Поиск ассоциативных правил с помощью алгоритма Apriori

Пока профессор Буковски занимается выбором помощницы, его ассистент продолжает свою экспедицию по Африке. Перед ним стоит задача наблюдать за небольшой стаей обезьян. Свои наблюдения ассистент оформляет в виде почасовой транзакционной таблицы. Каждая строка таблицы соответствует часу наблюдения, в столбце перечислено, чем занималась обезьянка в течение этого часа.

В таблице приведены результаты наблюдений:

Время	Чем занималась обезьянка
10:00	Ела, спала, чистила шерстку
11:00	Играла, чистила шерстку
12:00	Ела, спала
13:00	Играла, чистила шерстку
14:00	Ела, спала
15:00	Ела, играла

Ассистенту необходимо выяснить повадки обезьян – какие занятия обезьянки чаще всего делают одновременно. Поможет ему решить эту задачу алгоритм **`Apriori`**, предназначенный для поиска часто встречающихся сочетаний событий.

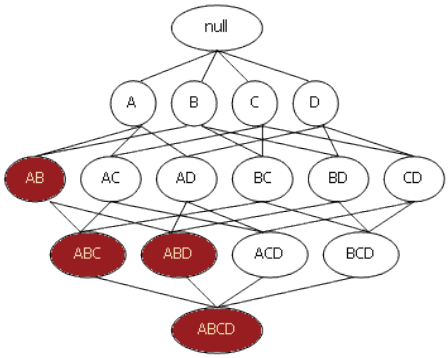
Алгоритм Apriori

Алгоритм **`Apriori`** предназначен для поиска ассоциативных правил. Примитивный подход к решению данной задачи – простой перебор всех возможных наборов элементов. Это потребует $O(2^n)$ операций, где **`n`** – количество элементов. **`Apriori`** использует одно из свойств поддержки, гласящее: поддержка любого набора элементов не может превышать минимальной поддержки любого из его подмножеств. Например, поддержка 3-элементного набора {Хлеб, Масло, Молоко} будет всегда меньше или равна поддержке 2-элементных наборов {Хлеб, Масло}, {Хлеб, Молоко}, {Масло, Молоко}. Дело в том, что любая транзакция, содержащая {Хлеб, Масло, Молоко}, также должна содержать {Хлеб, Масло}, {Хлеб, Молоко}, {Масло, Молоко}, причем обратное не верно.

Идею алгоритма **`Apriori`** демонстрирует рисунок ниже. Предположим, что набор из элементов {A, B} имеет поддержку ниже заданного порога и, соответственно, не является часто встречающимся. Тогда все его супермножества также не являются часто встречающимися и отбрасываются. Вся эта ветвь, начиная с {A, B}, выделена фоном. Использование этой эвристики позволяет существенно сократить пространство поиска.

На первом шаге алгоритма подсчитываются 1-элементные часто встречающиеся наборы. Для этого необходимо пройтись по всему набору данных и подсчитать для них поддержку, т.е. сколько раз встречается в базе.

Следующие шаги будут состоять из двух частей: генерации потенциально часто встречающихся наборов элементов (их называют кандидатами) и подсчета поддержки для кандидатов.



Описанный выше алгоритм можно записать в виде следующего псевдокода:

