第四章 关联分析

目录
CONTENTS

4.1 关联分析基础概念
4.2 关联分析算法
4.3 关联分析高级概念

Chapter 4.1

关联分析基础概念

关联分析基础概念

- 购物篮分析

关联分析基础概念

购物篮数据

- 例子中的交易数据通常采用表1的形式记录，被称作**购物篮事务**（Market basket transaction）,表中每一行对应一个事务，包含一个唯一标识 TID 和对应顾客购买的商品的集合。

表1. 购物篮数据示例

TID	项集
1	{面包，牛奶}
2	{面包，尿布，啤酒，鸡蛋}
3	{牛奶，尿布，啤酒，可乐}
4	{面包，牛奶，尿布，啤酒}
5	{面包，牛奶，尿布，可乐}

关联分析基础概念

购物篮数据

- 购物篮数据可以采用表2所示的二元形式表示，其中每行对应一个事务，每列对应一个**项**（Item）。因为通常认为项在事务中出现的情况更重要，所以项是**非对称的二元属性**。
- 项的二元表示是购物篮数据最简单、最直观的表示形式，但是忽略了项的数量和价格的因素。

表2. 购物篮数据的二元表示

TID	面包	牛奶	尿布	啤酒	鸡蛋	可乐
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

关联分析基础概念

购物篮数据

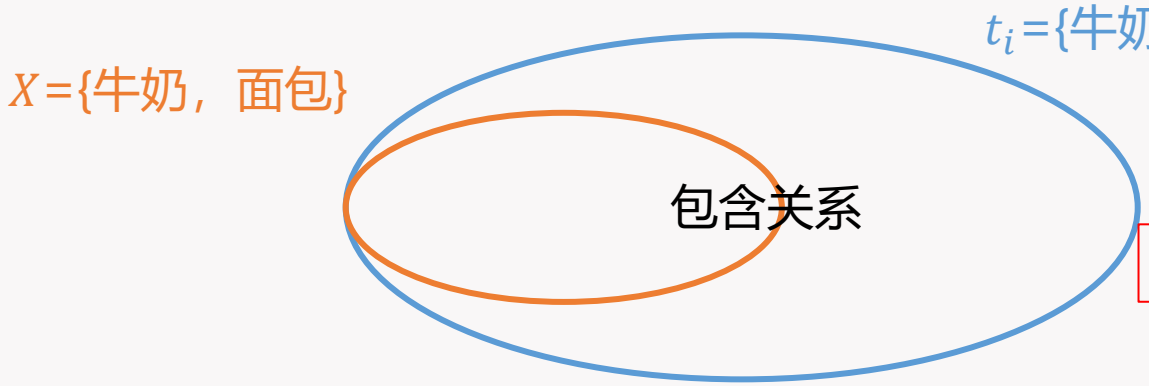
- 令 $I = \{i_1, i_2, \ldots, i_d\}$ 表示购物篮数据中所有项的集合，$T = \{t_1, t_2, \ldots, t_N\}$ 表示购物篮数据中所有事务的集合。
- 包含 0 个或多个项的集合被称作**项集**（Itemset）。如果一个项集包含 k 个项，则称它为 k-项集。如果项集不包含任何项，则被称作空集。

		i_1	i_2	i_3	i_4	i_5	i_6
	TID	面包	牛奶	尿布	啤酒	鸡蛋	可乐
t_1	1	1	1	0	0	0	0
t_2	2	1	0	1	1	1	0
t_3	3	0	1	1	1	0	1
t_4	4	1	1	1	1	0	0
t_5	5	1	1	1	0	0	1

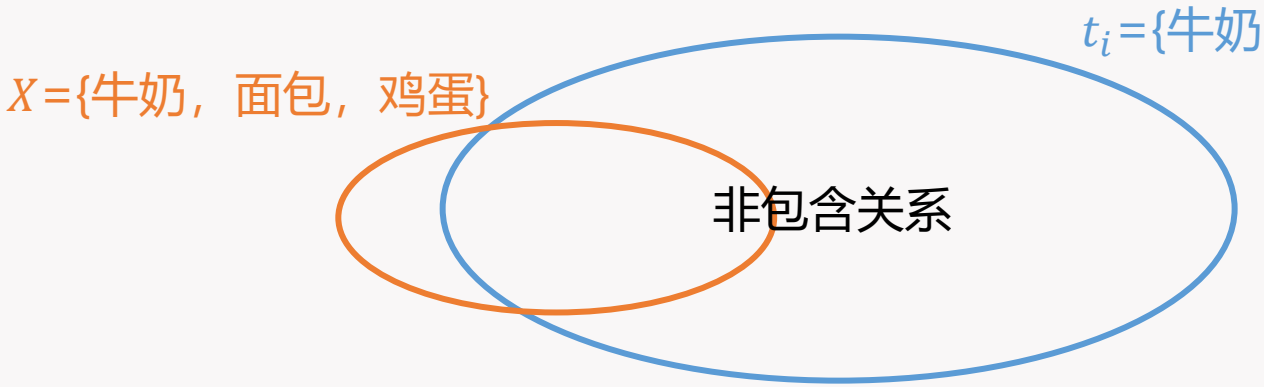
关联分析基础概念

项集与事务的包含关系

- 如果项集 X 是事务 t_i 中包含的项的子集，则称事务 t_i 包含项集 X 。数学上，事务 t_i 包含项集 X，当且仅当 $X \subseteq t_i$。



隐含的包含关系： $\forall i, t_i \subseteq I$



关联分析基础概念

支持度计数

- 项集的一个重要性质是可以根据与事务的包含关系进行支持度计数。数学上，项集 X 的**支持度计数**表示为

$$\sigma(X) = |\{ t_i \mid X \subseteq t_i , t_i \in T\}|$$

其中|·|表示统计集合中元素的个数。如果项集 X 的支持度计数**大于等于**最小支持度阈值 p_{sup}，则称 X 是**频繁项集**（Frequent itemset）。

- 示例：

TID	面包	牛奶	尿布	啤酒	鸡蛋	可乐
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

项集 X ={面包，尿布，啤酒}的支持度计数是多少?

包含 X 的事务有 t_2和t_4。所以

$\sigma(X) = |\{t_2, t_4\}|$=2

关联分析基础概念

关联规则相关概念

- **关联规则**（Association rule）是形如 $X \rightarrow Y$ 的蕴含表达式，其中 X 和 Y 指的是不相交的项集，即 $X \cap Y = \emptyset$。关联规则的强度可以使用其在事务集 T 上的**支持度**（Support）和**置信度**（Confidence）进行度量。

- 支持度用来确定 X 和 Y 在给定事务集上的同时出现的频繁程度，定义为

$$s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$$

- 置信度用来确定 Y 在包含 X 的事务中出现的频繁程度，定义为

$$c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

- 支持度反映了关联规则的有用性，置信度反映了关联规则的确定性。

关联分析基础概念

关联规则相关概念

- 支持度表示事务集中包含项集 $X \cup Y$ 的事务的百分比，也可以表示为

$$s(X \rightarrow Y) = P(X \cup Y)$$

- 置信度表示事务集中包含项集 X 的事务中包含项集 Y 的事务的百分比，也可以表示为

$$c(X \rightarrow Y) = P(Y \mid X)$$

- 一般情况下，当一个规则满足最小支持度阈值 p_{sup} 时，这个规则被认为是有趣的，否则被称作**非频繁的**。特别的，当一个规则**同时满足**最小支持度阈值和最小置信度阈值时被称为**强规则**（Strong rule）。

关联分析基础概念

关联规则相关概念

- 示例：

TID	面包	牛奶	尿布	啤酒	鸡蛋	可乐
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	1
5	1	1	1	0	0	1

- 项集 X_1 = {面包，牛奶}，Y_1 = {尿布}，规则 $X_1 \rightarrow Y_1$ 的支持度和置信度是多少？
- 项集 X_2 = {尿布，牛奶}，Y_2 = {可乐}，规则 $X_2 \rightarrow Y_2$ 的支持度和置信度是多少？
- 最小支持度阈值是30%，最小置信度阈值是80%，以上哪个关联规则是强规则？

$$s(X_1 \rightarrow Y_1) = \frac{\sigma(X_1 \cup Y_1)}{N} = \frac{2}{5} = 40\% \qquad c(X_1 \rightarrow Y_1) = \frac{\sigma(X_1 \cup Y_1)}{\sigma(X_1)} = \frac{2}{3} = 66.67\%$$

$$s(X_2 \rightarrow Y_2) = \frac{\sigma(X_2 \cup Y_2)}{N} = \frac{3}{5} = 60\% \qquad c(X_2 \rightarrow Y_2) = \frac{\sigma(X_2 \cup Y_2)}{\sigma(X_2)} = \frac{3}{3} = 100\%$$

关联分析基础概念

关联规则挖掘问题的形式描述

- 给定事务集 T，关联规则发现是指找出支持度**大于等于**最小支持度阈值 p_{sup}，并且置信度**大于等于**最小置信度阈值 p_{con} 的所有**强规则**。支持度和置信度的定义为

$$s(X \to Y) = \frac{\sigma(X \cup Y)}{N}$$

和

$$c(X \to Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

其中 $\sigma(X)$ 表示项集 X 在事务集 T 上的支持度计数，表示为

$$\sigma(X) = |\{ t_i \mid X \subseteq t_i , t_i \in T\}|$$

Chapter 4.2

关联分析算法

关联分析算法

关联分析的朴素算法

- 对于所有项的集合 I，列出所有的可能的项集 X，然后列出项集之间所有可能的规则，计算每个规则的支持度和置信度。这种做法的缺点如下：
  - 计算代价非常高，对于包含 d 个项的集合 I，规则的总数为

$$n = 3^d - 2^{d+1} + 1$$

当 d 很大时，规则的数量达到了天文数字！对于一个包含10个项的事务集，总规则数为 57,002 个。

  - 大部分规则的支持度和置信度都不会超过阈值，所以大部分的计算都是无用的。

关联分析算法

关联分析算法的通用策略

- 大多数关联分析算法都将关联规则挖掘任务分解为以下两个子任务：
  - **找出所有的频繁项集**：发现所有满足最小支持度阈值的项集，即找出所有频繁项集；
  - **由频繁项集产生强关联规则**：从上一步发现的频繁项集中提取所有置信度高于最小置信度阈值的规则，即找出所有的强规则。
- 通常情况下，第一步的计算花销远远高于第二步的计算花销，因此关联分析算法的总体性能由第一步决定。

关联分析算法

- 研究路线

找出所有的强规则（遍历）

提高效率

找出所有的频繁项集

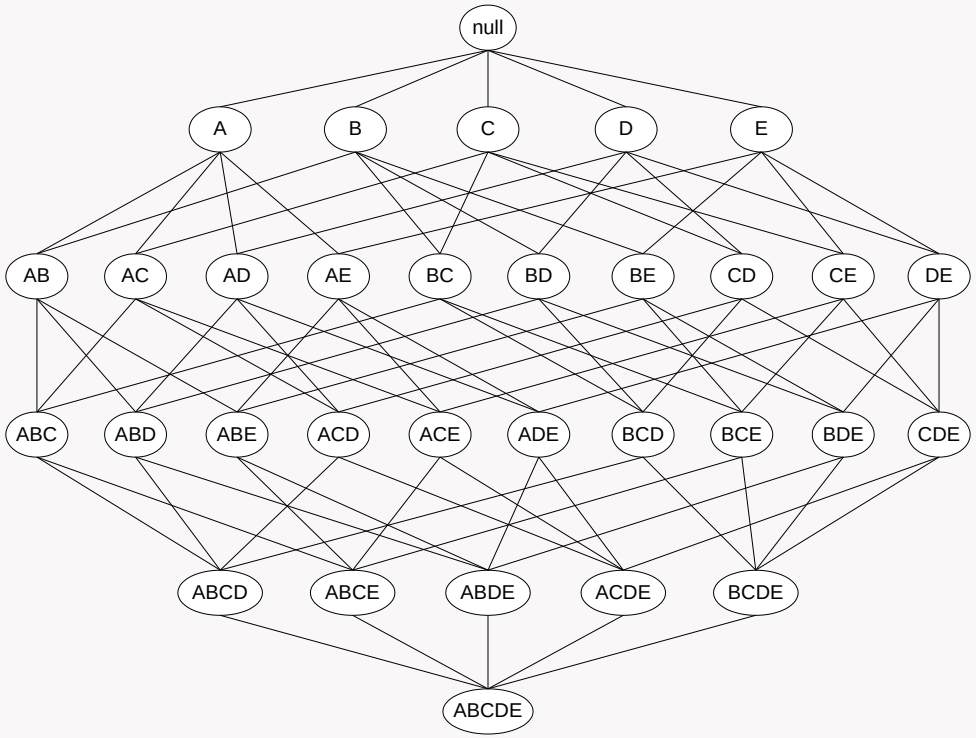
项集出现的次数大于等于最小支持度阈值

利用频繁项集产生强规则

关联规则满足最小支持度阈值和最小置信度阈值

频繁项集挖掘方法

- 在寻找频繁项集时，可以使用**格结构**（Lattice structure）枚举所有可能的项集。但是一个包含了 d 个项的 I 可能产生 $2^d - 1$ 个非空项集，指数规模的搜索空间极大增加了挖掘频繁项集的开销。



关联分析算法

Apriori算法

- Apriori算法由Agrawal 和 Srikant 于1993年提出的，为**布尔型**关联规则挖掘频繁项集的原创性算法。
- Apriori 意为先验的，来源于频繁项集先验原理的使用。
- Apriori是一种迭代方法，利用 k-项集去探索 (k+1)-项集：
  - 首先扫描事务集，找出频繁1-项集的集合，记为 F_1；
  - 利用频繁1-项集（ F_1 ）找出频繁2-项集，记为 F_2 ；
  - 重复第二个步骤，直到不能再找到频繁 k-项集（ F_k ）。

利用 F_{k-1} 找出 F_k 包含两个步骤：连接步和剪枝步

关联分析算法

Apriori算法-连接步

- 为了找出频繁k-项集，需要利用频繁(k-1)-项集构造候选k-项集的集合：
  - 遍历法
  - $F_k \times F_1$连接法
  - $F_k \times F_k$连接法
- 候选项集指的是需要通过支持度计数来判断是否是频繁的项集。

连接步的要求

- 构造候选k-项集时需要考虑以下几个要求：
  - 应当避免产生太多不必要的候选项集，不必要的候选项集指的是至少有一个子集是非频繁的（基于后续的先验原理可被排除）；
  - 应当确保产生的候选项集的集合是完全的，即包含所有可能的频繁项集；
  - 应当避免产生重复的候选项集，比如根据频繁2-项集生成候选3-项集时，{b,c}和{b,d}，{b,c}和{c,d}，{b,d}和{c,d}合并的结果都为{b,c,d}，所以重复产生候选项集会增加无用的计算开销。

关联分析算法

连接步的实现

- **遍历法**：使用1-项集生成候选(k+1)-项集。
- 优点：
  - 简单直观；
  - 候选项集的集合是完全的；
  - 不产生重复的候选项集。
- 缺点：
  - 产生了太多不必要的候选项集。因为遍历法会产生C_d^{k+1}个候选项集，d 是 I 的大小。当 d 很大时会产生非常多的候选项集，但是绝大多数都不可能是频繁项集。

1-项集

项集
{面包}
{牛奶}
{尿布}
{啤酒}
{鸡蛋}
{可乐}

候选3-项集

项集
{面包，牛奶，尿布}
{面包，牛奶，啤酒}
{面包，牛奶，鸡蛋}
{面包，牛奶，可乐}
{面包，尿布，啤酒}
……

关联分析算法

连接步的实现

- $F_k \times F_1$**法**：使用频繁k-项集和频繁1-项集构造候选(k+1)-项集。
- 优点：
  - 候选项集的集合是完全的；
  - 极大地减少了不必要的候选项集的数目；
- 缺点：
  - 产生了大量的重复候选项集。

频繁2-项集 × 频繁1-项集 → 候选3-项集

项集
{面包，牛奶}
{面包，尿布}
{牛奶，尿布}
{尿布，啤酒}

×

项集
{面包}
{牛奶}
{尿布}
{啤酒}

→

项集
{面包，牛奶，尿布}
{面包，牛奶，啤酒}
{面包，尿布，啤酒}
{牛奶，尿布，啤酒}

关联分析算法

连接步的实现

- **$F_k \times F_k$法**：使用频繁k-项集构造候选(k+1)-项集。假设项按某种顺序排序，两个频繁k-项集分别表示为$A = \{a_1, a_2, \ldots, a_k\}$ 和 $B = \{b_1, b_2, \ldots, b_k\}$，合并 A 和 B，如果它们满足以下条件：

$$a_i = b_i (i = 1,2, \ldots, k-1)\ and\ a_k \neq b_k$$

- 优点：
  - 候选项集的集合是完全的；
  - 更少的不必要候选项集；
  - 更少的重复候选项集。

频繁2-项集 × 频繁2-项集 → 候选3-项集

项集
{面包，牛奶}
{面包，尿布}
{牛奶，尿布}
{尿布，啤酒}

×

项集
{面包，牛奶}
{面包，尿布}
{牛奶，尿布}
{尿布，啤酒}

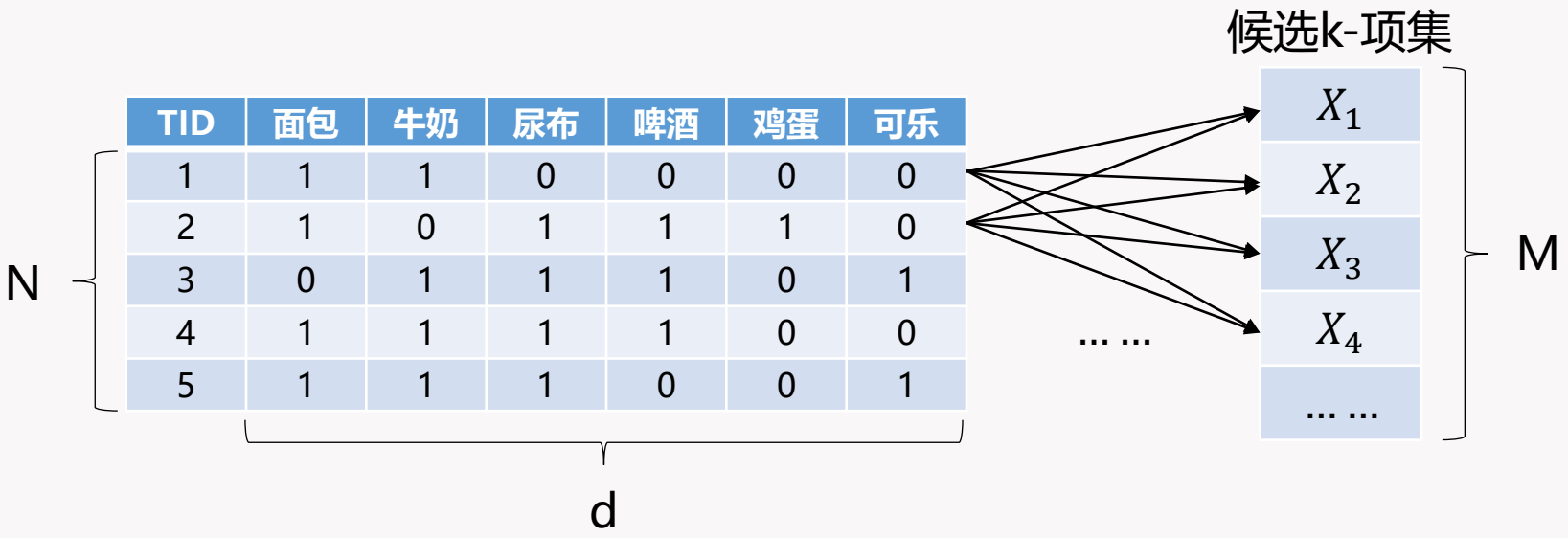
→

项集
{面包，牛奶，尿布}

排序顺序：面包 | 牛奶 | 尿布 | 啤酒 | 鸡蛋 | 可乐

Apriori算法-剪枝步

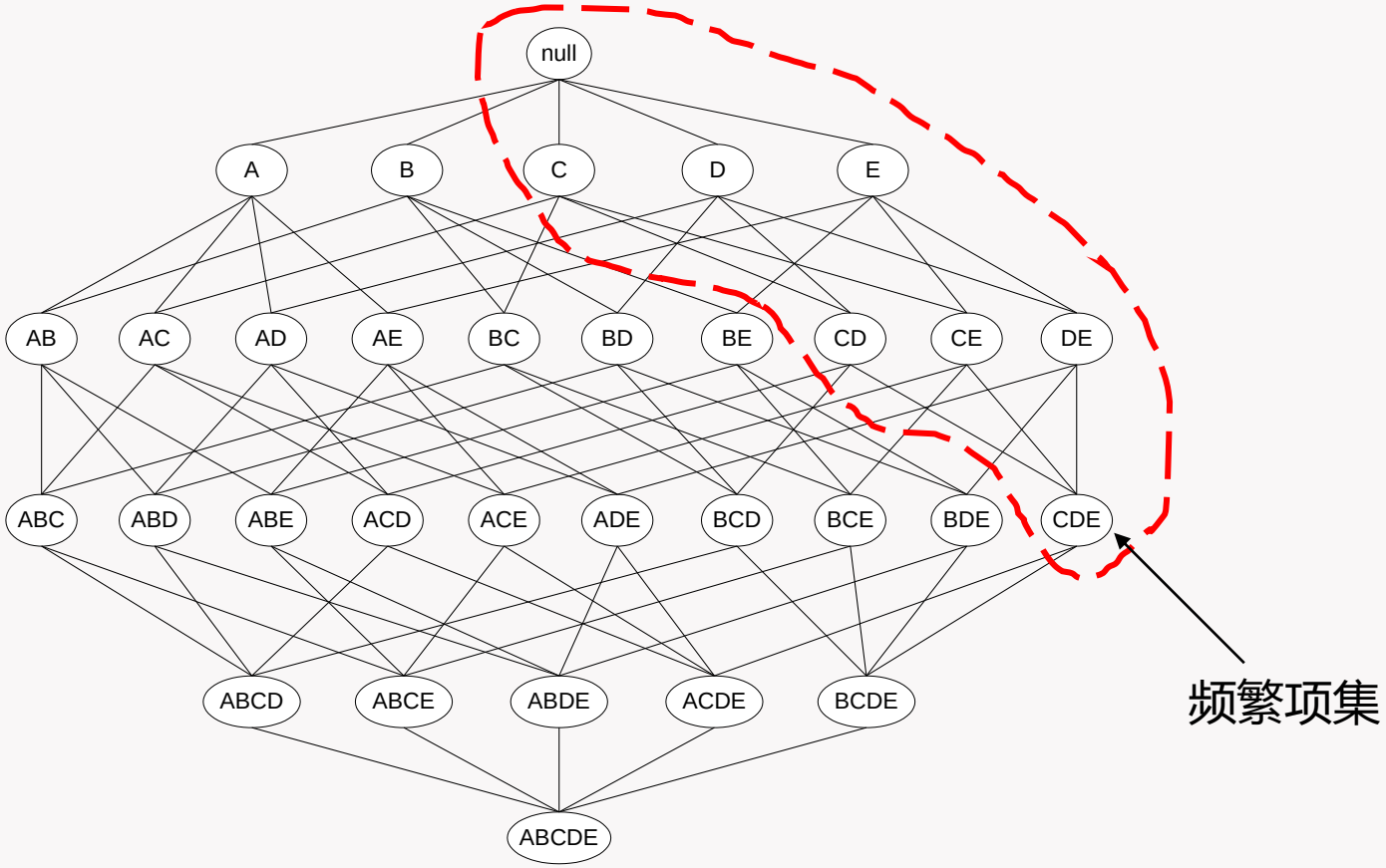
- 确定候选k-项集后，需要扫描事务集判断每个候选k-项集是否是频繁的，运算复杂度为 $O(NMd)$。
- 为了压缩候选k-项集，Apriori算法借助于频繁项集的先验原理，引入了**剪枝步**。



关联分析算法

先验原理

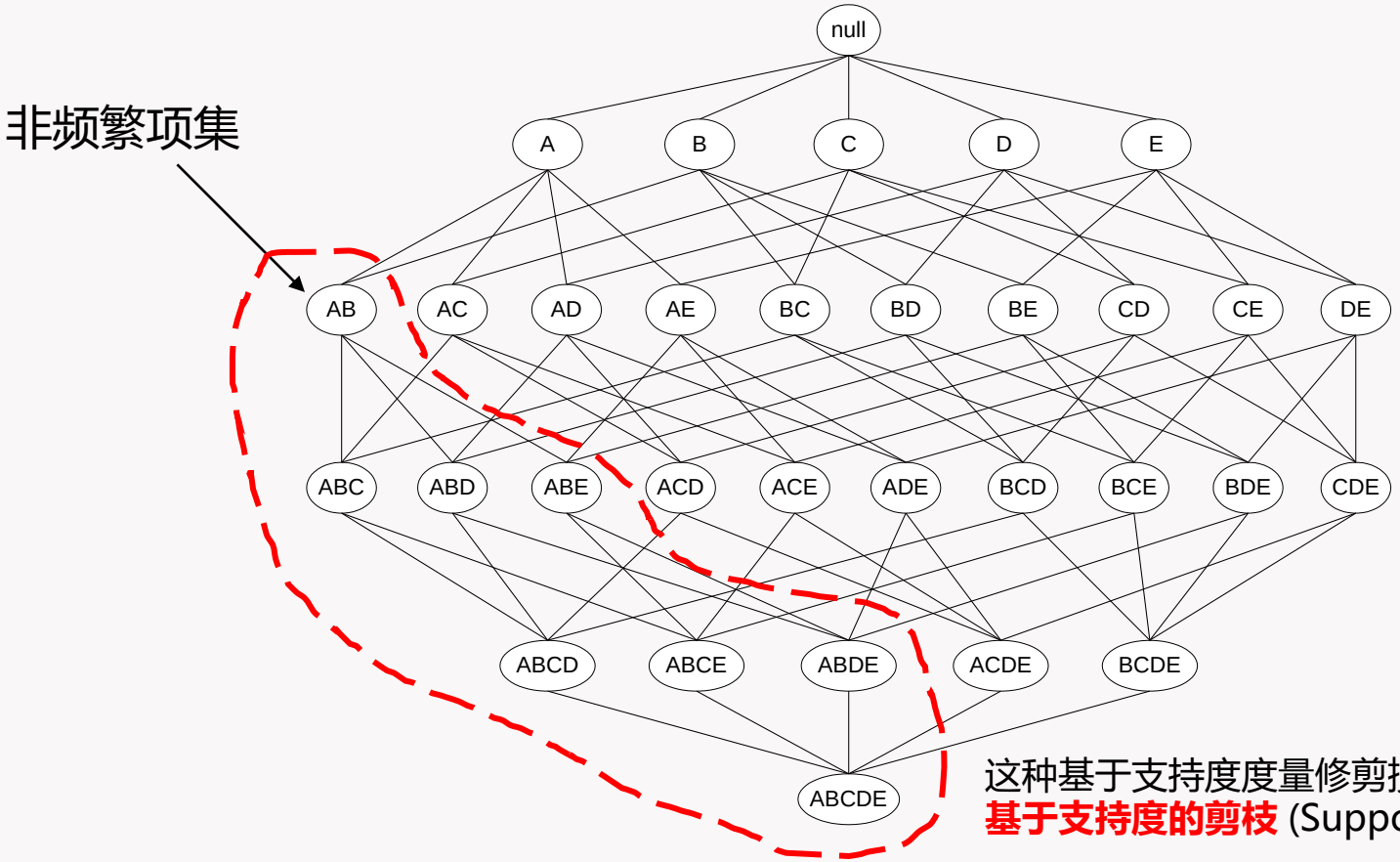
- **如果一个项集是频繁的，则它的所有子集一定也是频繁的。**



关联分析算法

先验原理

- **如果一个项集是非频繁的，那么它的所有超集也是非频繁的。**



这种基于支持度度量修剪搜索空间的策略称为**基于支持度的剪枝** (Support-based pruning)

