第三课

Tedu.cn 达内教育

关联规则

Apriori
Fpgrowth

Apriori

“啤酒与尿布”的故事产生于20世纪90年代的美国沃尔玛超市。沃尔玛的超市管理人员分析销售数据时发现了一个令人难于理解的现象：在某些特定的情况下，“啤酒”与“尿布”两件看上去毫无关系的商品会经常出现在同一个购物篮中，这种独特的销售现象引起了管理人员的注意，经过后续调查发现。

原来，美国的妇女通常在家照顾孩子，所以她们经常会嘱咐丈夫在下班回家的路上为孩子买尿布，而丈夫在买尿布的同时又会顺手购买自己爱喝的啤酒。这样就会出现啤酒与尿布这两件看上去不相干的商品经常会出现在同一个购物篮的现象。

这个发现为商家带来了大量的利润，但是如何从浩如烟海却又杂乱无章的数据中，发现啤酒和尿布销售之间的联系呢？

Apriori算法产生

1993年美国学者Agrawal提出通过分析购物篮中的商品集合，从而找出商品之间关联关系的关联算法，并根据商品之间的关系，找出客户的购买行为。Agrawal从数学及计算机算法角度提出了商品关联关系的计算方法——Apriori算法。

沃尔玛从上个世纪90年代尝试将Aprior算 法引入到POS机数据分析中，并获得了成功，于是产生了“啤酒与尿布”的故事。

Apriori

如何寻找？

在历史购物记录中，一些商品总是在一起购买。但人看上去不是那么的直观的，而是隐蔽的。让计算机做这事，设计算法让计算机自动去找，找到这样的模式(规律)。

目标:寻找那些总是一起出现商品。

mahout实战—>机器学习实战

《mahout实战》与《机器学习实战》一起该买的记录数占所有商品记录总数的比例——支持度(整体)

买了《mahout实战》与《机器学习实战》一起该买的记录数占所有购买《mahout实战》记录数的比例——置信度(局部)

需要达到一定的阈值

Apriori

支持度、置信度越大，商品出现一起购买的次数就越多，可信度就越大。

支持度：在所有的商品记录中有2%量是购买《mahout实战》与《机器学习实战》
置信度：买《mahout实战》的顾客中有60%的顾客购买了《机器学习实战》

作用：找到商品购买记录中反复一起出现的商品，帮能助营销人员做更好的策略，帮助顾客方便购买。

策略：

1、同时购买的商品放一起
2、同时购买的商品放两端

Apriori

支持度、置信度转化为数学语言进行计算：

A表示《mahout实战》B表示《机器学习实战》

support(A->B) = P(AB) （《mahout实战》和《机器学习实战》一起买占总的购买记录的比例）

confidence(A->B) = P(B|A) （购买了《mahout实战》后，买《机器学习实战》占的比例）

项集：项的集合称为项集，即商品的组合。

k项集：k件商品的组合，不关心商品件数，仅商品的种类。

项集频率：商品的购买记录数，简称为项集频率，支持度计数。

注意，定义项集的支持度有时称为相对支持度，而出现的频率称为绝对支持度。

频繁项集：如果项集的相对支持度满足给定的最小支持度阈值，则该项集是频繁项集。

强关联规则:满足给定支持度和置信度阈值的关联规则

Apriori

$$\text{sup } port(A \Rightarrow B) = P(AB) = \text{sup}\_\ port(A \cup B)$$

$$confidence(A \Rightarrow B) = P(B \mid A) = \frac{P(AB)}{P(A)} = \frac{\text{sup } port(A \cup B)}{\text{sup } port(A)} = \frac{\text{sup } port\ \_count(AUB)}{\text{sup } port\ \_count(A)}$$

A=>B的置信度可以由A于A U B的支持度计数计算推出。满足最小支持度计数的项集为频繁项集。

找关联规则问题，归结为找频繁项集。

注意:A=>B,B=>A的不同

交易ID	购买的商品
1000	A, B, C
2000	A, C
3000	A, D
4000	B, E

设：最小支持度为50%
最小可信度为60%

则可得到：

$A \Rightarrow C$ [50%, 66.7%]

$C \Rightarrow A$ [50%, 100%]

