哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

关联规则分析

李春山

2020年10月20日

主要内容

- **关联规则分析概述**
- **Apriori 算法**
- **FP-Tree 算法**
- **小结**

关联规则分析概述

概述-什么是关联规则分析?

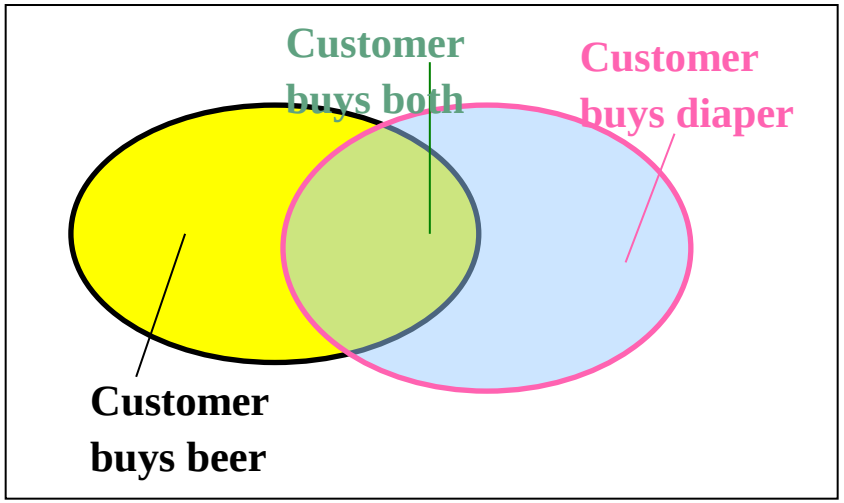
- **关联规则分析：**
  - 在事务数据库，关系数据库和其他信息存储中的大量数据集中发现有价值的、频繁出现的模式、关联和相关性。
  - 给定一组交易/记录, 关联规则分析将发现若干规则，这些规则利用记录中其他item的出现情况预测新item的出现
- **应用：**
  - 购物篮分析、医疗诊断、气象预测等、捆绑销售等

概述-样例

市场购物篮交易数据

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

{Diaper} → {Beer},
{Milk, Bread} → {Eggs,Coke},
{Beer, Bread} → {Milk},



概述-购物篮分析

- 如果问题的全域是商店中所有商品的集合，则对每种商品都可以用一个布尔量来表示该商品是否被顾客购买，则每个购物篮都可以用一个布尔向量表示（0001001100）；而通过分析布尔向量则可以得到商品被频繁关联或被同时购买的模式，这些模式就可以用关联规则表示（布尔量表示丢失了什么信息?）

概述-关联规则：定义

- **给定：**
  - 项集：$I=\{i_1,i_2,\ldots,i_n\}$
  - 任务相关数据D 是数据库事务的集合，每个事务T 则是项的集合，使得 $T \subseteq I$
  - 每个事务由事务标识符TID标识；
  - X,Y 为两个项集，事务T 包含X当且仅当 $X \subseteq T$
- **则关联规则是如下蕴涵式：**

$$X \Rightarrow Y \ [s,\ c]$$

  - 其中 $X \subset I,\ Y \subset I$ 并且$X \cap Y = \Phi$，规则$X \Rightarrow Y$ 在事务集D中成立，并且具有支持度s和置信度c

概述-关联规则：定义

- 规则评价度量
  - 支持度
    - X 和 Y同时在一条记录中出现的比率
  - 置信度
    - 度量X出现的情况下，Y出现的比率

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example:

$$\{\text{Milk, Diaper}\} \Rightarrow \text{Beer}$$

$$s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|\text{T}|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67$$

概述-关联规则分析的线路图

- **两步方法:**
  1. 生成频繁项集
     - 产生所有support ≥ minsup的项集
  2. 生成规则
     - 从每个频繁项集中生成高置信度的规则, 其中每条规则都是一个频繁项集的二项划分
- **生成频繁项集是时间花费最大的操作**

概述-关联规则

- **关联规则算法的目标是在给定的交易数据集T中侦测具有相关性的规则**
  - 支持度 ≥ *minsup* threshold
  - 置信度 ≥ *minconf* threshold
- **Brute-force approach:**
  - 列出所有的关联规则
  - 计算每个规则的支持度和置信度
  - 将支持度和置信度小于阈值的规则删除

