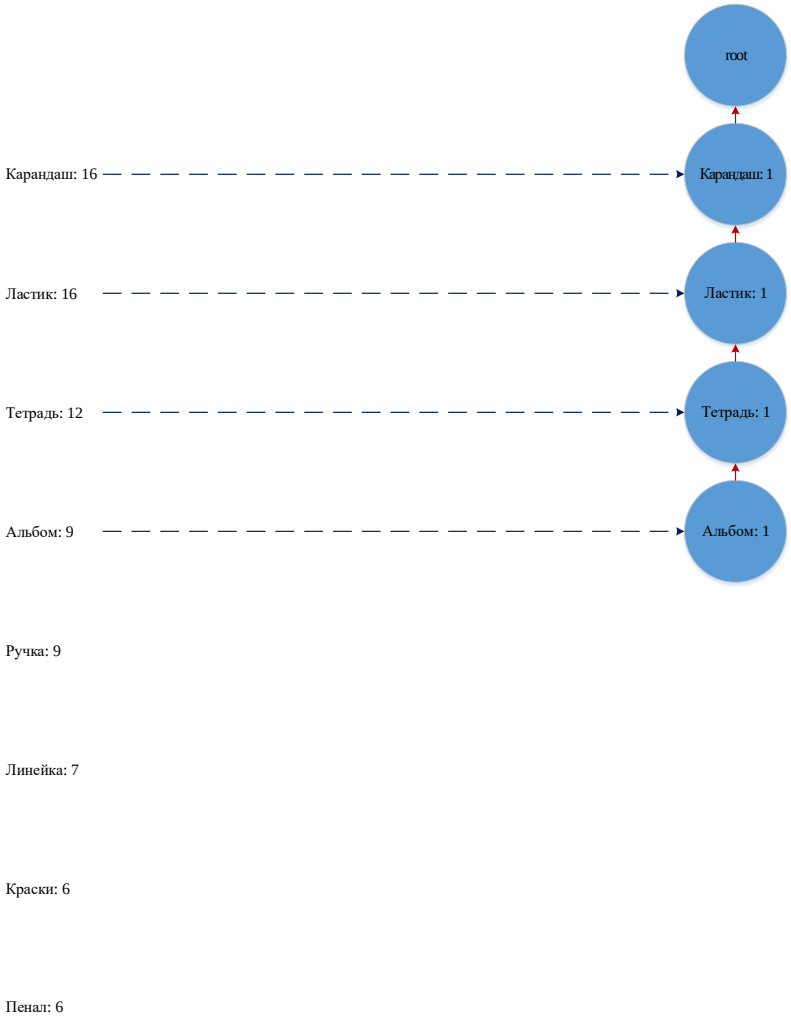


б

Рисунок 1. Шаги построения FP-дерева на основе транзакции № 1 (начало)



а



б

Рисунок 2. Шаги построения FP-дерева на основе транзакции № 1 (продолжение)

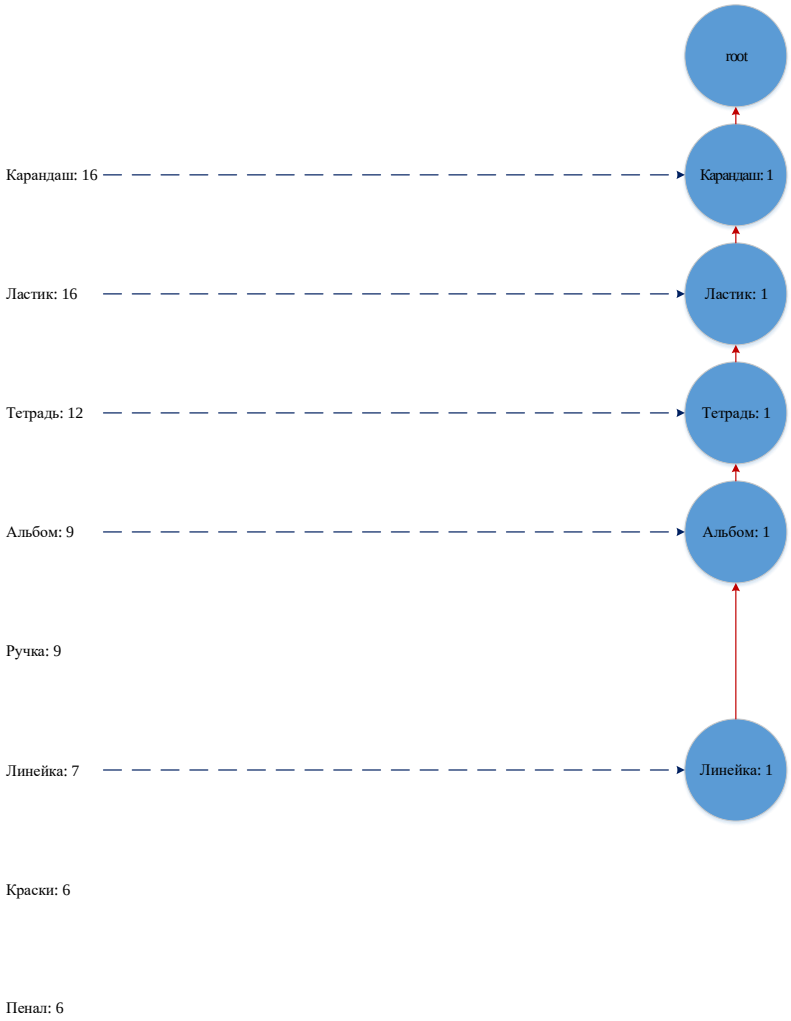


Рисунок 3. Шаги построения FP-дерева на основе транзакции № 1 (окончание)

Сплошными бордовыми линиями со стрелки будем отмечать связи дочерних узлов с родительскими.

Пунктирными темно-синими линиями со стрелками будем отмечать связи между узлами дерева, соответствующими одним и тем же предметам, с целью формирования односвязанных списков.

Сначала возьмем предмет Карандаш из транзакции №1. Поскольку он является первым, то создаем для него узел и соединяем с родительским (корневым) узлом (рисунок 1, а).

Затем возьмем следующий предмет – предмет Ластик. Поскольку FP-дерево пока не содержит узлов с именем Ластик, добавляем в FP-дерево новый узел Ластик, который будет потомком узла Карандаш (рисунок 1, б).

Затем возьмем следующий предмет – предмет Тетрадь.

Рассуждая аналогичным образом, добавляем в FP-дерево новый узел Тетрадь, который будет потомком узла Ластик (рисунок 2, а).

Таким же образом создадим узлы для предметов Альбом (рисунок 2, б) и Линейка (рисунок 3) из транзакции № 1. Для каждого нового узла определим значение его счетчика равным 1.

На этом использование транзакции №1 для построения FP-дерева завершено.

Для транзакции №2, содержащей предметы из предметного набора (Карандаш, Ластик, Тетрадь, Ручка, Линейка, Пенал) выбираем первый предмет – предмет Карандаш. Поскольку узел с таким именем уже существует, то в соответствии с правилом построения FP-дерева новый узел не создается, а счетчик поддержки существующего увеличивается на 1 (рисунок 4, а).

При добавлении следующего предмета – предмета Ластик – используем это же правило: поскольку узел Ластик является дочерним по отношению к текущему (т.е. к узлу Карандаш), то новый узел не создается, а счетчик поддержки существующего увеличивается на 1 (рисунок 4, б).

При добавлении следующего предмета – предмета Тетрадь – используем это же правило: поскольку узел Тетрадь является дочерним по отношению к текущему (т.е. к узлу Ластик), то новый узел не создается, а счетчик поддержки существующего увеличивается на 1 (рисунок 5).

Следующий предмет – предмет Ручка – в соответствии с правилом построения FP-дерева породит в FP-дереве новый узел, дочерний к узлу Тетрадь (рисунок 6).

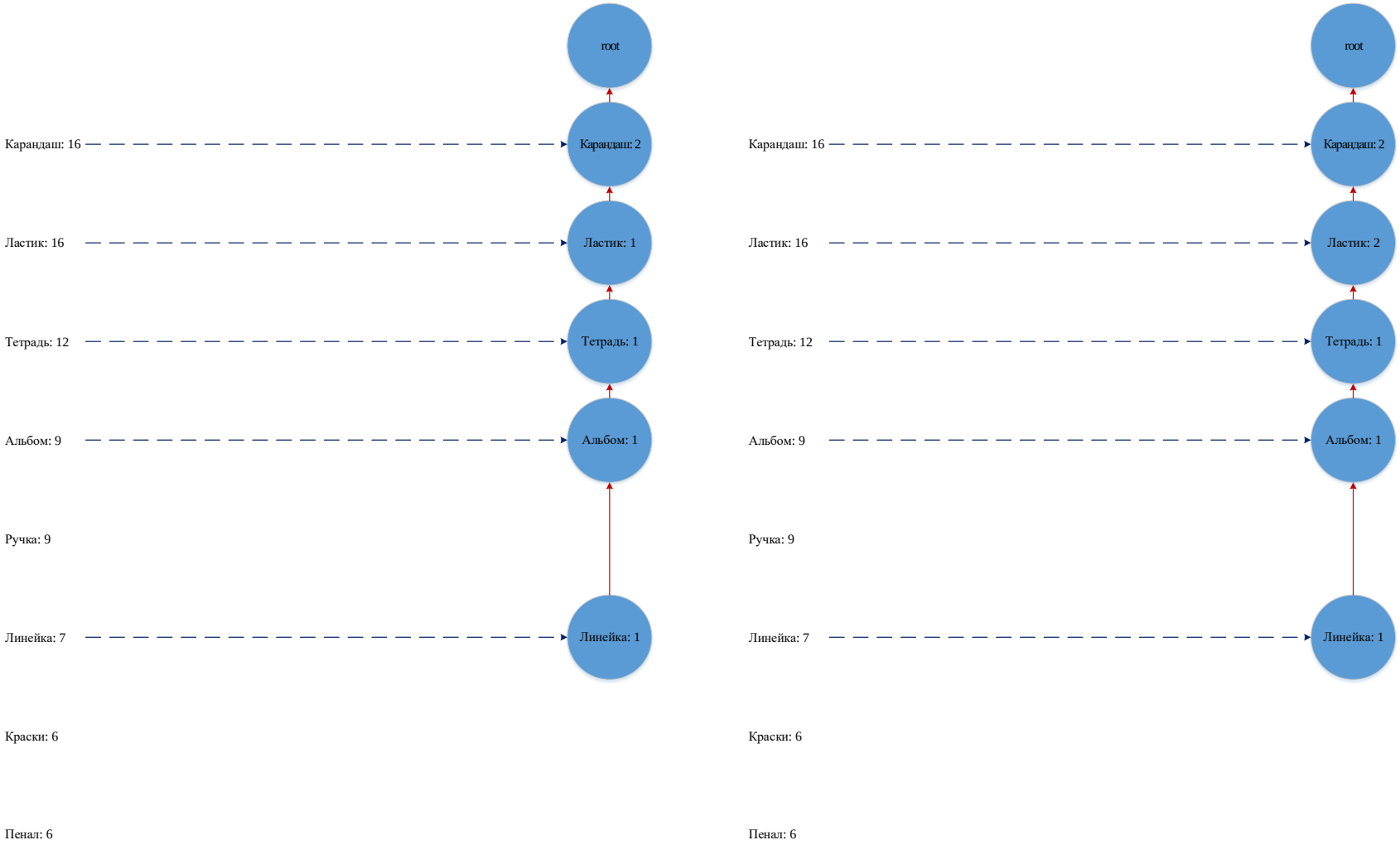


Рисунок 4. Шаги построения FP-дерева на основе транзакции № 2 (начало): добавление предметов Карандаш и Ластик без создания новых узлов

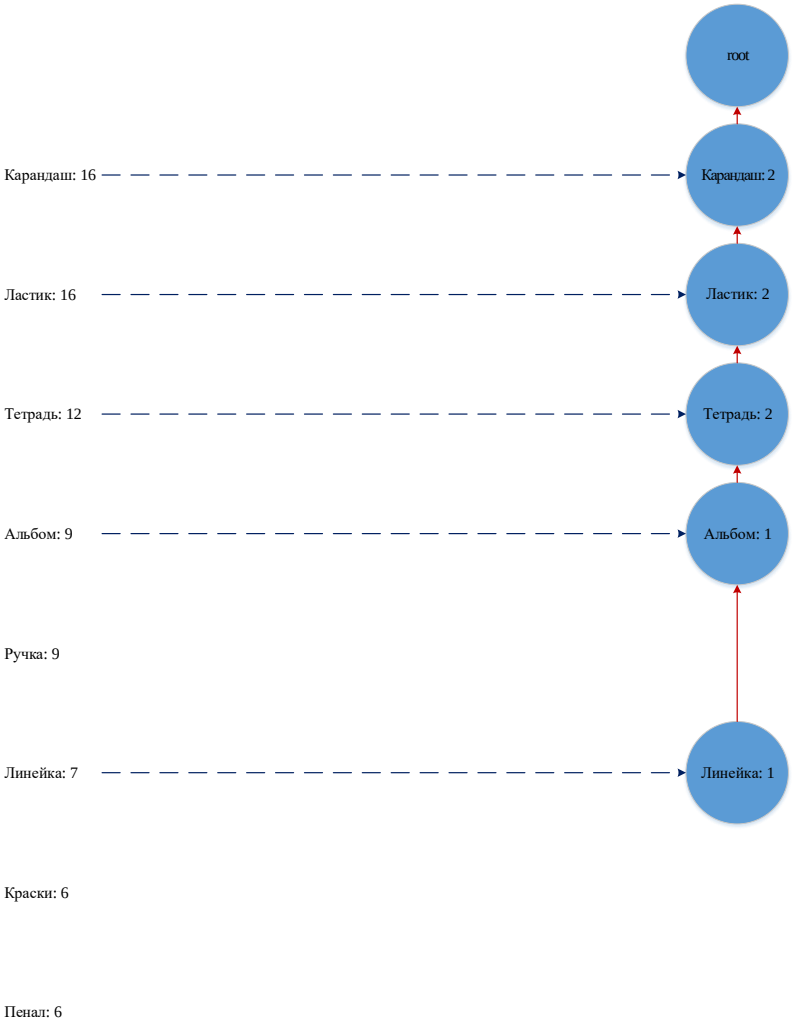


Рисунок 5. Шаги построения FP-дерева на основе транзакции № 2 (продолжение): добавление предмета Тетрадь без создания нового узла

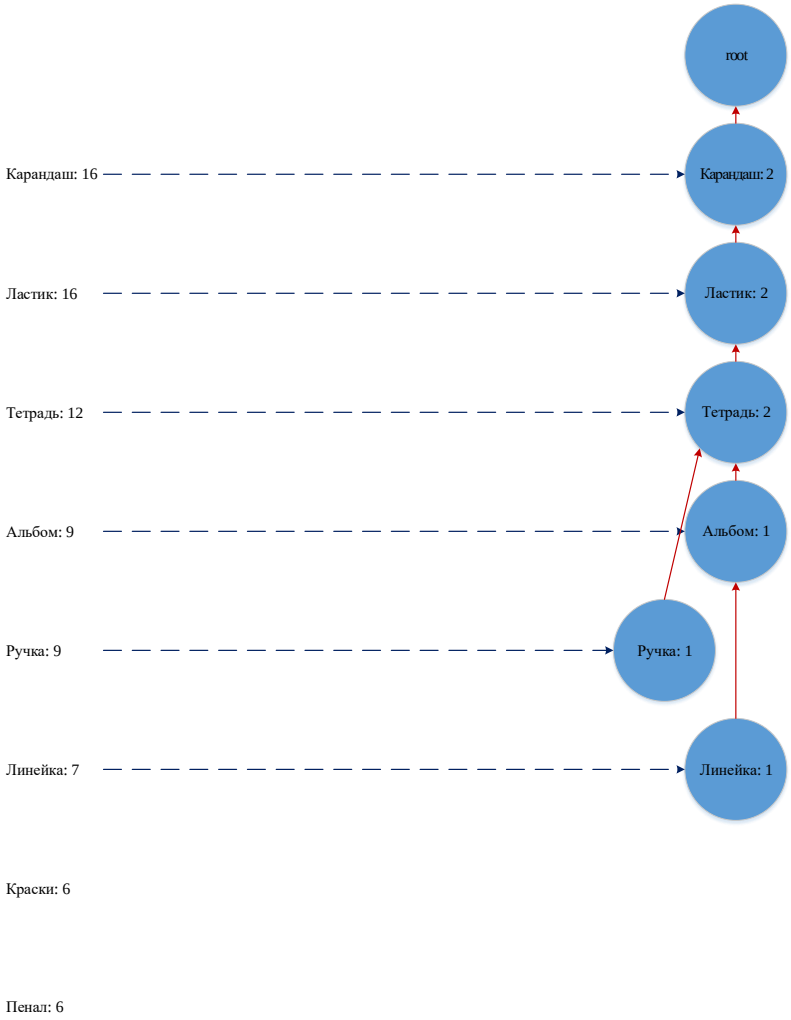


Рисунок 6. Шаги построения FP-дерева на основе транзакции № 2 (продолжение): добавление предмета Ручка с созданием нового узла