Apriori

明确问题

1、要找总是在一起出现的商品组合
2、提出衡量标准支持度、置信度(达到一定的阈值)
3、给出支持度、置信度直观计算方法
4、在计算方法中起决定因素的是频繁项集
5、由频繁项集轻松找到强关联规则

找关联规则--------->找频繁项集

步骤：

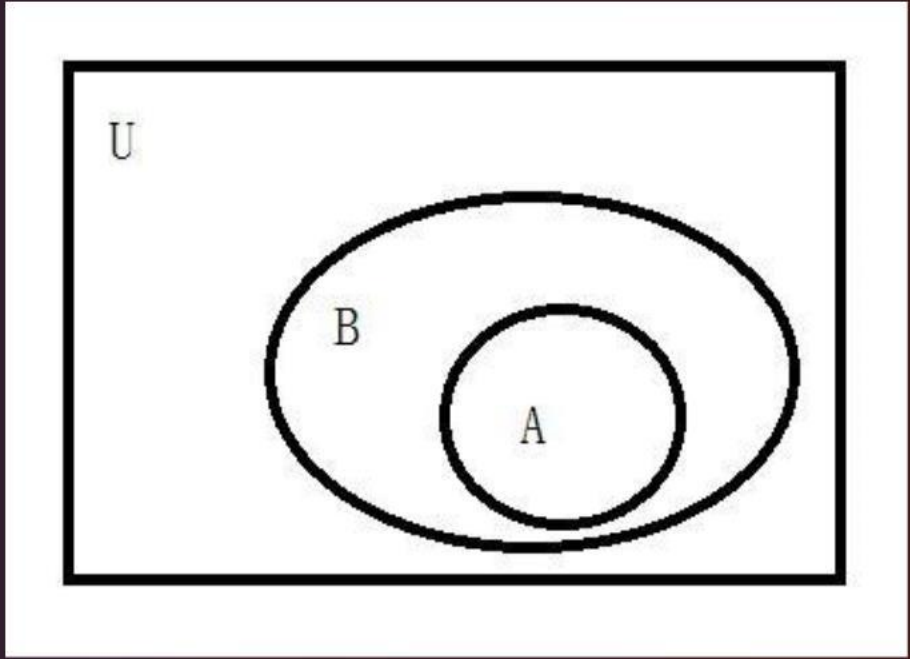
1. 找出所有的频繁项集；这个项集出现的次数至少与要求的最小计数一样。如在100次购买记录中，至少一起出现
2. 由频繁项集产生强关联规则；这些关联规则满足最小支持度与最小置信度。