关联分析算法

Apriori算法-剪枝步

- 如果候选k-项集中存在一个(k-1)项的子集不在频繁(k-1)-项集中，则该候选k-项集是非频繁的。
- 示例：
  - 频繁2-项集：{面包，牛奶}，{面包，尿布}，{面包，啤酒}，{牛奶，尿布}
  - 候选3-项集：{面包，牛奶，尿布}，~~{面包，尿布，啤酒}~~

关联分析算法

Apriori算法

- Apriori是一种迭代方法，利用 k-项集去探索 (k+1)-项集：
  - 首先扫描事务集，找出频繁1-项集的集合，记为 L_1；
  - 连接步：利用频繁1-项集（L_1）找出候选2-项集（C_2）；
  - 剪枝步：去除候选2-项集（C_2）中不可能为频繁2-项集的项集；
  - 筛选频繁项集：扫描事务集，统计候选2-项集（C_2）中项集的支持度计数，与支持度计数阈值比较并得到频繁2-项集（L_2）；
  - 重复第二至第五个步骤，直到不能再找到频繁 k-项集（L_k）。

关联分析算法

支持度计数

- 为了判断每个候选项集是否是频繁项集，需要遍历事务集并计算每个候选项集的支持度计数。遍历法将每个事务依次和所有的候选项集进行比较，当事务和候选项集的数量很多时，这种方法的计算代价过高。为了进一步减少运算复杂度，可采用以下技术：

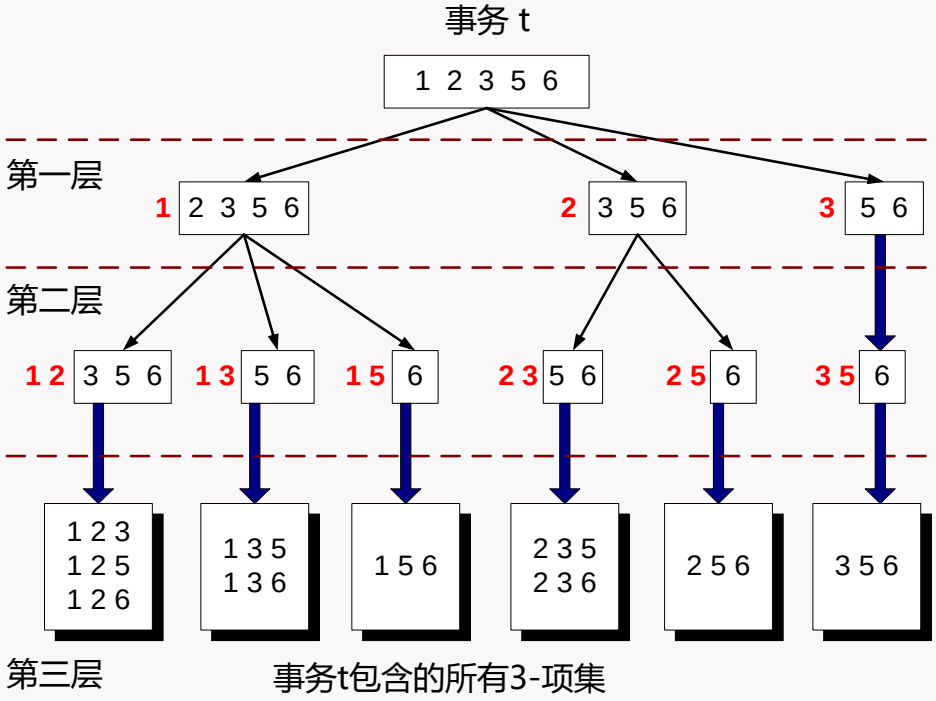
- 构造事务的项集树
- 构造候选项集的hash树

关联分析算法

基于事务的支持度计数实现

- 对于每个事务 t，枚举事务 t 包含的所有k-项集并增加它们的支持度计数。假定每个项按顺序排序，则项集的枚举先指定最小项，再指定较大的项。最后比较事务的3-项集与候选3-项集，计算候选3-项集的支持度计数。

面包	牛奶	尿布	啤酒	鸡蛋	可乐
1	2	3	4	5	6



关联分析算法

基于hash树的支持度计数实现

- 以hash树的结构来存储所有的候选项集，每个事务不再和每个候选项集比较，而是和hash树中特定的候选项集比较。
- 输入参数：
  - hash函数，$h(x) = x \ mod \ 3$；
  - 叶节点最大尺寸，3。

关联分析算法

基于hash树的支持度计数实现

- 以hash树的结构来存储所有的候选项集，每个事务不再和每个候选项集比较，而是和hash树中特定的候选项集比较。

- 首先对项进行编码：

面包	牛奶	尿布	啤酒	鸡蛋	可乐	蛋糕	饼干	饮料
1	2	3	4	5	6	7	8	9

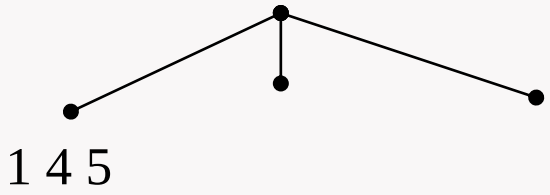
- 假设候选3-项集为

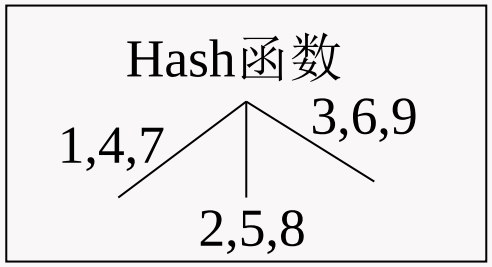
项集		
{1，4，5}	{1，5，9}	{3，5，6}
{1，2，4}	{1，3，6}	{3，5，7}
{4，5，7}	{2，3，4}	{6，8，9}
{1，2，5}	{5，6，7}	{3，6，7}
{4，5，8}	{3，4，5}	{3，6，8}

关联分析算法

基于hash树的支持度计数实现

- 使用hash函数：$h(x) = x \ mod \ 3$ 确定沿着当前结点的哪个分支向下。
- 对于项集{1，4，5}，根据第一项是1，进入hash树左分支。



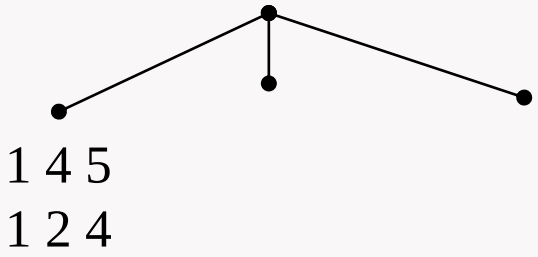


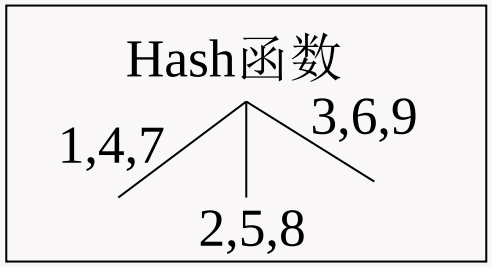
项集		
{1，4，5}	{1，5，9}	{3，5，6}
{1，2，4}	{1，3，6}	{3，5，7}
{4，5，7}	{2，3，4}	{6，8，9}
{1，2，5}	{5，6，7}	{3，6，7}
{4，5，8}	{3，4，5}	{3，6，8}

关联分析算法

基于hash树的支持度计数实现

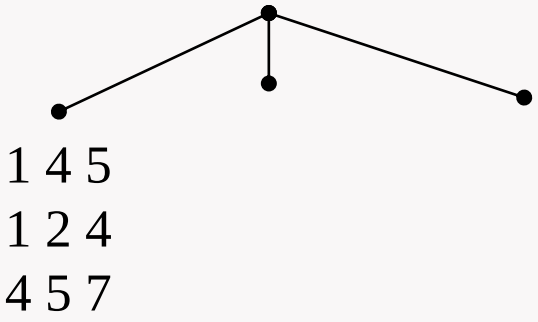
- 使用hash函数：$h(x) = x \ mod \ 3$ 确定沿着当前结点的哪个分支向下。
- 对于项集{1，2，4}，根据第一项是1，进入hash树左分支。

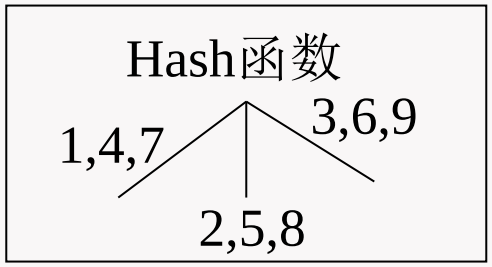




项集		
{1，4，5}	{1，5，9}	{3，5，6}
{1，2，4}	{1，3，6}	{3，5，7}
{4，5，7}	{2，3，4}	{6，8，9}
{1，2，5}	{5，6，7}	{3，6，7}
{4，5，8}	{3，4，5}	{3，6，8}

- 基于hash树的支持度计数实现
  - 使用hash函数：$h(x) = x \ mod \ 3$ 确定沿着当前结点的哪个分支向下。
  - 对于项集{4，5，7}，根据第一项是4，进入hash树左分支。



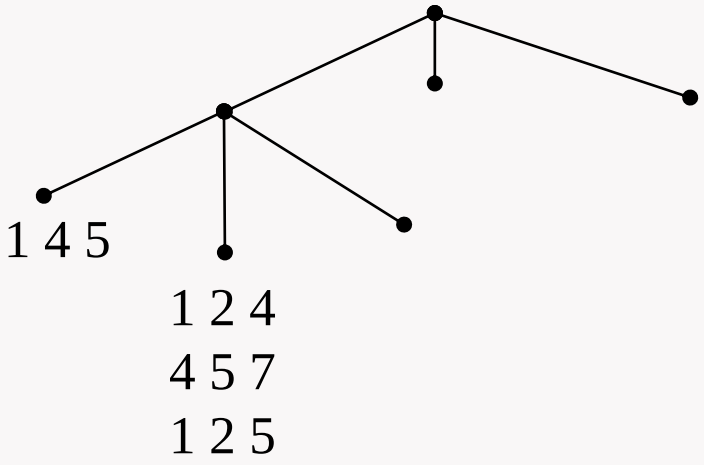


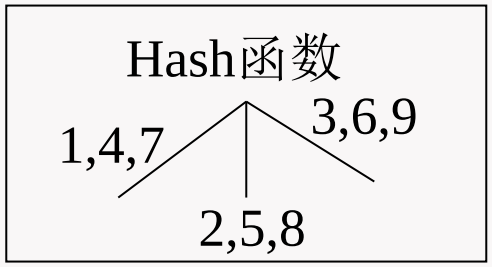
项集		
{1，4，5}	{1，5，9}	{3，5，6}
{1，2，4}	{1，3，6}	{3，5，7}
{4，5，7}	{2，3，4}	{6，8，9}
{1，2，5}	{5，6，7}	{3，6，7}
{4，5，8}	{3，4，5}	{3，6，8}

关联分析算法

基于hash树的支持度计数实现

- 使用hash函数：$h(x) = x \ mod \ 3$ 确定沿着当前结点的哪个分支向下。
- 对于项集{1，2，5}，根据第一项是1，进入hash树左分支。此时左分支中包含的候选项集的个数超过最大尺寸，根据第二项使用hash函数分裂。

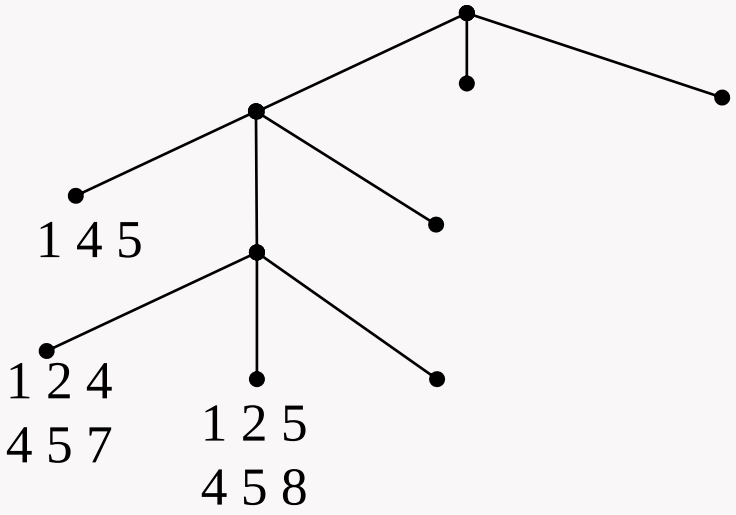


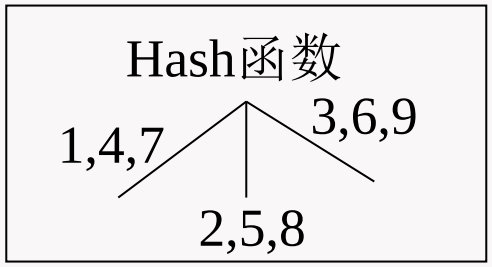


项集		
{1，4，5}	{1，5，9}	{3，5，6}
{1，2，4}	{1，3，6}	{3，5，7}
{4，5，7}	{2，3，4}	{6，8，9}
{1，2，5}	{5，6，7}	{3，6，7}
{4，5，8}	{3，4，5}	{3，6，8}

基于hash树的支持度计数实现

- 使用hash函数：$h(x) = x \ mod \ 3$ 确定沿着当前结点的哪个分支向下。
- 对于项集{4，5，8}，根据第一项是4，进入hash树左分支。根据第二项是5，进入hash树的中分支。因为中分支超过叶节点的最大尺寸，根据第三项进行分裂。

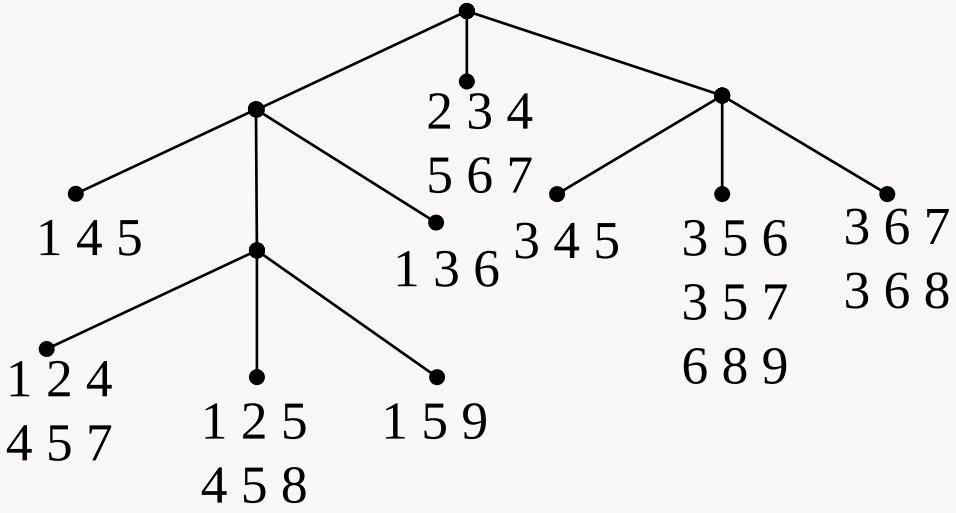


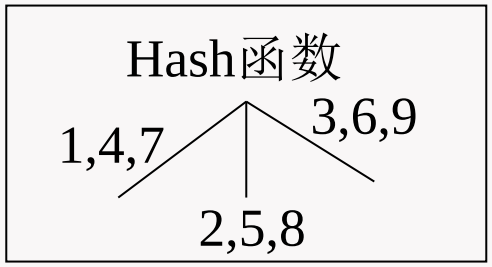


项集		
{1，4，5}	{1，5，9}	{3，5，6}
{1，2，4}	{1，3，6}	{3，5，7}
{4，5，7}	{2，3，4}	{6，8，9}
{1，2，5}	{5，6，7}	{3，6，7}
{4，5，8}	{3，4，5}	{3，6，8}

关联分析算法

- 基于hash树的支持度计数实现
  - 使用hash函数：$h(x) = x \ mod \ 3$ 确定沿着当前结点的哪个分支向下。
  - 依次输入剩下的项集，最终得到完整的hash树。

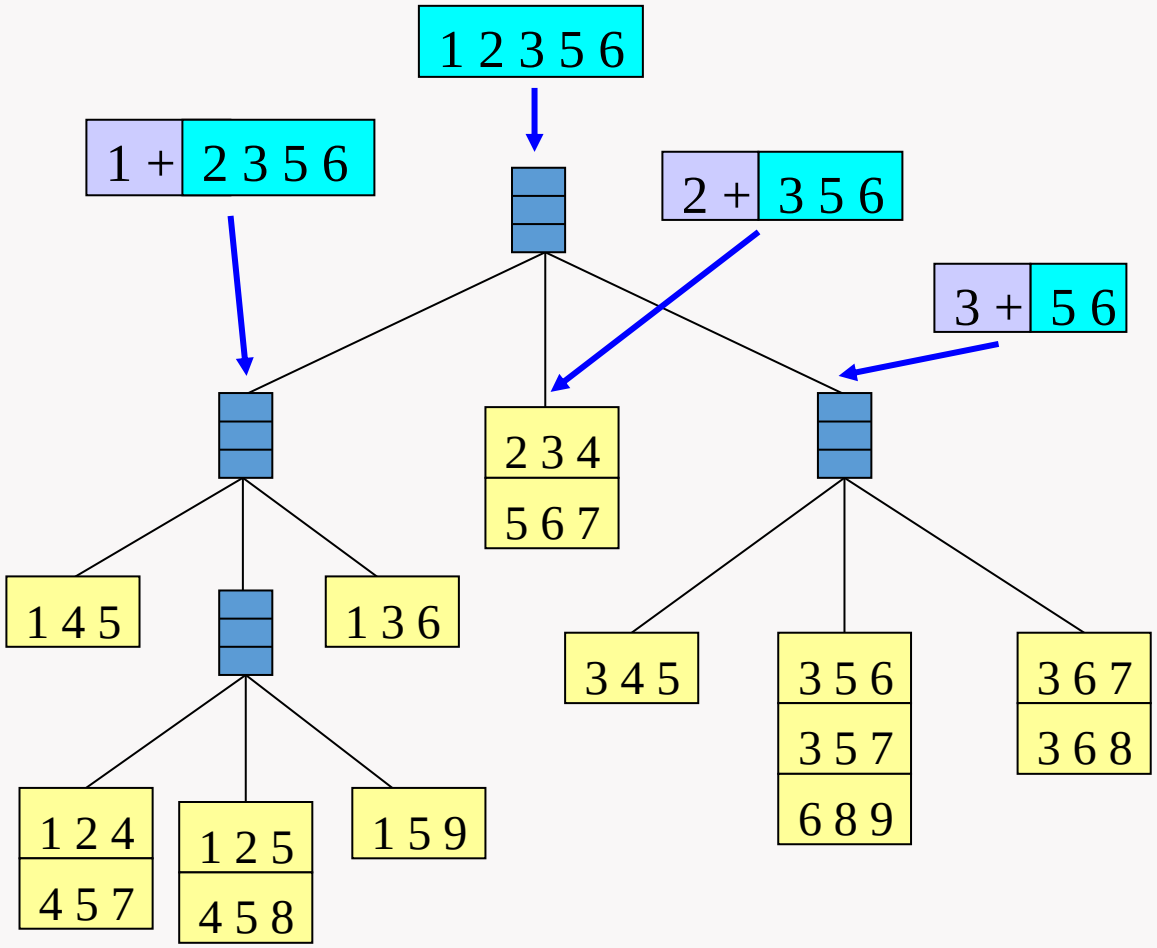


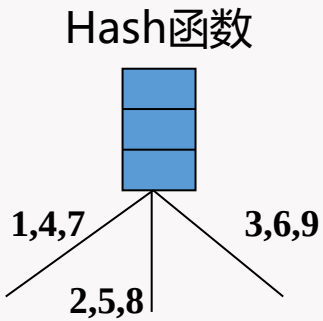


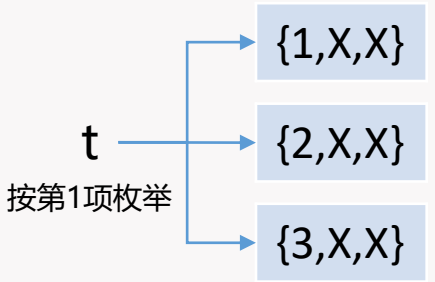
项集		
{1，4，5}	{1，5，9}	{3，5，6}
{1，2，4}	{1，3，6}	{3，5，7}
{4，5，7}	{2，3，4}	{6，8，9}
{1，2，5}	{5，6，7}	{3，6，7}
{4，5，8}	{3，4，5}	{3，6，8}