4.1 概述-挖掘关联规则

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Rules:

{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

Observations:

- 上述所有规则都是同一item集的二项划分:
  {Milk, Diaper, Beer}
- 来自同一item集的规则拥有相同的支持度，但拥有不同的置信度
- 因此我们需要独立的看待支持度和置信度

概述-关联规则的分类

- 根据规则中所处理的值类型
  - 布尔关联规则
  - 量化关联规则

$$age(X,"30...39") \wedge income(X,"42k...48k") \Rightarrow buys(X,"computer")$$

- 根据规则中设计的数据维
  - 单维关联规则
  - 多维关联规则

$$buys(X,"computer") \Rightarrow buys(X,"software")$$

概述-关联规则的分类

- 根据规则集所涉及的抽象层
  - 单层关联规则
  - 多层关联规则

$$age(X,"30...39") \Rightarrow buys(X,"laptop\_computer")$$
$$age(X,"30...39") \Rightarrow buys(X,"computer")$$

- 根据关联挖掘的各种扩展
  - 挖掘最大的频繁模式（该模式的任何真超模式都是非频繁的）
  - 挖掘频繁闭项集（一个项集c是频繁闭项集，如果不存在其真超集c`，使得每个包含c的事务也包含c`）

概述-由事务数据库挖掘单维布尔关联规则

- 最简单的关联规则挖掘，即单维、单层、布尔关联规则的挖掘。

Transaction ID	Items Bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

最小支持度 50%
最小置信度 50%

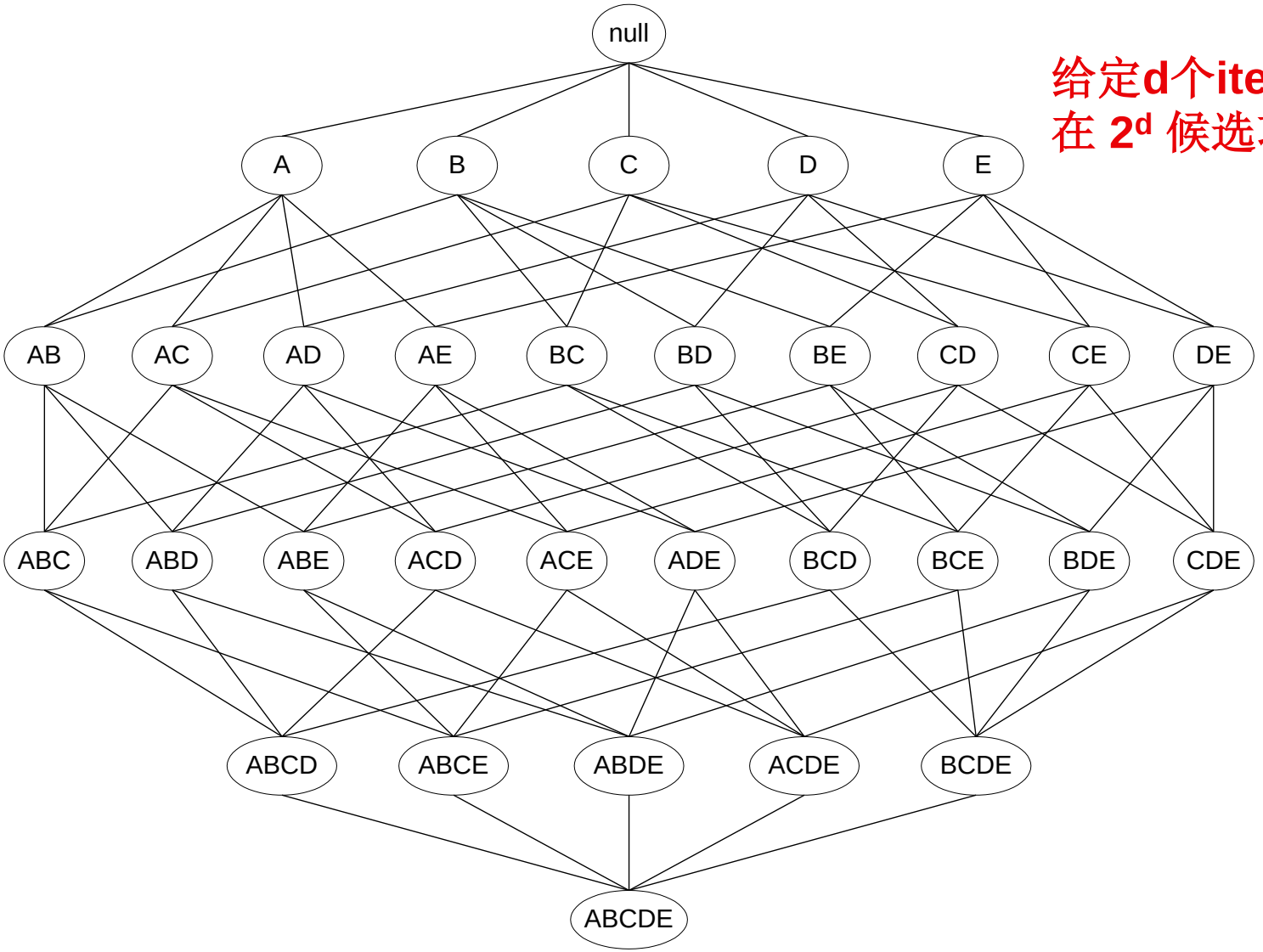
Frequent Itemset	Support
{A}	75%
{B}	50%
{C}	50%
{A,C}	50%