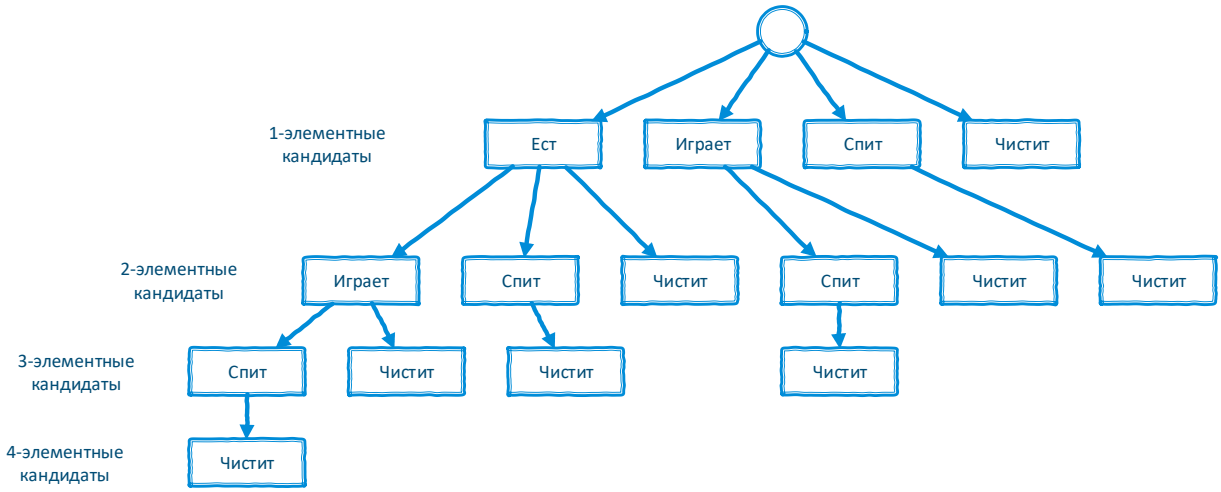
```
Расчёт поддержки support_ для 1-элементных кандидатов
Отбор 1-элементных кандидатов one_element_candidates, поддержка support_ которых выше порога minimal_support
Прошедшие кандидаты accepted_candidates = one_element_candidates

Цикл по k от 2 до n, где n – количество наблюдаемых событий:
  Генерация кандидатов multi_element_candidates размера k из кандидатов accepted_candidates размера k-1

  Цикл по всем сгенерированным кандидатам multi_element_candidates:
    Если поддержка текущего кандидата больше порога minimal_support:
      Добавление кандидата в список прошедших кандидатов accepted_candidates
      Сохранение как ассоциативного правила в rules

Возврат найденных ассоциативных правил rules
```

Процесс генерации кандидатов можно изобразить в виде дерева:



Под кандидатом понимается сочетание действий. Сначала рассматриваются 1-элементные кандидаты, то есть каждое действие обезьяны по отдельности. Из 1-элементных кандидатов генерируются 2-элементные кандидаты путем присоединения действий, которые находятся справа по списку (см. рисунок выше). Процедура генерации кандидатов повторяется до N – общего количества видов действий. Из дерева видно, что кандидат, состоящий из N действий, будет только один. Для генерации k-элементных кандидатов используются k-1-элементные кандидаты.

Порядок выполнения работы

Перейдите в каталог со скриптами данной практики.

Откройте для редактирования файл **`run.py`**. Заполните матрицу **X** исходными данными, которые приведены в таблице выше. Каждая строка в матрице **X** соответствует строке в приведенной таблице. Каждый столбец соответствует наблюдаемому событию. Таким образом, если ассистент

наблюдает 4 события (ест, играет, спит, чистит шерстку), то матрица **X** будет содержать 4 столбца. **ij**-элемент в матрице **X** будет равен **1**, если обезьянка в **i**-е время выполняла **j**-е действие.

Откройте для редактирования файл **Apriori.py**. В начале функции **apriori** посчитайте поддержу для одноэлементных кандидатов. Необходимые пояснения для вычисления этой поддержки описаны в комментарии.

Найдите в файле **Apriori.py** функцию **support**, которая вычисляет поддержку для кандидата **candidate**. Пользуясь приведенным к функции комментарием, реализуйте вычисление поддержки для кандидата и верните результат.

В функции **apriori** найдите фрагмент, где производится сравнение вычисленной для **i**-го кандидата поддержки с минимальной поддержкой. Если кандидат проходит проверку, то есть поддержка для него выше минимального порога, его необходимо внести в список **accepted_candidates**. Допишите код, который добавляет **i**-го кандидата из **multi_element_candidates** в список **accepted_candidates**.

После того, как Вы дополнили весь необходимый код согласно приведенной инструкции, запустите в **Python** файл **run.py**. Программа должна вывести обнаруженные ассоциативные правила:

```
ест   спит  -> 0.500000
```

Прочитайте весь получившийся код вместе с комментариями и разберитесь, как он работает.