- 基于hash树的支持度计数实现

➢ 假设事务 t={1,2,3,5,6}，使用事务 t 进行候选3-项集的支持度计数。

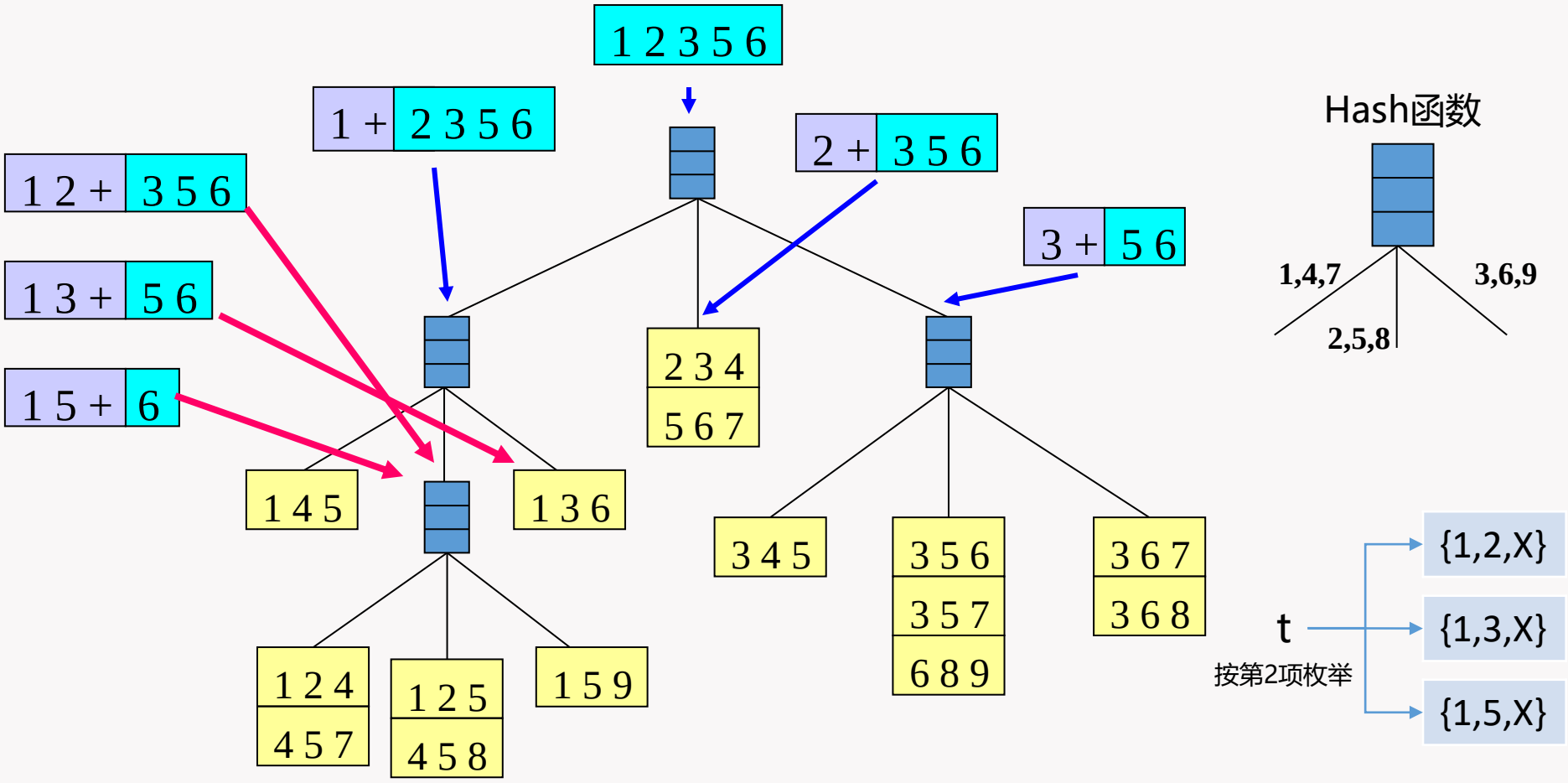






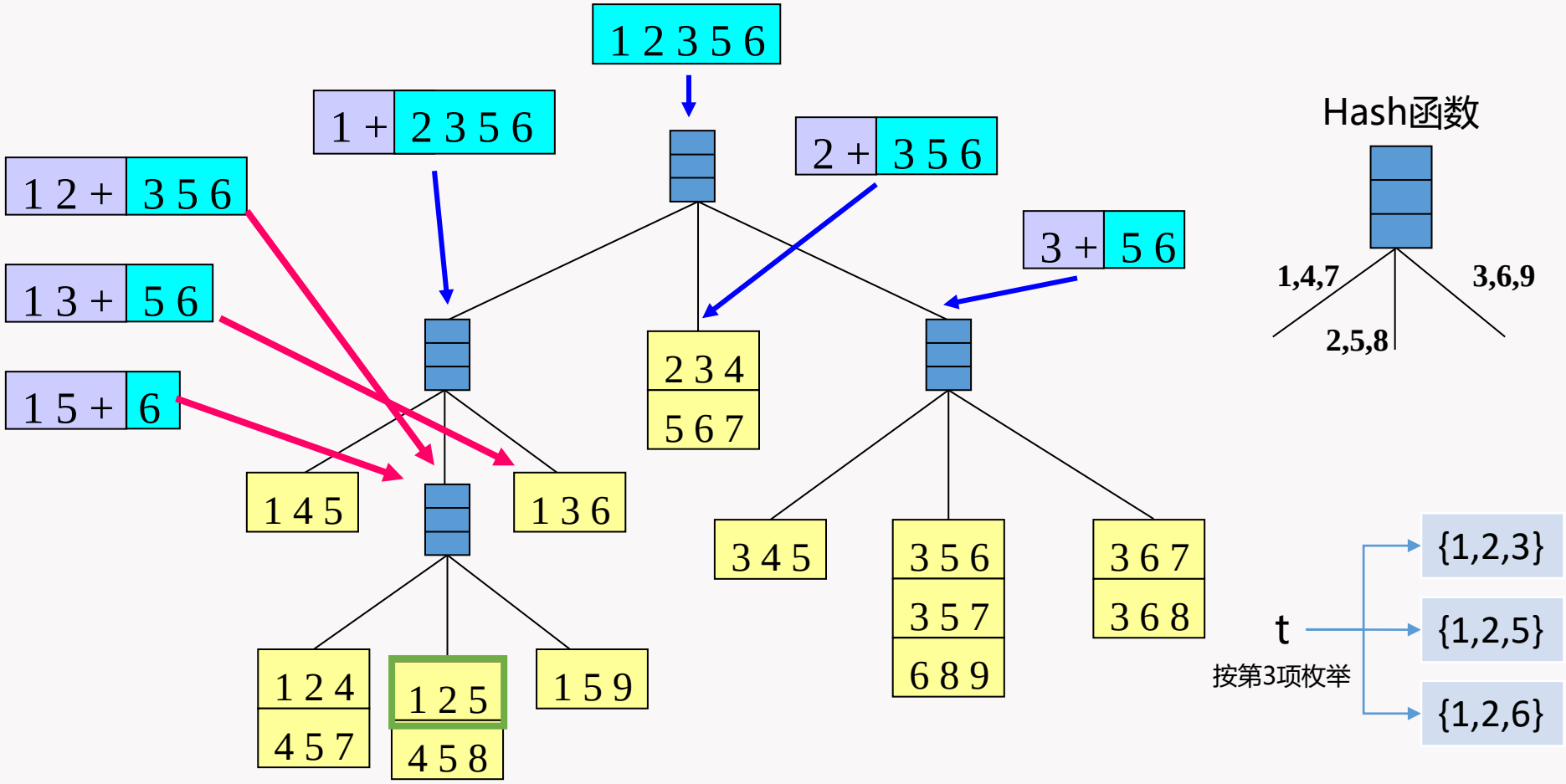
关联分析算法

- 基于hash树的支持度计数实现
  - 假设事务 t={1,2,3,5,6}，使用事务 t 进行候选3-项集的支持度计数。



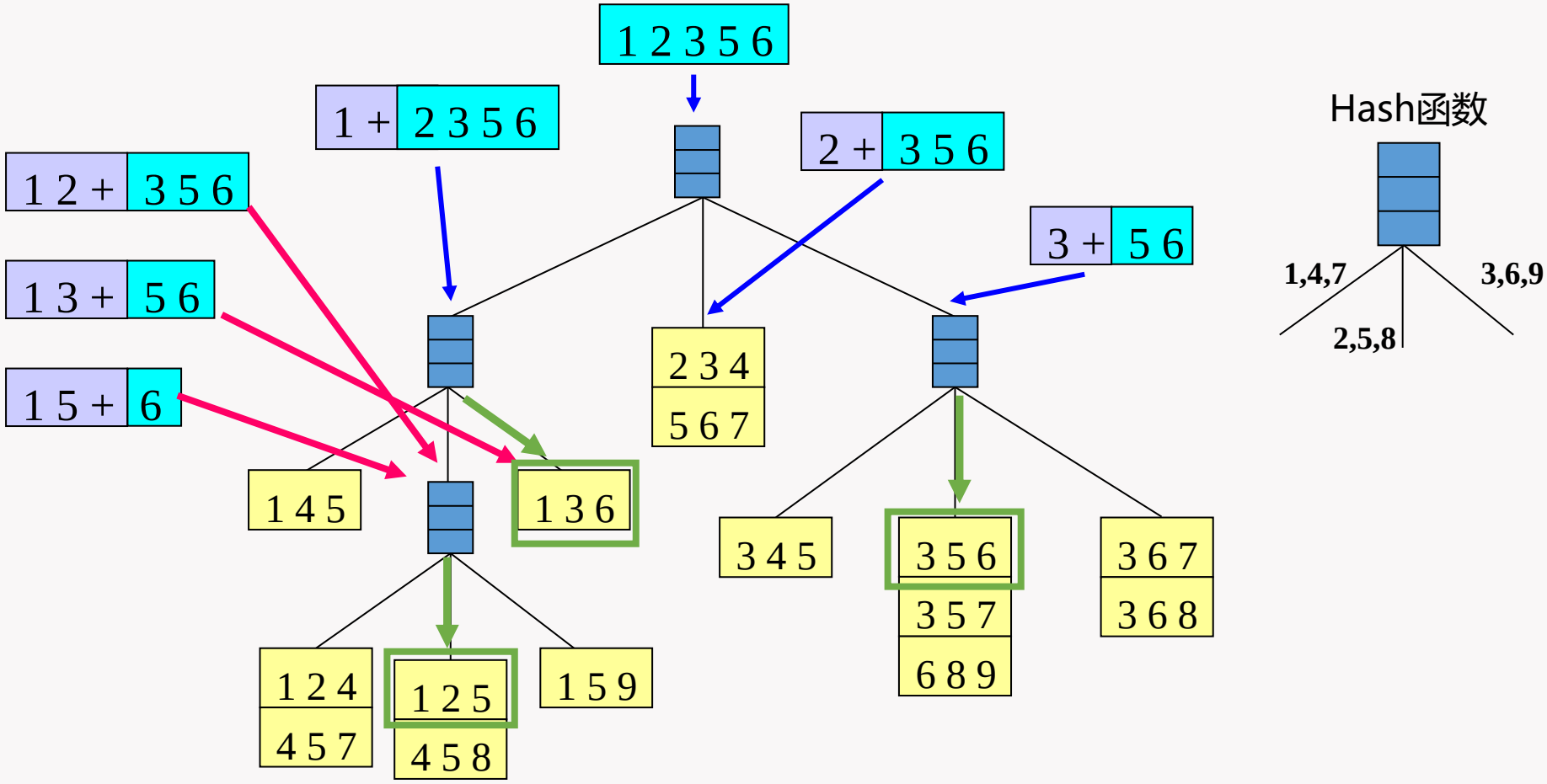
关联分析算法

- 基于hash树的支持度计数实现
  - 假设事务 t={1,2,3,5,6}，使用事务 t 进行候选3-项集的支持度计数。



关联分析算法

- 基于hash树的支持度计数实现
  - 假设事务 t={1,2,3,5,6}，使用事务 t 进行候选3-项集的支持度计数。



关联分析算法

Apriori算法回顾

- Apriori是一种迭代方法，利用 k-项集去探索 (k+1)-项集：
  - 首先扫描事务集，找出频繁1-项集的集合，记为 L_1；
  - 连接步：利用频繁1-项集（ L_1 ）找出候选2-项集（ C_2 ）；
  - 剪枝步：去除候选2-项集（ C_2 ）中不可能为频繁2-项集的项集；
  - 筛选频繁项集：扫描事务集，统计候选2-项集（ C_2 ）中项集的支持度计数，与支持度计数阈值比较并得到频繁2-项集（ L_2 ）；
  - 重复第二至第五个步骤，直到不能再找到频繁 k-项集（ L_k ）。

关联分析算法

Apriori算法示例

- 使用Apriori算法给出购物篮数据的频繁项集，假设支持度计数超过 2 就是频繁的。

TID	面包	牛奶	尿布	啤酒	鸡蛋
1	1	1	0	0	1
2	0	1	0	1	0
3	0	1	1	0	0
4	1	1	0	1	0
5	1	0	1	0	0
6	0	1	1	0	0
7	1	0	1	0	0
8	1	1	1	0	1
9	1	1	1	0	0

1-项集	支持度计数
{面包}	6
{牛奶}	7
{尿布}	6
{啤酒}	2
{鸡蛋}	2

2-项集	支持度计数
{面包，牛奶}	4
{面包，尿布}	4
{面包，啤酒}	1
{面包，鸡蛋}	2
{牛奶，尿布}	4
{牛奶，啤酒}	2
{牛奶，鸡蛋}	2
{尿布，啤酒}	0
{尿布，鸡蛋}	1
{啤酒，鸡蛋}	0

3-项集	支持度计数
{面包，牛奶，尿布}	2
{面包，牛奶，鸡蛋}	2
{面包，尿布，鸡蛋}	
{牛奶，尿布，啤酒}	
{牛奶，尿布，鸡蛋}	
{牛奶，啤酒，鸡蛋}	

被剪枝

4-项集
{面包，牛奶，尿布，鸡蛋}

Apriori算法的性能

- 为了找出购物篮数据中所有的频繁项集，遍历法和Apriori算法需要计算的项集数为：

  - 遍历法需要计算

$$C_5^1 + C_5^2 + C_5^3 + C_5^4 + C_5^5 = 5 + 10 + 10 + 5 + 1 = 31$$

个候选项集；

  - Apriori算法计算

$$5 + 10 + 2 = 17$$

个候选项集。

- 相较于遍历法，Apriori算法计算的候选项数减少了45%！

关联分析算法

Apriori算法产生频繁项集

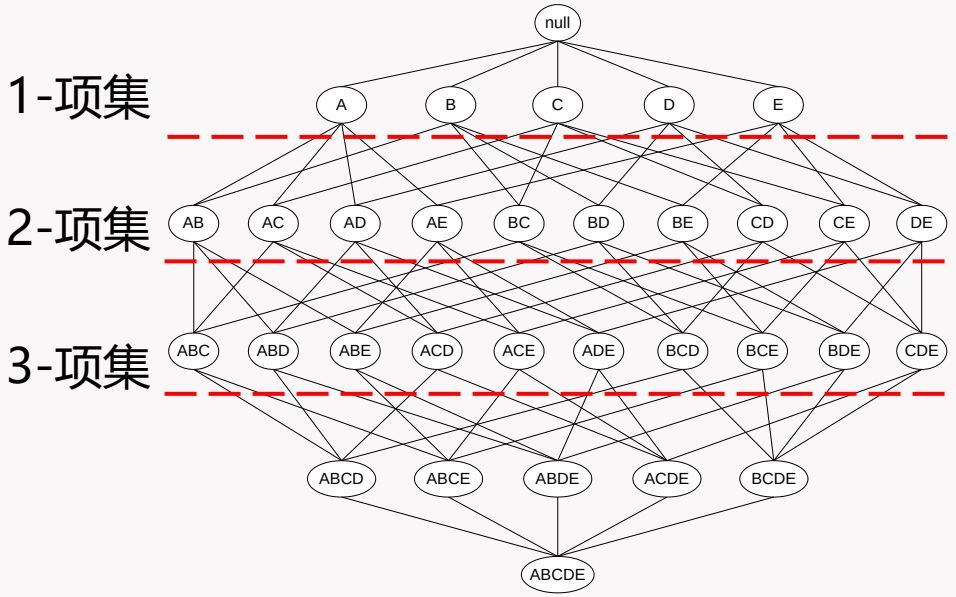
Apriori_frequentItemsets (T, I, m) ← T：事务集，I： 项的集合，m：最小支持度阈值

```
1   k=1
2   F_1 = {i| i ∈ I, σ(i) ≥ m}                  ← 计算频繁1-项集
3   Repeat
4     If F_k == ∅ then                          ← 中止条件判断
5       Return ∪_k F_k
6     End If
7     C_{k+1} = apriori_gen(F_k)                ← 基于频繁k-项集产生候选k+1-项集，对非频繁k+1-项集剪枝
8     For every t ∈ T do
9       C_t = select(C_{k+1}, t)                ← 筛选 t 包含的所有候选k+1-项集
10      For every c ∈ C_t do
11        σ(c)+=1                               ← 对筛选后的候选k+1-项集增加支持度计数
12      End For
13    End For
14    F_{k+1} = {c| c ∈ C_{k+1}, σ(c) ≥ m}      ← 从候选k+1-项集中提取频繁k+1-项集
15    k=k+1
16  End Repeat
```

关联分析算法

Apriori算法的特点

- Apriori算法是在格结构上是**逐层**（Level-wise）扫描的，每次遍历格结构中的一层；
- Apriori算法使用**产生-测试**（Generate-and-test）策略发现频繁项集，即产生候选项集（连接步和剪枝步），测试每个候选项集的支持度计数，最后得到频繁项集。

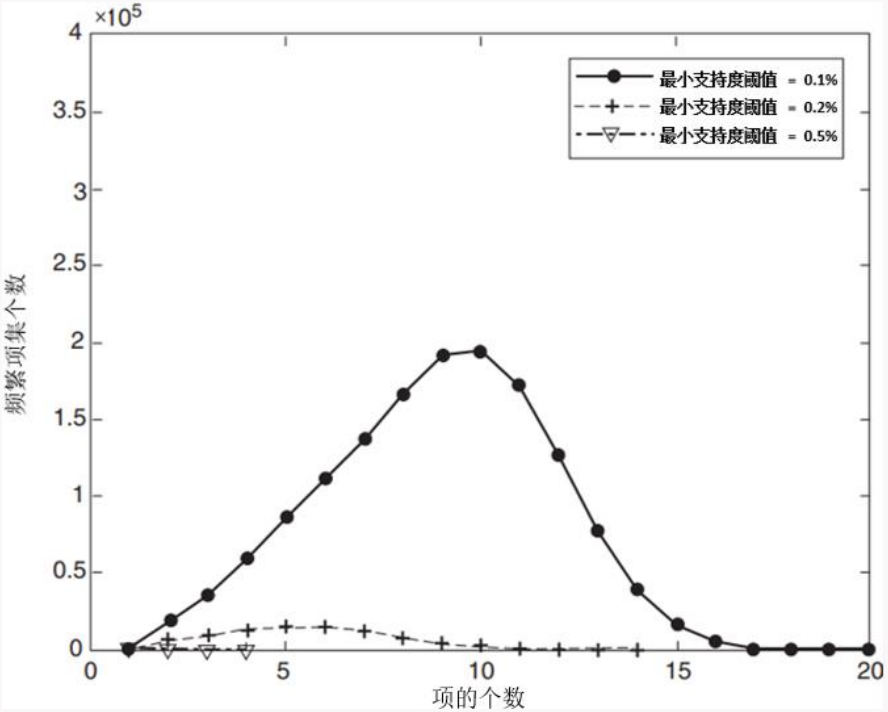


关联分析算法

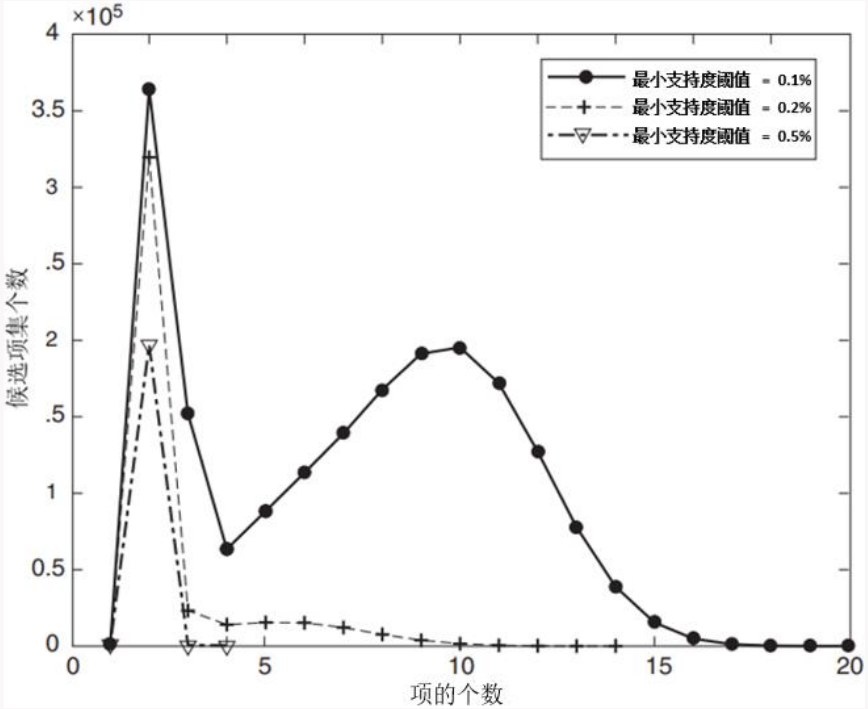
- 影响Apriori算法的计算复杂度的因素
- Apriori算法的计算复杂度受如下因素影响：
  - 支持度计数阈值
  - 事务集的项数
  - 事务集的事务数量
  - 事务的平均宽度

- 影响Apriori算法的计算复杂度的因素

- **最小支持度阈值**：更低的支持度阈值通常导致更多的频繁项集，更长的频繁项集最大长度，更多的迭代次数以及更多的事务集扫描次数。



不同阈值下，频繁项集个数与项集中项个数的曲线



不同阈值下，候选项集个数与项集中项个数的曲线

- 影响Apriori算法的计算复杂度的因素
- **项数**：随着事务集中项的个数的增加，频繁项集和候选项集的数目都会急剧增加，同样会极大地提高了Apriori算法的计算复杂度。
  - 假设频繁1-项集中项的个数是20，则候选2-项集的个数是

$$C_{20}^{2} = 100$$

  - 假设频繁1-项集中项的个数是300，则候选2-项集的个数是

$$C_{300}^{2} = 44850$$

影响Apriori算法的计算复杂度的因素

- **事务集中事务的数量**：Apriori算法每次在对候选项集计算支持度计数时，都需要遍历一遍事务表。Apriori算法在整个计算过程中需要频繁的遍历事务表，所以它的运行时间随着事务表中事务数量的增加而增加。
- **事务集中事务的平均宽度**：事务的平均宽度指的是平均一个事务所包含项的数目。对于密集事务表，事务的平均宽度可能很大，这导致包含更多项的项集可能是频繁的，使得Apriori算法需要更多次的迭代才能找到所有的频繁项集。

- 提高Apriori算法效率的方法

- **事务压缩**：不包含任何频繁k-项集的事务不可能包含任何频繁(k+1)-项集。因此，可对这些事务进行标记，在产生任何j-项集（j>k）时都不需要扫描它们。
- **事务集抽样**：选取事务集的子集并在子集中搜索频繁项集，通过牺牲精度的方式提高效率，但是可能会丢失一些频繁项集。

关联分析算法

研究路线

找出所有的强规则（遍历）

提高效率

- 找出所有的频繁项集
  - 项集出现的次数大于等于最小支持度阈值
  - 提高效率
    - Apriori算法
- **利用频繁项集产生强规则**
  - 关联规则满足最小支持度阈值和最小置信度阈值

利用频繁项集产生强规则

- 从事务集中找出所有的频繁项集后，就可以直接由它们产生强关联规则。忽略前件和后件为空的规则（$\emptyset \rightarrow Y, X \rightarrow \emptyset$），一个包含 k 个项的频繁项集可以产生$2^k - 2$个关联规则这些关联规则必然满足支持度阈值。

- 设 F = {1，2，3}是频繁项集，则 F 可以产生 6 个关联规则：

关联规则		
{1} →{2，3}	{2} →{1，3}	{3} →{1，2}
{1，2} →{3}	{1，3} →{2}	{2，3} →{1}

这些关联规则的支持度都和 F 的支持度计数有关，且**计算置信度不需要再次扫描事务集**，因为前件和后件的支持度计数在寻找频繁项集时就已经得到。

先验原理

- **如果规则 $X \to F - X$ 不满足置信度阈值，则 $X' \to F - X'$ 的规则一定也不满足置信度阈值。**其中 X' 是 X 的子集，即 $X' \subset X$。 F 是频繁项集。

证明：

$\because X'$是 X 的子集

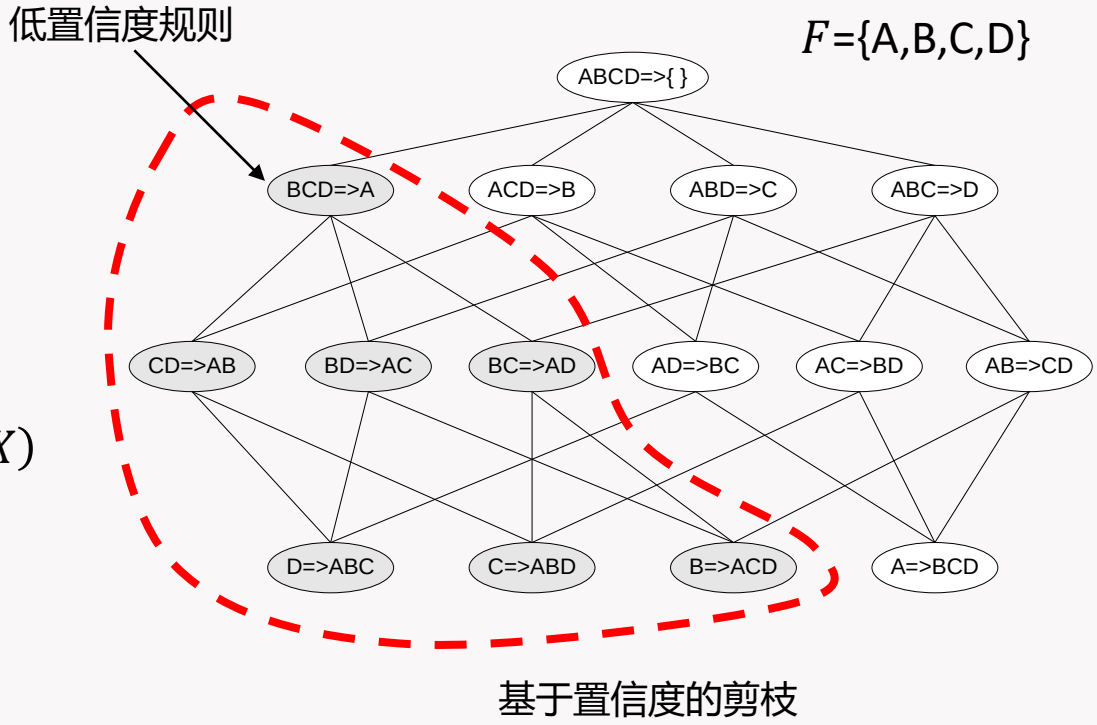
$\therefore \sigma(X') \geq \sigma(X)$

$$\because c(X \to F - X) = \frac{\sigma(\mathrm{F})}{\sigma(X)}$$

$$c(X' \to F - X') = \frac{\sigma(F)}{\sigma(X')}$$

$$\therefore c(X' \to F - X') \leq c(X \to F - X)$$

得证。



先验原理

- **如果规则 $X \rightarrow F - X$ 满足置信度阈值，则 $X \rightarrow Y'$ 的规则也一定满足置信度阈值。**

  其中 Y' 是 $F - X$ 的子集，即 $Y' \subset F - X$。 F 是频繁项集。

关联分析算法

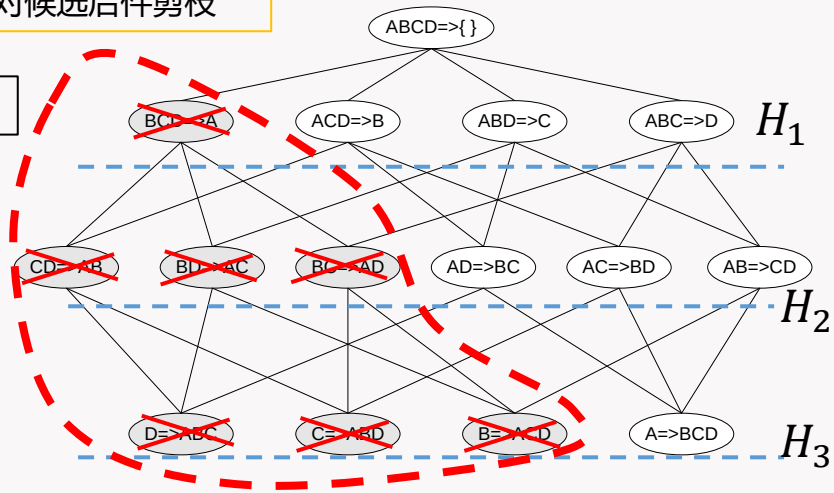
Apriori算法产生强关联规则

Apriori_StrongRules(F, H_0, p_{con}, R)

F：频繁项集，H_0： 包含0项的规则后件集合，p_{con}：最小置信度阈值，R：强关联规则集合

```
1   k=|F|                                  频繁项集的大小
2   m=|H_0|                                规则后件的大小
3   If k>m+1 then
4     H_{m+1}=apriori_gen(H_m)             基于m-项集的后件产生候选(m+1)项集的后件，并对候选后件剪枝
5     For every h_{m+1} ∈ H_{m+1} do
6       conf =σ(F)/σ(F − h_{m+1})          计算置信度
7       If conf ≥ p_con then
8         R=R ∪ (F − h_{m+1} → h_{m+1})
9       Else
10        Delete h_{m+1} from H_{m+1}
11      End if
12    End for
13    Apriori_StrongRules(F, H_{m+1}, p_con, R)   迭代找出下一层的强关联规则
14  End if
```

基于频繁k-项集产生候选k+1-项集，对非频繁k+1-项集剪枝



关联分析算法

Apriori算法

Apriori(T, I, m, p_{con})