Apriori

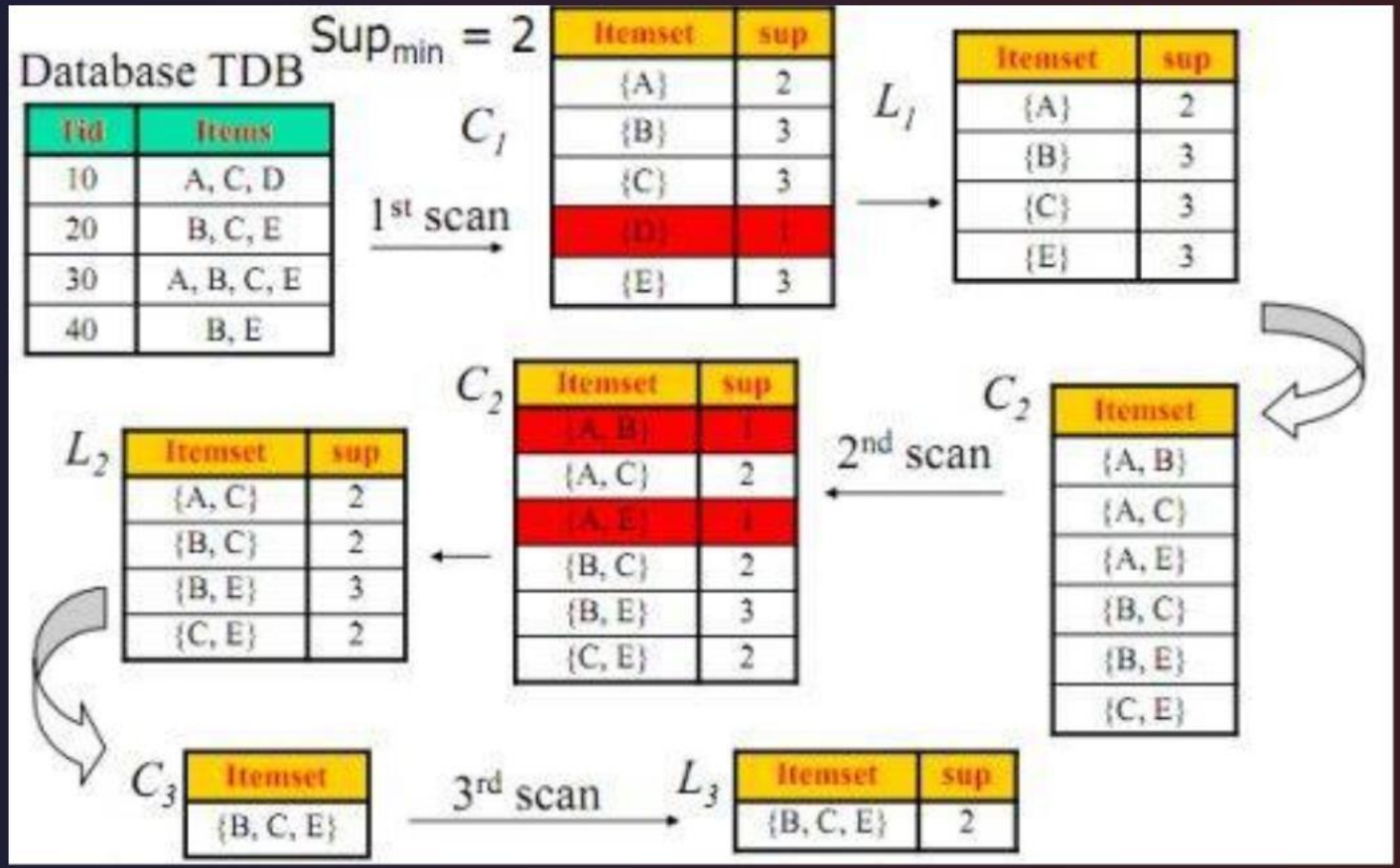
先验性质:频繁项集的所有非空子集也一定是频繁的。
逆否命题:若一个项集是非频繁的,则它的任何超集也是非频繁的。

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E



Apriori


Database TDB
Tid Items
10 A, C, D
20 B, C, E
30 A, B, C, E
40 B, E
Sup_min = 2
C1
1st scan
Itemset sup
{A} 2
{B} 3
{C} 3
{D} 1
{E} 3
L1
Itemset sup
{A} 2
{B} 3
{C} 3
{E} 3
C2
Itemset sup
{A, B} 1
{A, C} 2
{A, E} 1
{B, C} 2
{B, E} 3
{C, E} 2
2nd scan
C2
Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}
L2
Itemset sup
{A, C} 2
{B, C} 2
{B, E} 3
{C, E} 2
C3
Itemset
{B, C, E}
3rd scan
L3
Itemset sup
{B, C, E} 2