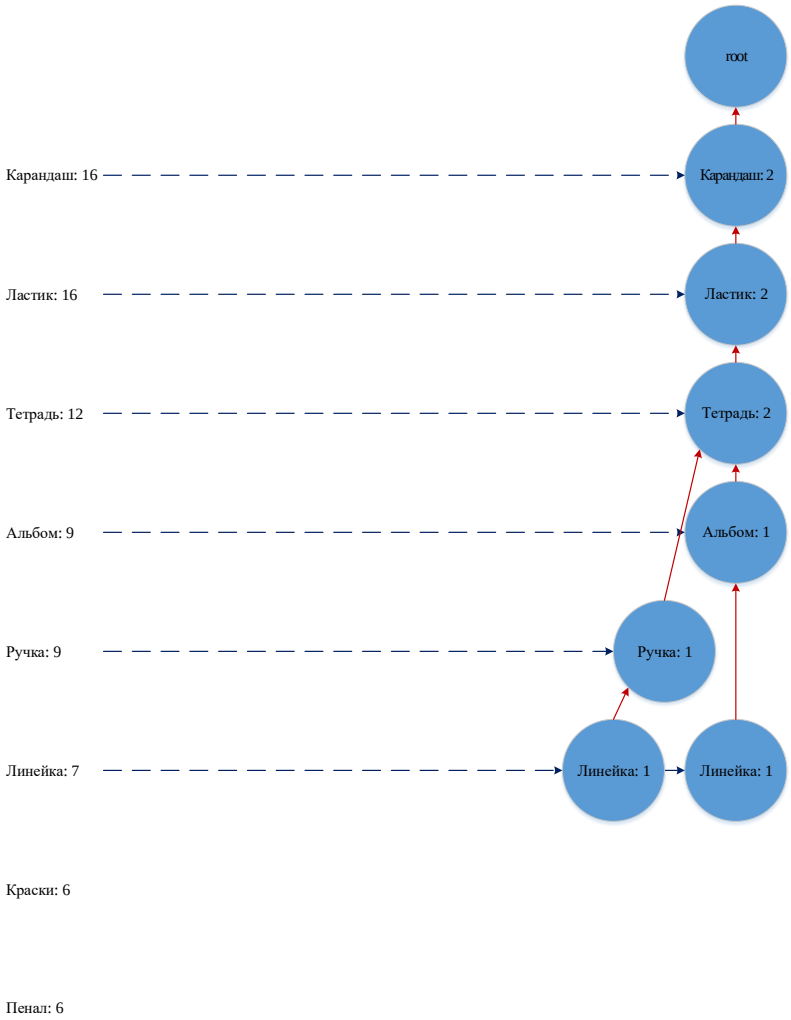


Рисунок 7. Шаги построения FP-дерева на основе транзакции № 2 (продолжение): добавление предмета Линейка с созданием нового узла

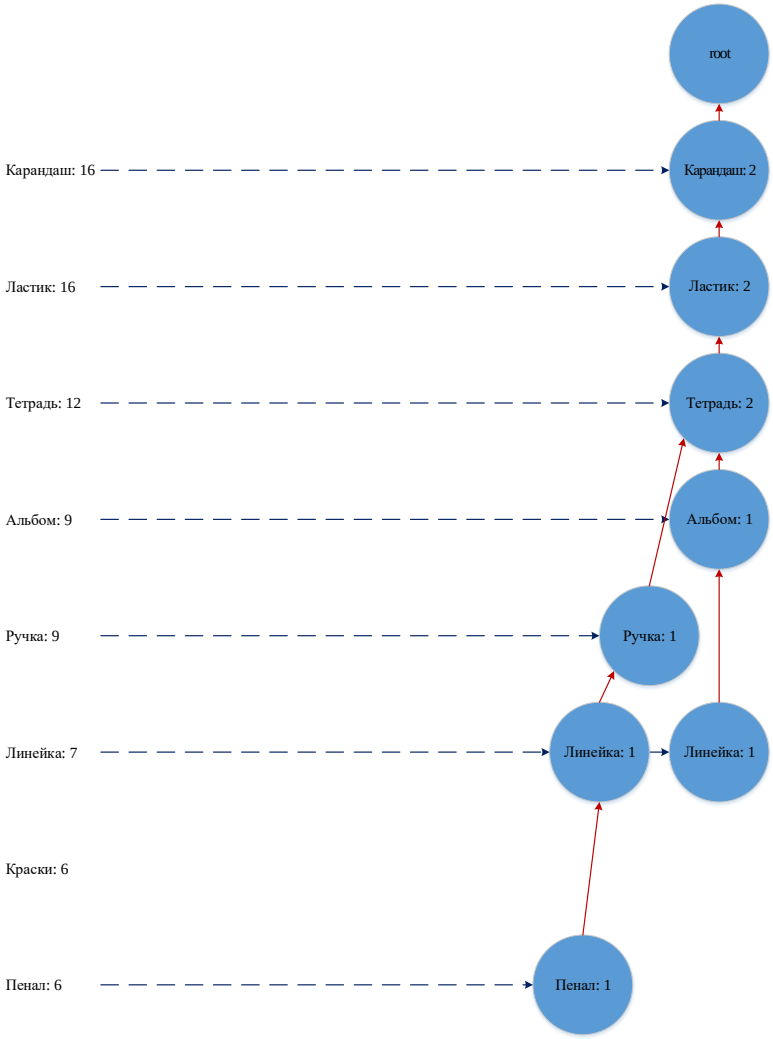


Рисунок 8. Шаги построения FP-дерева на основе транзакции № 2 (окончание): добавление предмета Пенал с созданием нового узла

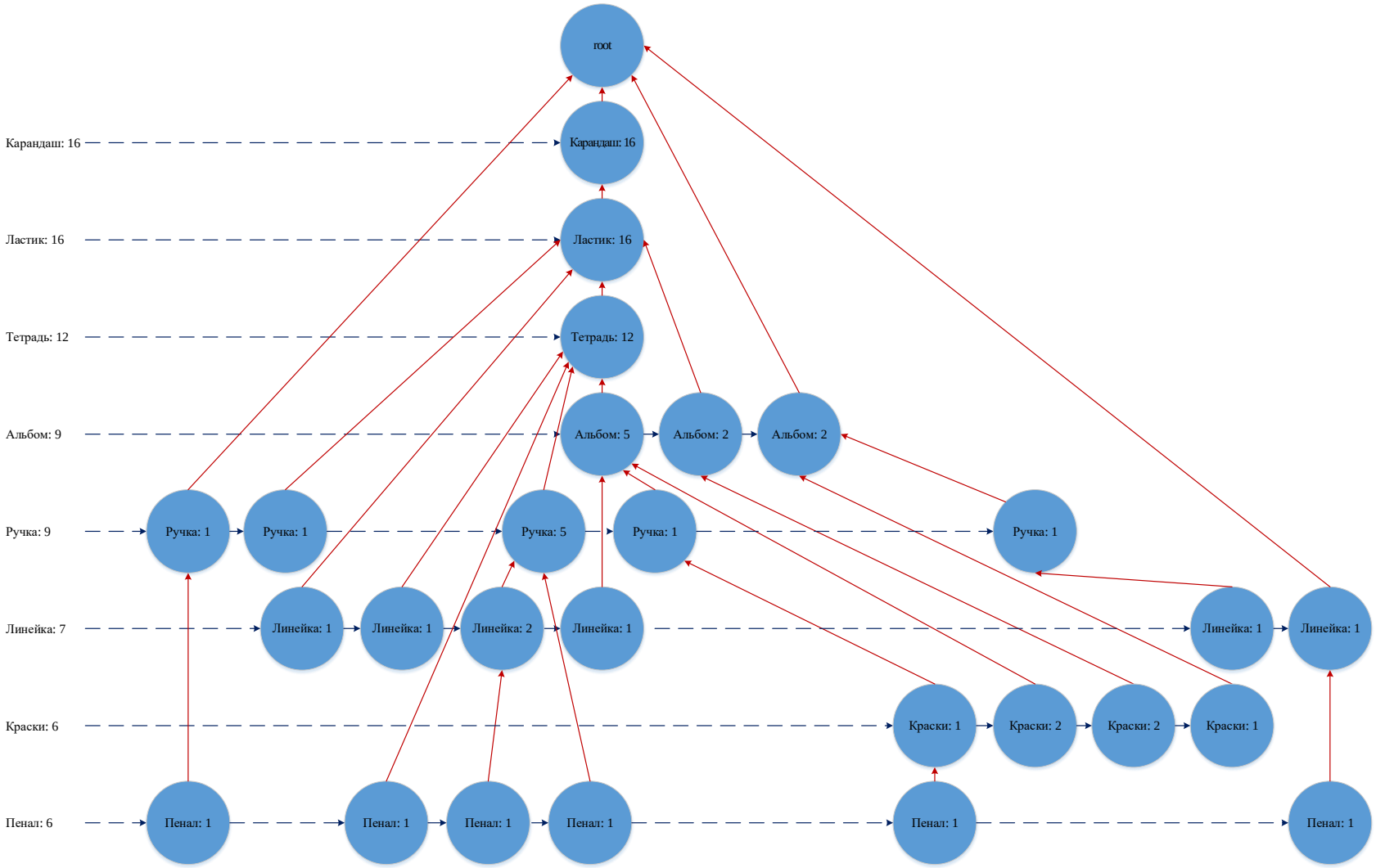


Рисунок 9. Итоговое FP-дерево

В соответствии с правилом построения FP-дерева предмет Линейка породит в FP-дереве новый узел, дочерний к узлу Ручка (рисунок 7), а предмет Пенал породит в FP-дереве новый узел, дочерний к узлу Линейка (рисунок 8).

На этом использование транзакции №2 для построения FP-дерева завершено.

Рассуждая аналогичным образом, можно обработать оставшиеся транзакции под номерами 3 – 20.

На рисунке 9 приведено итоговое FP-дерево, построенное на основе транзакций из таблицы 14.

Можно увидеть, что на любом уровне сумма частот появления одного и того же предмета в разных узлах дерева равна общей частоте этого предмета, указанной через двоеточение от названия предмета (в левой части на рисунке 9).

Таким образом, после двух сканирований БД транзакций и выполнения соответствующей обработки предметных наборов было получено итоговое FP-дерево, которое в компактном виде представляет информацию о частых предметных наборах.

Итоговое FP-дерево обычно имеет меньшие размеры, чем несжатые исходные данные, т.к. многие транзакции имеют общие предметы (и, следовательно, общие префиксы). При этом возможны следующие варианты, характеризующие размеры FP-дерева.

1. Лучший вариант: все транзации имеют один и тот же набор предметов. Тогда в FP-дереве имеется один единственный путь.

2. Худший вариант: все транзации содержат уникальные наборы предметов, т.е. у них нет общих предметов. Тогда размеры FP-дерева такие же, как размеры исходных данных, и требования к памяти – велики, т.к. необходимо хранить указатели для узлов и счетчики.

Размеры FP-дерева зависят от того, как упорядочены предметы. Обычно используется упорядочение по убыванию значений поддержки, однако это не всегда приводит к получению FP-дерева наименьших размеров.

Этап 2. Извлечение частых предметных наборов из FP-дерева.

При реализации этого этапа осуществляется движение снизу вверх – от листьев FP-дерева к корню.

Поиск частых предметных наборов выполняется рекурсивно.

При этом используется парадигма «разделяй и властвуй», заключающаяся в рекурсивном разбиении решаемой задачи на две или более подзадачи того же

типа, но меньшего размера, и комбинировании их решений для получения ответа к исходной задаче.

Разбиения выполняются до тех пор, пока все подзадачи не окажутся элементарными.

Для предмета в FP-дереве, представленного узлом, можно указать путь, определяемый последовательностью узлов, через которые необходимо пройти от корневого узла до узла, связанного с этим предметом.

Если предмет представлен в нескольких ветвях FP-дерева, то таких путей будет насколько.

Например, для FP-дерева на рисунке 9 для предмета Линейка можно указать 6 путей:

{Карандаш, Ластик, Линейка},

{Карандаш, Ластик, Тетрадь, Линейка},

{Карандаш, Ластик, Тетрадь, Ручка, Линейка},

{Карандаш, Ластик, Тетрадь, Альбом, Линейка},

{Альбом, Ручка, Линейка},

{Линейка}.

Такой набор путей называется **условным базисом предмета** (*conditional base*).

Каждый путь в базисе состоит из двух частей – префикса и суффикса.

Префикс – последовательность узлов, которые образуют путь.

Префикс – любое начало пути, в том числе и пустое.

Будем говорить, что префикс является собственным, если он не совпадает с пустым путем и полным путем.

Суффикс – любой конец пути, в том числе пустой.

Будем говорить, что суффикс является собственным, если он не совпадает с пустым путем и полным путем.

Пусть суффикс – узел (возможно, лист), к которому «прокладывается» путь.

В условном базисе все пути будут иметь различные префиксы и одинаковый суффикс.

Например, в пути {Карандаш, Ластик, Линейка} в качестве префикса, несовпадающего с полным путем, можно назвать поднабор {Карандаш, Ластик|}, а в качестве суффикса – поднабор из одного предмета, т.е. однопредметный набор {Линейка}.