T：事务集，I： 项的集合，m：最小支持度阈值，p_{con}：最小置信度阈值

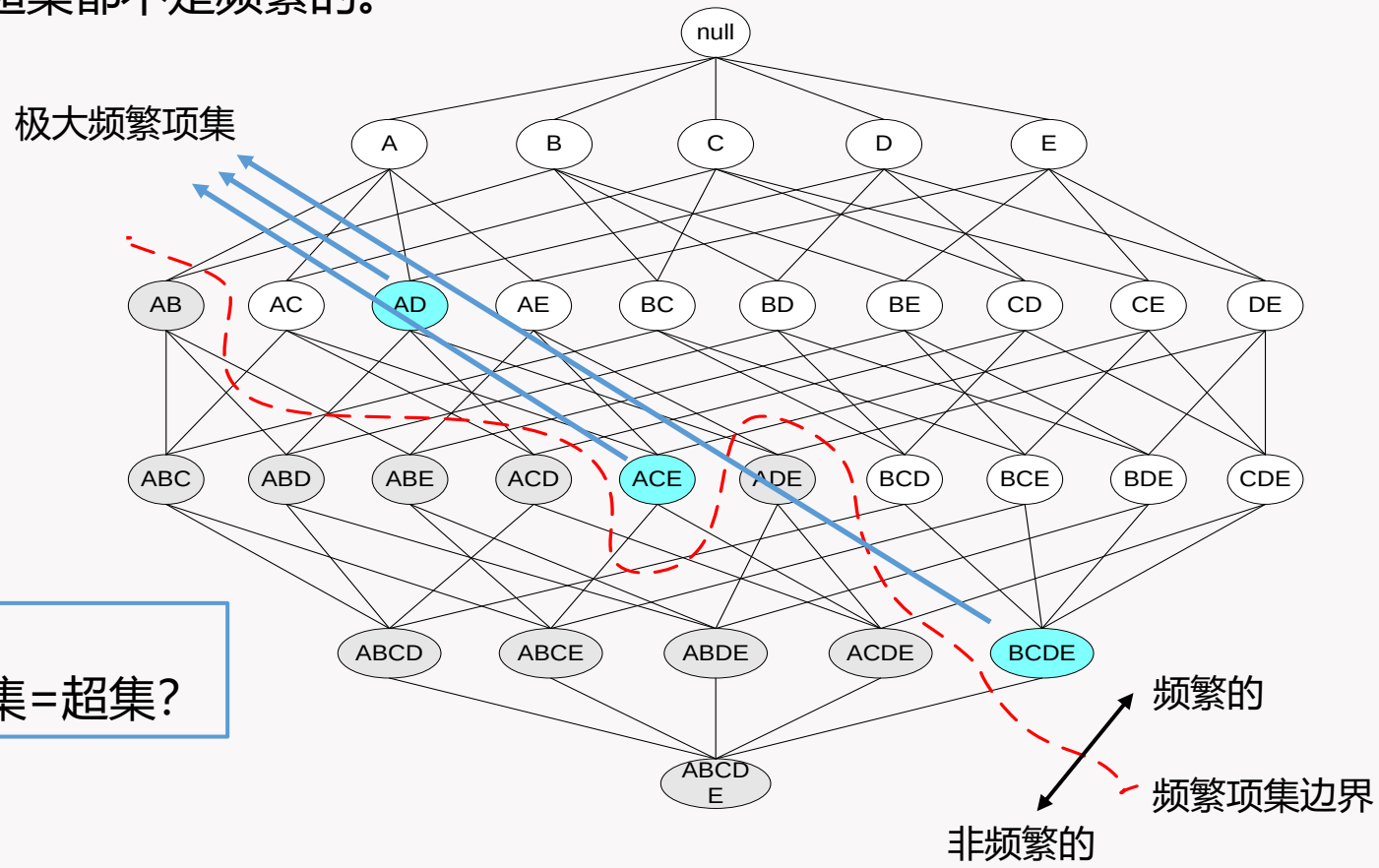
```
1  F=Apriori_frequentItemsets (T, I, m)
2  R={ }
3  H_0={ }
4  For every f ∈ F do
5     Apriori_StrongRules(f, H_0, p_con, R)
6  End For
7  Return R
```

频繁项集的两种紧凑表示

- 在实际应用中，当最小支持度阈值较低或是事务集规模更大且更稠密时，会挖掘出大量的频繁项集，可能导致无法计算和存储。
- 因此，从这些频繁项集中识别出**可以推导出其他所有的频繁项集的**、**较小的**、**具有代表性**的项集是有必要的，常用的两种代表性的紧凑表示：
  - 极大频繁项集
  - 闭频繁项集

关联分析算法

- 频繁项集的两种紧凑表示
- **极大频繁项集**（Maximal frequent itemset）指的是这样的频繁项集，它的所有直接超集都不是频繁的。



定义中
直接超集=超集?

频繁项集的两种紧凑表示

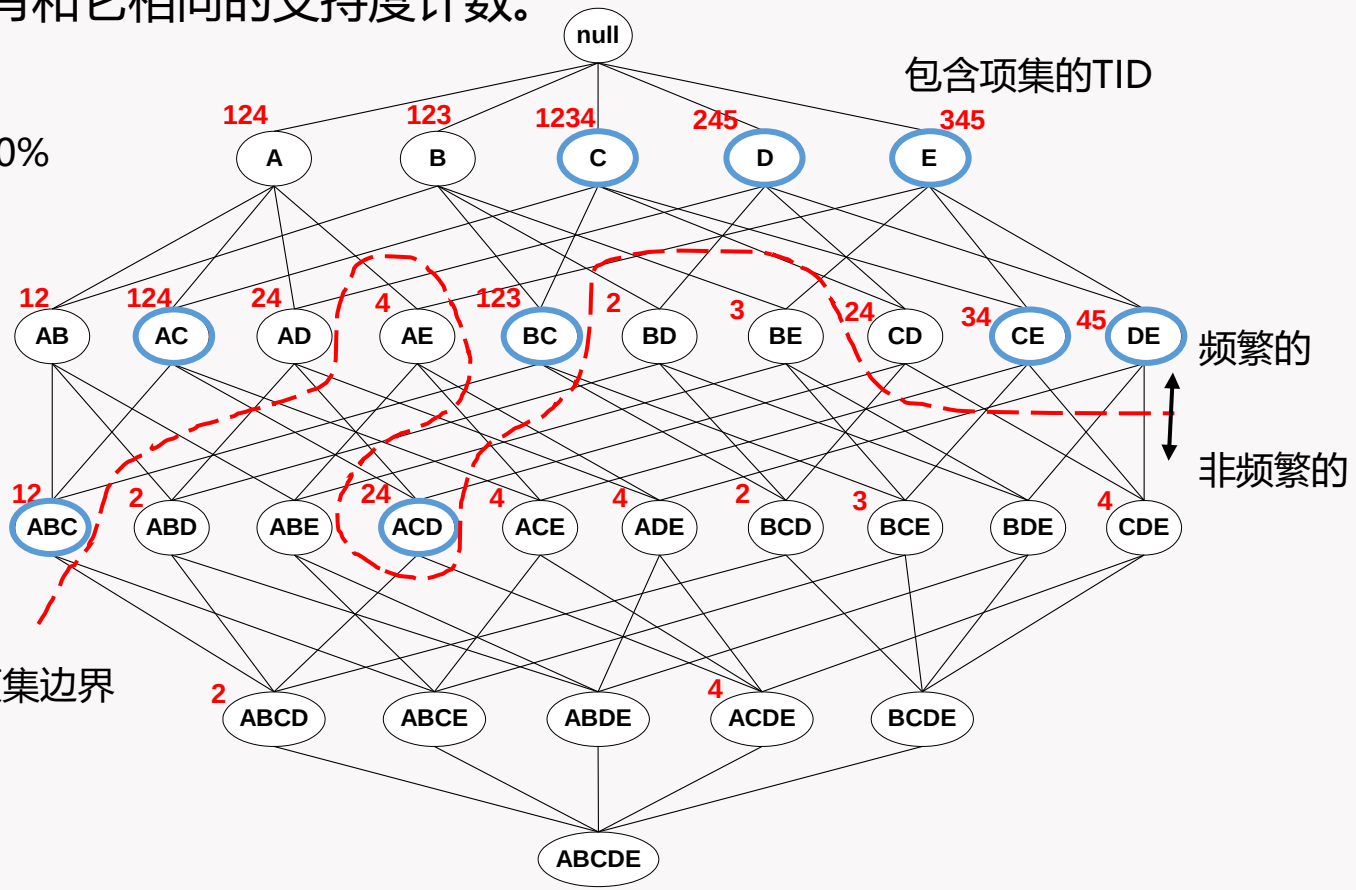
- 极大频繁项集可以有效的提供频繁项集的紧凑表示，因为从极大频繁项集可以导出所有包含的频繁项集。但是极大频繁项集不包含子集的支持度信息，所以需要再遍历一遍事务集以确定非极大的频繁项集的支持度计数。

关联分析算法

- 频繁项集的两种紧凑表示
- **闭频繁项集**（Closed frequent itemset）指的是这样的频繁项集，它的所有直接超集都不具有和它相同的支持度计数。

最小支持度阈值=40%

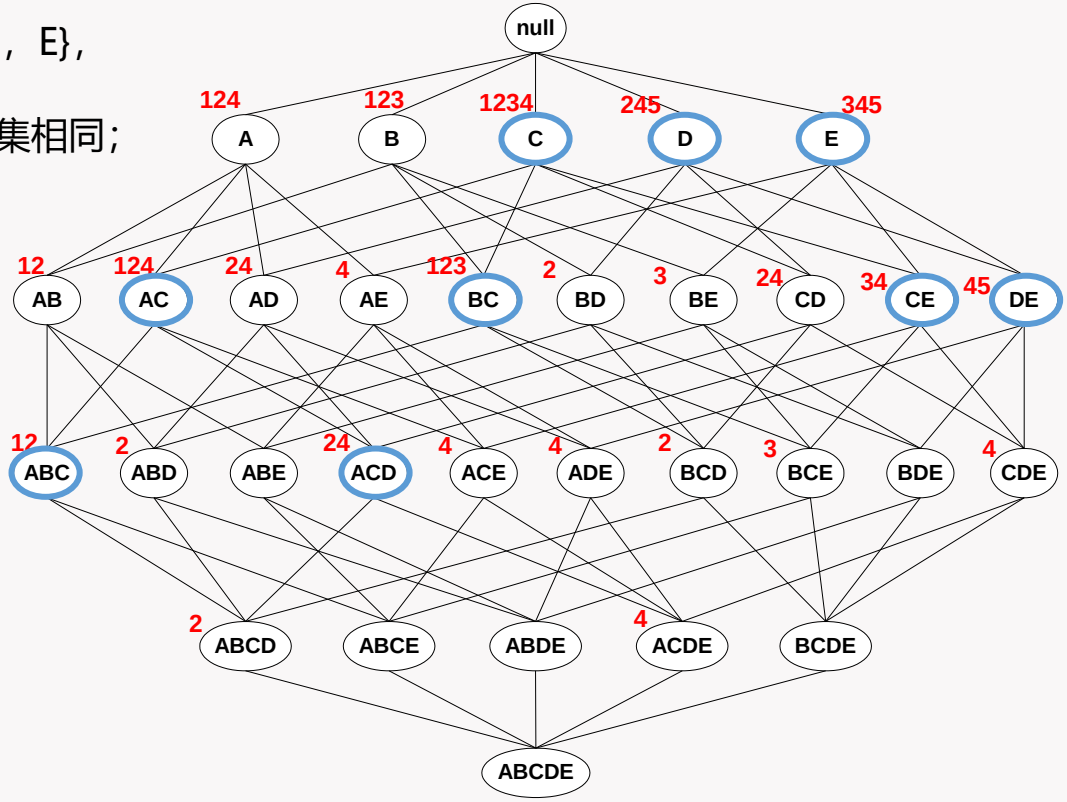
TID	项集
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



关联分析算法

频繁项集的两种紧凑表示

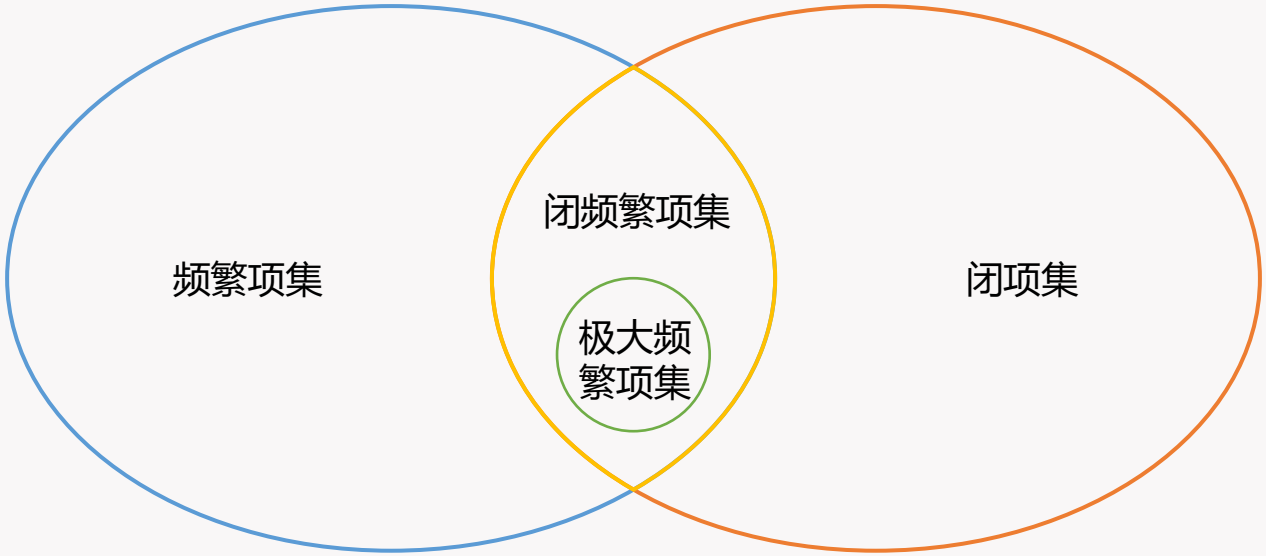
- 可以使用闭频繁项集的支持度计数确定非闭的频繁项集的支持度计数。
  - 对于频繁项集{A，B}，因为它是非闭的，所以它的支持度计数一定与它的某个超集相同；
  - 它的超集有{A，B，C}，{A，B，D}和{A，B，E}，
    所以需要确定{A，B}的支持度计数与哪个超集相同；
  - 先验原理表明，包含{A，B}的超集的事务
    一定包含{A，B}，所以{A，B}的支持度计数
    一定等于它的超集的最大支持度计数；
  - 所以{A，B}的支持度计数是2。



关联分析算法

频繁项集的两种紧凑表示

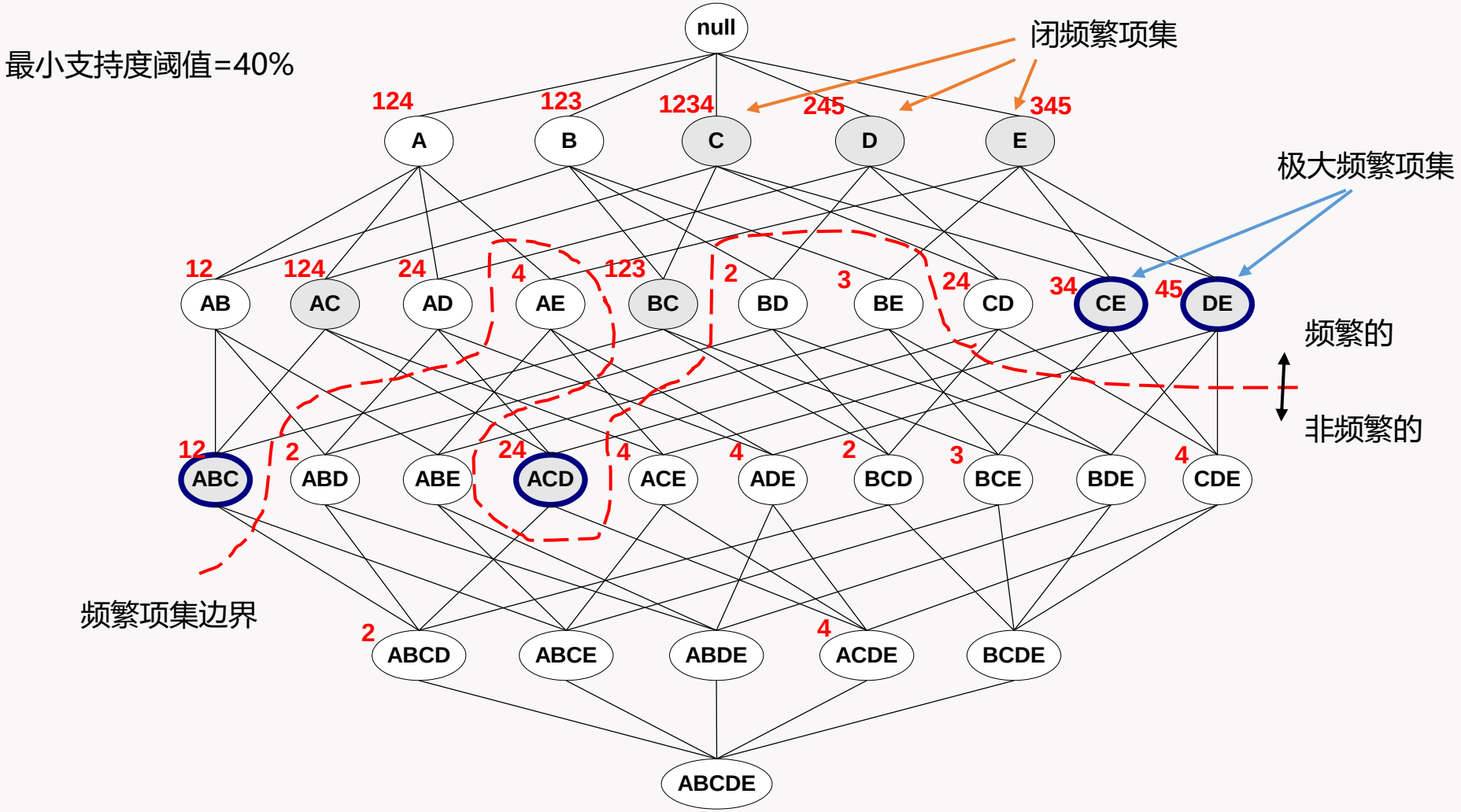
- 极大频繁项集都是闭的，因为任何极大频繁项集都不可能与它的直接超集具有相同的支持度计数。



闭项集：指一个项集X,它的直接超集的支持度计数都不等于它本身的支持度计数。

关联分析算法

- 频繁项集的两种紧凑表示



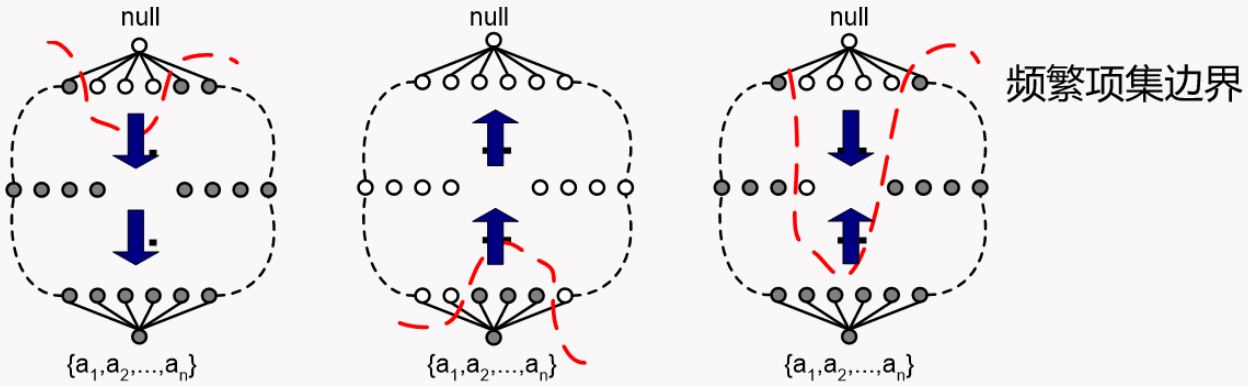
关联分析算法

- 极大频繁项集和闭频繁项集
  - 优点
    - 极大频繁项集给出了判断项集频繁性的最简表示；
    - 闭频繁项集保留了频繁项集的所有信息。
  - 缺点
    - 极大频繁项集损失了频繁项集的支持度计数信息；
    - 闭频繁项集保留了更多项集，需要更多的存储资源。

其他产生频繁项集的策略

搜索方向策略

- Apriori算法采用了“从无到有”的搜索策略，这种策略更适合（a）中的频繁项集分布；
- “从有到无”的搜索策略从格结构的底层开始，使用先验原理**剪掉极大频繁项集**的所有子集，仅保留非频繁项集。这种策略更适合（b）中频繁项集分布；
- 组合“从无到有”和“从有到无”的双向搜素策略可以更快的确定频繁项集的边界。



关联分析算法

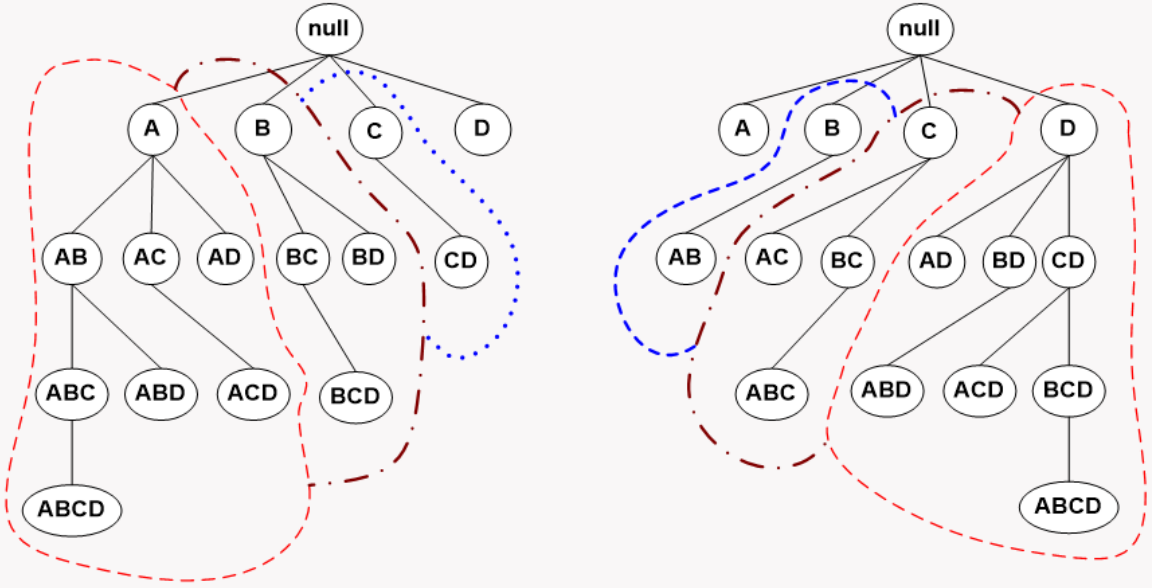
其他产生频繁项集的策略

- 搜索层次策略

- Apriori算法采用逐层的方式搜索格结构，可以看作以项集的大小作为格结构分层的依据；
- 也可以按照前缀或后缀对格结构进行划分，如果两个项集共享长度为 k 的相同前缀或后缀，则它们属于同一个层次。

优点:
每次加载部分事务集可以并行

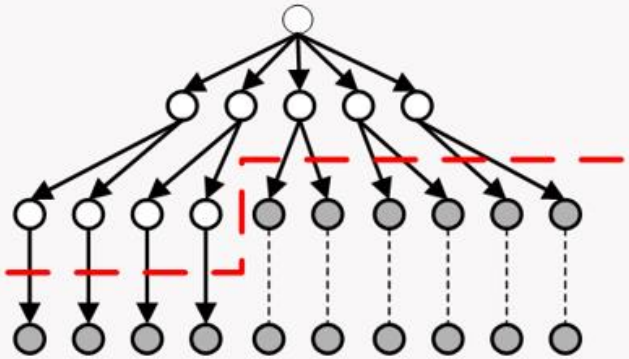
缺点:
额外的计算



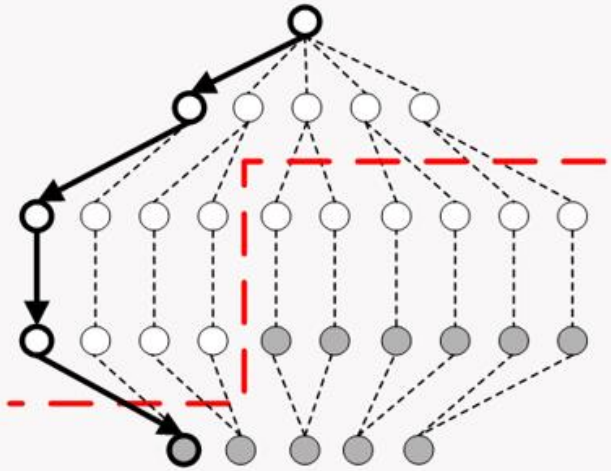
(a) 按前缀分层　　(b) 按后缀分层

其他产生频繁项集的策略

- 搜索优先策略
  - Apriori算法采用了广度优先的方法遍历格结构；
  - 以深度优先的方法遍历格结构，可以更快的发现极大频繁项集并更快地检测出频繁项集边界，适用于在大规模的事务集上对频繁项集的初步探索。



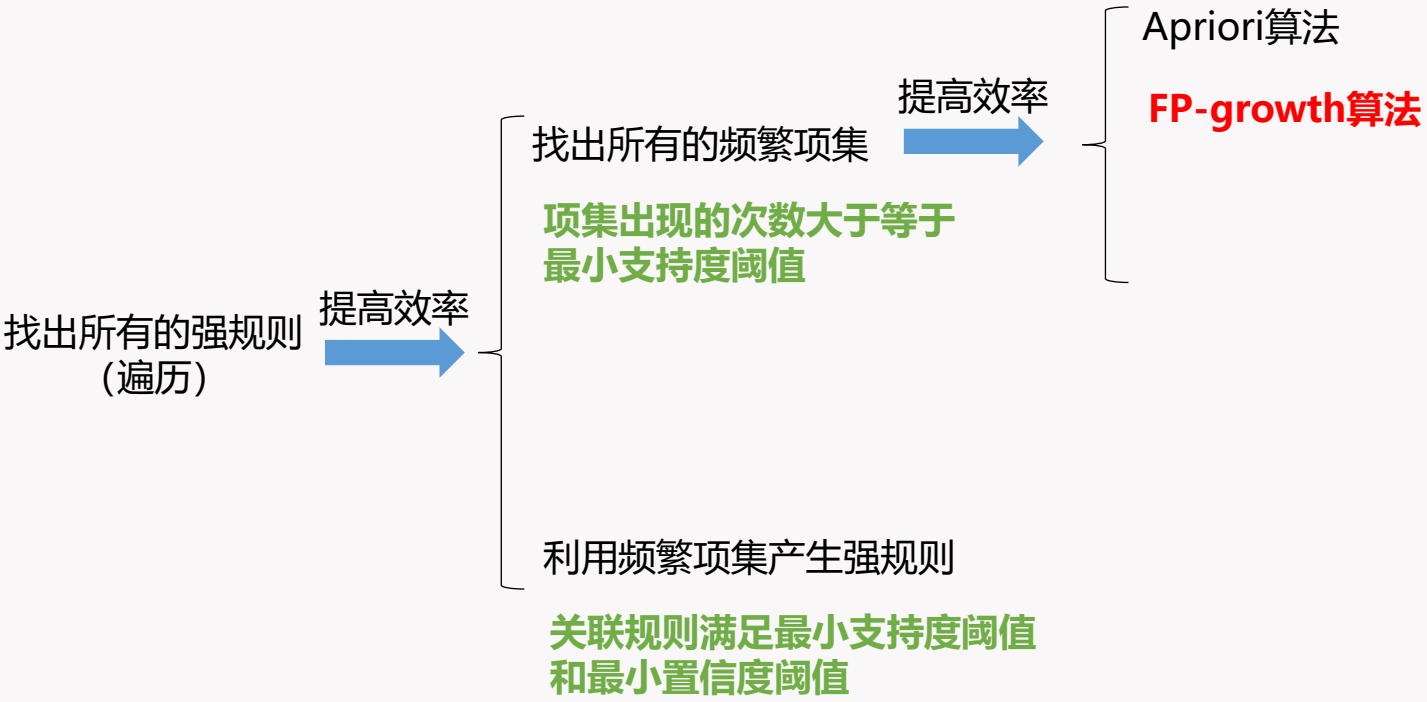
(a) 广度优先



(b) 深度优先

关联分析算法

- 研究路线


找出所有的强规则
（遍历）
提高效率
找出所有的频繁项集
项集出现的次数大于等于
最小支持度阈值
提高效率
Apriori算法
FP-growth算法
利用频繁项集产生强规则
关联规则满足最小支持度阈值
和最小置信度阈值

Apriori算法的局限性

- Apriori算法采用“产生-测试”的范式且显著压缩了候选项集的规模，但是在以下两种情况下面临着高昂的计算代价的难题：
  - 产生阶段：它仍然可能产生大量的候选项集。例如，如果有 10^4 个频繁1-项集，则Apriori算法会产生 $C_{10^4}^2 \approx 5 \times 10^7$ 个候选2-项集；
  - 测试阶段：它需要重复地扫描整个事务集，以确定候选项集的支持度计数。
- 是否可以设计这样一种方法，能挖掘出全部的频繁项集但是能够避免这种代价昂贵的候选项集产生过程?

