Chapter 3 Association Rule

👉 定义

给定集合,T,,关联规则发现是**找出支持度**,$\geq min_sup$,**并且置信度**,$\geq min_conf$,**的所有规则**

关联规则是形如,$X \rightarrow Y$,的表达式(,X和Y,是不相交的项集)

👉 相关概念

- 频繁模式——数据库中频繁出现的项集
- 项集(Itemset)——包含0个或多个项的集合
- 支持度计数(,σ,)——包含特定项集的事务个数
- 支持度——包含特定项集的事务数/总事务数
- 频繁项集(Frequent Itemset)——满足**最小支持度阈值**,min_sup,的所有项集

👉 关联规则的强度

- 支持度——确定项集的**频繁程度**

$$support(X \rightarrow Y) = P(X \bigcup Y)$$

- 置信度——确定Y在包含X的事务中出现的频繁程度
  - 注意在置信度的计算中忽略了Y的作用

$$confidence(X \rightarrow Y) = P(Y|X) = \frac{support(X \bigcup Y)}{support(X)}$$

👉 关联规则挖掘一般步骤

1. 频繁项集产生
2. 规则的产生(在频繁项集中提取所有高置信度的规则，这些规则称为强规则)

频繁项集的产生

BF

👉 不考虑 时间复杂度太高 指数级

👉 降低产生频繁项集计算复杂度的方法

- 减少候选项集的数量——Apriori原理
- 减少比较的次数——使用高级的数据结构，或存储候选项集或压缩数据集，来减少比较次数

Apriori

👉 理论基础

- 如果一个项集是频繁的，则它的**所有子集**一定是频繁的
- 如果一个项集是非频繁的，则它的**所有超集**也一定是非频繁的

👉 剪枝原则

如果一个项集不是频繁的，不产生它的超集

这种基于**支持度度量**修剪**指数搜索空间**的策略称为**基于支持度的剪枝**

👉 方法

长度为k的**频繁项集**产生长度为k+1的候选项集(**连接，剪枝**)，扫描DB测试候选项集

连接的算法

```
procedure apriori_gen(L_{k-1}:frequent (k-1)-itemsets)
(1)   for each itemset l_1 ∈ L_{k-1}
(2)     for each itemset l_2 ∈ L_{k-1}
(3)       if (l_1[1] = l_2[1]) ∧ (l_1[2] = l_2[2]) ∧ ... ∧ (l_1[k-2] = l_2[k-2]) ∧ (l_1[k-1] < l_2[k-1]) then {
(4)         c = l_1 ⋈ l_2; // join step: generate candidates
(5)         if has_infrequent_subset(c, L_{k-1}) then
(6)           delete c; // prune step: remove unfruitful candidate
(7)         else add c to C_k;
(8)       }
(9)   return C_k;
```

1、连接步

剪枝的算法

```
procedure has_infrequent_subset(c: candidate k-itemset;
            L_{k-1}: frequent (k-1)-itemsets); // use prior knowledge
(1)   for each (k-1)-subset s of c
(2)     if s ∉ L_{k-1} then
(3)       return TRUE;
(4)   return FALSE;
```

2、剪枝步

👉 算法步骤

(1) L_1={频繁1项集};
(2) for(k=2;$L_{k-1}\neq\varnothing$;k++) do begin
(3) C_k=**apriori_gen**(L_{k-1}); //新的潜在频繁项集
(4) for all *transactions* $t\in D$ do begin
(5) C_t=subset(C_k,t); //找出t中包含的潜在的频繁项
(6) for all *candidates* $c\in C_t$ do
(7) *c.count*++;
(8) end
(9) end
(10) L_k={c∈C_k|c.count≥min_sup}
(11) end;
(12) Answer= $\bigcup_k L_k$

频繁模式挖掘面临的挑战

- 事务数据库的多遍扫描，IO开销大
  - 改进——减少扫描次数
- 数量巨大的候选项集
  - 改进——压缩候选数量
- 候选支持度计数繁重的工作量
  - 便于候选计数

提高Apriori算法效率

- 散列项集计数
  - 压缩候选项集
    - 一个哈希桶中计数小于最小支持度阈值的项集不可能是频繁的
- 事务压缩
  - 删除不可能对寻找频繁项集有用的事务
    - 根据k项集若不是频繁，则其超集不为频繁项集，将超集打上标签或直接删除
- 划分
  - 分而治之
- 采样
  - 选取事务数据库的一个样本进行关联规则挖掘
  - 牺牲精度换取效率

FP-Growth(Frequent-Pattern Growth)