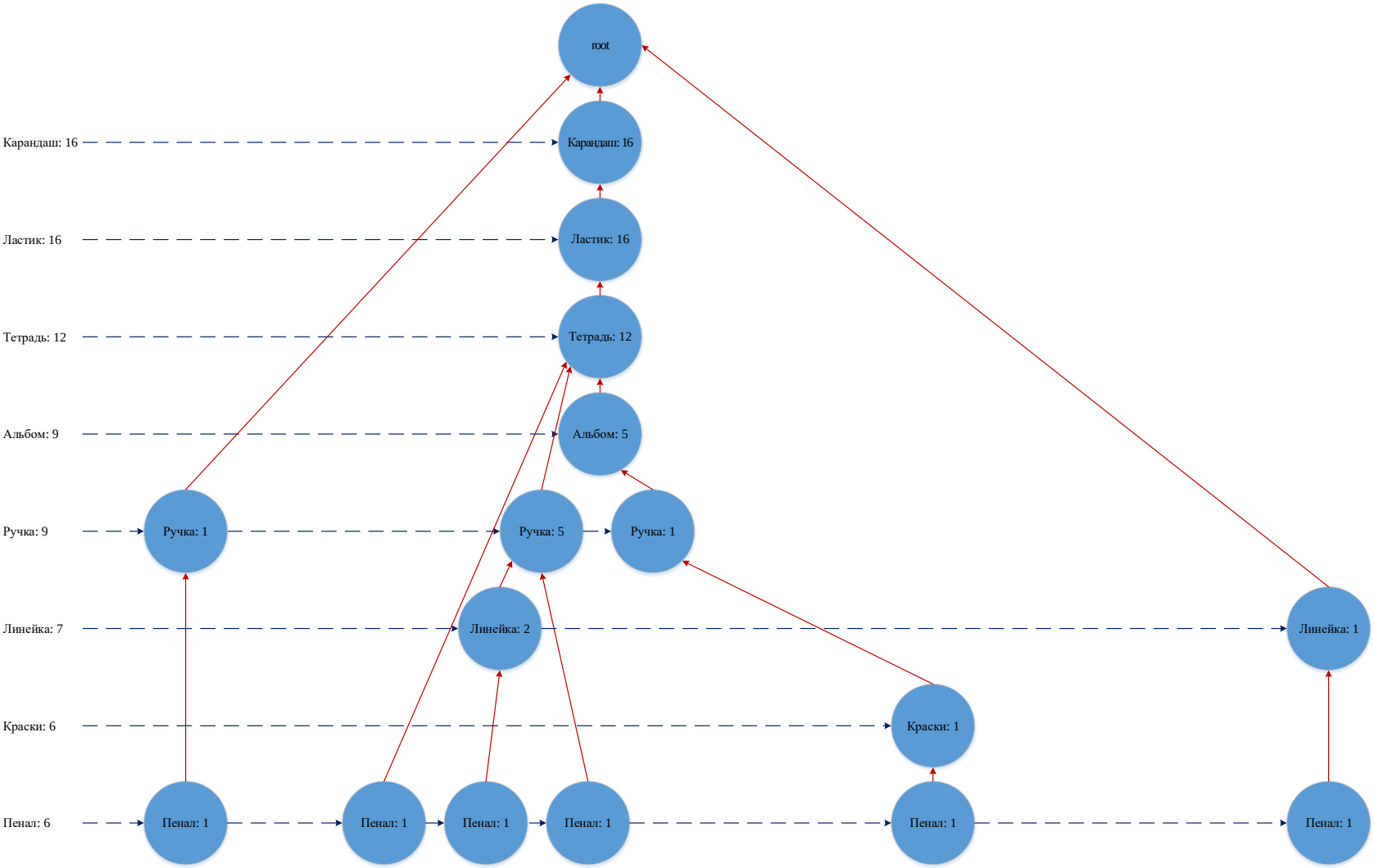


Рисунок 10. Префиксное поддерево путей, заканчивающихся предметом Пенал

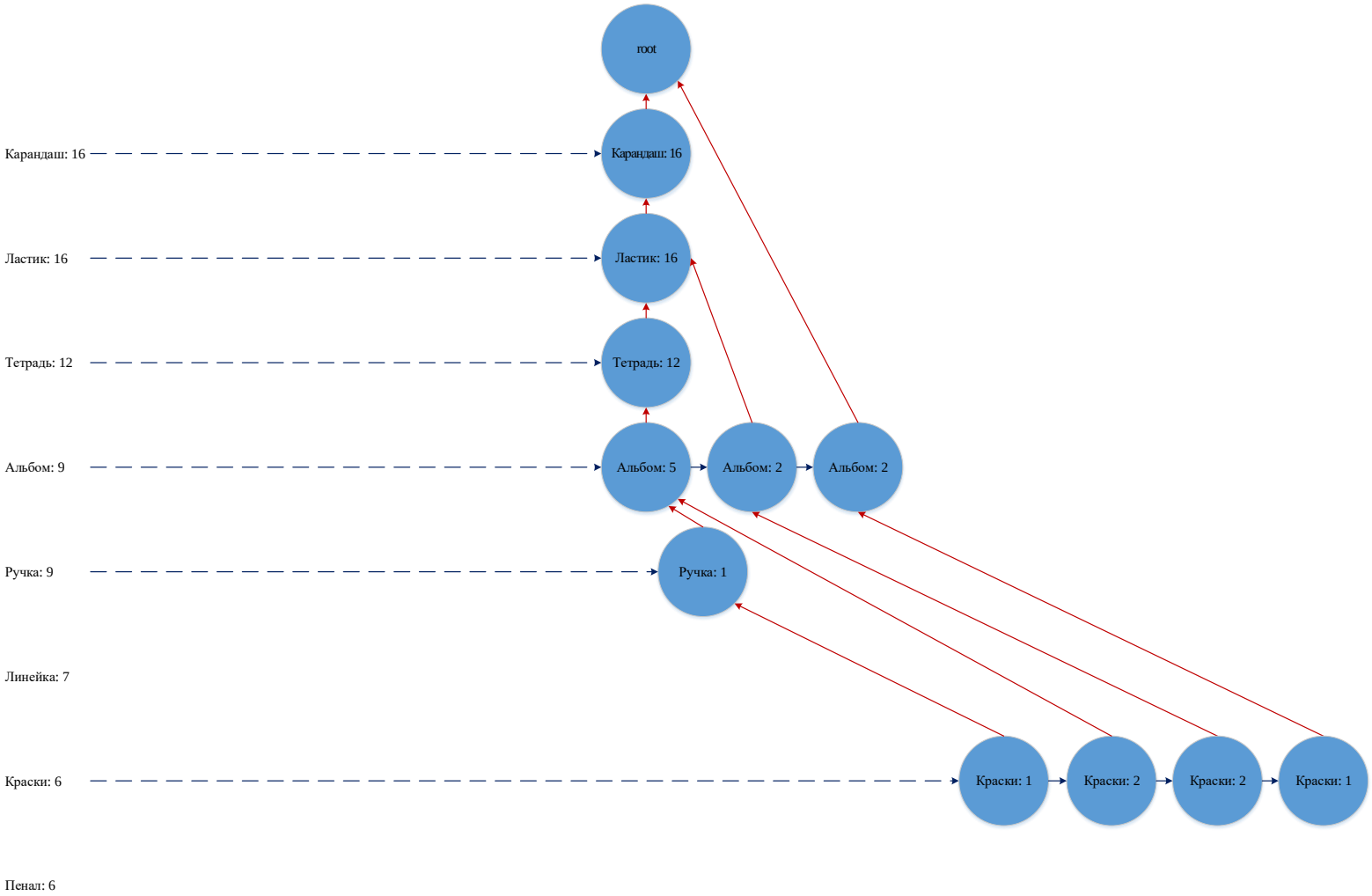


Рисунок 11. Префиксное поддерево путей, заканчивающихся предметом Краски

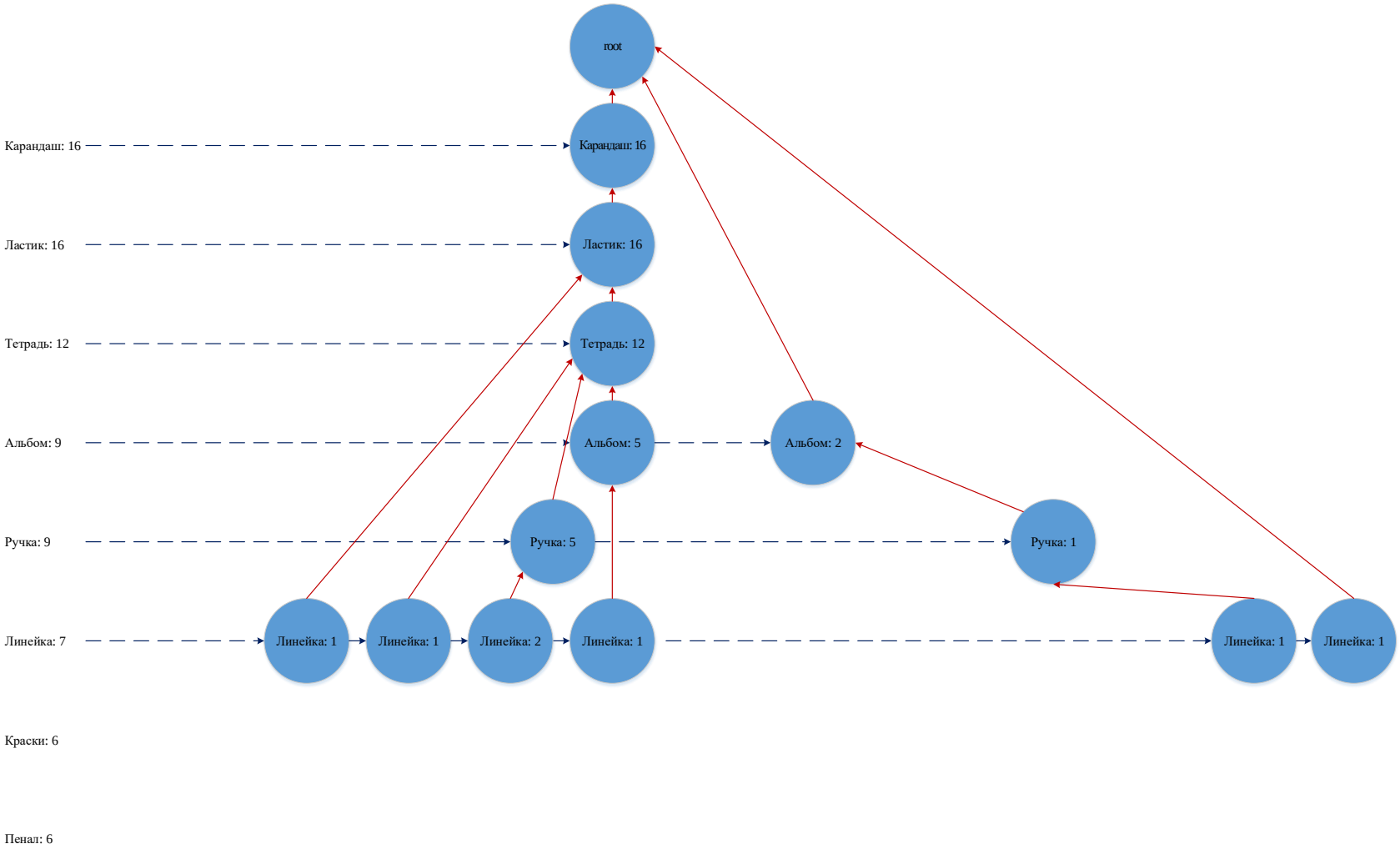


Рисунок 12. Префиксное поддерево путей, заканчивающихся предметом Линейка

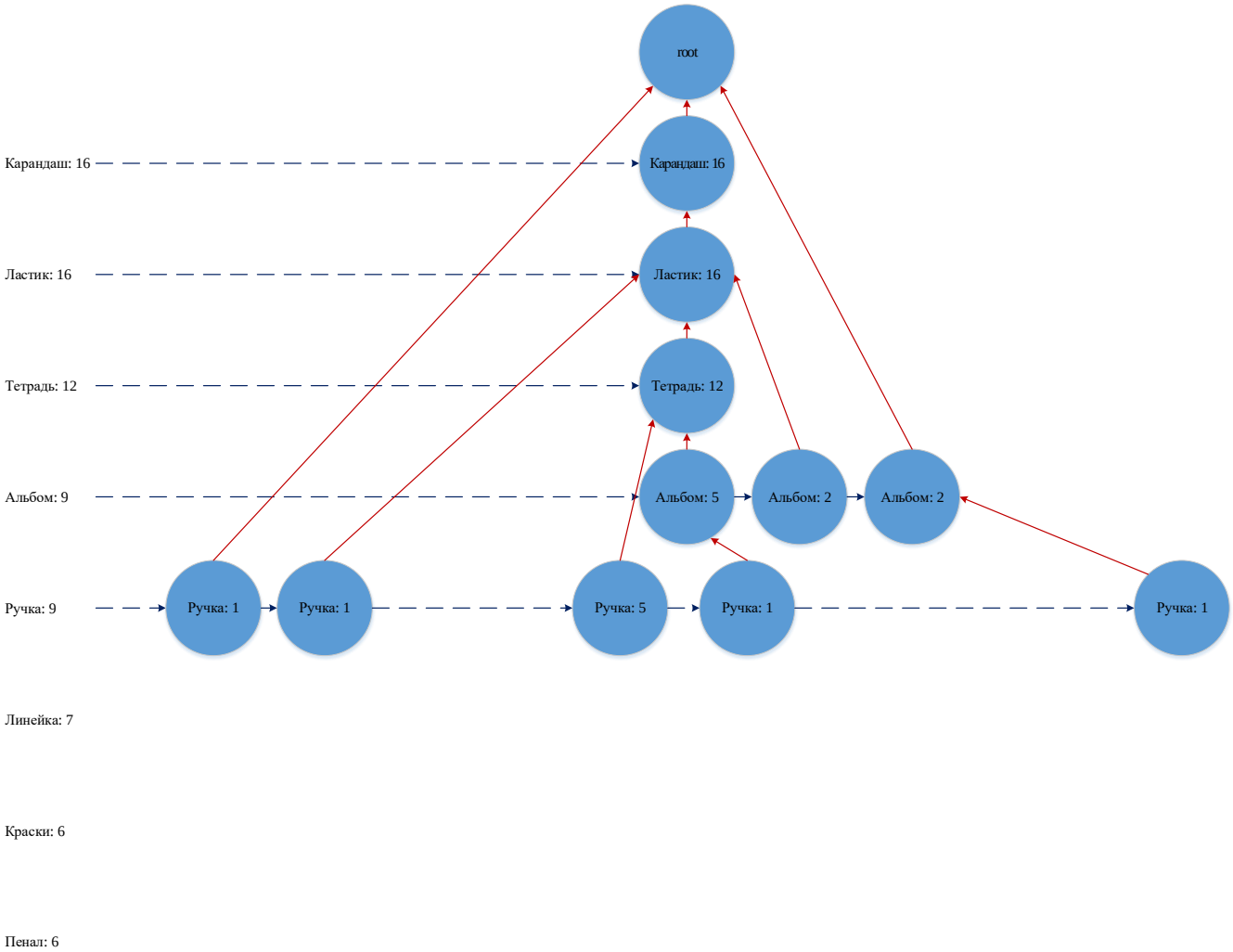


Рисунок 13. Префиксное поддерево путей, заканчивающихся предметом Ручка

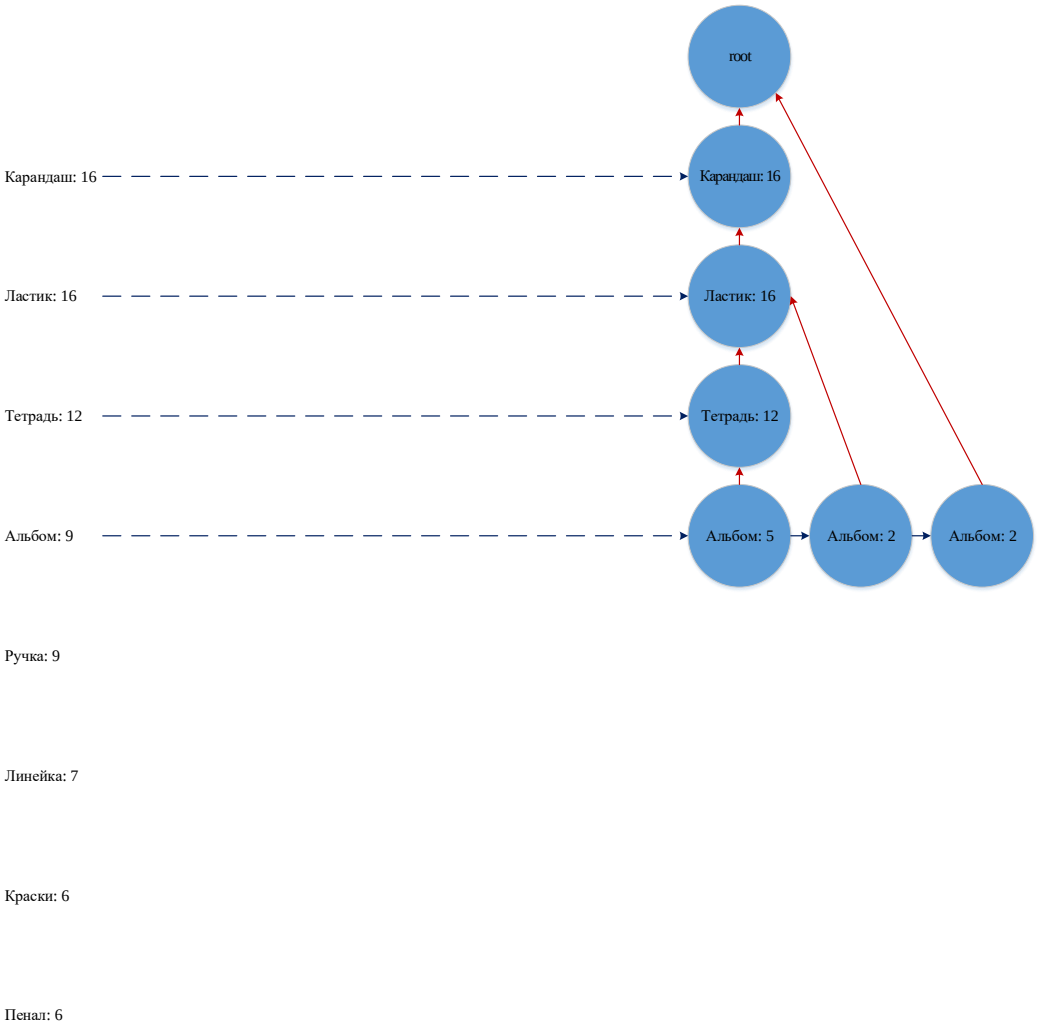


Рисунок 14. Префиксное поддерево путей, заканчивающихся предметом Альбом

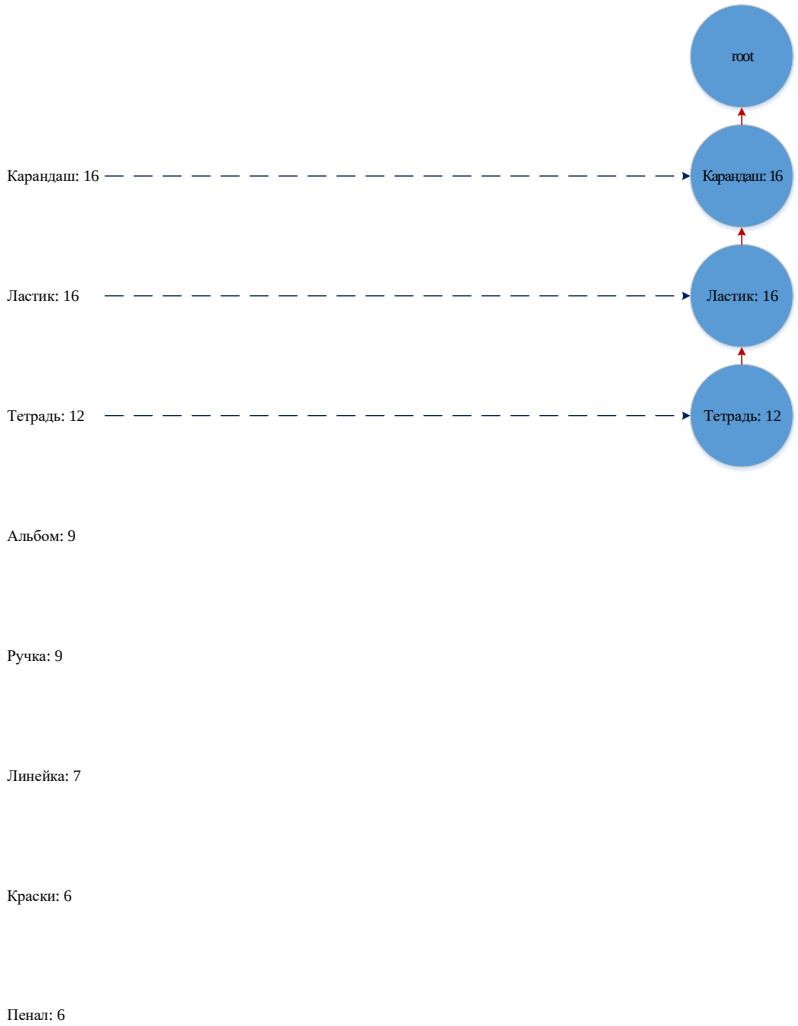


Рисунок 15. Префиксное поддерево путей, заканчивающихся предметом Тетрадь

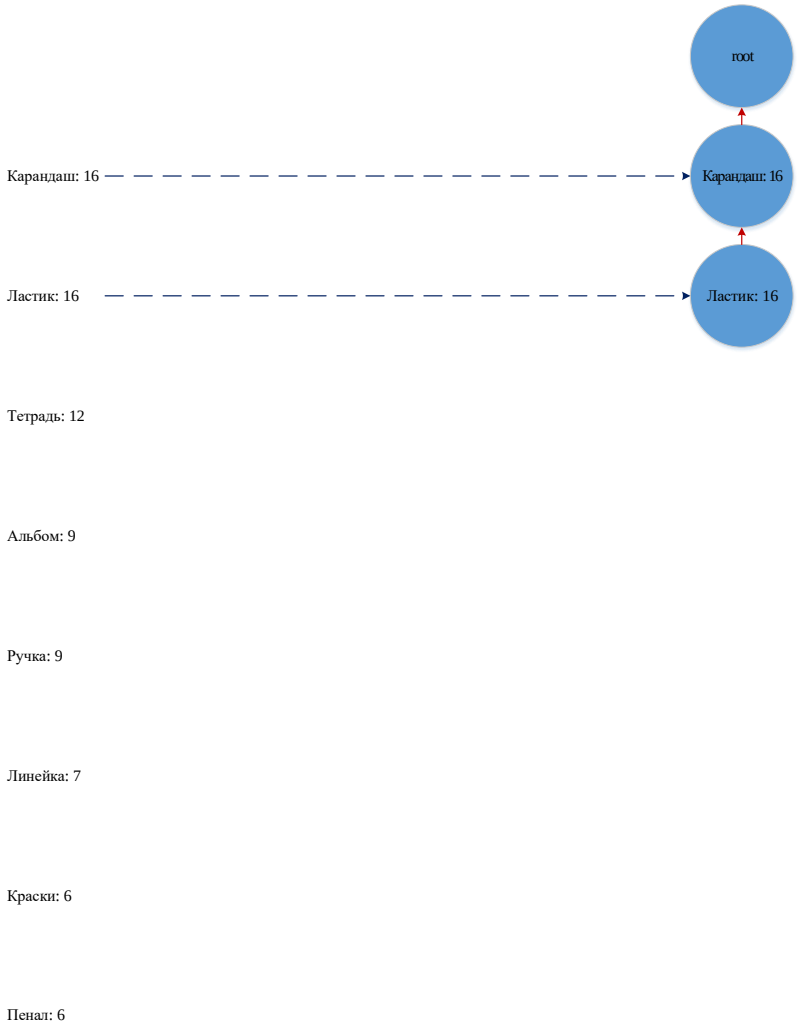


Рисунок 16. Префиксное поддерево путей, заканчивающихся предметом Ластик

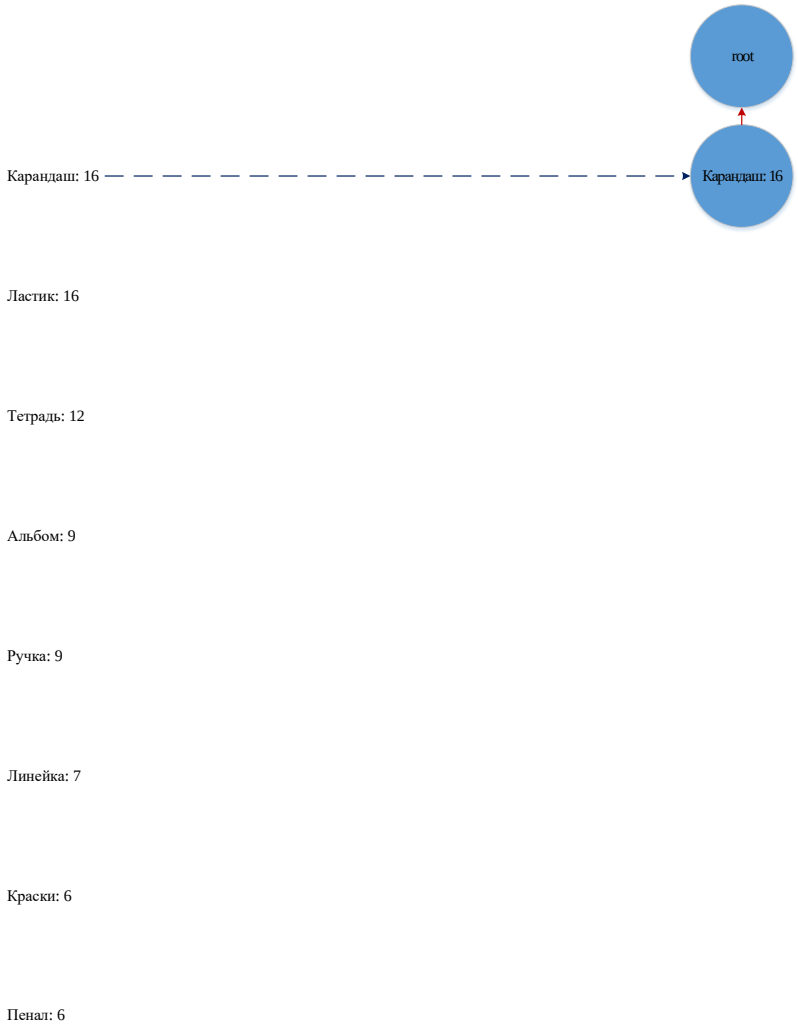


Рисунок 17. Префиксное поддерево путей, заканчивающихся предметом Карандаш

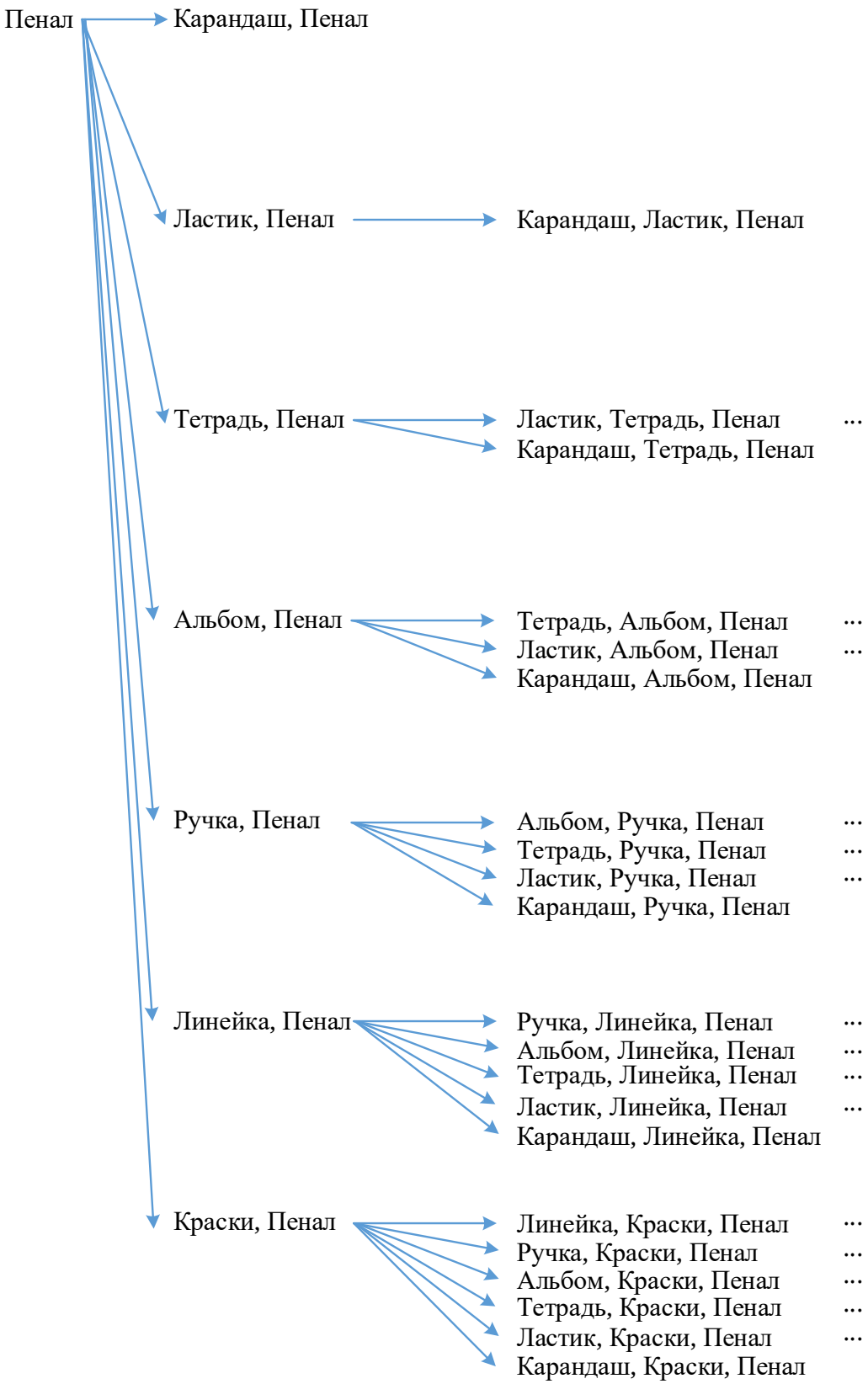


Рисунок 18. Варианты наборов предметов, потенциально определяемых путями, заканчивающимися предметом Пенал и подлежащих проверке на частость

1. Извлечем из FP-дерева префиксные поддеревья путей, заканчивающиеся предметом или набором предметов.

Сначала осуществим поиск частых предметных наборов среди наборов, определяемых путями, заканчивающимися узлами, расположенными на самом нижнем уровне путями, заканчивающимися связанными узлами, расположенными на двух нижних уровнях и т.п. Аналогичным образом выполним анализ всех путей от листьев к корню.

Например, для FP-дерева, изображенного на рисунке 9, необходимо сначала проанализировать пути, заканчивающиеся предметом Пенал, затем – предметами Краски, Пенал, затем – предметами Линейка, Пенал и т.п.