关联分析算法

频繁模式增长算法

- **频繁模式增长算法**（Frequent-Pattern growth，FP-growth）不同于Apriori算法采用“产生-测试”的范式从候选项集中生成频繁项集，而是使用了一种称作频繁模式树（FP树）的紧凑数据结构组织数据，不需要产生候选项集就可以**直接从FP树中提取频繁项集。**
- FP-growth算法主要包含以下两个过程：
  - 构造事务集的FP树表示形式；
  - 利用FP树产生频繁项集。

FP树表示法

- FP树是一种输入数据的**压缩表示**，采用逐个读入事务的方式将每个事务映射到树中的一条路径上。由于不同的事务可能会有若干个相同的项，所以它们的路径会出现重叠。**路径相互重叠的越多，则FP树获得的压缩效果越好。**

FP树表示法

- 假设事务集如右表所示，它包含了9个事务和6个项。FP树初始时仅包含一个根节点，用符号 null 标记。随后采用如下方法扩充FP树：
  - 第一次扫描事务集，去除非频繁的项，将每个事务中保留的项按支持度计数递减的顺序排列；
  - 第二次扫描事务集，依次读入每个事务，构造FP树。

TID	项集
1	{A,B,E}
2	{B,D}
3	{B,C}
4	{A,B,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{A,B,C,E}
9	{A,B,C,F}

关联分析算法

FP树表示法

- 假设最小支持度阈值为 22.2% 。扫描事务集，去除非频繁的项，将每个事务中保留的项按支持度计数递减的顺序排列。

TID	项集
1	{A,B,E}
2	{B,D}
3	{B,C}
4	{A,B,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{A,B,C,E}
9	{A,B,C,F}

A	B	C	D	E	F
6	7	6	2	2	1

B	A	C	D	E
7	6	6	2	2

TID	项集
1	{B,A,E}
2	{B,D}
3	{B,C}
4	{B,A,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{B,A,C,E}
9	{B,A,C}

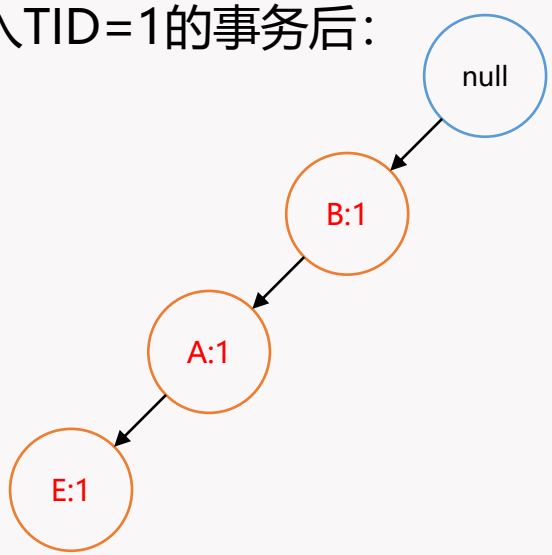
关联分析算法

FP树表示法

- 扫描事务集，依次读入每个事务，从根节点开始构造FP树，树中每个节点记录项的名称和经过该节点的次数。

TID	项集
1	{B,A,E}
2	{B,D}
3	{B,C}
4	{B,A,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{B,A,C,E}
9	{B,A,C}

读入TID=1的事务后：



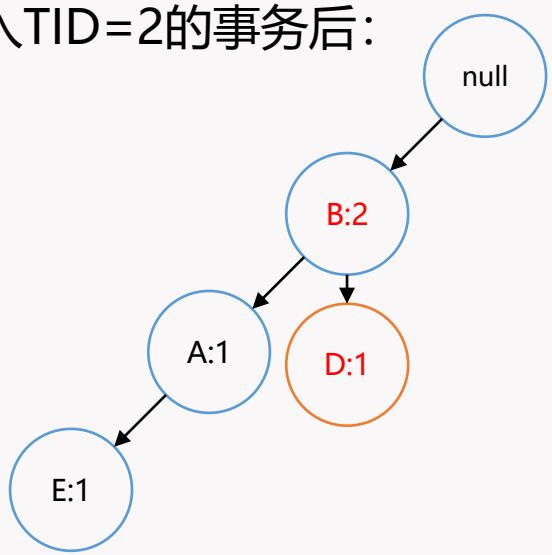
关联分析算法

FP树表示法

- 扫描事务集，依次读入每个事务，从根节点开始构造FP树，树中每个节点记录项的名称和经过该节点的次数。

TID	项集
1	{B,A,E}
2	{B,D}
3	{B,C}
4	{B,A,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{B,A,C,E}
9	{B,A,C}

读入TID=2的事务后：

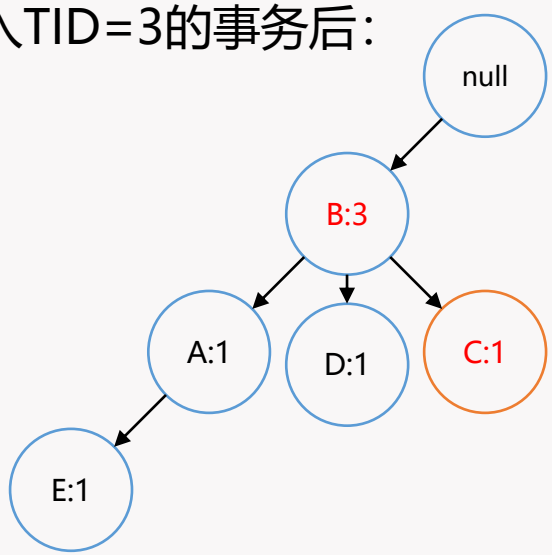


FP树表示法

- 扫描事务集，依次读入每个事务，从根节点开始构造FP树，树中每个节点记录项的名称和经过该节点的次数。

TID	项集
1	{B,A,E}
2	{B,D}
3	{B,C}
4	{B,A,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{B,A,C,E}
9	{B,A,C}

读入TID=3的事务后：

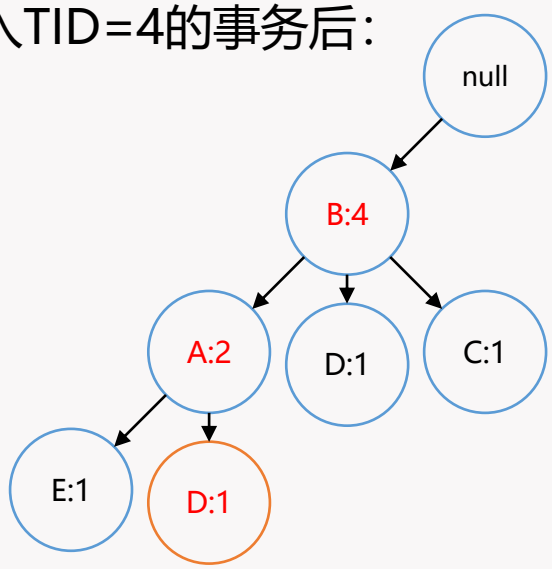


FP树表示法

- 扫描事务集，依次读入每个事务，从根节点开始构造FP树，树中每个节点记录项的名称和经过该节点的次数。

TID	项集
1	{B,A,E}
2	{B,D}
3	{B,C}
4	{B,A,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{B,A,C,E}
9	{B,A,C}

读入TID=4的事务后：

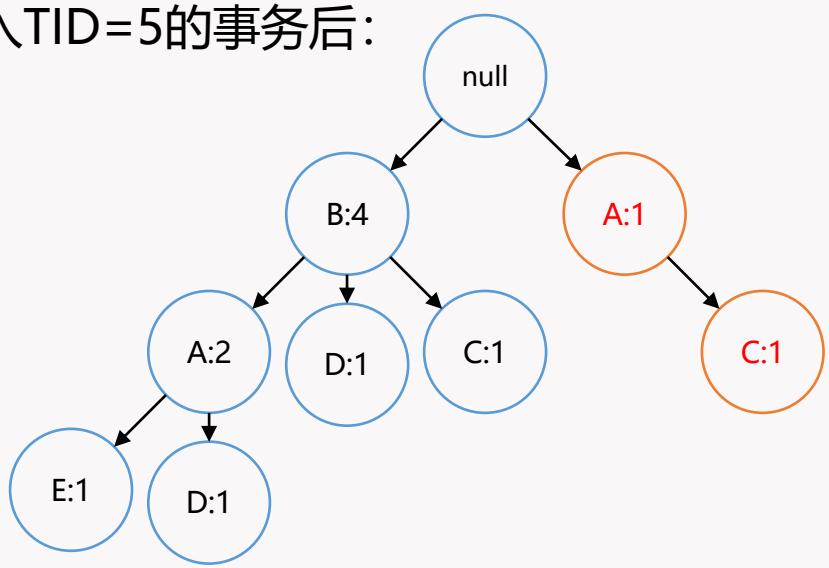


FP树表示法

- 扫描事务集，依次读入每个事务，从根节点开始构造FP树，树中每个节点记录项的名称和经过该节点的次数。

TID	项集
1	{B,A,E}
2	{B,D}
3	{B,C}
4	{B,A,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{B,A,C,E}
9	{B,A,C}

读入TID=5的事务后：



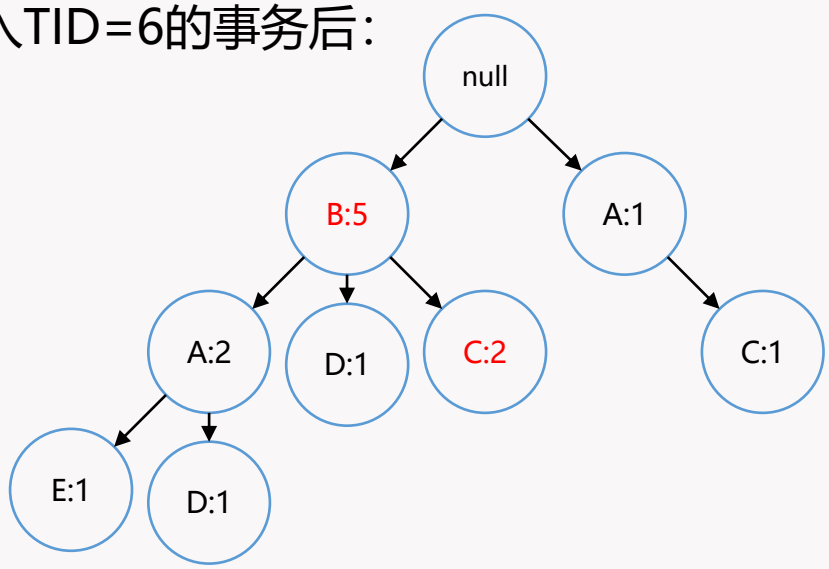
关联分析算法

FP树表示法

- 扫描事务集，依次读入每个事务，从根节点开始构造FP树，树中每个节点记录项的名称和经过该节点的次数。

TID	项集
1	{B,A,E}
2	{B,D}
3	{B,C}
4	{B,A,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{B,A,C,E}
9	{B,A,C}

读入TID=6的事务后：

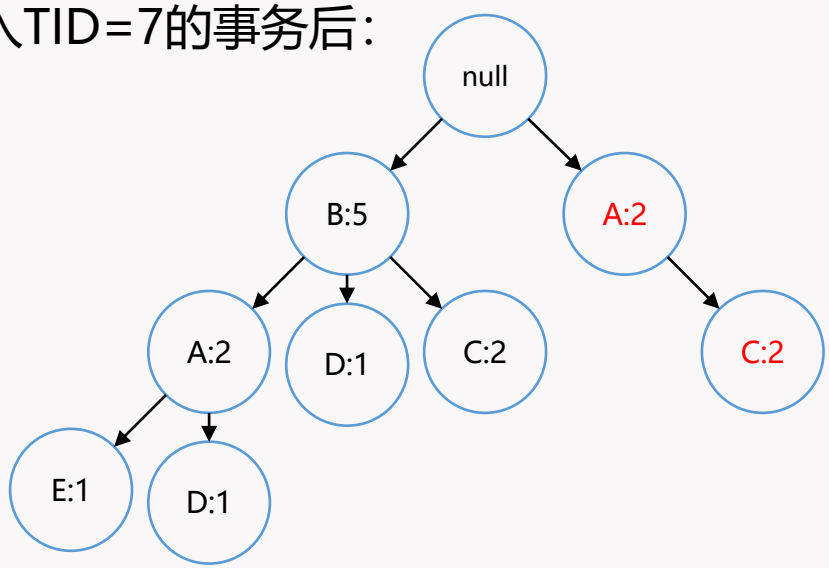


FP树表示法

- 扫描事务集，依次读入每个事务，从根节点开始构造FP树，树中每个节点记录项的名称和经过该节点的次数。

TID	项集
1	{B,A,E}
2	{B,D}
3	{B,C}
4	{B,A,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{B,A,C,E}
9	{B,A,C}

读入TID=7的事务后：

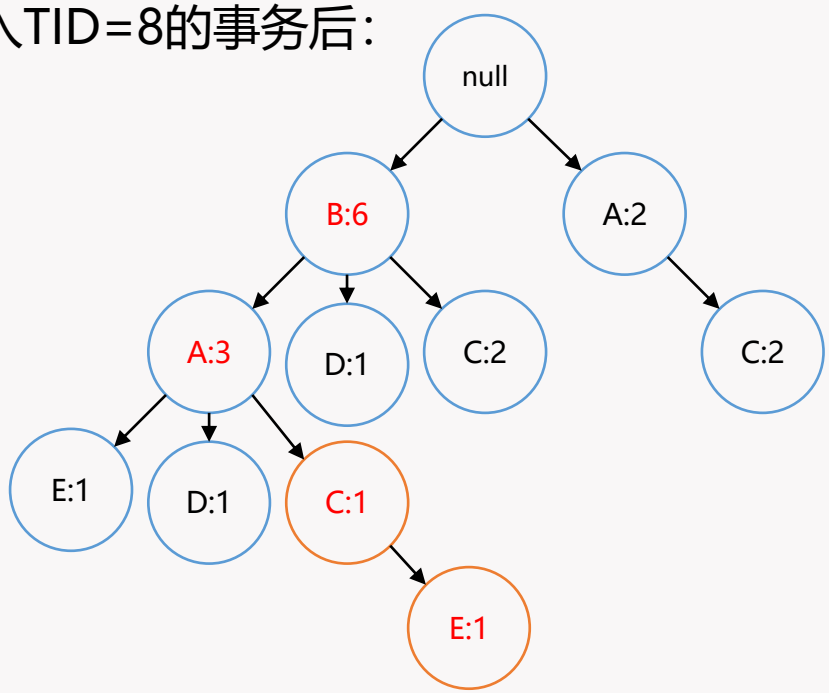


FP树表示法

- 扫描事务集，依次读入每个事务，从根节点开始构造FP树，树中每个节点记录项的名称和经过该节点的次数。

TID	项集
1	{B,A,E}
2	{B,D}
3	{B,C}
4	{B,A,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{B,A,C,E}
9	{B,A,C}

读入TID=8的事务后：

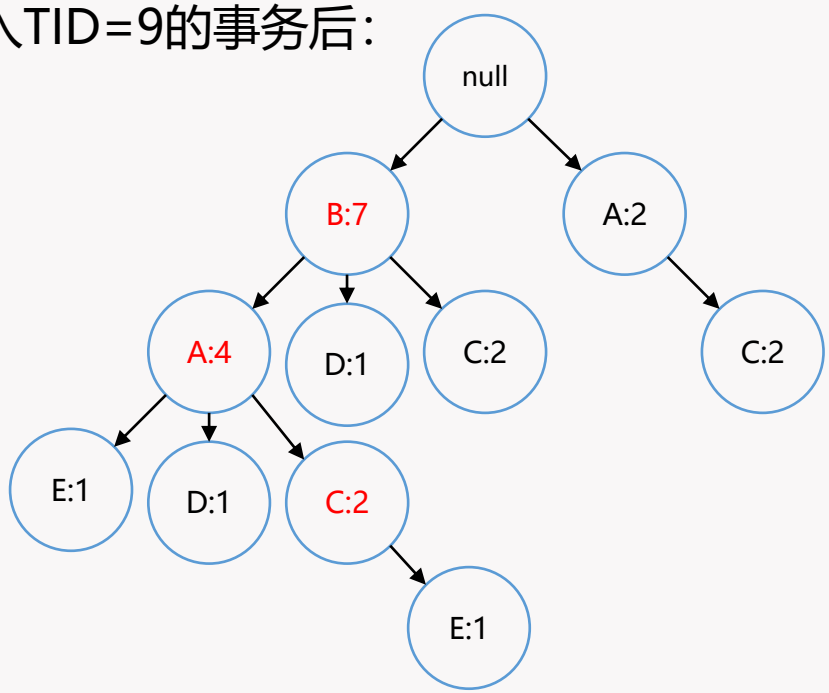


FP树表示法

- 扫描事务集，依次读入每个事务，从根节点开始构造FP树，树中每个节点记录项的名称和经过该节点的次数。

TID	项集
1	{B,A,E}
2	{B,D}
3	{B,C}
4	{B,A,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{B,A,C,E}
9	{B,A,C}

读入TID=9的事务后：

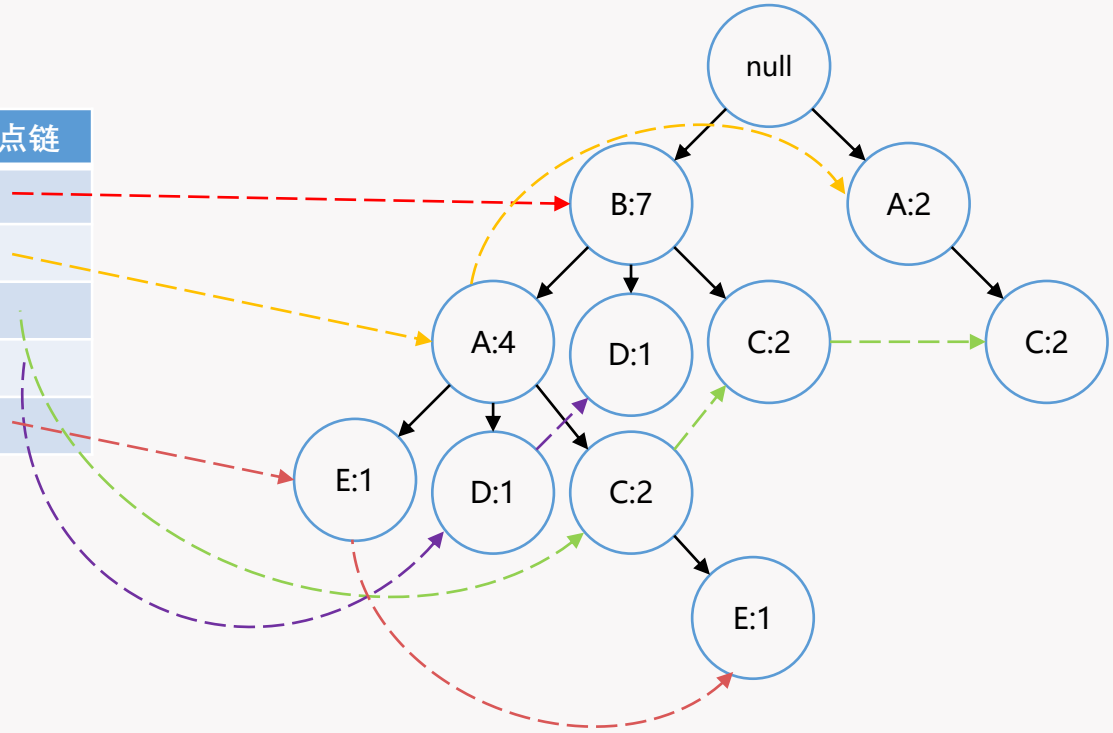


关联分析算法

FP树表示法

- 为了方便FP树的遍历，通常会创建一个项头表（Item head table），使每项通过一个**结点链**指向它在树中的位置。

项	支持度计数	结点链
B	7	
A	6	
C	6	
D	2	
E	2	



FP树表示法

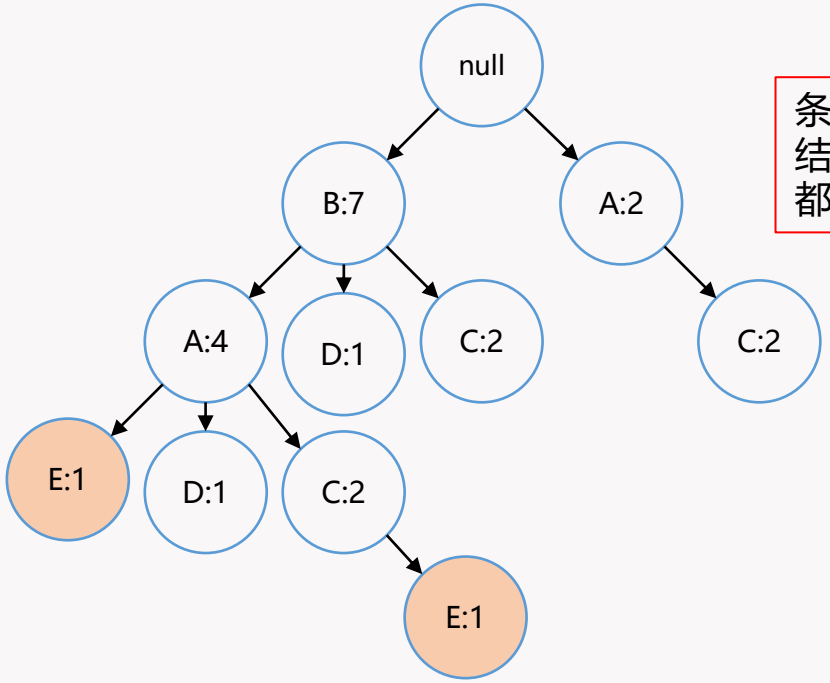
- 一般情况下，FP树可以在一定程度上压缩事务集，因为事务集中的某些项被不同的事务共享。
- 最好情况下，所有事务都包含相同的项，此时FP树只包含一条结点路径。
- 最坏情况下，所有事务都互不相同，此时FP树不但没有压缩事务集，还因为存储了每个结点的指针和计数，额外增加了存储的空间。
- 一般情况下，将频繁项按照支持度计数递增的顺序排序的话，会导致FP树更加茂盛。但是**将频繁项按照支持度计数递减的顺序排序后，所构造的FP树并不一定是最简洁的**!