👉 不同于Apriori先产生候选项集再产生频繁项集的方法，使用**FP树**数据结构组织数据，**直接从该结构中提取频繁项集**

👉 基本思想

- 将代表频繁项集的数据库压缩到FP树上
- 将FP树划分为一组条件数据库，挖掘每个条件数据库获取频繁项集

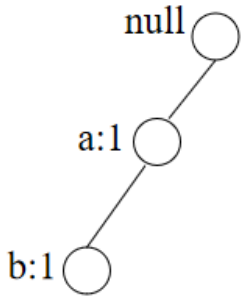
👉 构造FP树

- 支持度排序——扫描一次数据库，将获得的频繁1项集按照**支持度降序**排列，丢弃非频繁项
- 构建FP树——第二次扫描数据库 将每个事务都压缩到FP树上

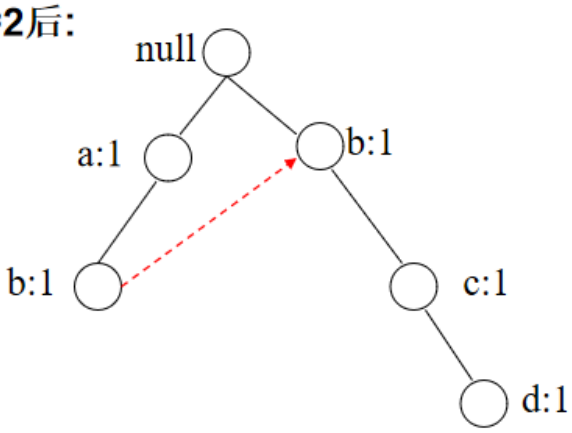
最小支持计数 = 2

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

读入事务 **TID=1**后:



读入事务 **TID=2**后:



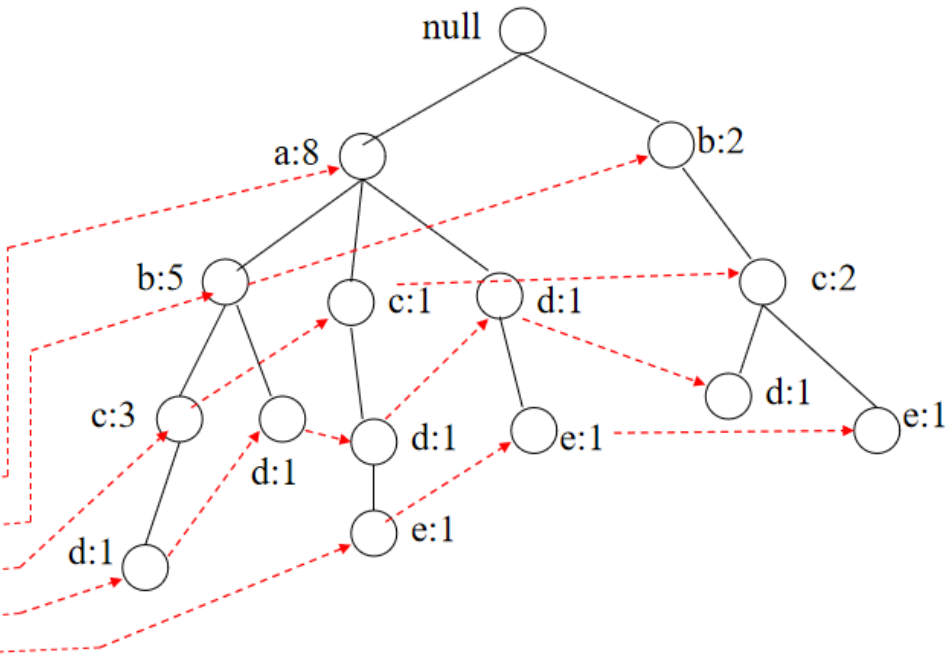
直到每个事务都映射到FP树的一条路径

继续该过程，直到每个事务都映射到**FP**树的一条路径。

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

Header table

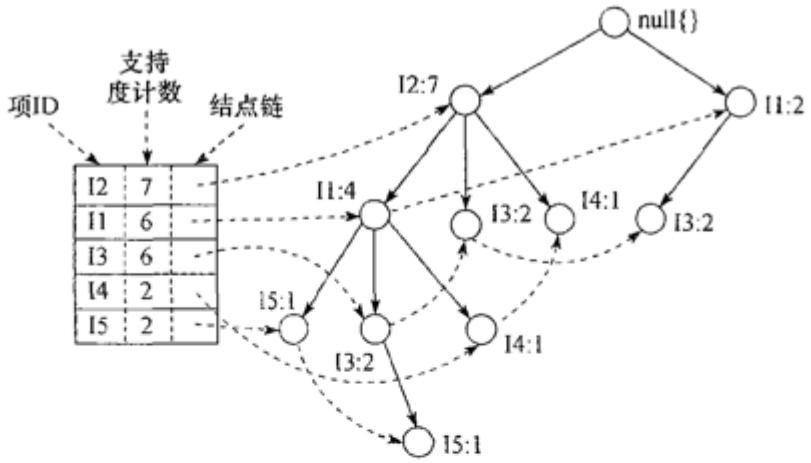
Item	Count	P
a	8	
b	7	
c	6	
d	5	
e	3	



注意每一个频繁1项集都有一个对应的链表(总体是一个桶)，指向树中对应的位置，以便可以快速访问

👉 构建**条件模式基**

以每一个频繁1项集作为**后缀**，找出路径中以**该项为后缀**的**前缀路径**



项	条件模式基
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}
I4	{{I2, I1: 1}, {I2: 1}}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}
I1	{{I2: 4}}

感觉要是构造算法 这一步两层循环

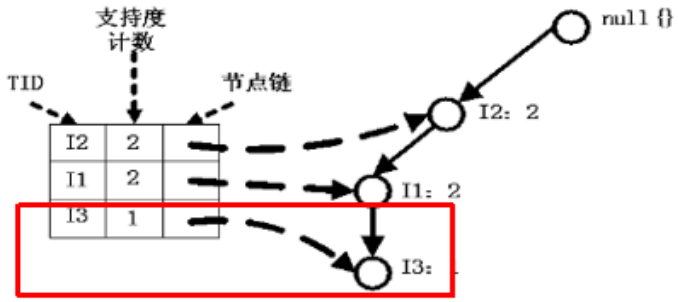
- 外层循环遍历频繁1项集
- 内层循环遍历每一项的链表，要追溯到`null`根结点为止

条件模式基相当于一个桶

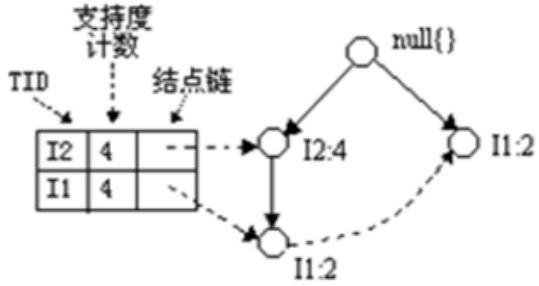
👉 构建**条件FP树**

将每一个条件模式基看成一个**子数据库**，这里要根据最小支持度阈值删除某些项

项	条件模式基	条件 FP 树
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	⟨I2: 2, I1: 2⟩
I4	{{I2, I1: 1}, {I2: 1}}	⟨I2: 2⟩
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩
I1	{{I2: 4}}	⟨I2: 4⟩



项 I5



项 I3

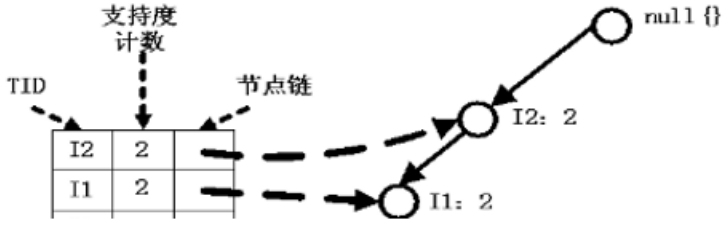
在项I5时，由于I3不满足最小支持度阈值，所以删掉

在项I4时，I1也是同理被删掉

👉 在条件FP树上挖掘频繁项集

- 若条件FP树为单路径，则产生该路径下所有模式的组合

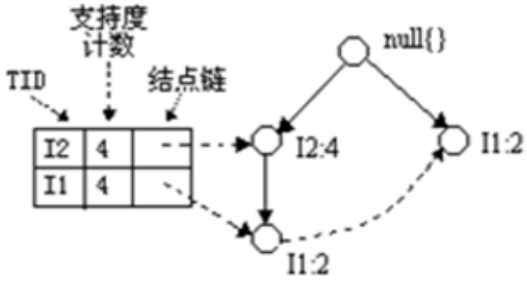
1)如果（条件）FP树为单个路径，则产生该路径下所有模式的组合。



e.g. I5: {I2, I5:2} , {I1,I5:2}, {I2,I1,I5:2}

- 若条件FP树为多路径，则针对树的每一个频繁1项集，产生对应的**条件模式基**获取频繁模式，如此处在I3项前分别加上I2、I1项，以(I2、I3和I1、I3)为后缀，寻找它们的前缀路径，从而构建新的FP树