В результате можно извлечь префиксные поддеревья путей, заканчивающиеся предметом или набором предметов. При этом целесообразно опираться на односвязанные списки, применяемые для связывания узлов дерева, соответствующих одному и тому же предмету.

Так, если в рассматриваемом примере было определено 8 частых предметов, то можно выделить 8 префиксных поддеревьев путей, имеющих в качестве листьев некоторый конкретный предмет (рисунки 10 – 17).

2. Обработаем каждое префиксное поддерево путей рекурсивно для извлечения частых предметных наборов, а затем объединим найденные решения.

На рисунке 18 показаны некоторые варианты наборов предметов, потенциально определяемых путями, заканчивающимися предметом Пенал и подлежащих проверке на частость.

Рассмотрим префиксное поддерево путей для предмета Пенал (рисунок 10) и найдем все частые предметные наборы, содержащие этот предмет.

Предмет Пенал является частым, т.к. встречается в наборе транзакций 6 раз (это число можно также найти, просуммировав частоты, приписанные узлам дерева, соответствующим предмету Пенал).

Т.к. предмет Пенал является частым, проверим на частость наборы, соответствующие путям, заканчивающимся предметом Пенал. Можно выписать 7 вариантов двухпредметных наборов (рисунок 18).

В таком случае можно говорить о рекурсивной декомпозиции задачи.

Построим *условные FP-деревья* (conditional FP-tree) для частых предметов (или частых наборов предметов).

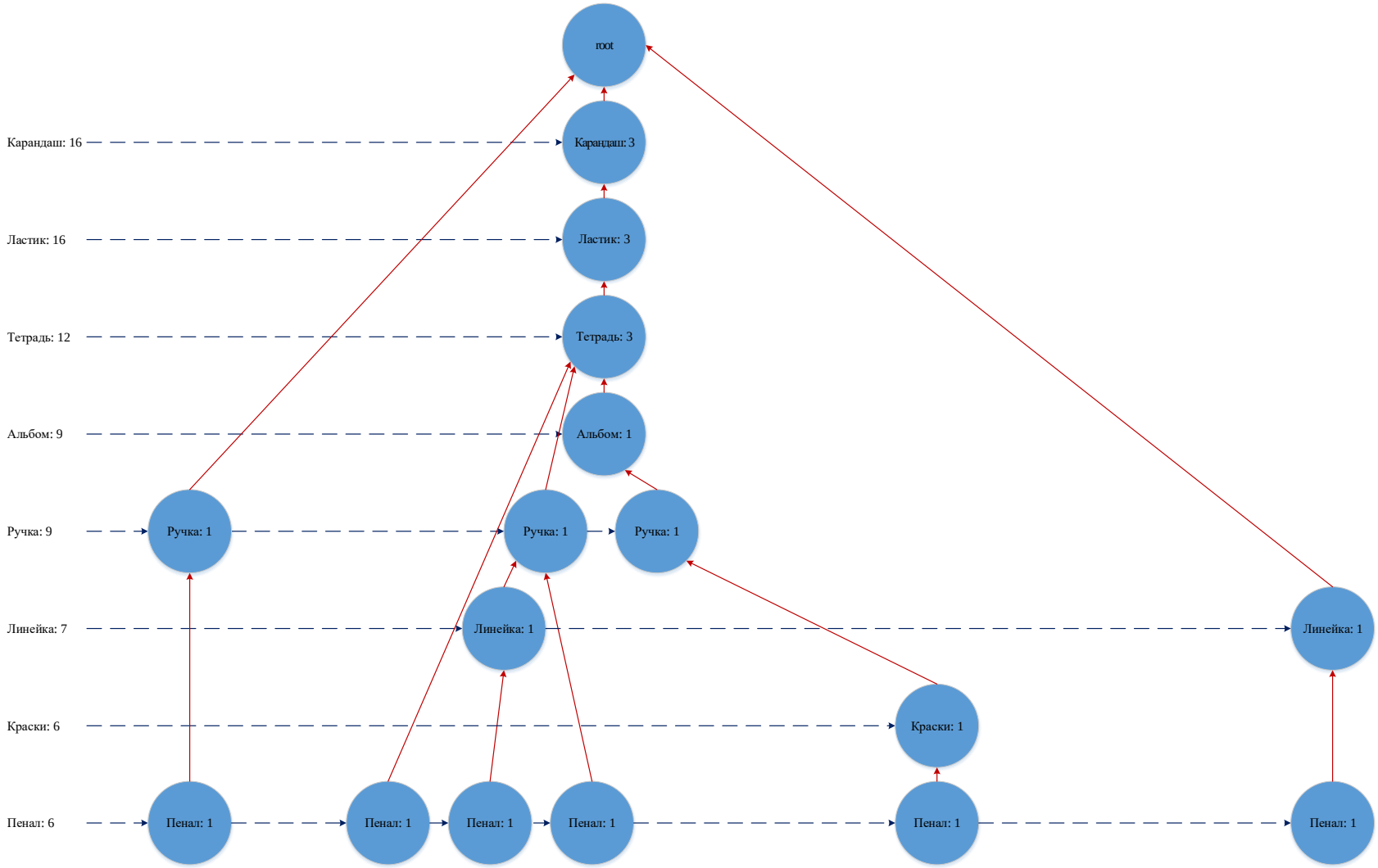


Рисунок 19. Формирование условного FP-дерева для предмета Пенал (начало): обновление значений счетчиков поддержки

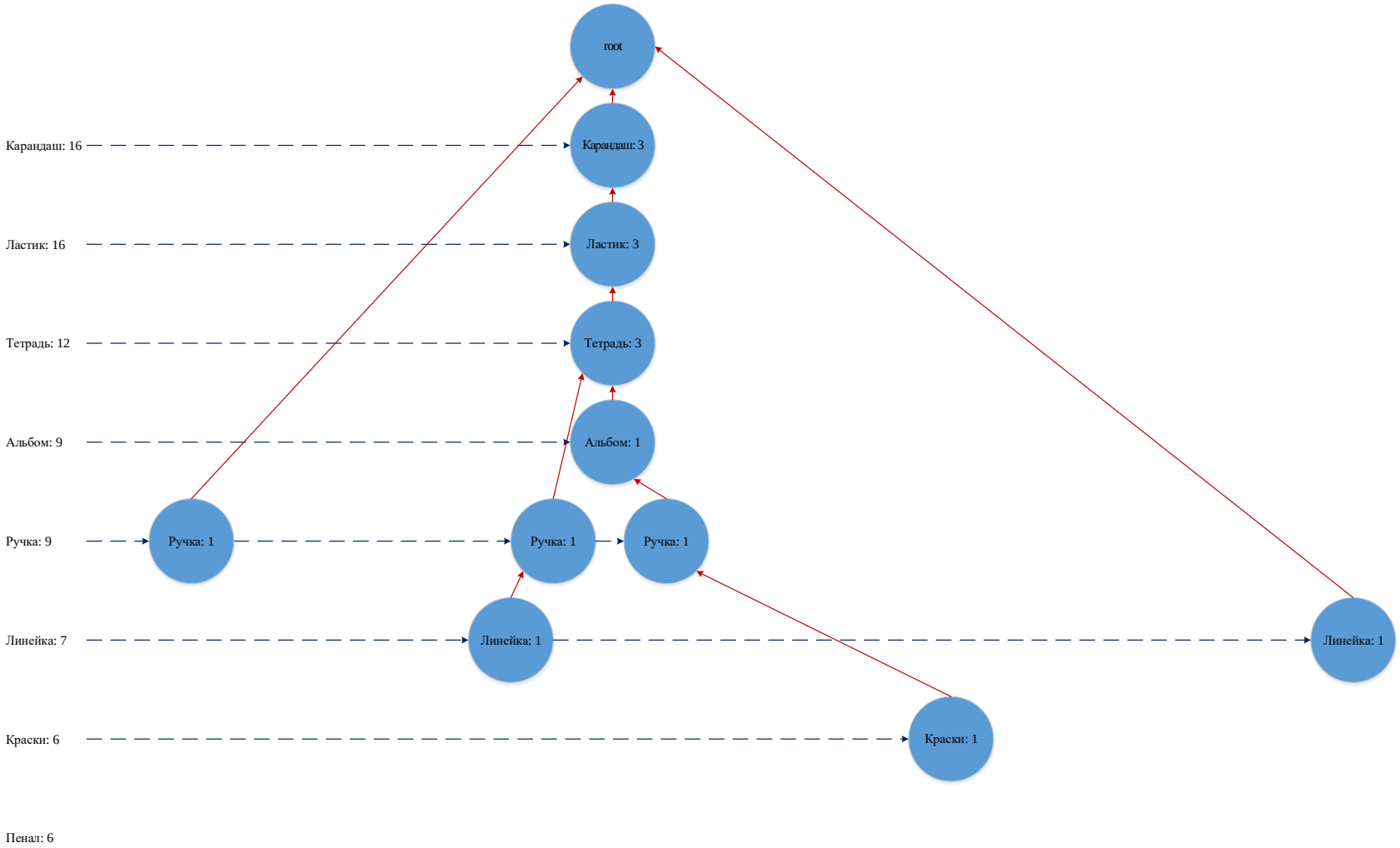


Рисунок 20. Формирование условного FP-дерева для предмета Пенал (продолжение): удаление узлов для предмета Пенал