利用FP树产生频繁项集

- 构造FP树时采用了自根节点至叶节点的构造方式，利用FP树产生频繁项集则采用了自叶节点至根节点的产生方式。主要包含3个步骤：
  - 选择支持度计数最小的频繁项，构造它的**条件模式基**（Conditional pattern base）；
  - 根据条件模式基构造该频繁项的**条件FP树**（Conditional FP tree）；
  - 利用条件FP树得到频繁项集。

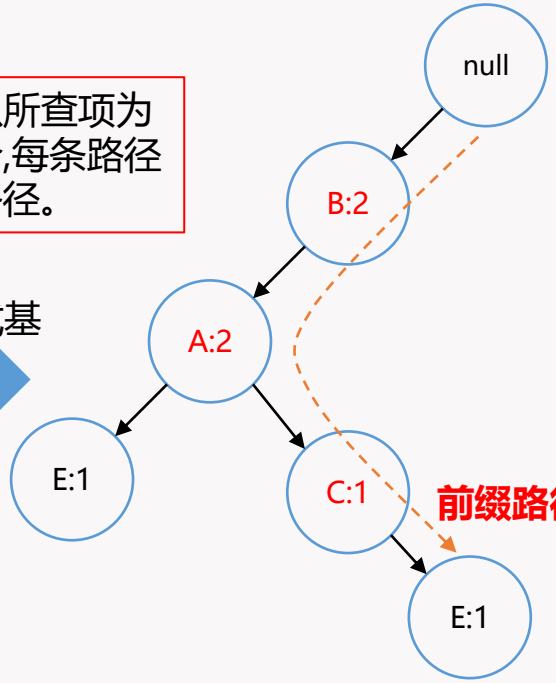
关联分析算法

利用FP树产生频繁项集



条件模式基是以所查项为结尾的路径集合,每条路径都是一条前缀路径。

E的条件模式基



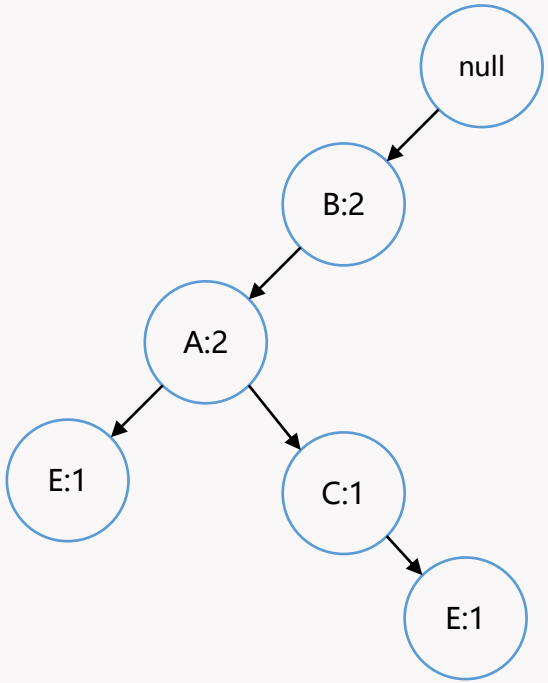
前缀路径（prefix path）

后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}		
D			
C			
A			
B			

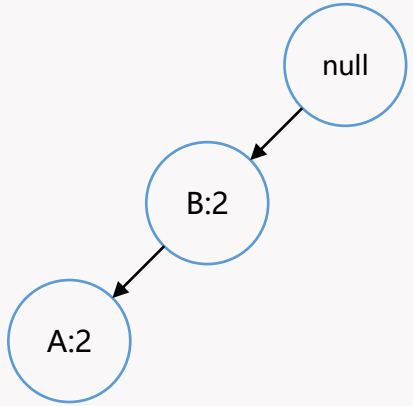
TID	项集	TID	项集
1	**{B,A,E}**	6	{B,C}
2	{B,D}	7	{A,C}
3	{B,C}	**8**	**{B,A,C,E}**
4	{B,A,D}	9	{B,A,C}
5	{A,C}		

关联分析算法

- 利用FP树产生频繁项集



E的条件FP树



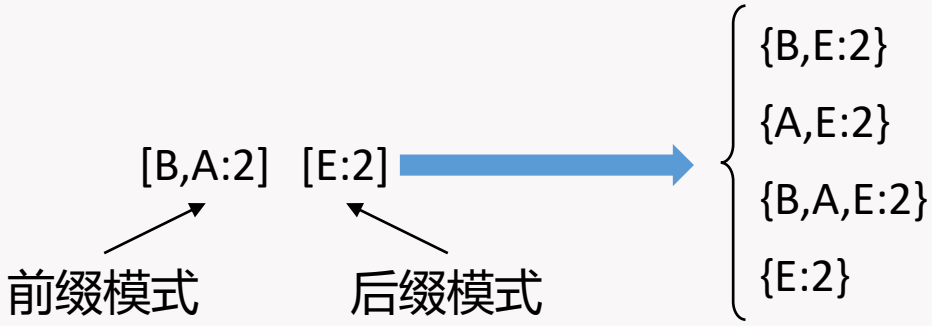
如果一个FP树**仅包含单个前缀路径**，那么可以通过**列举路径的所有组合来生成频繁项集**，其支持度计数为包含项的支持度计数的最小值。

后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	
D			
C			
A			
B			

关联分析算法

- 利用FP树产生频繁项集

项	支持度计数
B	7
A	6
C	6
D	2
E	2

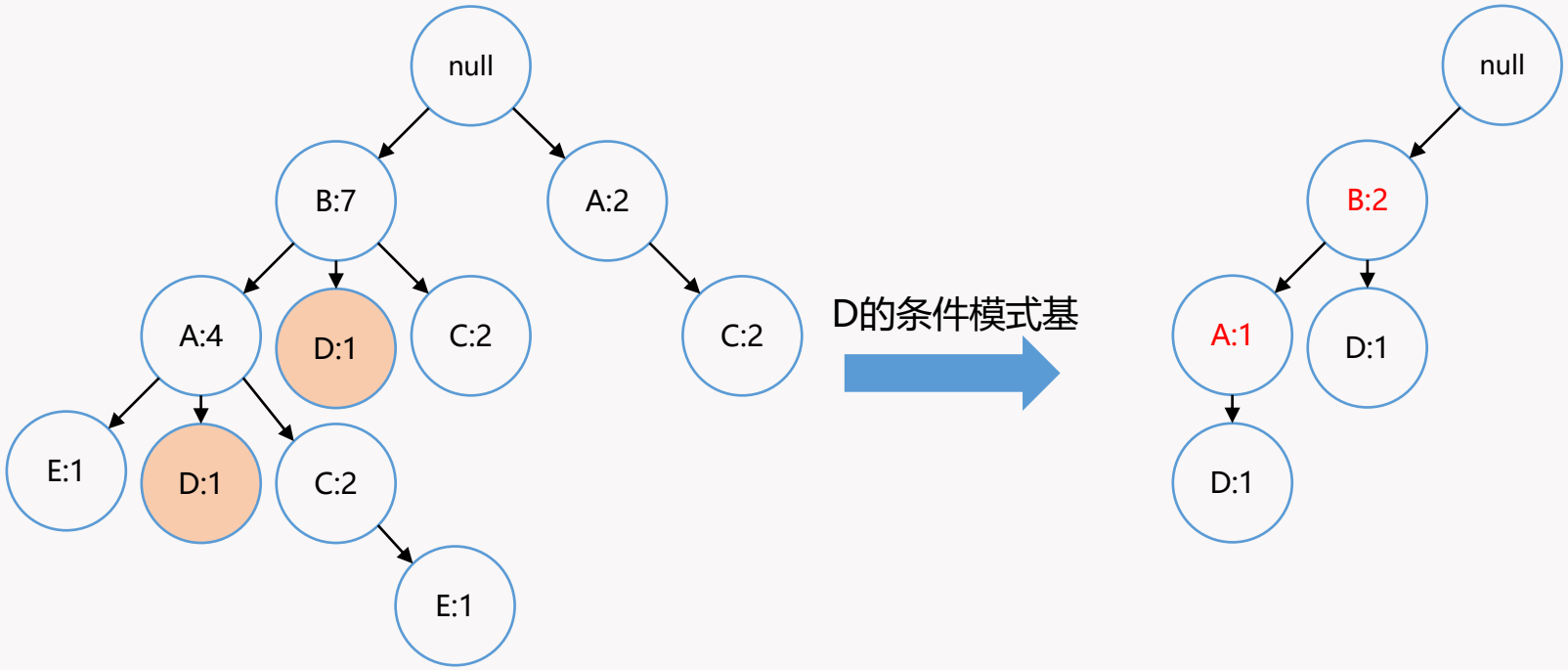


后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D			
C			
A			
B			

TID	项集	TID	项集
1	{B,A,E}	6	{B,C}
2	{B,D}	7	{A,C}
3	{B,C}	8	{B,A,C,E}
4	{B,A,D}	9	{B,A,C}
5	{A,C}		

关联分析算法

- 利用FP树产生频繁项集

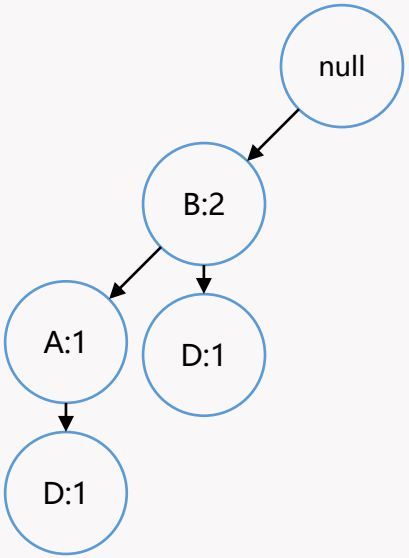


后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D	{{B,A:1},{B:1}}		
C			
A			
B			

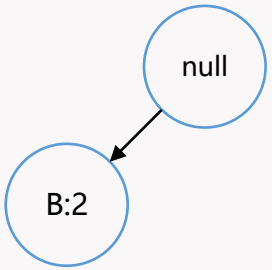
TID	项集	TID	项集
1	{B,A,E}	6	{B,C}
2	**{B,D}**	7	{A,C}
3	{B,C}	8	{B,A,C,E}
4	**{B,A,D}**	9	{B,A,C}
5	{A,C}		

关联分析算法

- 利用FP树产生频繁项集



D的条件FP树

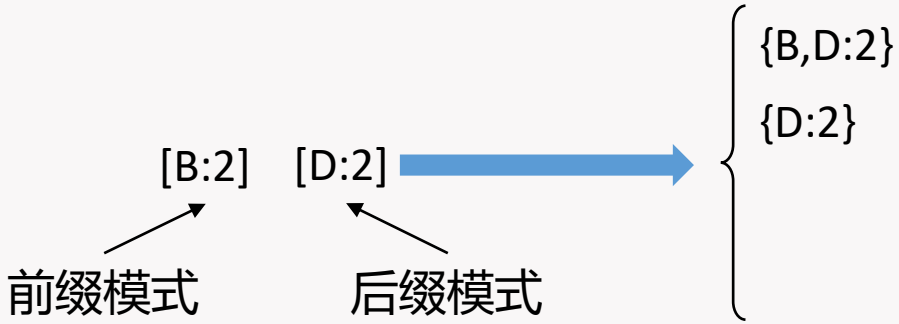


后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D	{{B,A:1},{B:1}}	<B:2>	
C			
A			
B			

关联分析算法

- 利用FP树产生频繁项集

项	支持度计数
B	7
A	6
C	6
D	2
E	2

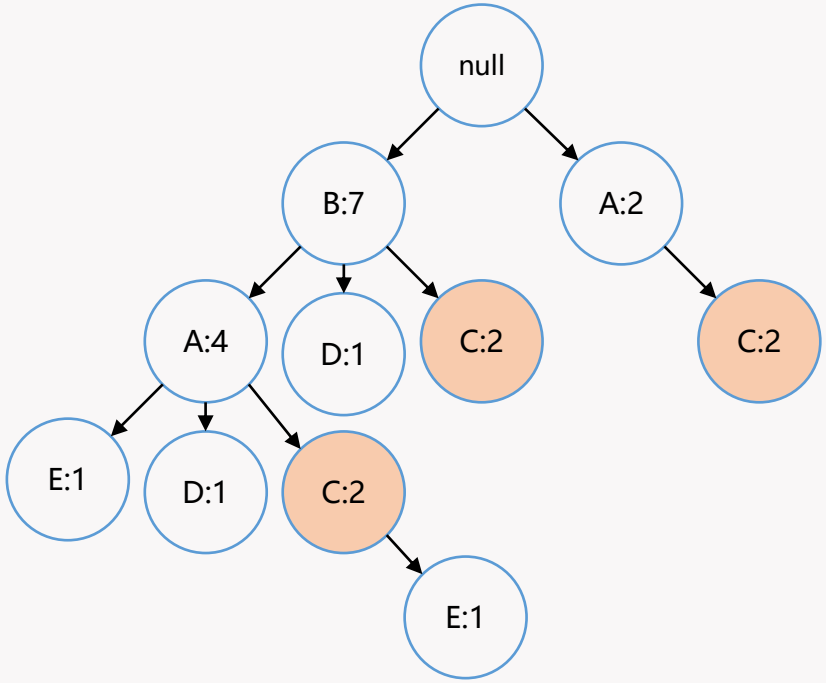


后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}
C			
A			
B			

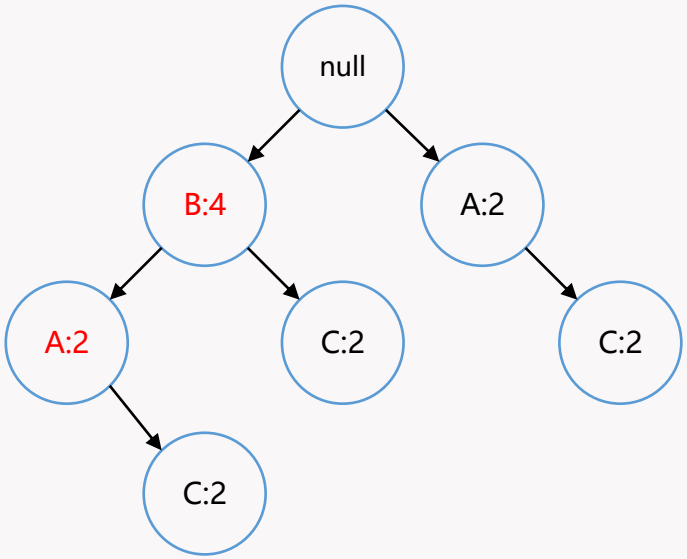
TID	项集	TID	项集
1	{B,A,E}	6	{B,C}
2	**{B,D}**	7	{A,C}
3	{B,C}	8	{B,A,C,E}
4	**{B,A,D}**	9	{B,A,C}
5	{A,C}		

关联分析算法

- 利用FP树产生频繁项集



C的条件模式基

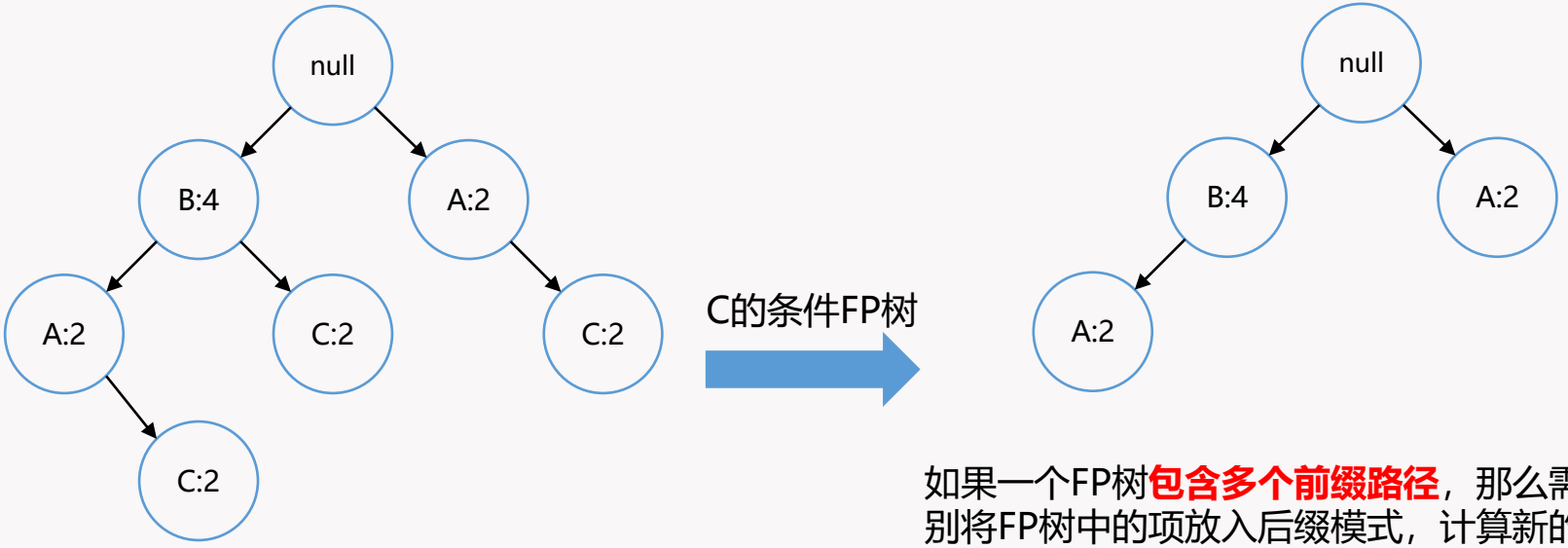


后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}
C	{{B,A:2},{B:2},{A:2}}		
A			
B			

TID	项集	TID	项集
1	{B,A,E}	**6**	**{B,C}**
2	{B,D}	**7**	**{A,C}**
3	**{B,C}**	**8**	**{B,A,C,E}**
4	{B,A,D}	**9**	**{B,A,C}**
5	**{A,C}**		

关联分析算法

- 利用FP树产生频繁项集



如果一个FP树**包含多个前缀路径**，那么需要分别将FP树中的项放入后缀模式，计算新的后缀模式的条件FP树。

后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}
C	{{B,A:2},{B:2},{A:2}}	<B:4,A:2>,<A:2>	{C:6}
A			
B			

关联分析算法

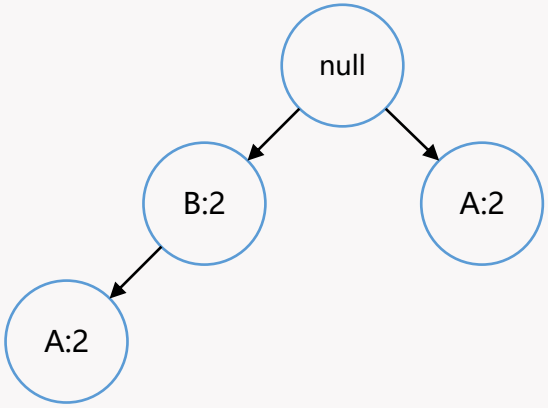
- 利用FP树产生频繁项集

从C的条件FP树中得到A的条件模式基

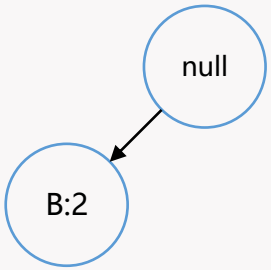
后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}
C	{{B,A:2},{B:2},{A:2}}	<B:4,A:2>,<A:2>	{C:6}
A			
B			

关联分析算法

- 利用FP树产生频繁项集



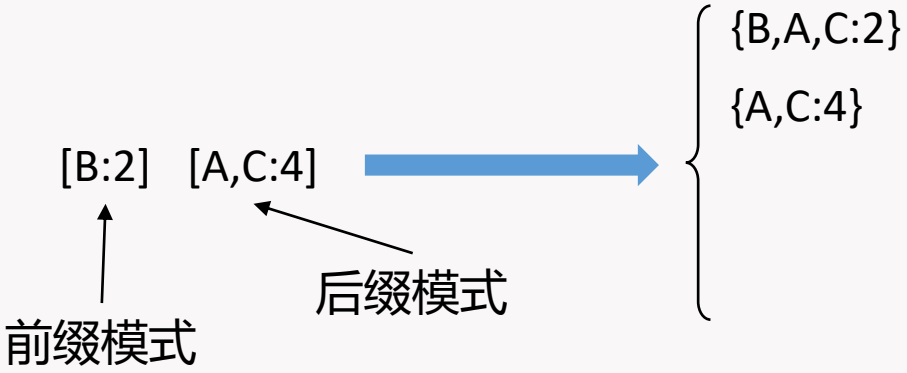
AC的条件FP树



后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}
C	{{B,A:2},{B:2},{A:2}}	<B:4,A:2>,<A:2>	{C:6}
A			
B			

关联分析算法

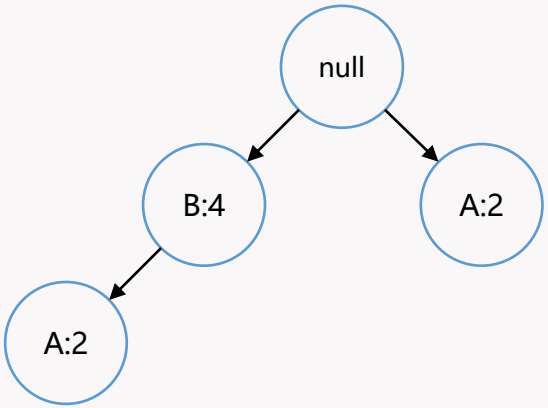
- 利用FP树产生频繁项集

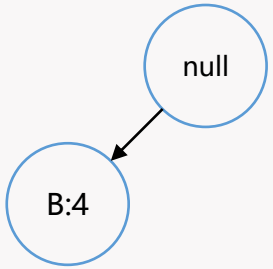


后缀模式	条件模式基	条件FP树	频繁模式	TID	项集	TID	项集
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}	1	{B,A,E}	6	{B,C}
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}	2	{B,D}	**7**	**{A,C}**
C	{{B,A:2},{B:2},{A:2}}	<B:4,A:2>,<A:2>	{C:6}{B,A,C:2},{A,C:4}	3	{B,C}	**8**	**{B,A,C,E}**
A				4	{B,A,D}	**9**	**{B,A,C}**
B				**5**	**{A,C}**		

关联分析算法

- 利用FP树产生频繁项集



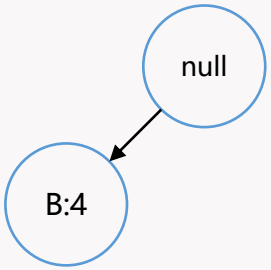


从C的条件FP树中得到B的条件模式基

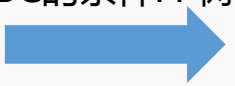
后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}
C	{{B,A:2},{B:2},{A:2}}	<B:4,A:2>,<A:2>	{C:6}{B,A,C:2},{A,C:4}
A			
B			

关联分析算法

- 利用FP树产生频繁项集



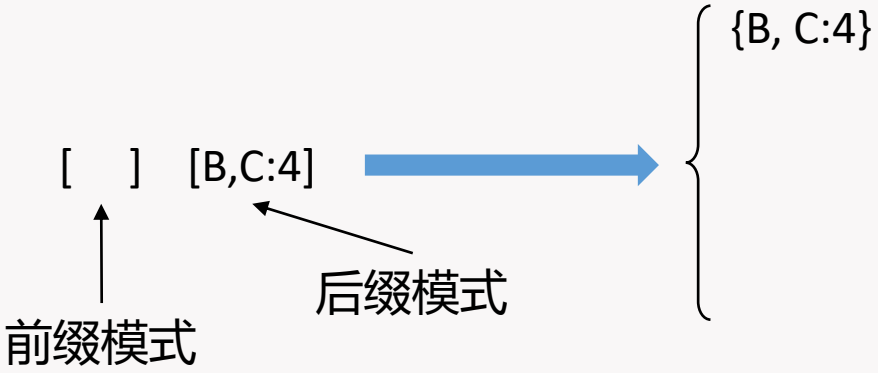
BC的条件FP树



后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}
C	{{B,A:2},{B:2},{A:2}}	<B:4,A:2>,<A:2>	{C:6}{B,A,C:2},{A,C:4}
A			
B			

关联分析算法

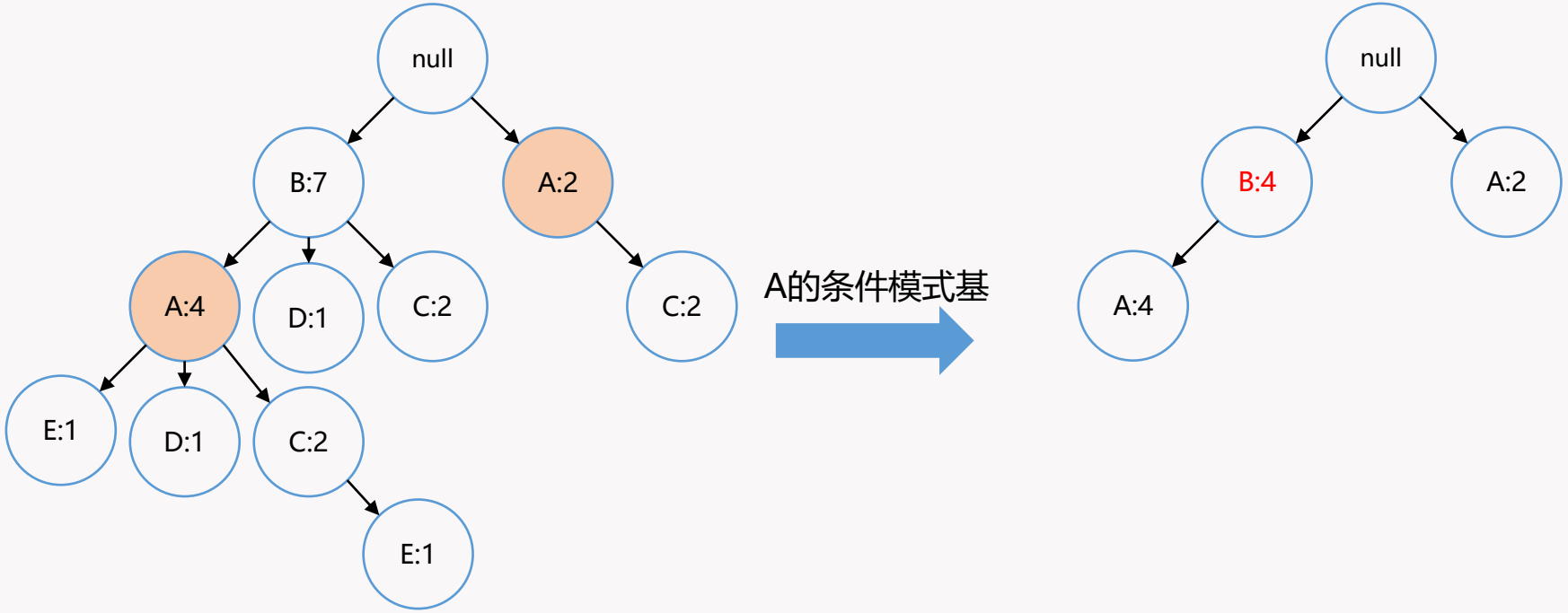
- 利用FP树产生频繁项集



后缀模式	条件模式基	条件FP树	频繁模式	TID	项集	TID	项集
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}	1	{B,A,E}	**6**	**{B,C}**
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}	2	{B,D}	**7**	**{A,C}**
C	{{B,A:2},{B:2},{A:2}}	<B:4,A:2>,<A:2>	{C:6}{B,A,C:2},{A,C:4},{B,C:4}	**3**	**{B,C}**	**8**	**{B,A,C,E}**
A				4	{B,A,D}	**9**	**{B,A,C}**
B				**5**	**{A,C}**		