Apriori

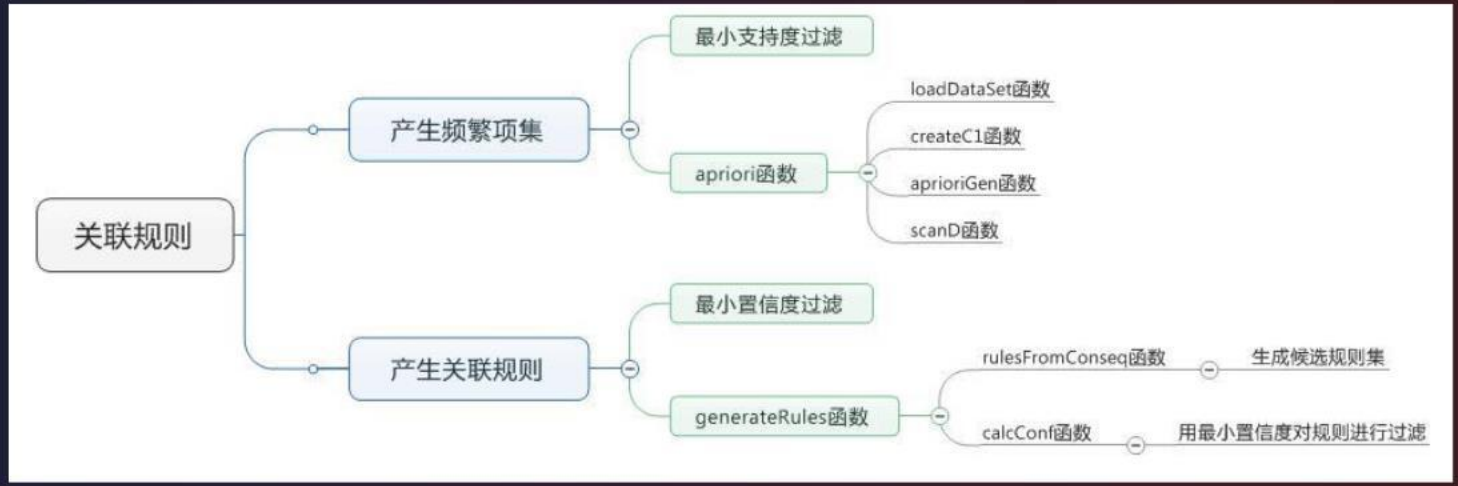
挑战
- 多次数据库扫描
- 巨大数量的候补项集
- 繁琐的支持度计算

改善Apriori: 基本想法
- 减少扫描数据库的次数
- 减少候选项集的数量
- 简化候选项集的支持度计算

Apriori

产生频繁项集(最小支持度过滤) Apriori 函数

产生关联规则(最小置信度过滤) generateRules 函数



Apriori

```python
#生成原始数据，用于测试
def loadDataSet():
    return [[1, 3, 4],
            [2, 3, 5],
            [1, 2, 3, 5],
            [2, 5]]
```

```python
#遍历数据集每项物品，建立1-项集
def createC1(dataSet):
    #记录每项物品的列表
    C1 = []
    #遍历每条记录
    for transaction in dataSet:
        #遍历每条记录中的物品
        for item in transaction:
            #判断如果该物品没在列表中
            if not [item] in C1:
                #将该物品加入到列表中
                C1.append([item])
    #对所有物品进行排序
    C1.sort()
    #将列表元素映射到frozenset()中，返回列表。
    #frozenset数据类型，指被冰冻的集合，
    #集合一旦完全建立，就不能被修改。
    #即用户不能修改他们。
    return map(frozenset, C1)
```

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

C_1

1st scan →

Itemset
{A}
{B}
{C}
{D}
{E}

Apriori

Tedu.cn 达内教育

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

C_1

1st scan

Itemset
{A}
{B}
{C}
{D}
{E}

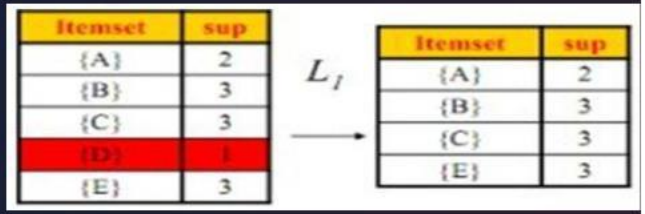
Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

```
#输入：数据集D、候选集Ck、最小支持度。
#候选集Ck由上一层(第k-1层)的频繁项集Lk-1组合得到。
#用最小支持度minSupport对候选集Ck过滤
#输出：本层(第k层)的频繁项集Lk，每项的支持度

#例如，由频繁1-项集(L1)内部组合生成候选集(C2)
#去除不满足最小支持度的项，得到频繁2-项集(L2)
def scanD(D, Ck, minSupport):
    #建立字典<key,value>
    #候选集Ck中每项及在所有物品记录中出现的次数
    #key-->候选集中的每项
    #value-->该物品在所有物品记录中出现的次数
    ssCnt = {}


    #对比候选集中的每项与原物品记录，统计出现的次数
    #遍历每条物品记录
    for tid in D:
        #遍历候选集Ck中的每一项，用于对比
        for can in Ck:
            #如果候选集Ck中该项在该条物品记录出现
            #即当前项是当前物品记录的子集
            if can.issubset(tid):
                #如果选集Ck中该项第一次被统计到，次数记为1
                if not ssCnt.has_key(can): ssCnt[can]=1
                #否则次数在原有基础上
                else: ssCnt[can] += 1
```