Рисунок 21. Формирование условного FP-дерева для предмета Пенал (окончание): удаление узлов для нечастых предметов

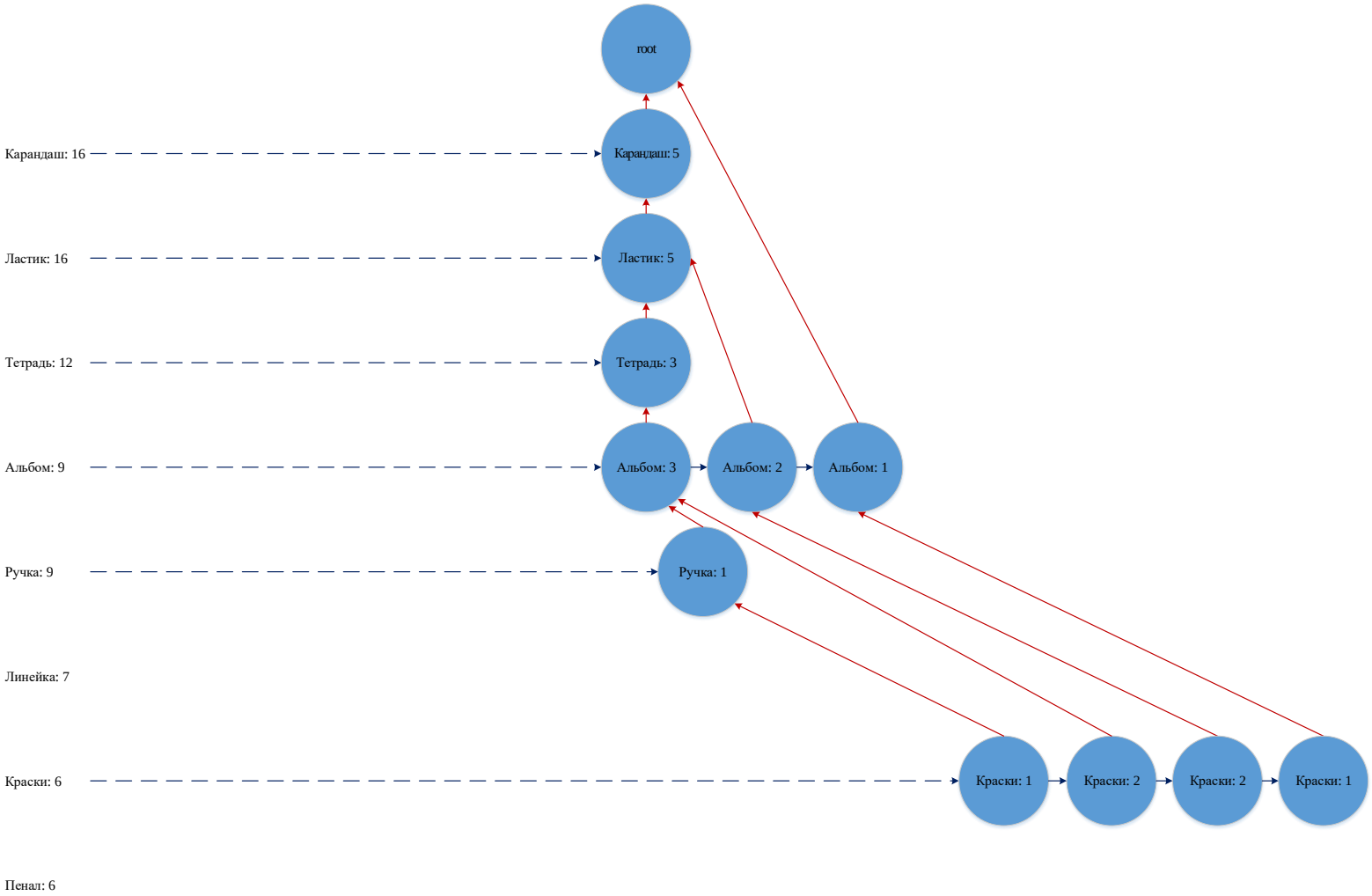


Рисунок 22. Формирование условного FP-дерева для предмета Краски (начало): обновление значений счетчиков поддержки

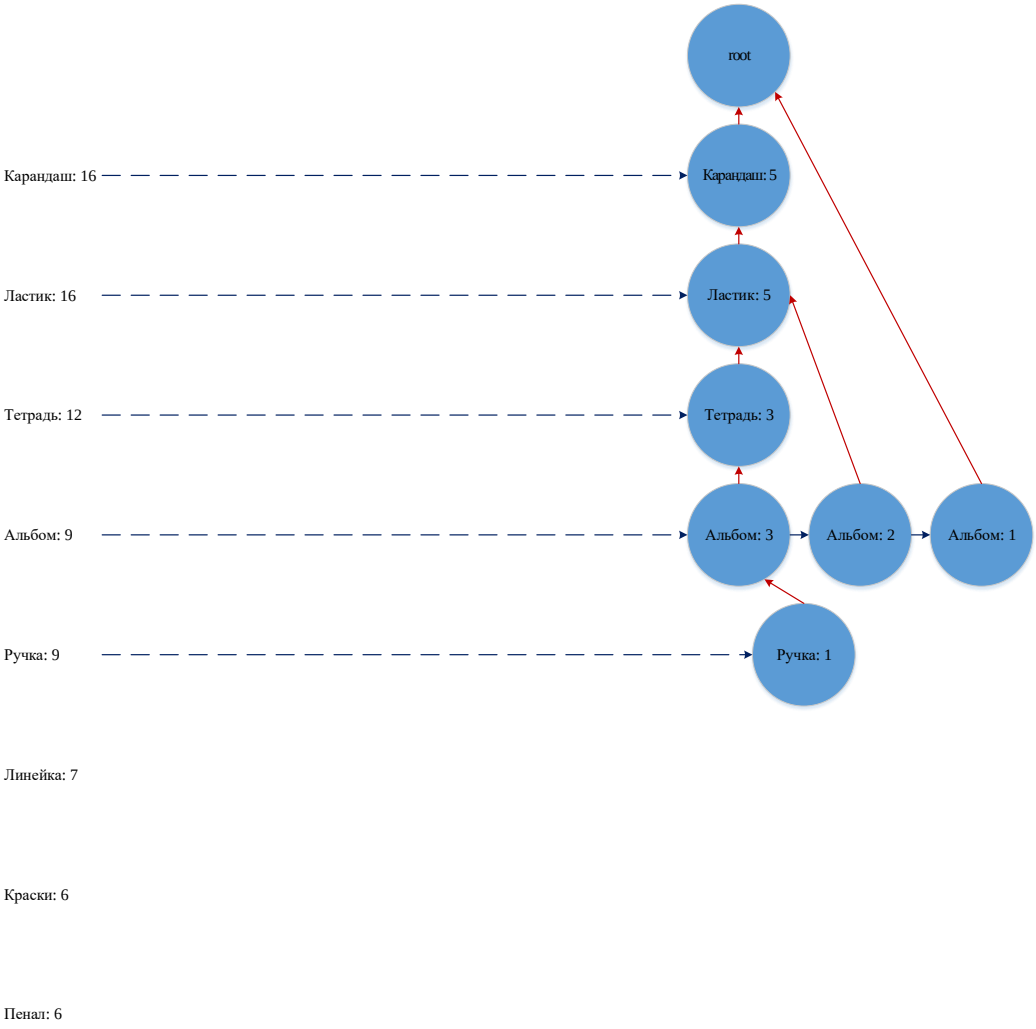


Рисунок 23. Формирование условного FP-дерева для предмета Краски (продолжение): удаление узлов для предмета Краски

Условное FP-дерево – дерево, которое построено на основе только тех транзакций, которые содержат определенный частый предмет (или частый набор предметов), при этом этот предмет (или набор предметов) удалены из дерева.

Построим, например, условное FP-дерево для предмета Пенал.

2.1. Для извлечения условного FP-дерева из префиксного дерева путей необходимо обновить счетчики поддержки вдоль префиксных путей от узла, соответствующего частому предмету (или частому набору предметов), чтобы учесть только те транзакции, которые содержат частый предмет (или частый набор предметов).

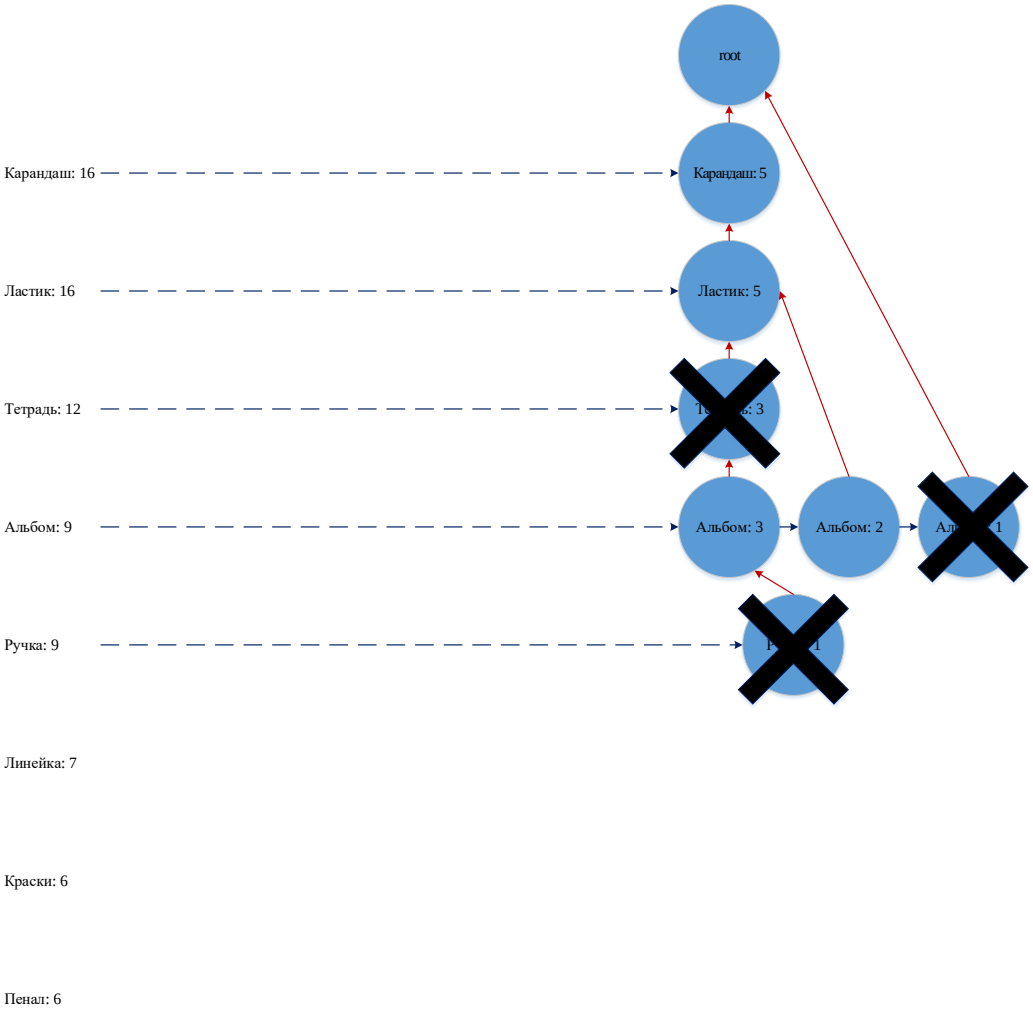


Рисунок 24. Формирование условного FP-дерева для предмета Краски (продолжение): удаление узлов для нечастых предметов

На рисунке 19 приведено префиксное дерева путей предмета Пенал с обновленными значениями счетчиков поддержки (изменены значения счетчиков поддержки у узлов с названиями предметов Карандаш, Ластик, Тетрадь, Альбом, Ручка (второй узел в ряду, считая слева), Линейка (первый узел в ряду, считая слева).

2.2. Для извлечения условного FP-дерева из префиксного дерева путей необходимо удалить все узлы, соответствующего частому предмету (или частому набору предметов).

На рисунке 20 приведено префиксное дерева путей предмета Пенал с удаленными узлами для предмета Пенал.

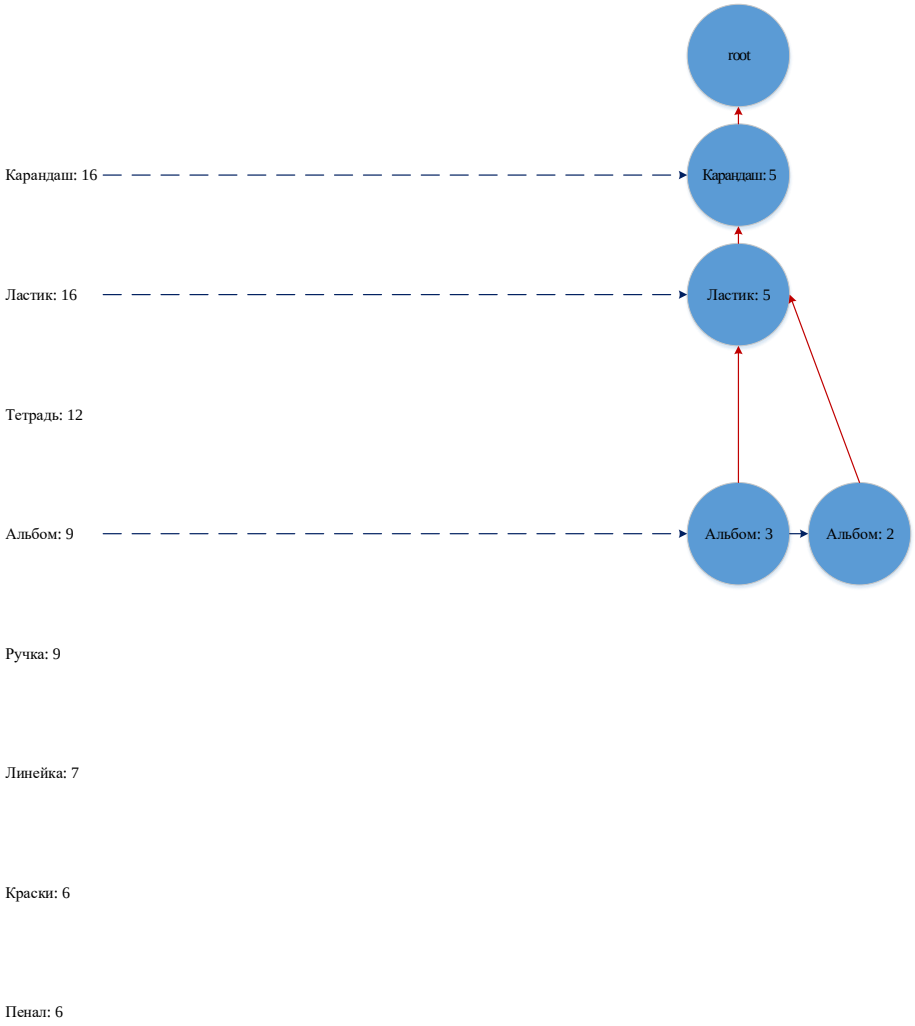


Рисунок 25. Формирование условного FP-дерева для предмета Краски (окончание): итоговое условное FP-дерево

2.3. Для извлечения условного FP-дерева из префиксного дерева путей необходимо удалить все узлы, соответствующего нечастым предметам (или нечастым наборам предметов).

На рисунке 21 приведено условное FP-дерево для предмета Пенал. Оно оказалось пустым, т.к. все предметы, соответствующие вершинам этого условного FP-дерева, не являются частыми.

Построим условные FP-деревья для остальных 7 частых предметов.

Рассмотрим предмет Краски. Построим для него условное FP-дерево.

На рисунках 22 – 25 показаны шаги формирования условного FP-дерева. При этом на рисунке 24 отмечены как вычеркнутые узлы предметов Тетрадь, Ручка, а также – узел предмета Альбом (правая ветвь от корня), т.к. суммарная поддержка которых меньше 5.