关联分析算法

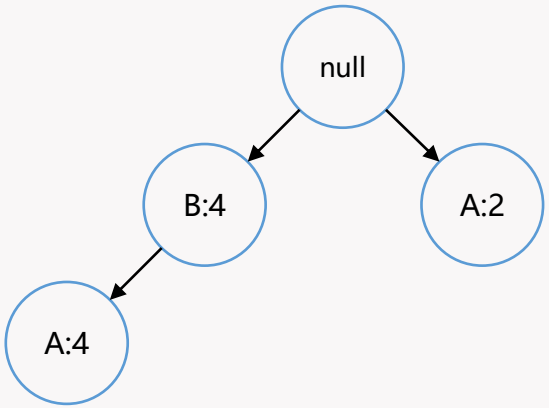
- 利用FP树产生频繁项集



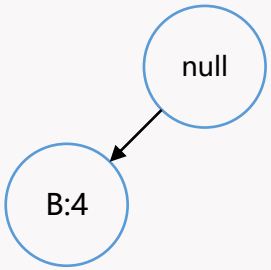
后缀模式	条件模式基	条件FP树	频繁模式	TID	项集	TID	项集
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}	1	{B,A,E}	6	{B,C}
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}	2	{B,D}	7	{A,C}
C	{{B,A:2},{B:2},{A:2}}	<B:4,A:2>,<A:2>	{C:6}{B,A,C:2},{A,C:4},{B,C:4}	3	{B,C}	8	{B,A,C,E}
A	{{B:4}}			4	{B,A,D}	9	{B,A,C}
B				5	{A,C}		

关联分析算法

- 利用FP树产生频繁项集



A的条件FP树

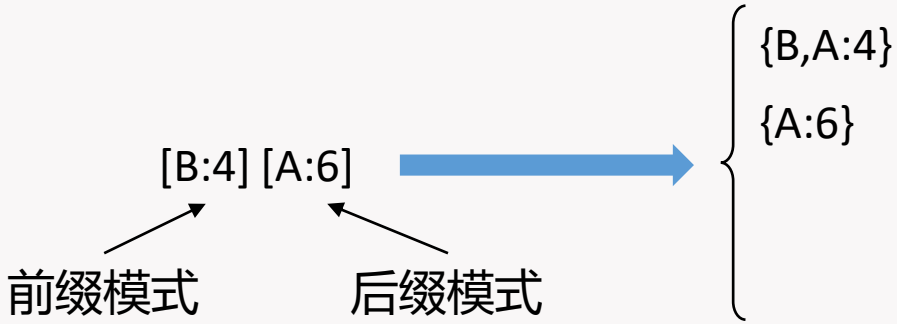


后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}
C	{{B,A:2},{B:2},{A:2}}	<B:4,A:2>,<A:2>	{C:6}{B,A,C:2},{A,C:4},{B,C:4}
A	{{B:4}}	<B:4>	
B			

关联分析算法

- 利用FP树产生频繁项集

项	支持度计数
B	7
A	6
C	6
D	2
E	2

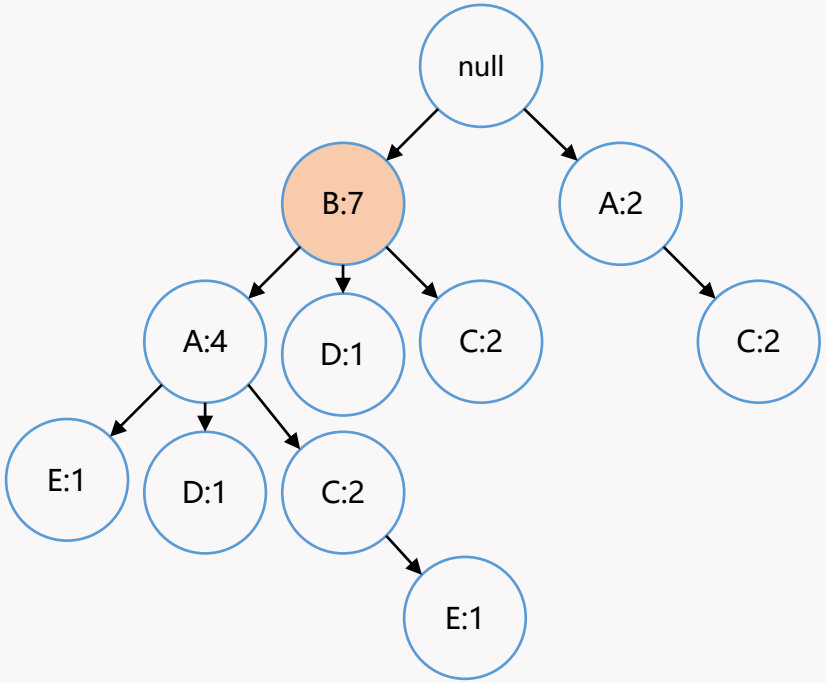


后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}
C	{{B,A:2},{B:2},{A:2}}	<B:4,A:2>,<A:2>	{C:6}{B,A,C:2},{A,C:4},{B,C:4}
A	{{B:4}}	<B:4>	{B,A:4},{A:6}
B			

TID	项集	TID	项集
1	**{B,A,E}**	6	{B,C}
2	{B,D}	7	{A,C}
3	{B,C}	**8**	**{B,A,C,E}**
4	**{B,A,D}**	**9**	**{B,A,C}**
5	{A,C}		

关联分析算法

- 利用FP树产生频繁项集



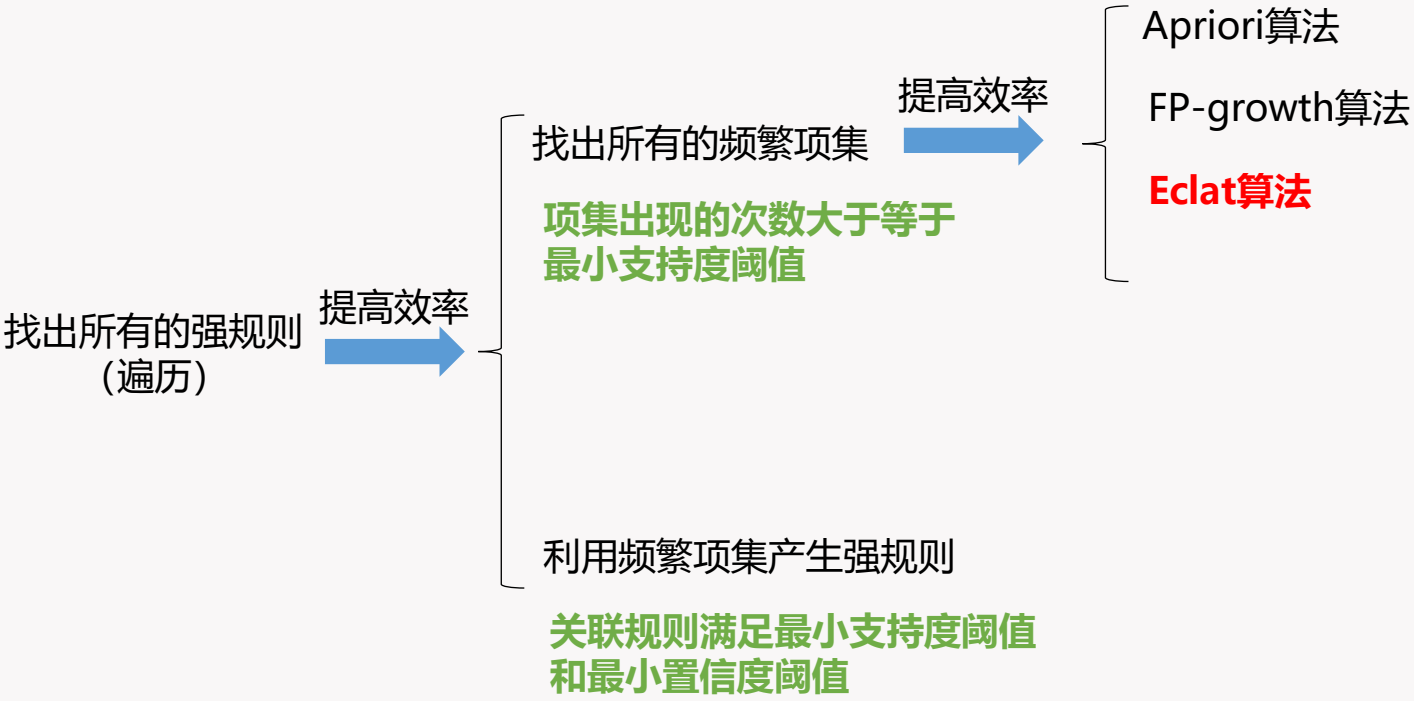
后缀模式	条件模式基	条件FP树	频繁模式
E	{{B,A:1},{B,A,C:1}}	<B,A:2>	{B,E:2},{A,E:2},{B,A,E:2},{E:2}
D	{{B,A:1},{B:1}}	<B:2>	{B,D:2},{D:2}
C	{{B,A:2},{B:2},{A:2}}	<B:4,A:2>,<A:2>	{C:6}{B,A,C:2},{A,C:4},{B,C:4}
A	{{B:4}}	<B:4>	{B,A:4},{A:6}
B	{}	<>	{B:7}

FP-growth的特点

- 优点：
  - 无论挖掘长的或短的频繁项集，FP-growth算法都是有效且可伸缩的；
  - FP-growth算法的性能依赖于事务集的**压缩因子**（Compaction factor），某些情况下FP-growth算法比标准的Apriori算法快几个数量级；
  - FP-growth算法仅需要扫描两次事务集，采用分治方法递归实现。
- 缺点：
  - 实现复杂；
  - 如果生成的FP树非常茂盛，则算法的性能显著下降，因为会产生大量的子问题，并且要合并每个子问题返回的结果。

关联分析算法

研究路线


找出所有的强规则
（遍历）
提高效率
找出所有的频繁项集
项集出现的次数大于等于
最小支持度阈值
提高效率
Apriori算法
FP-growth算法
Eclat算法
利用频繁项集产生强规则
关联规则满足最小支持度阈值
和最小置信度阈值

从垂直数据格式中挖掘频繁项集

- Apriori算法和FP-growth算法都是从**水平数据格式**（Horizontal data format）中挖掘频繁项集。数据也可以采用**垂直数据格式**（Vertical data format）的形式组织，通常使用**等价类变换**（Equivalence CLAss Transformation，Eclat）算法在垂直数据格式上挖掘频繁项集。

水平数据格式

TID	项集
1	{A,B,E}
2	{B,D}
3	{B,C}
4	{A,B,D}
5	{A,C}
6	{B,C}
7	{A,C}
8	{A,B,C,E}
9	{A,B,C,F}

垂直数据格式

项	TID集
A	{1,4,5,7,8,9}
B	{1,2,3,4,6,8,9}
C	{3,5,6,7,8,9}
D	{2,4}
E	{1,8}
F	{9}

水平数据格式转换成垂直数据格式时需要扫描一次事务集！

关联分析算法

- 从垂直数据格式中挖掘频繁项集
- Eclat算法和Apriori算法的过程类似，主要有以下两个区别：
  - Apriori算法保留每个项集的支持度计数，而Eclat算法保留每个项集的TID集；
  - Apriori算法在判断候选项集是否频繁时扫描事务集并统计支持度计数，而Eclat算法判断候选项集是否频繁时计算TID集的交集。

关联分析算法

- 从垂直数据格式中挖掘频繁项集
- Eclat算法示例，假设最小支持度阈值为2。

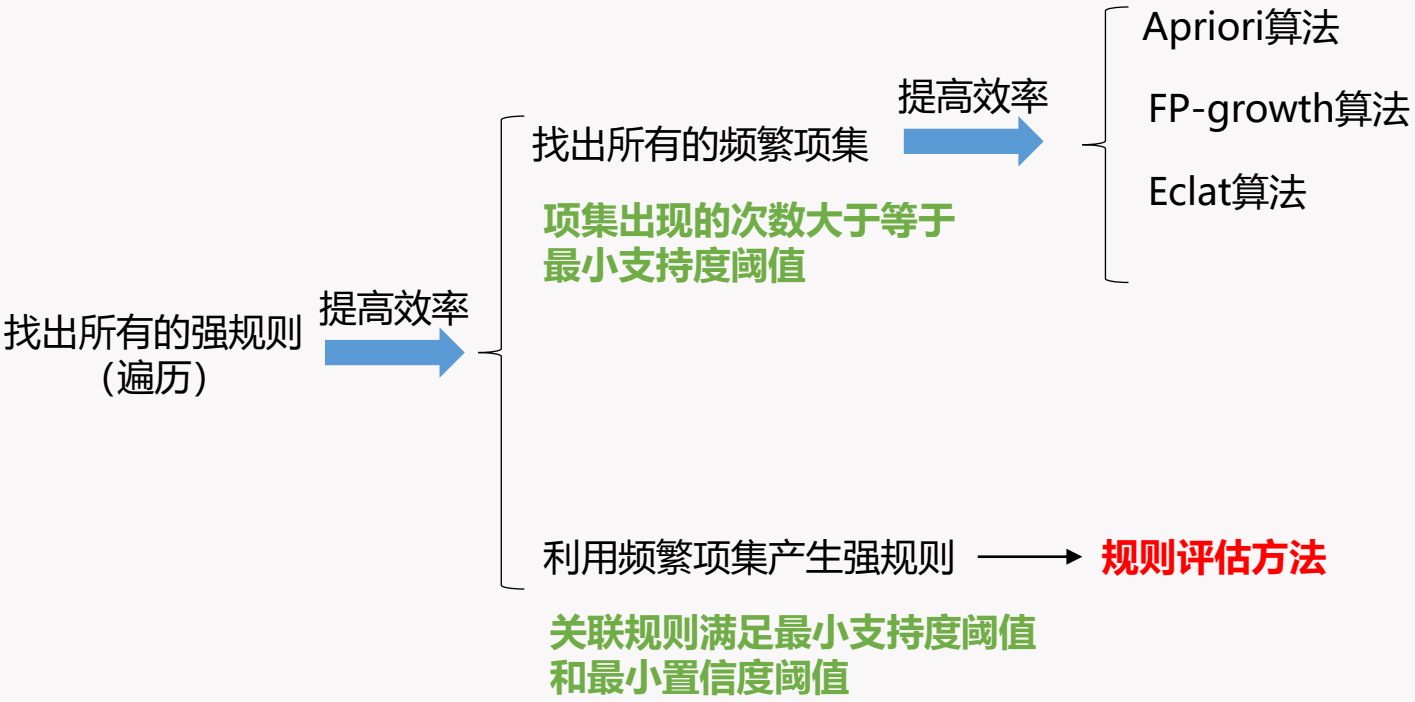
项	TID集
A	{1,4,5,7,8,9}
B	{1,2,3,4,6,8,9}
C	{3,5,6,7,8,9}
D	{2,4}
E	{1,8}
F	{9}

项集	TID集
A,B	{1,4,8,9}
A,C	{5,7,8,9}
A,D	{4}
A,E	{1,8}
B,C	{3,6,8,9}
B,D	{2,4}
B,E	{1,8}
C,D	{}
C,E	{8}
D,E	{}

项集	TID集
A,B,C	{8,9}
A,B,D	{4}
A,B,E	{1,8}
B,C,D	
B,C,E	
C,D,E	

关联分析算法

研究路线


找出所有的强规则
（遍历）
提高效率
找出所有的频繁项集
项集出现的次数大于等于
最小支持度阈值
提高效率
Apriori算法
FP-growth算法
Eclat算法
利用频繁项集产生强规则
规则评估方法
关联规则满足最小支持度阈值
和最小置信度阈值

规则评估方法

- 大部分关联规则挖掘算法都使用支持度-置信度框架。虽然最小支持度和置信度阈值可以帮助我们排除掉大量的无趣规则，但是满足支持度和置信度阈值的强规则也可能是无趣的和误导的。

支持度-置信度框架的缺点

- 假设我们对分析涉及购买计算机游戏（game）和录像（video）的事务集感兴趣。在所分析的10000个事务中，6000个顾客的事务中包含了计算机游戏，7500个顾客的事务中包含了录像，其中有4000个顾客的事务中同时包含了计算机游戏和录像。最小支持度阈值 p_{sup} 为 30%，最小置信度阈值 p_{con} 为 60%，则如下规则

$$\{game\} \rightarrow \{video\}$$

是强关联规则，因为它的支持度是40%，置信度是66.67%。但是购买 video 的概率是 75%，所以这个规则是误导的。

基于相关性的规则判断

- 使用相关性度量来扩充关联规则的支持度-置信度框架，得到如下的**相关规则**（Corelation rule）表示形式

$$X \rightarrow Y[support, confidence, correlation]$$

相关规则同时使用支持度和置信度，以及项集 X 和 Y 之间的相关性三种度量方式判断是否为强关联规则。

相关性度量

- **提升度**（Lift）是一种常用的相关性度量，项集 X 和 Y 的提升度表示为

$$lift(X,Y)=\frac{P(X\cup Y)}{P(X)P(Y)}=\frac{c(Y\rightarrow X)}{s(X)}$$

根据概率论，如果满足 $P(X\cup Y)=P(X)P(Y)$，则项集 X 与项集 Y 之间是独立的，否则 X 和 Y 之间是**依赖的**（Dependent）或**相关的**（Correlated）。

- 以提升度为 1 作为参考，我们有：
  - 当**提升度等于 1** 时，表示项集 X 与项集 Y 是**相互独立**的，所以不存在关联规则；
  - 当**提升度小于 1** 时，表示项集 X 与项集 Y 是**负相关**的，则关联规则是不可靠的；
  - 当**提升度大于 1** 时，表示项集 X 与项集 Y 是**正相关**的，则关联规则是有趣的。

相关性度量

- **杠杆度**（Leverage）是提升度的另一种表示形式，项集 X 和 Y 的杠杆度表示为

$$leverage(X,Y) = P(X \cup Y) - P(X)P(Y)$$

- 以杠杆度为 0 作为参考，我们有：
  - 当**杠杆度等于 0** 时，表示项集 X 与项集 Y 是**相互独立**的，所以不存在关联规则；
  - 当**杠杆度小于 0** 时，表示项集 X 与项集 Y 是**负相关**的，则关联规则是不可靠的；
  - 当**杠杆度大于 0** 时，表示项集 X 与项集 Y 是**正相关**的，则关联规则是有趣的。

相关性度量

- **全置信度**（All confidence）也是一种常用的相关性度量，项集 X 和 Y 的全置信度表示为

$$all\_conf(X,Y)=\frac{s(X\cup Y)}{\max\{s(X),s(Y)\}}=\frac{\sigma(X\cup Y)}{\max\{\sigma(X),\sigma(Y)\}}=\min\{P(X|Y),P(Y|X)\}$$

全置信度又称为关联规则 $X\rightarrow Y$ 和 $Y\rightarrow X$ 的最小置信度。

- 与全置信度对应，项集 X 和 Y 的**最大置信度**（Max confidence）定义为

$$max\_conf(X,Y)=max\{P(X|Y),P(Y|X)\}$$

相关性度量

- 全置信度和最大置信度都只关注了项集 X 和 Y 的某个关联规则的置信度，而**Kulczynski（Kulc）度量**和**余弦（Cosine）度量**则同时利用了关联规则 $X \to Y$ 和 $Y \to X$ 的置信度。Kulc度量表示为

$$Kulc(X,Y) = \frac{1}{2}(c(Y \to X) + c(X \to Y)) = \frac{1}{2}(P(X|Y) + P(Y|X))$$

- 余弦度量表示为

$$Cosine(X,Y) = \frac{P(X \cup Y)}{\sqrt{P(X)P(Y)}} = \frac{\sigma(X \cup Y)}{\sqrt{\sigma(X)\sigma(Y)}} = \sqrt{c(Y \to X)c(X \to Y)} = \sqrt{P(X|Y)P(Y|X)}$$

相依表

- 使用相关性度量时都涉及到了项集的概率计算，所以通常做法是先列出**相依表**（Contingency table），然后根据相依表中的频数来计算相应的概率，比如

- $P(X) = \frac{\sigma(X)}{N} = \frac{f_X}{N}$
- $P(Y|X) = \frac{\sigma(X \cup Y)}{\sigma(X)} = \frac{f_{XY}}{f_X}$
- $P(Y \cup X) = \frac{\sigma(X \cup Y)}{N} = \frac{f_{XY}}{N}$

项集 X 和 Y 的相依表

	X	$\bar{X}$	求和
Y	f_{XY}	$f_{\bar{X}Y}$	f_Y
$\bar{Y}$	$f_{X\bar{Y}}$	$f_{\bar{X}\bar{Y}}$	$f_{\bar{Y}}$
求和	f_X	$f_{\bar{X}}$	N

相关性度量

- 假设 $\overline{game}$ 表示不包含计算机游戏的事务，$\overline{video}$ 表示不包含视频的事务，相依表为

$video$ 和 $game$ 的相依表

	$game$	$\overline{game}$	求和
$video$	4000	2500	6500
$\overline{video}$	2000	500	2500
求和	6000	3000	9000

- 分别计算提升度、杠杆度、全置信度、最大置信度、Kulc度量和余弦度量。

相关性度量方法的评估

项集 $video$ 和 $game$ 的在六个事务集上的相关性度量

事务集	g, v	$\overline{g}, v$	$g, \overline{v}$	$\overline{g}, \overline{v}$	提升度	全置信度	最大置信度	Kluc	余弦
D_1	10000	1000	1000	100000	9.26	0.91	0.91	0.91	0.91
D_2	10000	1000	1000	100	1	0.91	0.91	0.91	0.91
D_3	100	1000	1000	100000	8.44	0.09	0.09	0.09	0.09
D_4	1000	1000	1000	100000	25.75	0.50	0.50	0.50	0.50
D_5	1000	100	10000	100000	9.18	0.09	0.91	0.50	0.29
D_6	1000	10	100000	100000	1.97	0.01	0.99	0.50	0.10

相关性度量方法的评估

项集 $video$ 和 $game$ 的在五个事务集上的相关性度量

事务集	g, v	$\bar{g}, v$	$g, \bar{v}$	$\bar{g}, \bar{v}$	提升度	全置信度	最大置信度	Kluc	余弦
D_1	10000	1000	1000	**100000**	9.26	0.91	0.91	0.91	0.91
D_2	10000	1000	1000	**100**	1	0.91	0.91	0.91	0.91
D_3	100	1000	1000	100000	8.44	0.09	0.09	0.09	0.09
D_4	1000	1000	1000	100000	25.75	0.50	0.50	0.50	0.50
D_5	1000	100	10000	100000	9.18	0.09	0.91	0.50	0.29
D_6	1000	10	100000	100000	1.97	0.01	0.99	0.50	0.10

- 对比事务集 D_1 和 D_2，仅有 $\bar{g}, \bar{v}$ 项不同。五个相关性度量的数值，仅有提升度随着 $\bar{g}, \bar{v}$ 改变而改变。

相关性度量方法的评估

- 对于不包容任何考察项集的事务，我们称之为**零事务**（Null-transaction）。例如示例中的事务集 D_1 ，关于项集 $\{video, game\}$ 的零事务个数是100000。
- 如果一种相关性度量方法不受零事务的影响，则称这种度量是**零不变的**（Null-invariant）。零不变性是一种度量大型事务集中的关联模式的**重要性质**，全置信度、最大置信度、kulc度量和余弦度量都是零不变度量。

相关性度量方法的评估

项集 $video$ 和 $game$ 的在五个事务集上的相关性度量

事务集	g, v	$\overline{g}, v$	$g, \overline{v}$	$\overline{g}, \overline{v}$	提升度	全置信度	最大置信度	Kluc	余弦
D_1	**10000**	1000	1000	100000	9.26	0.91	0.91	0.91	0.91
D_2	10000	1000	1000	100	1	0.91	0.91	0.91	0.91
D_3	**100**	1000	1000	100000	8.44	0.09	0.09	0.09	0.09
D_4	**1000**	1000	1000	100000	25.75	0.50	0.50	0.50	0.50
D_5	1000	100	10000	100000	9.18	0.09	0.91	0.50	0.29
D_6	1000	10	100000	100000	1.97	0.01	0.99	0.50	0.10

- 对比事务集 D_1、D_3、D_4，仅有 $\boldsymbol{g, v}$ 项不同。零不变度量随着 $\boldsymbol{g, v}$ 的增加而增加，但是提升度却得出自相矛盾的结果。

关联分析算法

相关性度量方法的评估

项集 $video$ 和 $game$ 的在五个事务集上的相关性度量

事务集	g, v	$\overline{g}, v$	$g, \overline{v}$	$\overline{g}, \overline{v}$	提升度	全置信度	最大置信度	Kluc	余弦
D_1	10000	1000	1000	100000	9.26	0.91	0.91	0.91	0.91
D_2	10000	1000	1000	100	1	0.91	0.91	0.91	0.91
D_3	100	1000	1000	100000	8.44	0.09	0.09	0.09	0.09
D_4	1000	1000	1000	100000	25.75	0.50	0.50	0.50	0.50
D_5	1000	**100**	**10000**	100000	9.18	0.09	0.91	0.50	0.29
D_6	1000	**10**	**100000**	100000	1.97	0.01	0.99	0.50	0.10

- 对比事务集 D_5、D_6，四个零不变度量给出不同的结果。原因是 $P(video|game)$ 和 $P(game|video)$ 的不一致。

相关性度量方法的评估

- 当项集 X 和 Y 蕴含的规则的概率不一致时，引入**不平衡比**（Imbalance Ratio，IR）的概念，用于评估规则蕴含式中两个项集 X 和 Y 的不平衡程度。IR定义为

$$IR(X,Y)=\frac{|\sigma(X)-\sigma(Y)|}{\sigma(X)+\sigma(Y)-\sigma(X\cap Y)}$$

  - 分子表示项集 X 和 Y 的支持度计数之差，分母表示包含项集 X 或 Y 的事务总数。

相关性度量方法的评估

项集 $video$ 和 $game$ 的在五个事务集上的相关性度量

事务集	g, v	$\overline{g}, v$	$g, \overline{v}$	$\overline{g}, \overline{v}$	提升度	全置信度	最大置信度	Kluc	余弦
D_1	10000	1000	1000	100000	9.26	0.91	0.91	0.91	0.91
D_2	10000	1000	1000	100	1	0.91	0.91	0.91	0.91
D_3	100	1000	1000	100000	8.44	0.09	0.09	0.09	0.09
D_4	1000	1000	1000	100000	25.75	0.50	0.50	0.50	0.50
D_5	1000	100	10000	100000	9.18	0.09	0.91	0.50	0.29
D_6	1000	10	100000	100000	1.97	0.01	0.99	0.50	0.10