Apriori



Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

```
#数据集中总的记录数，物品购买记录总数，
numItems = float(len(D))
#记录经最小支持度过滤后的频繁项集
retList = []
#记录候选集中满足条件的项的支持度<key,value>结构
#key-->候选集中满足条件的项
#value-->该项支持度
supportData = {}

#遍历候选集中的每项出现次数
for key in ssCnt:
    #计算每项的支持度
    support = ssCnt[key]/numItems
    #用最小支持度过滤，
    if support >= minSupport:
        #保留满足条件物品组合
        #使用retList.insert(0,key)
        #在列表的首部插入新的集合，
        #只是为了让列表看起来有组织。
        retList.insert(0,key)
    #记录该项的支持度
    #注意：候选集中所有项的支持度均被保存下来了
    #不仅仅是满足最小支持度的项，其他项也被保存
    supportData[key] = support
#返回满足条件的物品项，以及每项的支持度
return retList, supportData
```

Apriori


```python
#生成原始数据，用于测试
def loadDataSet():
    return [[1, 3, 4],
            [2, 3, 5],
            [1, 2, 3, 5],
            [2, 5]]
```

1)

```
C1=createC1(dataSet)

print C1
```

```
[frozenset([1]), frozenset([2]), frozenset([3]), frozenset([4]), frozenset([5])]
```