- 对规则$A \Rightarrow C$，其支持度 $\sup port(A \Rightarrow C) = P(A \cup C)$ =50%
- 置信度

$$confidence(A \Rightarrow C) = P(C \mid A) = P(A \cup C) / P(A) = \sup port(A \cup C) / \sup port(A) = 66.6\%$$

Apriori Algorithm

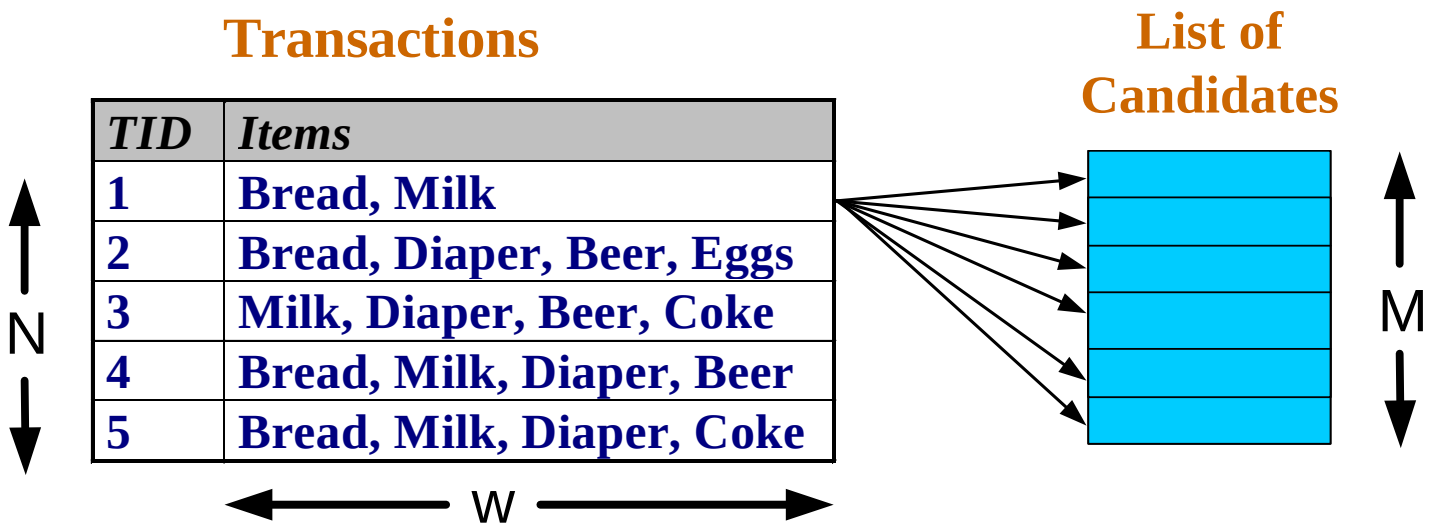
Apriori 算法-生成频繁项集



给定d个item, 可能存在 2^d 候选项集!