- 对于事务集 D_5，有

$$P(game|video) = 90.91\%, \quad P(video|game) = 9.09\%, \quad IR(video, game) = 0.89$$

全置信度和余弦度量认为是负相关而最大置信度则是正相关，Kluc度量认为是中性。

相关性度量方法的评估

项集 $video$ 和 $game$ 的在五个事务集上的相关性度量

事务集	g, v	$\overline{g}, v$	$g, \overline{v}$	$\overline{g}, \overline{v}$	提升度	全置信度	最大置信度	Kluc	余弦
D_1	10000	1000	1000	100000	9.26	0.91	0.91	0.91	0.91
D_2	10000	1000	1000	100	1	0.91	0.91	0.91	0.91
D_3	100	1000	1000	100000	8.44	0.09	0.09	0.09	0.09
D_4	1000	1000	1000	100000	25.75	0.50	0.50	0.50	0.50
D_5	1000	100	10000	100000	9.18	0.09	0.91	0.50	0.29
D_6	1000	10	100000	100000	1.97	0.01	0.99	0.50	0.10

- 对于事务集 D_6，有

$$P(game|video) = 99.01\%, \quad P(video|game) = 0.99\%, \quad IR(video, game) = 0.99$$

随着不平衡程度的增加，余弦度量给出更倾向于负相关的结论。

相关性度量方法的评估

- 对于“不平衡”的规则，Kluc度量可能会给出更公平的评价。在示例中，Kluc度量给出了中性的评价，但从 IR 可以看出“不平衡”的程度。所以**推荐**同时使用Kluc度量和 IR 度量。
- 余弦度量与提升度的区别在于，余弦度量对 $P(X)P(Y)$ 的计算结果开平方，所以余弦度量不受事务总数的影响，因此余弦度量是零不变的。

Chapter 4.3

关联分析高级概念

关联分析高级概念

高级关联分析

- 上一节介绍的关联规则挖掘算法都认为项是非对称的二元属性。我们更关注于事务中出现的项，并且只有支持度计数超过最小支持度阈值的项或项集认为是频繁的、有趣的。
- 本节我们对关联规则挖掘算法进行拓展，将其应用在对称的二元属性、标称属性和数值属性，以及更复杂的数据以进行关联规则的挖掘。

关联分析高级概念

- 处理对称的二元属性和标称属性
- 对于对称的二元属性和标称属性，对属性的每一个观测值都创建一个非对称的二元项。

学历
{高中，专科，本科，硕士，博士}

高中	专科	本科	硕士	博士
{1，0}	{1，0}	{1，0}	{1，0}	{1，0}

性别
{男，女}

男	女
{1，0}	{1，0}

关联分析高级概念

处理对称的二元属性和标称属性

- 将标称属性和对称的二元属性转换为非对称的二元属性时，需要考虑以下问题：
  - 某些观测值在事务集中出现的次数较少，导致其对应的项是非频繁的，所以我们在挖掘频繁模式时无法找到与这些观测值相关的关联规则。一般的解决方法是概念分层，或是仅合并部分观测值；
  - 某些项之间是互斥的，所以应该避免产生包含多个来自同一个属性的项的项集，因为该项集的支持度计数必然为零，比如{本科，硕士}这一项集。

关联分析高级概念

- 处理数值属性
  - 包含数值属性的关联规则通常称为**量化关联规则**（Quantitative association rule），处理数值属性时常采用以下三种方法：
    - 基于离散化的方法
    - 基于统计学的方法
    - 非离散化的方法

基于离散化的数值属性处理方法

- 离散化是处理数值属性的常用方法，这种方法将数值属性的取值范围分组，形成有限个区间。将离散化后的区间映射到非对称的二元属性，我们就可以使用已有的关联分析算法挖掘关联规则。
- 年龄属性离散化为[10,20)， [20,30)， [30,40)， [40,50)，[50,60)，然后将离散化后的年龄属性转换为五个非对称的二元属性。

关联分析高级概念

基于离散化的数值属性处理方法

- 离散化后区间的长度是一个关键的超参数，不同的区间长度会产生不同的关联规则。假设最小支持度阈值是5%，最小置信度阈值是65%。

年龄	上网	不上网
[12,16)	12	13
[16,20)	11	2
[20,24)	11	3
[24,28)	12	13
[28,32)	14	12
[32,36)	15	12
[36,40)	16	14
[40,44)	16	14
[44,48)	4	10
[48,52)	5	11
[52,56)	5	10
[56,60)	4	11
总计	125	125

- 对于关联规则：{年龄∈[12,36)}→{上网}

$$sup = \frac{75}{125+125} = 30.00\%,\quad con = \frac{75}{75+55} = 57.69\%$$

- 对于关联规则：{年龄∈[36,60)}→{不上网}

$$sup = \frac{70}{125+125} = 28.00\%,\quad con = \frac{70}{50+70} = 58.33\%$$

- 区间太宽可能因为置信度不足而丢弃某些规则。
- 采用区间分裂的方法缓解这个问题。

关联分析高级概念

基于离散化的数值属性处理方法

- 离散化后区间的长度是一个关键的超参数，不同的区间长度会产生不同的关联规则。假设最小支持度阈值是5%，最小置信度阈值是65%。

年龄	上网	不上网
[12,16)	12	13
[16,20)	11	2
[20,24)	11	3
[24,28)	12	13
[28,32)	14	12
[32,36)	15	12
[36,40)	16	14
[40,44)	16	14
[44,48)	4	10
[48,52)	5	11
[52,56)	5	10
[56,60)	4	11
总计	125	125

- 对于关联规则：{年龄∈[16,20)}→{上网}

$$sup = \frac{11}{125+125} = 4.40\%,\quad con = \frac{11}{11+2} = 84.62\%$$

- 对于关联规则：{年龄∈[20,24)}→{上网}

$$sup = \frac{11}{125+125} = 4.40\%,\quad con = \frac{11}{11+3} = 78.57\%$$

- 区间太窄可能因为支持度不足而丢弃某些规则。
- 采用区间融合的方法缓解这个问题。

关联分析高级概念

- 基于统计的数值属性处理方法

- 统计量可以反映量化关联规则的统计性质，给出量化关联规则更准确的刻画和描述。

年收入[0,10)	年收入[10,20)	年收入[20,30)	…	网购	男	女	年龄
1	0	0	…	1	1	0	27
1	0	0	…	1	0	1	25
0	1	0	…	0	0	1	29
0	0	1	…	1	0	1	24

- 量化关联规则：{年收入∈[30,40),网购}→{年龄：均值=29}，这表明年收入在30万到40万并且网上购物的用户的平均年龄是29岁。

基于统计的数值属性处理方法

- 为了产生目标属性的基于统计的量化关联规则，需要指定用于刻画目标属性的相关属性。具体做法是：
  - 保留目标属性，对其他标称属性、数值属性二元化；
  - 使用Apriori算法或FP-growth算法在非对称的二元属性上产生频繁项集；
  - 在每个频繁项集上对目标属性计算均值、中位数、方差等统计量。
- 在基于统计的量化关联规则中，直接利用**频繁项集**和**目标属性的统计量**得到关联规则，所以**无法计算置信度**，而是采用**统计假设检验**的方法判断是否是有趣的。

关联分析高级概念

非离散的数值属性处理方法

- 在一些应用中，分析者对数值属性之间的关联更感兴趣，而不是数值属性的离散区间之间的关联，比如在文档中找出词的关联。

文档	数据	挖掘	关联	分析	系统	区间
d1	15	30	0	0	0	10
d2	10	10	0	0	20	5
d3	20	10	50	50	15	30
d4	5	0	0	0	15	5
d5	0	0	0	0	0	0

关联分析高级概念

非离散的数值属性处理方法

- 通常采用min支持度度量计算词之间的关联程度：
  - $\sup(\{数据, 挖掘\}) = \min(0.3,0.6) + \min(0.2,0.2) + \min(0.4,0.1) + \min(0.1,0) = 60\%$
  - $\sup(\{数据, 挖掘, 系统\}) = 40\%$
  - $\mathrm{con}(\{数据, 挖掘\} \rightarrow \{系统\}) = \frac{0.4}{0.6} = 66.67\%$

文档	数据	挖掘	关联	分析	系统	区间
d1	0.3	0.6	0.0	0.0	0.0	0.2
d2	0.2	0.2	0.0	0.0	0.4	0.1
d3	0.4	0.2	1.0	1.0	0.3	0.6
d4	0.1	0.0	0.0	0.0	0.3	0.1
d5	0.0	0.0	0.0	0.0	0.0	0.0

关联分析高级概念

- 非离散的数值属性处理方法
- 发现词之间关联的支持度度量方法需要满足以下要求：
  - 支持度随词的规范化频率增加而单调递增；
  - 支持度随包含该词的文档个数增加而单调递增；
  - 支持度具有反单调性。
- 使用新的支持度定义可以修改Apriori算法，比如使用min支持度度量的Apriori算法被称作min-Apriori算法。

关联分析高级概念

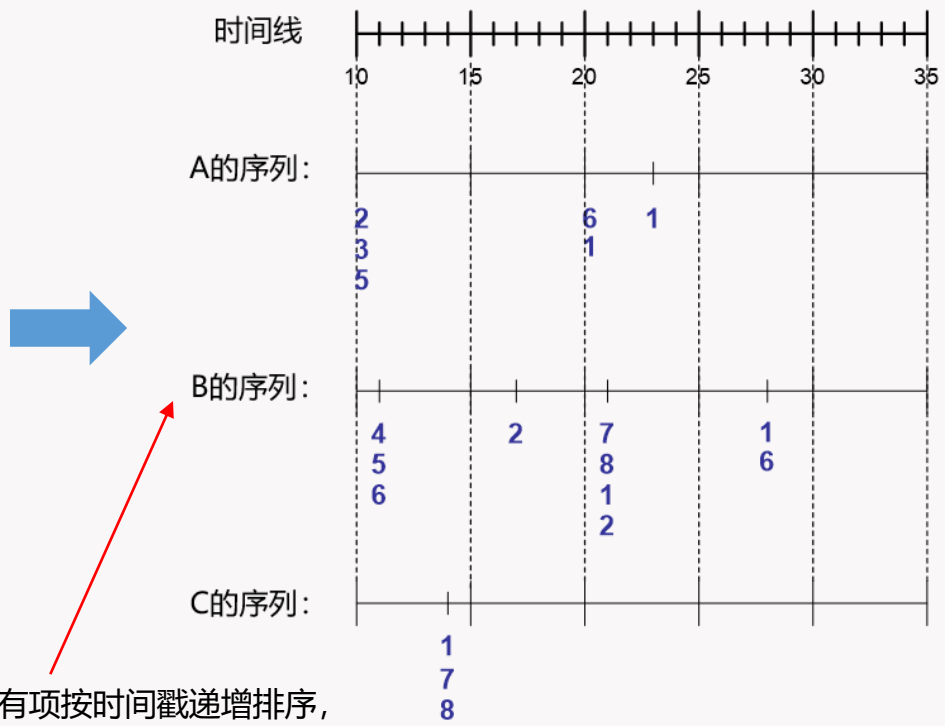
复杂数据上的关联分析

- 多年来广泛的研究极大地拓展了关联分析的理论与应用，关联分析的挖掘早已超越了事务数据。关联分析的拓展主要在以下三个方面：
  - 除了挖掘基本模式（频繁模式，闭模式，极大模式），还有挖掘多层和多维模式、不频繁模式或稀有模式、负模式等；
  - 除了基本的挖掘方法（Apripri、FP-growth，Eclat），还有基于约束的挖掘方法、基于异常规则的挖掘方法、分布式挖掘方法等；
  - 除了基本的事务数据，还有**序列数据**、结构（树、图）数据、空间数据、图像视频和多媒体数据等；

序列模式挖掘

- 之前讨论的关联分析算法都只强调项的共存关系，而忽略了它们的时序信息。时序模式的挖掘可以发现项之间的时序关联，这对于实时动态系统或预测特定的事件未来是否发生等都是极有价值的。

对象	时间戳	项集
A	10	2,3,5
A	20	6,1
A	23	1
B	11	4,5,6
B	17	2
B	21	7,8,1,2
B	28	1,6
C	14	1,8,7



将与对象B有关的所有项按时间戳递增排序，就得到了对象B的一个**序列**（Sequence）

序列模式挖掘

- 一般情况下，序列是**元素**（Element）的有序列表，记作$S =< s_1, s_2, \ldots, s_n >$，其中每个 s_j 是一个或多个项的集合，表示为$s_j = \{i_1, i_2, \ldots, i_{m_j}\}$，比如：
  - 购物篮序列：<{面包，牛奶}，{香蕉，番茄，卷心菜}，{牛奶，酸奶}>
  - 课程序列：<{计算机导论，高等数学}，{数据挖掘，数据库系统}，{计算机视觉，自然语言处理}>
- 购物篮序列中包含3个元素（事务），6个项
- 课程序列中包含3个元素（事务），5个项

序列模式挖掘

- 如果 T 中的每个有序元素都是 S 中一个有序元素的子集，则称序列 T 是序列 S 的**子序列**（Subsequence），也称序列 T 包含在序列 S 中。
- 如果存在整数$1 \le j_1 < j_2 < \cdots < j_m \le n$，使得 $t_1 \subseteq s_{j_1}$，$t_2 \subseteq s_{j_2}$，$t_3 \subseteq s_{j_3}$，…，$t_m \subseteq s_{j_m}$，则称序列 $T =< t_1, t_2, \ldots, t_m >$是序列 $S =< s_1, s_2, \ldots, s_n >$ 的子序列。

序列S	序列T	T是S的子序列
<{2,4},{3,5,6},{8}>	<{2},{3,6},{8}>	是
<{2,4},{3,5,6},{8}>	<{2},{8}>	是
<{1,2},{3,4}>	<{1},{2}>	否
<{2,4},{2,4},{2,4}>	<{2},{4}>	是

序列模式挖掘

- 假设 D 是包含一个或多个**数据序列**（Data sequence）的数据集，数据序列指的是与单个对象相关联的项的有序列表，那么序列 T 的支持度计数是所有包含 T 的数据序列的数量。
- 如果序列 T 的支持度大于等于最小支持度阈值，则称序列 T 是频繁的，或者称其是一个**序列模式**（频繁序列）。

对象	时间戳	项集
A	10	2,3,5
A	20	1,6
A	23	1
B	11	4,5,6
B	17	2
B	21	1,2,7,8
B	28	1,6
C	14	1,7,8
…	…	…

数据序列

对象	序列
A	<{2,3,5},{1,6},{1}>
B	<{4,5,6},{2},{1,2,7,8},{1,6}>
C	<{1,7,8},{2,3,4}>
D	<{2},{2,3,4},{4,5}>

最小支持度阈值=60%

序列模式:

< {1,2} >	sup=25%
< {2,3} >	sup=75%
< {2,4}>	sup=50%
< {3},{5}>	sup=25%
< {1},{2} >	sup=25%
< {2},{2} >	sup=50%
< {1},{2,3} >	sup=25%
< {2},{2,3} >	sup=25%
< {1,2},{2,3} >	sup= 0%

序列模式挖掘

- k-序列表示包含 k 个项（可重复）的所有序列组合：

- 1-序列： $< i_1 >, < i_2 >, < i_3 > \cdots, < i_N >$

- 2-序列： $< \{i_1, i_2\} >, < \{i_1, i_3\} >, < \{i_1, i_4\} > \cdots, < \{i_{N-1}, i_N\} >$

  $< \{i_1\}, \{i_1\} >, < \{i_1\}, \{i_2\} >, < \{i_1\}, \{i_3\} > \cdots, < \{i_{N-1}\}, \{i_N\} >$

- 3-序列： $< \{i_1, i_2, i_3\} >, < \{i_1, i_2, i_4\} > \cdots$

  $< \{i_1\}, \{i_1, i_2\} >, < \{i_1\}, \{i_1, i_3\} > \cdots$

  $< \{i_1, i_2\}, \{i_1\} >, < \{i_1, i_2\}, \{i_2\} > \cdots$

  $< \{i_1\}, \{i_1\}, \{i_1\} >, < \{i_1\}, \{i_1\}, \{i_2\} > \cdots$

序列模式挖掘

- 在挖掘频繁序列时，一种朴素的方法是列举所有的序列，然后分别计算其支持度，比如1-序列，2-序列，3-序列…… 。但序列的个数要远远多于项集的个数，原因是：
  - 一个项在项集中最多出现一次，但在序列中可以出现多次。比如给定两个项 a 和 b，只能产生2-项集{a,b}，但是却可以产生多个2-序列，比如<{a,b}>，<{a},{b}>，<{b},{a}>，<{a},{a}>等。

关联分析高级概念

- 序列模式挖掘
  - 先验原理对于序列模式同样成立，即包含特定 k+1-序列的任何数据序列必然包含该序列的所有 k-子序列。

- GSP算法中候选序列的产生

数据序列
<{A},{B},{F,G},{C},{D}>
<{B},{G},{D}>
<{B},{F},{G},{A,B}>
<{F},{A,B},{C},{D}>
<{A},{B,C},{G},{F},{D,E}>

最小支持度阈值=28.6%

项	支持度计数
A	4
B	5
C	3
D	4
E	1
F	4
G	4

A,B...,E代表项，不是之前表中的对象！

- GSP算法中候选序列的产生

两个元素的序列

	A	B	C	D	F	G
A	{A}{A}	{A}{B}	{A}{C}	{A}{D}	{A}{F}	{A}{G}
B	{B}{A}	{B}{B}	{B}{C}	{B}{D}	{B}{F}	{B}{G}
C	{C}{A}	{C}{B}	{C}{C}	{C}{D}	{C}{F}	{C}{G}
D	{D}{A}	{D}{B}	{D}{C}	{D}{D}	{D}{F}	{D}{G}
F	{F}{A}	{F}{B}	{F}{C}	{F}{D}	{F}{F}	{F}{G}
G	{G}{A}	{G}{B}	{G}{C}	{G}{D}	{G}{F}	{G}{G}

一个元素的序列

	A	B	C	D	F	G
A		{A,B}	{A,C}	{A,D}	{A,F}	{A,G}
B			{B,C}	{B,D}	{B,F}	{B,G}
C				{C,D}	{C,F}	{C,G}
D					{D,F}	{D,G}
F						{F,G}
G						

关联分析高级概念

GSP算法中候选序列的产生

{A},{A}	{A},{B}	{A},{C}
<{A},{B},{F,G},{C},{D}>	<{**A**},{**B**},{F,G},{C},{D}>	<{**A**},{B},{F,G},{**C**},{D}>
<{B},{G},{D}>	<{B},{G},{D}>	<{B},{G},{D}>
<{B},{F},{G},{A,B}>	<{B},{F},{G},{A,B}>	<{B},{F},{G},{A,B}>
<{F},{A,B},{C},{D}>	<{F},{A,B},{C},{D}>	<{F},{A,B},{C},{D}>
<{A},{B,C},{G},{F},{D,E}>	<{**A**},{**B**,C},{G},{F},{D,E}>	<{**A**},{B,**C**},{G},{F},{D,E}>

{A},{D}	{A},{F}	{A},{G}
<{**A**},{B},{F,G},{C},{**D**}>	<{**A**},{B},{**F**,G},{C},{D}>	<{**A**},{B},{F,**G**},{C},{D}>
<{B},{G},{D}>	<{B},{G},{D}>	<{B},{G},{D}>
<{B},{F},{G},{A,B}>	<{B},{F},{G},{A,B}>	<{B},{F},{G},{A,B}>
<{F},{**A**,B},{C},{**D**}>	<{F},{A,B},{C},{D}>	<{F},{A,B},{C},{D}>
<{**A**},{B,C},{G},{F},{**D**,E}>	<{**A**},{B,C},{G},{**F**},{D,E}>	<{**A**},{B,C},{**G**},{F},{D,E}>

频繁2-序列
<{A},{B}>
<{A},{C}>
<{A},{D}>
<{A},{F}>
<{A},{G}>
<{B},{C}>
<{B},{D}>
<{B},{F}>
<{B},{G}>
<{C},{D}>
<{F},{A}>
<{F},{B}>
<{F},{C}>
<{F},{D}>
<{G},{D}>
<{A,B}>

GSP算法中候选序列的产生

- k-序列 S 和 k-序列 T 合并，当且仅当 S 中去掉**第一个项**得到的 (k-1)-子序列和 T 中去掉**最后一个项**得到的 (k-1)-子序列相同。合并的结果有以下两种可能：
  - 如果 T 的最后两个项属于相同的元素，那么 T 中最后一个项合并到 S 中最后一个元素；
  - 如果 T 的最后两个项属于不同的元素，那么 T 中最后一个项形成独立的元素并加入到 S 。

- GSP算法中候选序列的产生

频繁2-序列	去首项	去尾项	频繁2-序列	去首项	去尾项
<{A},{B}>	<{B}>	<{A}>	<{B},{G}>	<{G}>	<{B}>
<{A},{C}>	<{C}>	<{A}>	<{C},{D}>	<{D}>	<{C}>
<{A},{D}>	<{D}>	<{A}>	<{F},{A}>	<{A}>	<{F}>
<{A},{F}>	<{F}>	<{A}>	<{F},{B}>	<{B}>	<{F}>
<{A},{G}>	<{G}>	<{A}>	<{F},{C}>	<{C}>	<{F}>
<{B},{C}>	<{C}>	<{B}>	<{F},{D}>	<{D}>	<{F}>
<{B},{D}>	<{D}>	<{B}>	<{G},{D}>	<{D}>	<{G}>
<{B},{F}>	<{F}>	<{B}>	**<{A,B}>**	**<{B}>**	**<{A}>**
				<{A}>	**<{B}>**

关联分析高级概念

- GSP算法中候选序列的产生

频繁2-序列1	频繁2-序列1去首项	频繁2-序列2	频繁2-序列2去尾项	候选3-序列	候选3-序列剪枝后	支持度计数	频繁3-序列
<{A},{B}>	<{B}>	<{B},{C}>	<{B}>	<{A},{B},{C}>	<{A},{B},{C}>	1	
<{A},{B}>	<{B}>	<{A,B}>	<{B}>	<{A},{A,B}>			
<{A},{F}>	<{F}>	<{F},{D}>	<{F}>	<{A},{F},{D}>	<{A},{F},{D}>	2	<{A},{F},{D}>
<{A,B}>	<{B}>	<{B},{C}>	<{B}>	<{A,B},{C}>	<{A,B},{C}>	1	
…	…	…	…	…	…	…	…

<{A},{A,B}>的子序列<{A},{A}>是非频繁序列，所以被剪枝!

数据序列
<{A},{B},{F,G},{C},{D}>
<{B},{G},{D}>
<{B},{F},{G},{A,B}>
<{F},{A,B},{C},{D}>
<{A},{B,C},{G},{F},{D,E}>

- GSP算法中候选序列的产生

频繁3-序列	去首项	去尾项	频繁3-序列	去首项	去尾项
<{A},{B},{D}>	<{B},{D}>	<{A},{B}>	<{B},{C},{D}>	<{C},{D}>	<{B},{C}>
<{A},{B},{F}>	<{B},{F}>	<{A},{B}>	<{B},{F},{D}>	<{F},{D}>	<{B},{F}>
<{A},{B},{G}>	<{B},{G}>	<{A},{B}>	<{B},{G},{D}>	<{G},{D}>	<{B},{G}>
<{A},{C},{D}>	<{C},{D}>	<{A},{C}>	<{F},{A,B}>	<{A,B}>	<{F},{A}>
<{A},{F},{D}>	<{F},{D}>	<{A},{F}>			<{F},{B}>
<{A},{G},{D}>	<{G},{D}>	<{A},{G}>	<{F},{C},{D}>	<{B},{D}>	<{B},{D}>

- GSP算法中候选序列的产生

频繁3-序列1	频繁3-序列1去首项	频繁3-序列2	频繁3-序列2去尾项	候选4-序列	候选4-序列剪枝后	支持度计数	频繁4-序列
<{A},{B},{F}>	<{B},{F}>	<{B},{F},{D}>	<{B},{F}>	<{A},{B},{F},{D}>	<{A},{B},{F},{D}>	2	<{A},{B},{F},{D}>
<{A},{B},{G}>	<{B},{G}>	<{B},{G},{D}>	<{B},{G}>	<{A},{B},{G},{D}>	<{A},{B},{G},{D}>	2	<{A},{B},{G},{D}>

数据序列
<{A},{B},{F,G},{C},{D}>
<{B},{G},{D}>
<{B},{F},{G},{A,B}>
<{F},{A,B},{C},{D}>
<{A},{B,C},{G},{F},{D,E}>

关联分析算法

GSP算法产生频繁序列

GSP_frequentSequence (T, I, m)

T：序列事务集，I： 项的集合，m：最小支持度阈值

```
1   k=1
2   F_1 = {i| i ∈ I, σ(i) ≥ m}        计算频繁1-序列
3   Repeat
4     If F_k == ∅ then                中止条件判断
5       Return ∪_k F_k
6     End If
7     C_{k+1} = GSP_gen(F_k)          基于频繁k-序列产生候选k+1-序列，对非频繁k+1-序列剪枝（2-序列单独生成）
8     For every t ∈ T do
9       C_t = select(C_{k+1}, t)      筛选 t 包含的所有候选k+1-项集
10      For every c ∈ C_t do
11        σ(c)+=1                     对筛选后的候选k+1-序列增加支持度计数
12      End For
13    End For
14    F_{k+1} = {c| c ∈ C_{k+1}, σ(c) ≥ m}   从候选k+1-序列中提取频繁k+1-序列
15    k=k+1
16  End Repeat
```

Chapter 4

本章小结

本章小结

关联分析

- 关联分析相关概念：事务集、项、支持度、置信度、频繁的定义、频繁项集的紧凑表示。
- 关联分析算法：Apriori算法相关、hash树、FP-growth、Eclat。
- 模式评估方法：提升度、杠杆度、全置信度、最大置信度、Kulc度量和余弦度量、相依表、零事务和零不变性、不平衡比。
- 关联分析高级概念：标称属性、对称的二元属性和数值属性的转化，序列模式挖掘。

第四章 完结