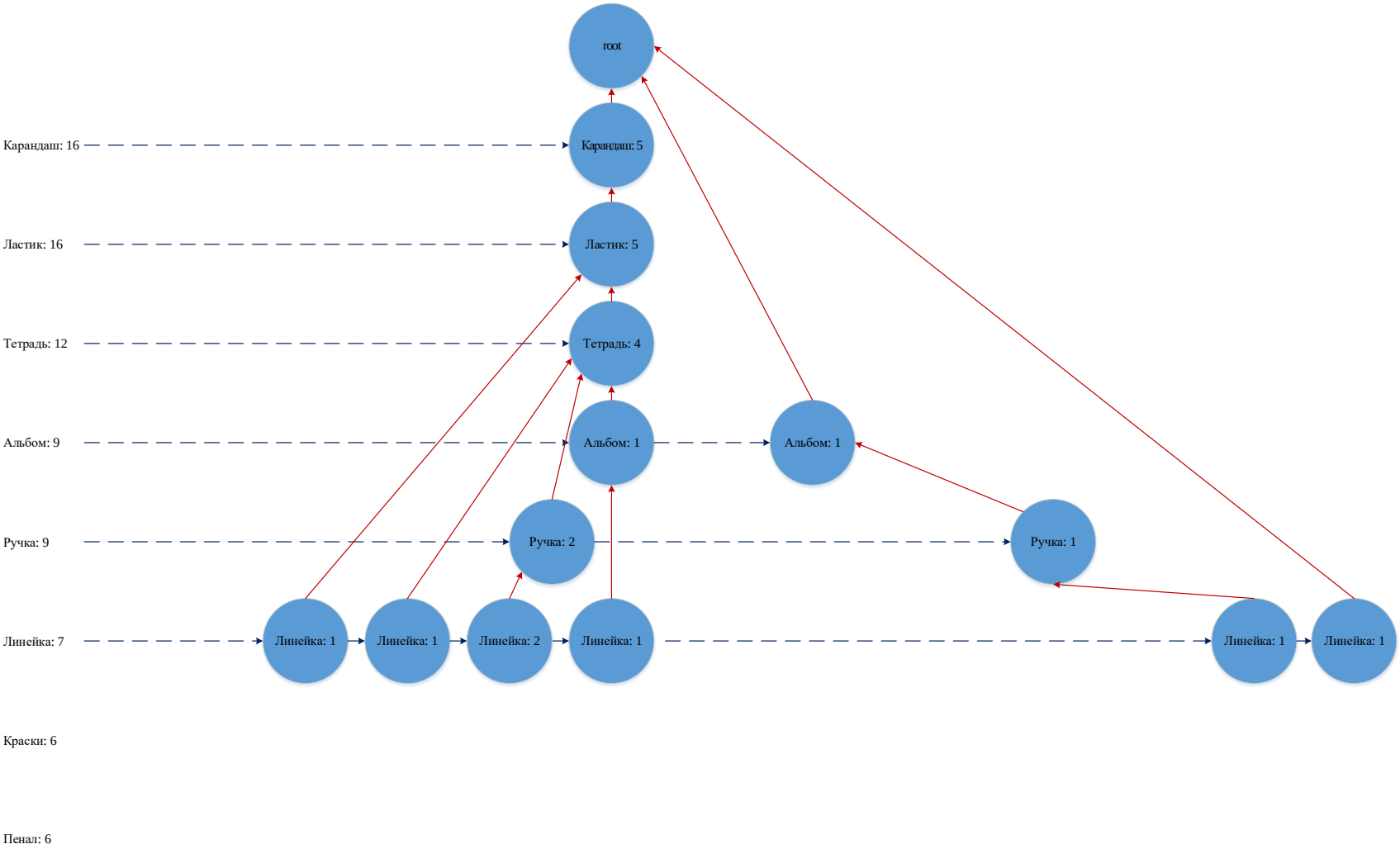


Рисунок 26. Формирование условного FP-дерева для предмета Линейка (начало): обновление значений счетчиков поддержки

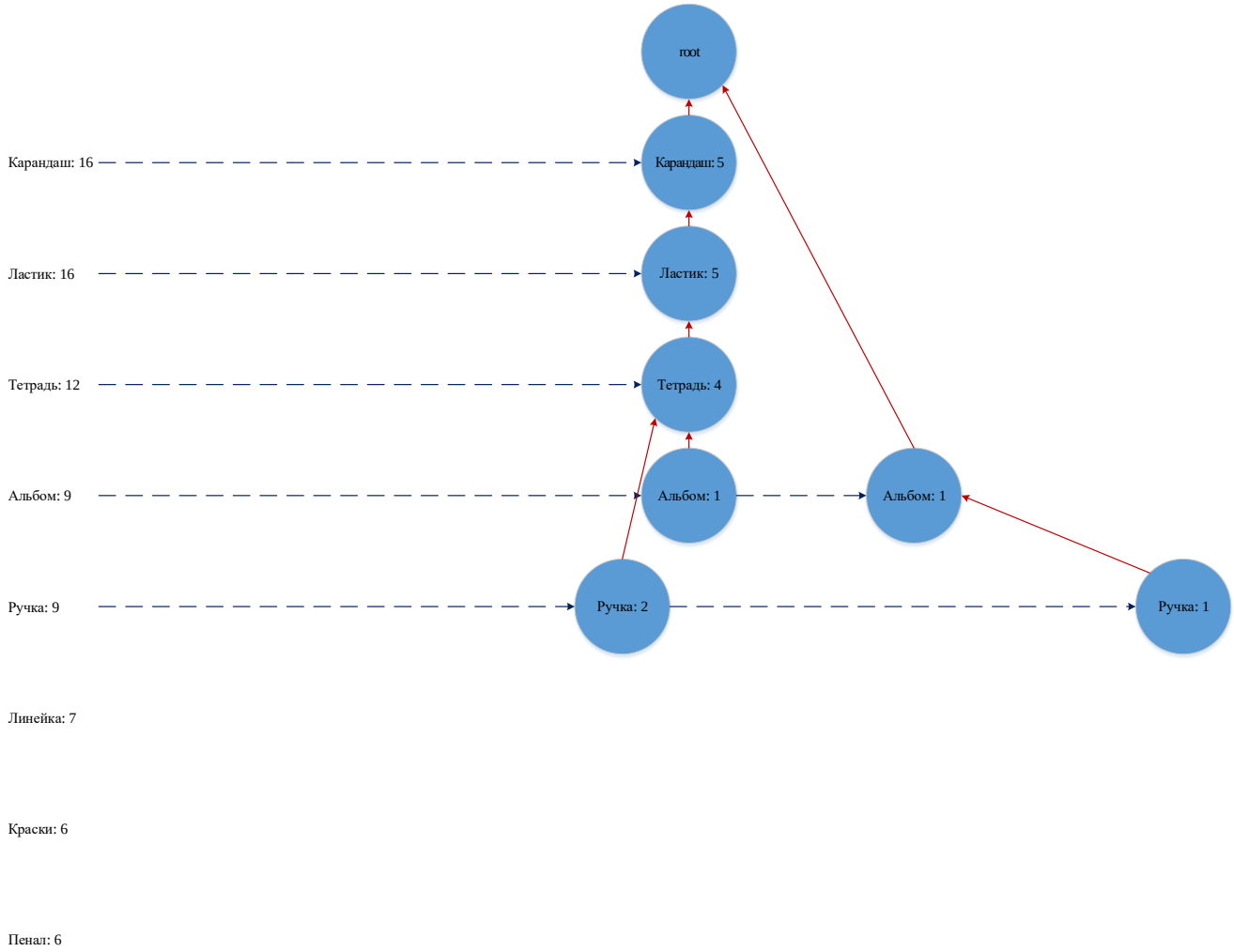


Рисунок 27. Формирование условного FP-дерева для предмета Линейка (продолжение): удаление узлов для предмета Линейка

В итоге частым предметным набором, найденным на основе условного FP-дерева для предмета Краски будет набор {Карандаш, Ластик, Альбом, Краски}.

Самый правый узел предмета Альбом, связанный с корнем дерева, удален потому, что при движении по этому пути (т.е. по пути от корня вправо) предмет Альбом не является частым.

Рассмотрим предмет Линейка. Построим для него условное FP-дерево.

На рисунках 23 – 29 показаны шаги формирования условного FP-дерева.

В итоге частым предметным набором, найденным на основе условного FP-дерева для предмета Линейка будет набор {Карандаш, Ластик, Линейка}.

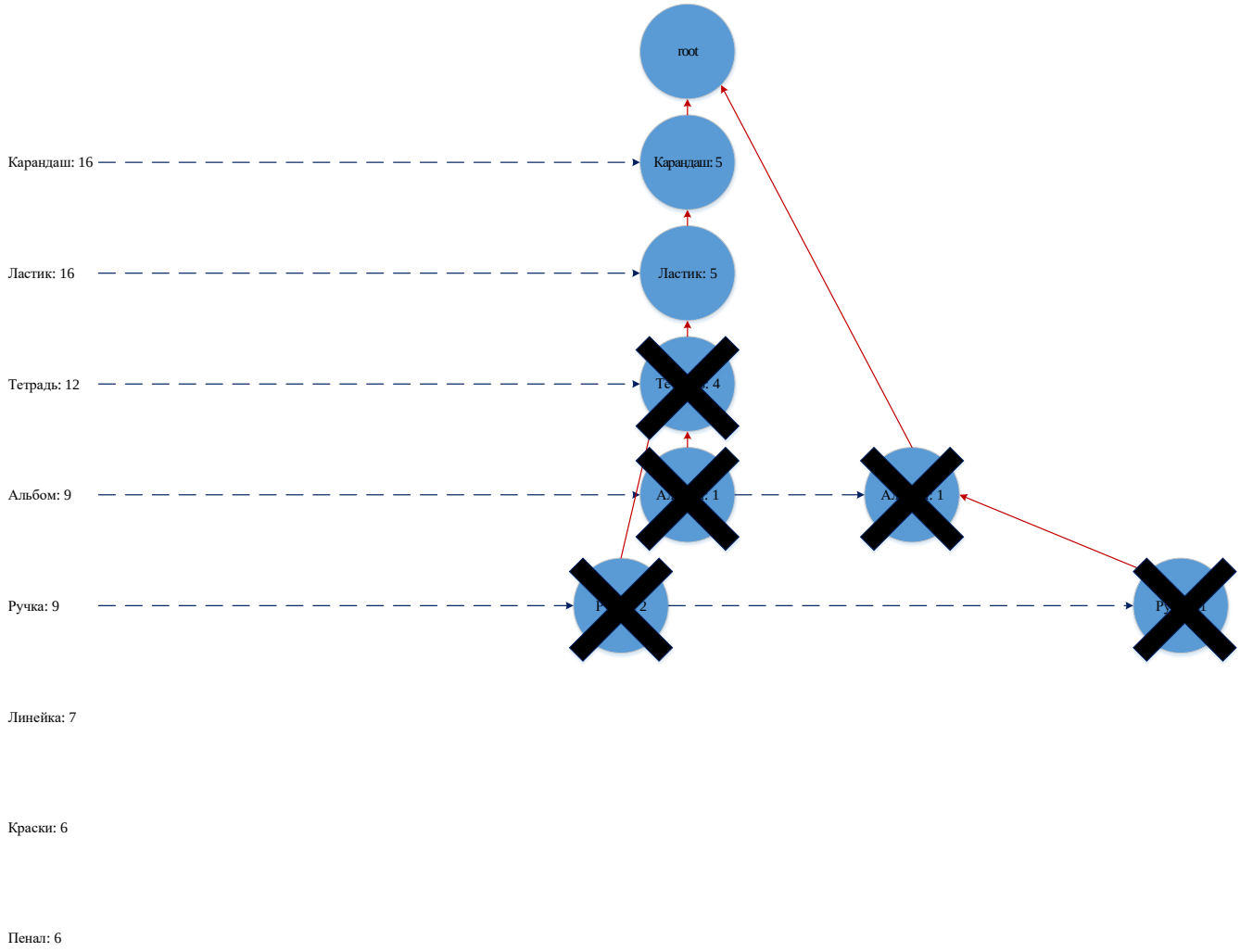


Рисунок 28. Формирование условного FP-дерева для предмета Линейка (продолжение): удаление узлов для нечастых предметов

Аналогичным образом можно выполнить формирование всех условных FP-деревьев сначала для частых однопредметных наборов на основе префиксных деревьев путей (рисунки 10 – 17), затем – для частых двухпредметных наборов и т.д., предполагая, что предметы в этих наборах соответствуют узлам деревьев в конце соответствующих путей.

В результате будут найдены все частые предметные наборы (двухпредметные, трехпредметные и четырехпредметные) в дополнение к уже найденным однопредметным частым наборам. При этом найденные частые наборы совпадут с наборами, найденными с применением алгоритма Apriori.

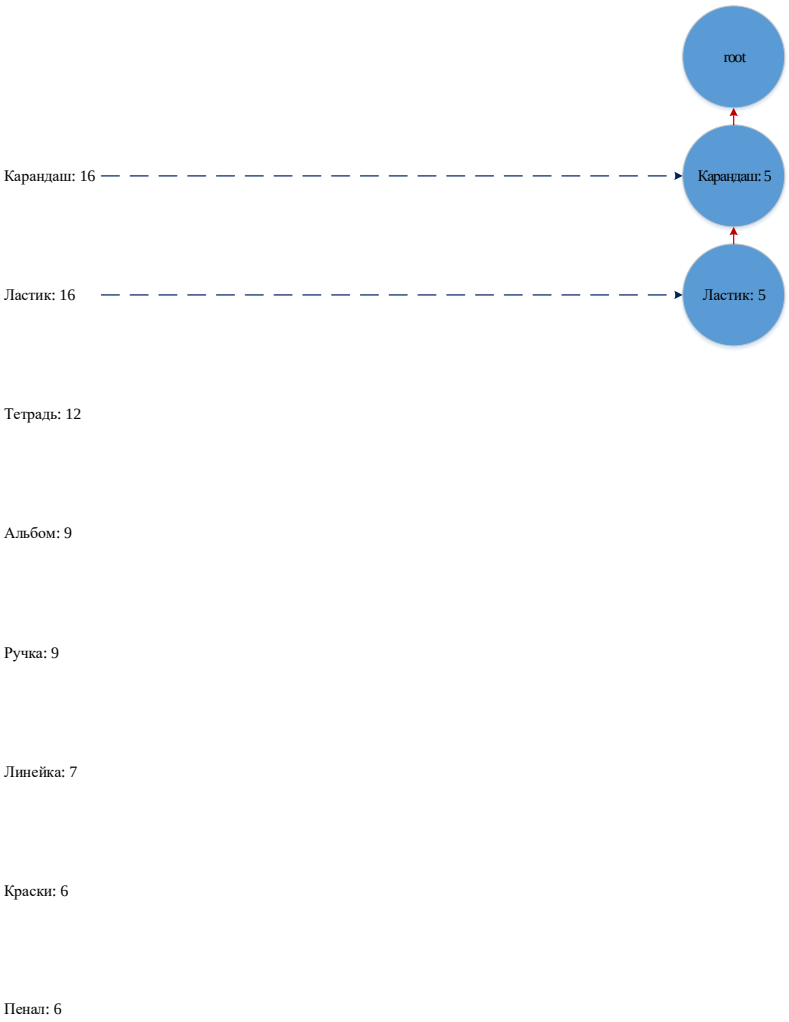


Рисунок 29. Формирование условного FP-дерева для предмета Линейка (окончание): итоговое условное FP-дерево

Множество частых двухпредметных наборов будет иметь вид:

F_2 {{Альбом, Карандаш}, {Альбом, Краски}, {Альбом, Ластик}, {Альбом, Тетрадь}, {Карандаш, Краски}, {Карандаш, Ластик}, {Карандаш, Линейка}, {Карандаш, Ручка}, { Карандаш, Тетрадь}, {Краски, Ластик}, {Ластик, Линейка}, {Ластик, Ручка}, {Ластик, Тетрадь}, {Ручка, Тетрадь}}.

Множество частых трехпредметных наборов будет иметь вид:

F_3 {{Альбом, Карандаш, Краски}, {Альбом, Карандаш, Ластик}, {Альбом, Карандаш, Тетрадь}, {Альбом, Краски, Ластик}, {Альбом, Ластик, Тетрадь}, {Карандаш, Краски, Ластик}, {Карандаш, Ластик, Линейка}, {Карандаш, Ластик, Ручка}, {Карандаш, Ластик, Тетрадь}, {Карандаш, Ручка, Тетрадь}, {Ластик, Ручка, Тетрадь}}.

Множество частых четырехпредметных наборов будет иметь вид:

F_4 {{Альбом, Карандаш, Краски, Ластик},

{Альбом, Карандаш, Ластик, Тетрадь},

{Карандаш, Ластик, Ручка, Тетрадь}}.

Сравнительный анализ результатов работы алгоритмов Apriori и FPGrowth показывает, что с увеличением числа транзакций в БД временные затраты на поиск частых предметных наборов растут для алгоритма FPGrowth намного медленнее, чем для алгоритма Apriori.

4. Алгоритм Apriori на языке Python

Для демонстрации принципов работы библиотек, реализующих поиск ассоциативных правил, будем использовать Jupyter Notebook.

На языке Python имеются различные реализации алгоритма Apriori, предполагающие, в том числе, использование отдельных функций, реализованных на других языках программирования, например, на C++, что существенным образом может влиять на общее время реализации алгоритма Apriori, затрачиваемое на выявление кандидатов в частые предметные наборы, самих частых предметных наборов, а также – искомых ассоциативных правил.