FPGrowth

相比Apriori算法需要多次扫描数据库，FPGrowth只需要对数据库扫描2次。

第1次扫描获得单个项目的频率，去掉不满足支持度要求的项，并对剩下的项排序。

第2次扫描建立一颗FP-Tree树。

FPGrowth

{A,F,G}的支持度数为3，支持度为3/4。

{F,G}的支持度数为4，支持度为4/4。

{A}的支持度数为3，支持度为3/4。

{F,G}=>{A}的置信度为：{A,F,G}的支持度数除以{F,G}的支持度数，即3/4

{A}=>{F,G}的置信度为：{A,F,G}的支持度数除以{A}的支持度数，即3/3

挖掘强关联规则是在满足一定支持度的情况下寻找置信度达到阈值的所有商品组合。

ID	Items
1	A,E,F,G
2	A,F,G
3	A,B,E,F,G
4	E,F,G

FPGrowth

我们要找出哪些总是一起购买的商品，比如人们买薯片的时候通常也会买鸡蛋，则[薯片，鸡蛋]就是一条频繁模式(规律)。

第一步：扫描数据库，每项商品按频数递减排序，删除频数小于最小支持度MinSup的商品。　（第一次扫描数据库）

薯片:7鸡蛋:7面包:7牛奶:6啤酒:4　（这里我们令MinSup=4）

以上结果就是频繁1项集，记为F1。

FPGrowth

F1中排序　薯片:7鸡蛋:7面包:7牛奶:6啤酒:4

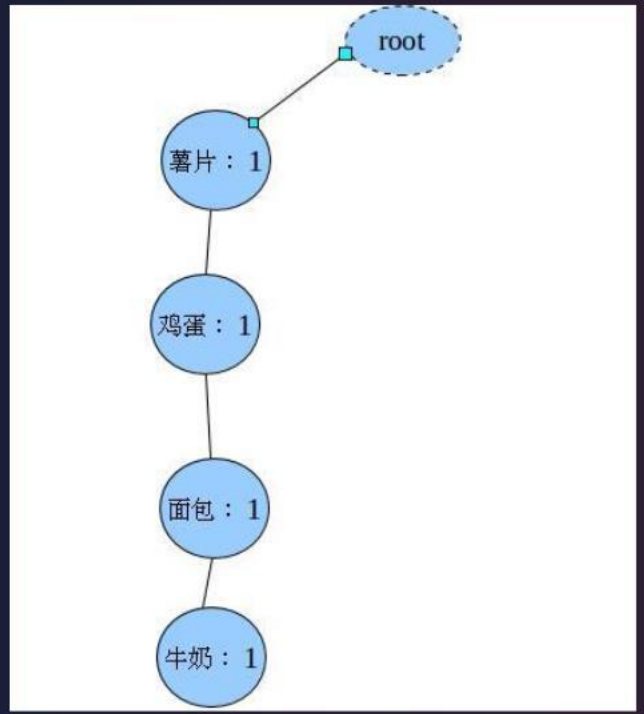
第二步：对于每一条购买记录，按照F1中的顺序重新排序。

FPGrowth

ID	Items
1	薯片,鸡蛋,面包,牛奶
2	薯片,鸡蛋,啤酒
3	面包,牛奶,啤酒
4	薯片,鸡蛋,面包,牛奶,啤酒
5	薯片,鸡蛋,面包
6	鸡蛋,面包,啤酒
7	薯片,面包,牛奶
8	薯片,鸡蛋,面包,牛奶
9	薯片,鸡蛋,牛奶