2) 如果（条件）FP树为多路径，则针对树的头表中的每个项， 产生对应模式获取频繁模式。



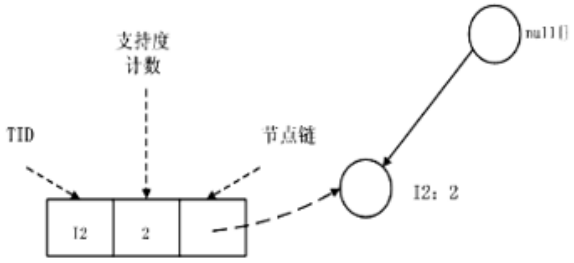
项 I3

{I2,I3: 4}

然后以 {I2,I3} 构造条件模式基 （空）

{I1,I3: 4}

然后以{I1,I3}构造条件模式基 {I2:2}
构造条件FP树



{I2，I1，I3：2}

👉 算法步骤

算法：FP-Growth。使用 FP 树，通过模式增长挖掘频繁模式。
输入：
- D：事务数据库。
- min_sup：最小支持度阈值。

输出：频繁模式的完全集。
方法：
1. 按以下步骤构造 FP 树：
   (a) 扫描事务数据库 D 一次。收集频繁项的集合 F 和它们的支持度计数。对 F 按支持度计数降序排序，结果为频繁项列表 L。
   (b) 创建 FP 树的根结点，以“null”标记它。对于 D 中每个事务 *Trans*，执行：
   选择 *Trans* 中的频繁项，并按 L 中的次序排序。设 *Trans* 排序后的频繁项列表为 $[p \mid P]$，其中 p 是第一个元素，而 P 是剩余元素的列表。调用 **insert_tree**$([p \mid P], T)$。该过程执行情况如下。如果 T 有子女 N 使得 $N.item\text{-}name = p.item\text{-}name$，则 N 的计数增加 1；否则，创建一个新结点 N，将其计数设置为 1，链接到它的父结点 T，并且通过结点链结构将其链接到具有相同 *item-name* 的结点。如果 P 非空，则递归地调用 **insert_tree**(P, N)。
2. FP 树的挖掘通过调用 **FP_growth**$(FP\_tree, null)$ 实现。该过程实现如下。

```
procedure FP_growth(Tree, α)
(1) if Tree 包含单个路径 P then
(2) for 路径 P 中结点的每个组合(记作 β)
(3) 产生模式 β∪α,其支持度计数 support_count 等于 β 中结点的最小支持度计数;
(4) else for Tree 的头表中的每个 a_i {
(5) 产生一个模式 β = a_i ∪ α,其支持度计数 support_count = a_i.support_count;
(6) 构造 β 的条件模式基,然后构造 β 的条件 FP 树 Tree_β;
(7) if Tree_β ≠ ∅ then
(8) 调用 FP_growth(Tree_β, β); }
```

规则产生

☞ 将一个频繁项集，Y，划分为两个非空的子集，X，和，$Y-X$，，使得，$X \rightarrow Y-X$，满足置信度阈值，min_conf，

☞ 计算关联规则的置信度**不需要再次扫描数据库**，因为分割后的两个子集一定是频繁项集(一个频繁项集的子集一定是频繁的)，这两个集合的支持度都是已经计算得到的

关联模式的评估

☞ 建立一组广泛接收的**评价**关联模式**质量**的**标准**是很重要的

- 通过统计论据建立
- 主观论据的建立

☞ 现有关联规则的挖掘算法依赖**支持度和置信度**来去除没有意义的模式。但是可能会出现误导

例子：假定希望分析爱喝咖啡和爱喝茶的人之间的关系。收集一组人关于饮料偏爱的信息，并汇总到下表6-8。

	Coffee	$\overline{\text{Coffee}}$	
Tea	150	50	200
$\overline{\text{Tea}}$	650	150	800
	800	200	1000

可以使用表中信息来评估关系规则{茶} →{咖啡}。

- ✓ 似乎喜欢喝茶的人也喜欢喝咖啡，因为该规则的**支持度**（15%）和**置信度**（75%）都相当高。
- ✓ 但是所有人中，不管他是否喝茶，喝咖啡的人的比例为80%。这意味着，一个人如果喝茶，则他喝咖啡的可能性由80%减到了75%。

👆原因在于：置信度忽略了**规则后件(即👆的咖啡)中项集的支持度**

为了解决这个问题，提出了新的度量

- 提升度
  - 关联规则置信度与规则后件项集支持度的比值
- 兴趣因子
  - 二元变量的度量

解决这个问题的一种方法是使用称作**提升度**（lift）的度量：

$$Lift(A \rightarrow B) = \frac{c(A \rightarrow B)}{s(B)}$$

计算规则置信度和规则后件中项集的支持度之间的比率

对于二元变量，提升度等价于另一种称作**兴趣因子**（interest factor）的客观度量，其定义如下：

$$I(A,B) = \frac{s(A,B)}{s(A) \times s(B)}$$