Рассмотрим, в частности, работу с двумя реализациями алгоритма Apriori, предложенными по ссылкам:

https://pypi.org/project/apriori-python/

и

https://pypi.org/project/efficient-apriori/.

4.1. Программная реализация apriori-python

Рассмотрим работу с программной реализацией алгоритма Apriori, предложенной по ссылке [8]:

https://pypi.org/project/apriori-python/.

Эта программная реализация алгоритма Apriori позиционируется как реализация авторского алгоритма Apriori [1] *чисто* на языке Python [9, 10] и именуется авторами как `apriori-python`.

Для установки библиотеки, обеспечивающей работу с программной реализацией `apriori-python`, необходимо выполнить команду:

```
pip install apriori_python
```

```
from apriori_python import apriori
transactions = [['Тетрадь', 'Ручка', 'Краски'],
                ['Тетрадь', 'Ручка', 'Кисть'],
                ['Краски', 'Ручка', 'Ластик']]
freqItemSet, rules = apriori(transactions, minSup=0.5, minConf=0.5)
print(rules)
```

```
[[{'Ручка'}, {'Тетрадь'}, 0.6666666666666666], [{'Ручка'}, {'Краски'}, 0.6666666666666666],
[{'Тетрадь'}, {'Ручка'}, 1.0], [{'Краски'}, {'Ручка'}, 1.0]]
```

Рисунок 30. Пример программного кода, реализующего поиск ассоциативных правил с применением программной реализации `apriori-python`

```
type(freqItemSet)
```

```
dict
```

```
freqItemSet
```

```
{1: {frozenset({'Краски'}), frozenset({'Ручка'}), frozenset({'Тетрадь'})},
 2: {frozenset({'Ручка', 'Тетрадь'}), frozenset({'Краски', 'Ручка'})}}
```

```
type(rules)
```

```
list
```

```
rules
```

```
[[{'Ручка'}, {'Тетрадь'}, 0.6666666666666666],
 [{'Ручка'}, {'Краски'}, 0.6666666666666666],
 [{'Тетрадь'}, {'Ручка'}, 1.0],
 [{'Краски'}, {'Ручка'}, 1.0]]
```

Рисунок 31. Примеры, позволяющие определить тип и содержимое выходных параметров в программной реализации `apriori-python`

```
rules[0]
```

```
[{'Ручка'}, {'Тетрадь'}, 0.6666666666666666]
```

```
rules[0][0]
```

```
{'Ручка'}
```

```
rules[0][1]
```

```
{'Тетрадь'}
```

```
rules[0][2]
```

```
0.6666666666666666
```

Рисунок 32. Пример обращения к компонентам списка, определяющим первое правило в программной реализации `apriori-python`

На рисунке 30 приведен пример программного кода, реализующего поиск ассоциативных правил с применением программной реализации apriori-python для демонстрационного набора из 3 транзакций:

```
transactions= [['Тетрадь', 'Ручка', 'Краски'],
               ['Тетрадь', 'Ручка', 'Кисть'],
               ['Краски', 'Ручка', 'Ластик']]
```

При этом предварительно осуществляется импорт функции apriori, реализующей алгоритм:

```
from Apriori_python import Apriori
```

При вызове функции apriori необходимо указать в списке её параметров список transactions, содержащий набор транзакций, а также параметры, определяющие пороговые (минимальные) значения поддержки minSup и достоверности minConf, присвоив им некоторые значения, например,

```
minSup=0.5, minConf=0.5.
```

Следует отметить, что именно эти значения предполагается использовать по умолчанию в программной реализации алгоритма.

В результате в качестве выходных параметров будут сформированы частые предметные наборы freqItemSet, а также ассоциативные правила rules, удовлетворяющие заданным пороговым значениям поддержки и достоверности.

На рисунке 31 приведены примеры, позволяющие определить тип и содержимое выходных параметров в программной реализации apriori-python. Так, freqItemSet – словарь, а rules – список.

На рисунке 32 приведен пример обращения к компонентам списка, определяющим первое ассоциативное правило. При этом можно обратиться к условию, следствию и достоверности правила как к отдельному элементу.

Найденные ассоциативные правила можно, например, вывести на печать командой

```
print(rules)
```

При этом для каждого ассоциативного правила будет выведено значение его достоверности.

Для ознакомления с программной реализацией алгоритма следует перейти по ссылке:

https://github.com/chonyy/apriori_python.

В программной реализации apriori_python задействовано много функций, однако наибольший интерес представляет функция apriori().

Функция apriori() возвращает частые предметные наборы и ассоциативные правила, удовлетворяющие заданным значениям минимальной поддержки minSup и минимальной достоверности minConf.

Функция apriori() предполагает, что набор транзакций представлен в виде списка.

Ниже приведена краткая информация по функции apriori.

Функция apriori.

Apriori-python.apriori(itemSetList, minSup, minConf)

Функция реализует классический алгоритм Apriori.

Функция работает в два этапа.

На этапе 1 формируются частые предметные наборы с заданным значением *минимальной поддержки* minSup.

На этапе 2 строятся ассоциативные правила с заданным значением *минимальной достоверности* minConf на основе частых предметных наборов, найденных на этапе 1.

Оба этапа реализуются посредством исследования пространства поиска, то есть создания всех возможных предметных наборов и оценки их поддержки. При этом удается эффективно сократить пространство поиска, решая предварительно, имеет ли предметный набор желаемую поддержку, прежде чем выполнять итерацию по всему набору данных и проверку.

Входные параметры:

параметр itemSetList, определяющий набор транзакций;

minSup, определяющий минимальное значение поддержки;

minConf, определяющий минимальное значение достоверности.

Выходные параметры:

параметр globalFreqItemSet, определяющий частые предметные наборы;

параметр rules, определяющий ассоциативные правила.

Следует отметить, что по аналогии с apriori() работает функция aprioriFromFile(), при этом предполагается, что набор транзакций считывается из файла.

4.2. Программная реализация efficient-apriori

Рассмотрим работу с программной реализацией алгоритма Apriori, предложенной по ссылке [11]:

https://pypi.org/project/efficient-apriori/.

Эта программная реализация алгоритма Apriori позиционируется как эффективная реализация авторского алгоритма Apriori [1] *чисто* на языке Python [12, 13] и именуется авторами как `Efficient Apriori` или `efficient-apriori`.

Для установки библиотеки, обеспечивающей работу с программной реализацией `efficient-apriori`, необходимо выполнить команду:

```
pip install efficient-apriori
```

На рисунке 33 приведен пример программного кода, реализующего поиск ассоциативных правил с применением программной реализации `apriori-python` для демонстрационного набора из 3 транзакций.

При этом предварительно осуществляется импорт функции `apriori`, реализующей алгоритм:

```
from efficient_Apriori import Apriori
```

При этом можно заменить некоторые отличия в названии входных параметров, а также в оформлении выведенных правил (по сравнению с программной реализацией `Apriori-python`).

На рисунке 34 приведены примеры, позволяющие определить тип и содержимое выходных параметров в программной реализации `efficient-Apriori`. Так, `freqItemSet` – словарь, а `rules` – список, но их организация несколько отлична от организации аналогичных параметров в программной реализации `Apriori-python`. В частности, значение достоверности для правила не хранится в выходном параметре `rules`.

На рисунке 35 приведен пример обращения к компонентам списка, определяющим первое ассоциативное правило.

При этом не возможно обратиться напрямую к условию или следствию ассоциативного правила как к отдельному элементы (в отличие от в программной реализации `Apriori-python`).

Найденные ассоциативные правила можно, например, вывести на печать командой

```
print(rules)
```

```
from efficient_apriori import apriori
transactions = [['Тетрадь', 'Ручка', 'Краски'],
                ['Тетрадь', 'Ручка', 'Кисть'],
                ['Краски', 'Ручка', 'Ластик']]
freqItemSet, rules = apriori(transactions, min_support=0.5, min_confidence=0.5)
print(rules)
```

```
[{Ручка} -> {Краски}, {Краски} -> {Ручка}, {Тетрадь} -> {Ручка}, {Ручка} -> {Тетрадь}]
```

Рисунок 33. Пример программного кода, реализующего поиск ассоциативных правил с применением программной реализации `efficient-apriori`

```
type(freqItemSet)
```

```
dict
```

```
freqItemSet
```

```
{1: {('Тетрадь',): 2, ('Краски',): 2, ('Ручка',): 3},
 2: {('Краски', 'Ручка'): 2, ('Ручка', 'Тетрадь'): 2}}
```

```
type(rules)
```

```
list
```

```
rules
```

```
[{Ручка} -> {Краски},
 {Краски} -> {Ручка},
 {Тетрадь} -> {Ручка},
 {Ручка} -> {Тетрадь}]
```

Рисунок 34. Примеры, позволяющие определить тип и содержимое выходных параметров в программной реализации `efficient-apriori`

```
rules[0]
```

```
{Ручка} -> {Краски}
```

Рисунок 35. Пример обращения к компоненте списка, определяющей первое ассоциативное правило в программной реализации `efficient-apriori`

```
rules_rhs = filter(lambda rule: len(rule.lhs) == 1 and len(rule.rhs) == 1, rules)
for rule in sorted(rules_rhs, key=lambda rule: rule.confidence):
  print(rule)
```

```
{Ручка} -> {Краски} (conf: 0.667, supp: 0.667, lift: 1.000, conv: 1.000)
{Ручка} -> {Тетрадь} (conf: 0.667, supp: 0.667, lift: 1.000, conv: 1.000)
{Краски} -> {Ручка} (conf: 1.000, supp: 0.667, lift: 1.000, conv: 0.000)
{Тетрадь} -> {Ручка} (conf: 1.000, supp: 0.667, lift: 1.000, conv: 0.000)
```

Рисунок 36. Пример вывода правил, удовлетворяющих заданным условиям, в программной реализации `efficient-apriori`

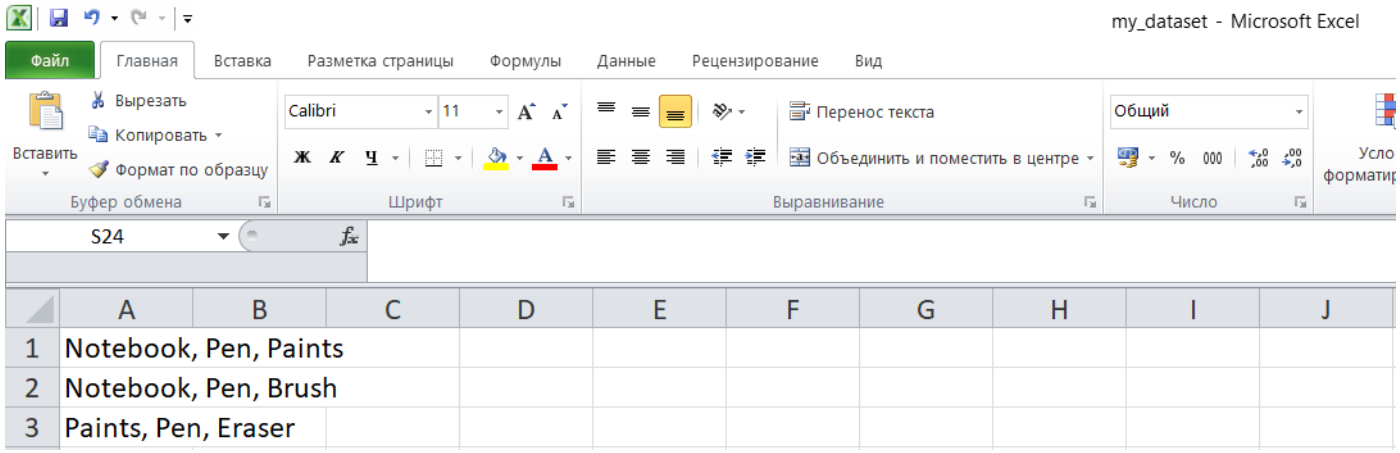


Рисунок 37. Фрагмент тестового csv-файла с данными

```
def data_generator(filename):
    """
    Data generator
    """
    def data_gen():
        with open(filename) as file:
            for line in file:
                yield tuple(k.strip() for k in line.split(','))

    return data_gen

transactions = data_generator('my_dataset.csv')
freqItemSet, rules = apriori(transactions, min_support=0.6, min_confidence=0.6)
```

Рисунок 38. Фрагмент программного кода, использующего в своей работе генератор

```
type(transactions)
```

```
function
```

```
print(freqItemSet)
```

```
{1: {('Pen',): 3, ('Paints',): 2, ('Notebook',): 2}, 2: {('Notebook', 'Pen'): 2, ('Paints', '
Pen'): 2}}
```

```
print(rules)
```

```
[{Pen} -> {Notebook}, {Notebook} -> {Pen}, {Pen} -> {Paints}, {Paints} -> {Pen}]
```

Рисунок 39. Результаты работы программной реализации `efficient-apriori` в случае применения генератора

При этом для каждого ассоциативного правила значение его достоверности не выводится, так как оно не хранится в выходном параметре `rules` (в отличие от программной реализации `Apriori-python`).

При необходимости можно выполнить вывод только тех ассоциативных правил, которые удовлетворяют тем или иным условиям, осуществив фильтрацию и сортировку.

Например, можно наложить ограничения на длину левой (`lhs` – left hand side) и правой (rhs – right hand side) частей ассоциативного правила, то есть на число предметов в них.

Кроме того, можно осуществить, например, вывод ассоциативных правил по убыванию значений некоторого показателя, например, по убыванию значений достоверности ассоциативных правил.

На рисунке 36 приведен пример вывода правил, упорядоченных по убыванию значений достоверности правил (`rule.confidence`).

При этом число предметов в условии и следствии правила равно ровно 1 («`len(rule.lhs) == 1 and len(rule.rhs) == 1`»).

При необходимости, можно для каждого правила посмотреть, что хранится в `rule.lhs`, `rule.rhs`, `rule.confidence` (достоверность), `rule.support` (поддержка), `rule.lift` (лифт), `rule.conviction` (убежденность).

Если необходимо работать с данными больших объемов, то есть с данными, которые слишком велики для размещения в памяти, целесообразно использовать функцию, возвращающую генератор вместо списка.

На рисунке 37 приведен фрагмент тестового csv-файла с данными.

На рисунке 38 приведен фрагмент программного кода, использующего в своей работе генератор.

На рисунке 39 приведены результаты работы программной реализации `efficient-apriori` в случае применения генератора.

При этом можно заменить, что тип у переменной `transactions` стал `function` (в то время как раньше тип определялся как `list`).

Для ознакомления с программной реализацией алгоритма следует перейти по ссылке [12]:

https://github.com/tommyod/Efficient-Apriori.

Хотя алгоритм программная реализация `efficient-apriori` работает со многими функциями, наибольший интерес представляют три функции:

`apriori()`,

itemsets_from_transactions(),

generate_rules_apriori().

Функция apriori() возвращает предметные наборы и ассоциативные правила, которые получаются посредством вызова функций itemsets_from_transactions() и generate_rules_apriori() соответственно. Ассоциативные правила возвращаются как экземпляры класса Rule.

Ниже приведена краткая информация по этим функциям и классу.

1. Функция apriori.

efficient_apriori.apriori(*transactions: Union[List[tuple], Callable], min_support: float = 0.5, min_confidence: float = 0.5, max_length: int = 8, verbosity: int = 0, output_transaction_ids: bool = False*)

Функция реализует классический алгоритм Apriori.

Функция работает в два этапа.

На этапе 1 формируются предметные наборы с заданным значением *минимальной поддержки* min_support.

На этапе 2 строятся ассоциативные правила с заданным значением *минимальной достоверности* min_confidence на основе предметных наборов, найденных на этапе 1.

Оба этапа реализуются посредством исследования пространства поиска, то есть создания всех возможных предметных наборов и оценки их поддержки. Алгоритм Apriori эффективно сокращает пространство поиска, решая предварительно, имеет ли предметный набор желаемую поддержку, прежде чем выполнять итерацию по всему набору данных и проверку.

На рисунке 40 приведена информация по параметрам функции apriori.

2. Функция itemsets_from_transactions.

efficient_apriori.itemsets_from_transactions(*transactions: Union[List[tuple], Callable], min_support: float, max_length: int = 8, verbosity: int = 0, output_transaction_ids: bool = False*)

Функция реализует формирование предметных наборов.

На рисунке 41 приведена информация по параметрам функции itemsets_from_transactions.