FPGrowth

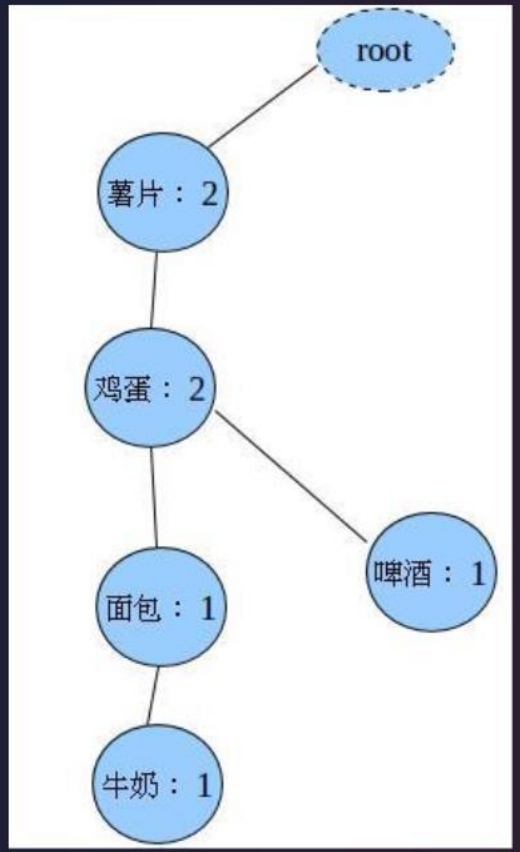
第三步：把第二步得到的各条记录插入到FP-Tree中。



1	薯片, 鸡蛋, 面包, 牛奶
2	薯片, 鸡蛋, 啤酒
3	面包, 牛奶, 啤酒

插入第一条（薯片, 鸡蛋, 面包, 牛奶）

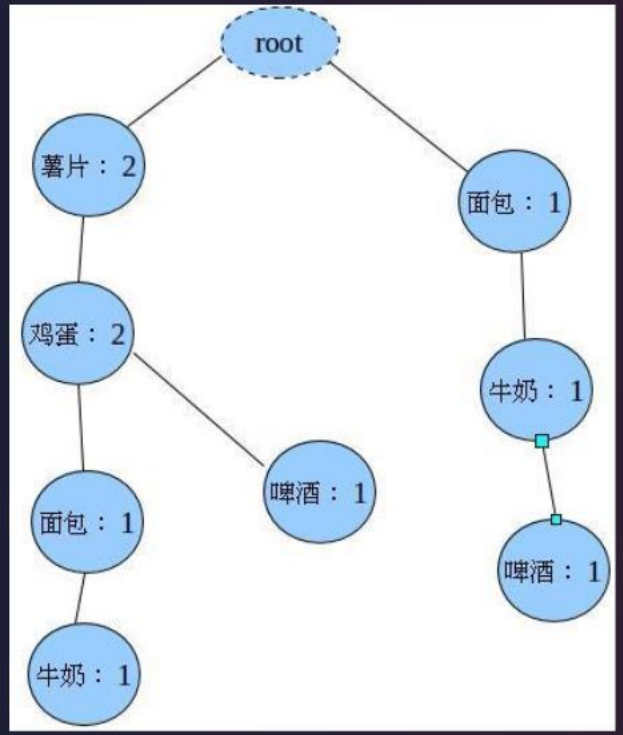
FPGrowth



1	薯片, 鸡蛋, 面包, 牛奶
2	薯片, 鸡蛋, 啤酒
3	面包, 牛奶, 啤酒

插入第二条记录（薯片, 鸡蛋, 啤酒）

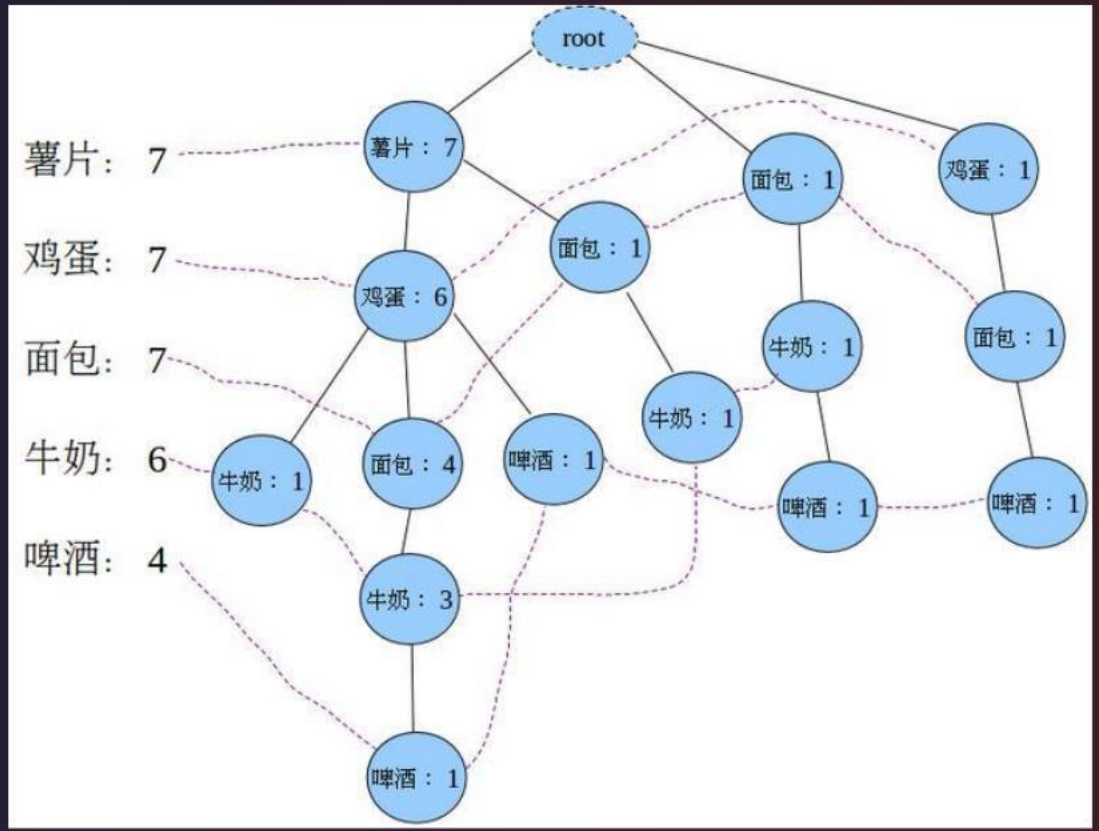
FPGrowth



1	薯片，鸡蛋，面包，牛奶
2	薯片，鸡蛋，啤酒
3	面包，牛奶，啤酒

插入第三条记录（面包，牛奶，啤酒）

FPGrowth


root
薯片： 7
鸡蛋： 7
面包： 7
牛奶： 6
啤酒： 4
薯片：7
面包：1
鸡蛋：1
面包：1
鸡蛋：6
牛奶：1
面包：1
牛奶：1
牛奶：1
面包：4
啤酒：1
啤酒：1
啤酒：1
牛奶：3
啤酒：1