3. Функция generate_rules_apriori.

efficient_apriori.generate_rules_apriori(*itemsets: Dict[int, Dict[tuple, int]], min_confidence: float, num_transactions: int, verbosity: int = 0*)

Функция реализует формирование ассоциативных правил.

```
Parameters
----------
transactions : list of tuples, list of itemsets.TransactionWithId,
    or a callable returning a generator. Use TransactionWithId's when
    the transactions have ids which should appear in the outputs.
    The transactions may be either a list of tuples, where the tuples must
    contain hashable items. Alternatively, a callable returning a generator
    may be passed. A generator is not sufficient, since the algorithm will
    exhaust it, and it needs to iterate over it several times. Therefore,
    a callable returning a generator must be passed.
min_support : float
    The minimum support of the rules returned. The support is frequency of
    which the items in the rule appear together in the data set.
min_confidence : float
    The minimum confidence of the rules returned. Given a rule X -> Y, the
    confidence is the probability of Y, given X, i.e. P(Y|X) = conf(X -> Y)
max_length : int
    The maximum length of the itemsets and the rules.
verbosity : int
    The level of detail printing when the algorithm runs. Either 0, 1 or 2.
output_transaction_ids : bool
    If set to true, the output contains the ids of transactions that
    contain a frequent itemset. The ids are the enumeration of the
    transactions in the sequence they appear.
```

Рисунок 40. Информация по параметрам функции `apriori`

```
Parameters
----------
transactions : a list of itemsets (tuples with hashable entries),
               or a function returning a generator
    A list of transactions. They can be of varying size. To pass through
    data without reading everything into memory at once, a callable
    returning a generator may also be passed.
min_support : float
    The minimum support of the itemsets, i.e. the minimum frequency as a
    percentage.
max_length : int
    The maximum length of the itemsets.
verbosity : int
    The level of detail printing when the algorithm runs. Either 0, 1 or 2.
output_transaction_ids : bool
    If set to true, the output contains the ids of transactions that
    contain a frequent itemset. The ids are the enumeration of the
    transactions in the sequence they appear.
```

Рисунок 41. Информация по параметрам функции `itemsets_from_transactions`

```
Parameters
----------
itemsets : dict of dicts
    The first level of the dictionary is of the form (length, dict of item
    sets). The second level is of the form (itemset, count_in_dataset)).
min_confidence :  float
    The minimum confidence required for the rule to be yielded.
num_transactions : int
    The number of transactions in the data set.
verbosity : int
    The level of detail printing when the algorithm runs. Either 0, 1 or 2.
```

Рисунок 42. Информация по параметрам функции `generate_rules_apriori`

```
Parameters
----------
lhs : tuple
    The left hand side (antecedent) of the rule. Each item in the tuple
    must be hashable, e.g. a string or an integer.
rhs : tuple
    The right hand side (consequent) of the rule.
count_full : int
    The count of the union of the lhs and rhs in the dataset.
count_lhs : int
    The count of the lhs in the dataset.
count_rhs : int
    The count of the rhs in the dataset.
num_transactions : int
    The number of transactions in the dataset.
```

Рисунок 43. Информация по параметрам класса `Rule`

На рисунке 42 приведена информация по параметрам функции `generate_rules_apriori`.

Класс `Rule`.

class `efficient_apriori.Rule`(*lhs: tuple, rhs: tuple, count_full: int = 0, count_lhs: int = 0, count_rhs: int = 0, num_transactions: int = 0*)

Класс предназначен для описания ассоциативного правила.

На рисунке 43 приведена информация по параметрам класса `Rule`.

```
from efficient_apriori import apriori
transactions = [['Тетрадь', 'Ручка', 'Краски'],
                ['Тетрадь', 'Ручка', 'Кисть'],
                ['Краски', 'Ручка', 'Ластик']]
freqItemSet, rules = apriori(transactions, output_transaction_ids=True)
print(freqItemSet)
```

```
{1: {('Тетрадь',): ItemsetCount(itemset_count=2, members={0, 1}), ('Краски',): ItemsetCount(i
temset_count=2, members={0, 2}), ('Ручка',): ItemsetCount(itemset_count=3, members={0, 1,
2})}, 2: {('Краски', 'Ручка'): ItemsetCount(itemset_count=2, members={0, 2}), ('Ручка', 'Тетр
адь'): ItemsetCount(itemset_count=2, members={0, 1})}}
```

Рисунок 44. Пример программного кода с выводом соответствующих частых предметных наборов, а также – информации о том, сколько раз они встретились и в каких транзакциях в программной реализации `efficient-apriori`

Если при выявлении ассоциативных правил необходимо знать, сколько раз встретился тот или иной частый предметный набор и в каких транзакциях он встретился, необходимо установить значение параметра `output_transaction_ids` равным `True`:

```
output_transaction_ids=True.
```

В этом случае для каждого предметного набора выводится информация об объекте `ItemsetCount`, в свойстве `itemset_count` которого хранится информация о числе транзакций, на основе которых сформирован частый предметный набор, а в свойстве `members` – информация о номерах `ids` таких транзакций.

На рисунке 44 приведен пример программного кода с выводом соответствующих частых предметных наборов, а также – информации о том, сколько раз они встретились (значение свойства `itemset_count`) и в каких транзакциях (значение свойства `members`).

Подробная документация по программной реализации `efficient-apriori` приведена по ссылке:

https://efficient-apriori.readthedocs.io/en/latest/.

5. Алгоритм FPGrowth на языке Python

На языке Python имеются различные реализации алгоритма FPGrowth.

Рассмотрим работу с программной реализацией алгоритма FPGrowth, предложенной по ссылке [14]:

https://pypi.org/project/fpgrowth-py/.

Для установки библиотеки, реализующей работу с алгоритмом FPGrowth, необходимо выполнить команду:

```
pip install fpgrowth_py
```

На рисунке 45 приведен пример программного кода, реализующего поиск ассоциативных правил с применением программной реализации `fpgrowth_py` для демонстрационного набора из 3 транзакций. При этом можно заметить некоторые отличия в названии входных параметров.

Оформление выведенных ассоциативных правил такое же, как в программной реализации `apriori_python`.

```
from fpgrowth_py import fpgrowth
transactions = [['Тетрадь', 'Ручка', 'Краски'],
                ['Тетрадь', 'Ручка', 'Кисть'],
                ['Краски', 'Ручка', 'Ластик']]
freqItemSet, rules = fpgrowth(transactions, minSupRatio=0.5, minConf=0.5)
print(rules)
```

```
[[{'Тетрадь'}, {'Ручка'}, 1.0], [{'Ручка'}, {'Тетрадь'}, 0.6666666666666666], [{'Краски'}, {'
Ручка'}, 1.0], [{'Ручка'}, {'Краски'}, 0.6666666666666666]]
```

Рисунок 45. Пример программного кода, реализующего поиск ассоциативных правил с применением программной реализации `fpgrowth_py`

```
type(freqItemSet)
```

```
list
```

```
freqItemSet
```

```
[{'Тетрадь'}, {'Ручка', 'Тетрадь'}, {'Краски'}, {'Краски', 'Ручка'}, {'Ручка'}]
```

```
type(rules)
```

```
list
```

```
rules
```

```
[[{'Тетрадь'}, {'Ручка'}, 1.0],
 [{'Ручка'}, {'Тетрадь'}, 0.6666666666666666],
 [{'Краски'}, {'Ручка'}, 1.0],
 [{'Ручка'}, {'Краски'}, 0.6666666666666666]]
```

Рисунок 46. Примеры, позволяющие определить тип и содержимое выходных параметров в программной реализации `fpgrowth_py`

На рисунке 46 приведены примеры, позволяющие определить тип и содержимое выходных параметров в программной реализации fpgrowth_py. Так, `freqItemSet` – список (в то время как в программных реализациях Apriori_python и efficient_apriori – это словарь), а `rules` – список, организация которого аналогична организации списка в программной реализации Apriori_python.

Найденные ассоциативные правила можно, например, вывести на печать командой

```
print(rules)
```

При этом для каждого ассоциативного правила будет выведено значение его достоверности.

Для ознакомления с программной реализацией алгоритма следует перейти по ссылке [15]:

https://github.com/chonyy/fpgrowth_py.

В программной реализации fpgrowth_py задействованы многие функции, наибольший интерес из которых представляет функция `fpgrowth()`.

Функция `fpgrowth()` возвращает частые предметные наборы и ассоциативные правила, удовлетворяющие заданным значениям минимальной поддержки `minSupRatio` и минимальной достоверности `minConf`.

Функция `fpgrowth()` предполагает, что набор транзакций представлен в виде списка.

Ниже приведена краткая информация по функции `fpgrowth`.

Функция `fpgrowth`.

`fpgrowth_py.fpgrowth` (*itemSetList, minSupRatio, minConf*)

Функция реализует классический FPGrowth алгоритм.

Функция работает в три этапа.

На этапе 1 формируется FP-дерево на основе набора транзакций с учетом заданного порогового минимального значения `minSup`, определяющее частоту, при которой одинарный предмет считается частым.

На этапе 2 из FP-дерева извлекаются частые предметные наборы с учетом заданного минимального значения `minSup`, определяющего частоту, при которой предметный набор считается частым.

На этапе 3 строятся ассоциативные правила с заданным значением *минимальной достоверности* `minConf` на основе частых предметных наборов, найденных на этапе 2.

Входные параметры:

параметр `itemSetList`, определяющий набор транзакций;

параметр `minSupRatio`, определяющий минимальное значение поддержки;

параметр `minConf`, определяющий минимальное значение достоверности.

По умолчанию значения параметров `minSupRatio` и `minConf` устанавливаются равными 0.5.

Пороговое минимальное значение `minSup`, определяющее частоту, при которой набор предметов считается частым, вычисляется как

`minSup = len(itemSetList) * minSupRatio`.

Выходные параметры:

параметр `freqItems`, определяющий частые предметные наборы;

параметр `rules`, определяющий ассоциативные правила.

Следует отметить, что по аналогии с `fpgrowth()` работает функция `fpgrowthFromFile()`, при этом предполагается, что набор транзакций считывается из файла.

В программной реализации `fpgrowth_py` используется класс `Node`, предназначенный для описания FP-дерева. В частности, в узле дерева хранится информация о названии соответствующего предмета, о частоте, с которой этот предмет встречается в транзакциях, имеющих общие пути от корня дерева до рассматриваемого узла, информация о родительском узле.

Дополнительные сведения по программной реализации алгоритма FPGrowth можно найти в [14 – 21].

6. Визуализация ассоциативных правил

Существенный интерес представляет реализованная на языке Python библиотека для визуализации ассоциативных правил, которая доступна по ссылке:

В частности, обратим внимание на реализацию, предложенную по ссылке [22]:

https://pypi.org/project/pyarmviz/

Для ознакомления с программной реализацией библиотеки для визуализации ассоциативных правил следует перейти по ссылке [23]:

https://github.com/chonyy/fpgrowth_py.

ВАРИАНТЫ И ЗАДАНИЯ

Необходимо выполнить задания по поиску ассоциативных правил в соответствии с предложенным вариантом.

Варианты

1. Тестовые варианты.

Вариант 1. Сформировать набор из 30 транзакций на основе чеков покупок в продуктовом магазине.

Вариант 2. Сформировать набор из 30 транзакций на основе чеков покупок в аптеке.

Вариант 3. Сформировать набор из 30 транзакций на основе чеков покупок в магазине компьютерной техники.

Вариант 4. Сформировать набор из 30 транзакций на основе чеков покупок в магазине одежды.

Вариант 5. Сформировать набор из 30 транзакций на основе продуктов, употребляемых на завтрак, обед и ужин.

Вариант 6. Сформировать набор из 30 транзакций на основе чеков покупок в магазине канцтоваров.

Вариант 7. Сформировать набор из 30 транзакций на основе чеков покупок в магазине бытовой техники.

Вариант 8. Сформировать набор из 30 транзакций на основе чеков покупок в кондитерском отделе.

Вариант 9. Сформировать набор из 30 транзакций на основе чеков покупок в магазине косметики.

Вариант 10. Сформировать набор из 30 транзакций на основе чеков покупок в магазине спорттоваров.

2. Варианты из репозиториев.

Вариант 1.

Набор данных:

http://archive.ics.uci.edu/ml/datasets/Online+Retail

Вариант 2.

Набор данных:

https://github.com/viktree/curly-octo-chainsaw/blob/master/BreadBasket_DMS.csv

P.S. Можно предложить собственные «большие» наборы данных.

Задания

1. Применить для тестовых вариантов и вариантов из репозиториев различные алгоритмы поиска ассоциативных правил при одинаковых начальных условиях (при одинаковых пороговых значениях для поддержки и достоверности) и сравнить полученные результаты. Для тестовых вариантов выполнить ручные расчеты (например, с применением MS Excel) и расчеты с применением программных библиотек на языке Python. Для вариантов из репозиториев выполнить расчеты с применением программных библиотек на языке Python.

В качестве алгоритмов поиска ассоциативных правил использовать алгоритмы:

– Apriori (https://pypi.org/project/apriori-python/);
– Efficient Apriori (https://pypi.org/project/efficient-apriori/);
– FPGrowth (https://pypi.org/project/fpgrowth-py/).

2. Сформировать базы ассоциативных правил с уровнем минимальной достоверности 60% и 80%. Вычислить для ассоциативных правил поддержку, достоверность, значимость.

3. Оценить время формирования искомых ассоциативных правил с применением различных алгоритмов и построить диаграммы, позволяющие выполнить сравнительный анализ.

4. Выполнить визуализацию ассоциативных правил (https://pypi.org/project/pyarmviz/).

Дополнительную информацию по работе с Python можно получить в [24, 25].

СПИСОК ЛИТЕРАТУРЫ

1. Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules // Proceedings of the 20th International Conference on Very Large Databases, 1994. P. 487 – 499.
2. Паклин Н., Орешков В. Бизнес-аналитика: от данных к знаниям: Учебное пособие. – 2-е изд., испр. – СПб.: Питер, 2013. – 704 с.
3. Apriori – масштабируемый алгоритм поиска ассоциативных правил [Электронный ресурс]. – Режим доступа: https://loginom.ru/blog/apriori, свободный (дата обращения 29.08.2021).
4. Воронцов К.В. Методы поиска ассоциативных правил [Электронный ресурс]. – Режим доступа: http://www.machinelearning.ru/wiki/images/archive/7/7c/20140621071835%21Voron-ML-AssocRules-slides.pdf, свободный (дата обращения 29.08.2021).
5. Tan J., Bu Y., Yang B. An Efficient Close Frequent Pattern Mining Algorithm // 2009 Second International Conference on Intelligent Computation Technology and Automation, 2009. – P. 528-531.
6. FPG – альтернативный алгоритм поиска ассоциативных правил [Электронный ресурс]. – Режим доступа: https://loginom.ru/blog/fpg, свободный (дата обращения 29.08.2021).
7. Frequent Pattern Growth (FP-Growth) Algorithm [Электронный ресурс]. – Режим доступа: https://wimleers.com/sites/wimleers.com/files/FP-Growth%20presentation%20handouts%20%E2%80%94%C2%A0Florian%20Verhein.pdf, свободный (дата обращения 29.08.2021).
8. A simple apriori algorithm python implementation [Электронный ресурс]. – Режим доступа: https://pypi.org/project/apriori-python/, свободный (дата обращения 29.08.2021).
9. Apriori_Python [Электронный ресурс]. – Режим доступа: https://github.com/chonyy/apriori_python, свободный (дата обращения 29.08.2021).
10. Apriori: Association Rule Mining In-depth Explanation and Python Implementation [Электронный ресурс]. – Режим доступа: https://towardsdatascience.com/apriori-association-rule-mining-explanation-and-python-implementation-290b42afdfc6, свободный (дата обращения 29.08.2021).
11. An efficient Python implementation of the Apriori algorithm [Электронный ресурс]. – Режим доступа: https://pypi.org/project/efficient-apriori/, свободный (дата обращения 29.08.2021).
12. Efficient-Apriori [Электронный ресурс]. – Режим доступа: https://efficient-apriori.readthedocs.io/en/latest/, свободный (дата обращения 29.08.2021).

13. Efficient-Apriori [Электронный ресурс]. – Режим доступа: https://github.com/tommyod/Efficient-Apriori, свободный (дата обращения 29.08.2021).

14. Python implementation of FP Growth algorithm [Электронный ресурс]. – Режим доступа: https://pypi.org/project/fpgrowth-py/, свободный (дата обращения 29.08.2021).

15. FPGrowth_py [Электронный ресурс]. – Режим доступа: https://github.com/chonyy/fpgrowth_py, свободный (дата обращения 29.08.2021).

16. FP Growth: Frequent Pattern Generation in Data Mining with Python Implementation [Электронный ресурс]. – Режим доступа: https://towardsdatascience.com/fp-growth-frequent-pattern-generation-in-data-mining-with-python-implementation-244e561ab1c3, свободный (дата обращения 29.08.2021).

17. Mlxtend [Электронный ресурс]. – Режим доступа: https://github.com/rasbt/mlxtend/tree/master/mlxtend/frequent_patterns, свободный (дата обращения 29.08.2021).

18. FP-Growth [Электронный ресурс]. – Режим доступа: https://fp-growth.readthedocs.io/en/latest/, свободный (дата обращения 29.08.2021).

19. Understand and Build FP-Growth Algorithm in Python [Электронный ресурс]. – Режим доступа: https://towardsdatascience.com/understand-and-build-fp-growth-algorithm-in-python-d8b989bab342, свободный (дата обращения 29.08.2021).

20. Frequent Pattern (FP) Growth Algorithm In Data Mining [Электронный ресурс]. – Режим доступа: https://www.softwaretestinghelp.com/fp-growth-algorithm-data-mining/, свободный (дата обращения 29.08.2021).

21. ML | Frequent Pattern Growth Algorithm [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/ml-frequent-pattern-growth-algorithm/, свободный (дата обращения 29.08.2021).

22. PyARMViz [Электронный ресурс]. – Режим доступа: https://pypi.org/project/pyarmviz/, свободный (дата обращения 29.08.2021).

23. PyARMViz [Электронный ресурс]. – Режим доступа: https://github.com/Mazeofthemind/PyARMViz/commits/masterz/, свободный (дата обращения 29.08.2021).

24. Лутц М. Изучаем Python. – М.: Диалектика. 2020. – Том. 1. 832 с.; Том. 2. 720 с.

25. Доусон М. Программируем на Python. – СПб.: Питер. 2020. – 416 с.

Сведения об авторах

Демидова Лилия Анатольевна, доктор технических наук, профессор, профессор кафедры корпоративных информационных систем Института информационных технологий РТУ МИРЭА.