4.2 Apriori 算法-生成频繁项集

- **Brute-force 算法:**
  - 蓝色表中每一行都是一个频繁项集的候选项
  - 通过扫描事务数据库并计算每个候选项的支持度



  - 在每条数据记录中匹配每个候选项
  - 复杂度~ O(NMw) => Expensive since $M = 2^d$!!!

Apriori 算法-生成频繁项集的策略

- **减少候选项的数目 (M)**
  - 完全搜索: $M=2^d$
  - 使用剪枝技术来消减M
- **减少交易记录的数量 (N)**
  - 随着项集数目的增加减少交易记录数目N
  - 使用DHP以及垂直型搜索算法
- **减少比较次数(NM)**
  - 使用高效的数据结构来存储候选项集和交易记录
  - 无需匹配每个候选项和交易记录

Apriori 算法-挖掘频繁模式扩展

- **频繁模式的向下闭合特性**
  - 任何频繁模式的子集一定是频繁的
  - 若 **{beer, diaper, nuts}** 是频繁模式, **{beer, diaper}也是频繁模式**
  - i.e., 每个包含 {beer, diaper, nuts}的记录也会包含{beer, diaper}
- **频繁模式挖掘算法:**
  - Apriori (Agrawal & Srikant@VLDB'94)
  - Freq. pattern growth (FPgrowth—Han, Pei & Yin @SIGMOD'00)

Apriori 算法

- **Apriori算法利用频繁项集性质的先验知识（prior knowledge），通过逐层搜的迭索代方法，即将k-项集用于探察(k+1)-项集，来穷尽数据集中的所有频繁项**
  - 先找到频繁1-项集集合L_1,然后用L_1找到频繁2-项集集合L_2，接着用L_2找L_3，直到找不到频繁k-项集，找每个L_k需要一次数据库扫描。
- **Apriori 剪枝原则: 频繁项集的子集是频繁的 (i.e. Anti-monotone) (Agrawal & Srikant @VLDB'94)**
- **Apriori算法是反单调的，即一个集合如果不能通过测试，则该集合的所有超集也不能通过相同的测试。**

Apriori 算法步骤

- Apriori算法由连接和剪枝两个步骤组成。
- 连接：为了找L_k，通过L_{k-1}与自己连接产生候选k-项集的集合，该*候选k项集*记为C_k。
  - L_{k-1}中的两个元素L1和L2可以执行连接操作 $l_1 \triangleright\triangleleft l_2$ 的条件是

$$(l_1[1]=l_2[1]) \wedge (l_1[2]=l_2[2]) \wedge \ldots \wedge (l_1[k-2]=l_2[k-2]) \wedge (l_1[k-1]<l_2[k-1])$$

- C_k是L_k的超集，即它的成员可能不是频繁的，但是所有频繁的k-项集都在C_k中（为什么？）。因此可以通过扫描数据库，通过计算每个k-项集的支持度来得到L_k 。
  - 为了减少计算量，可以使用*Apriori*性质，即如果一个k-项集的$(k-1)$-子集不在L_{k-1}中，则该候选不可能是频繁的，可以直接从C_k删除。

Apriori 算法的例子

$Sup_{min} = 2$

Database TDB

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

1st scan

C_1

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

L_1

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

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan

C_2

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

L_2

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

C_3

Itemset
{B, C, E}

3rd scan

L_3

Itemset	sup
{B, C, E}	2

Apriori 算法-使用Apiori性质由L_2产生C_3

- **连接：**
  - C3=L_2　　L_2= {{A,C},{B,C},{B,E}{C,E}}　　{{A,C},{B,C},{B,E}{C,E}} = {{A,B,C},{A,C,E},{B,C,E}}
- **使用Apriori性质剪枝：频繁项集的所有子集必须是频繁的，对候选项C_3，我们可以删除其子集为非频繁的选项：**
  - {A,B,C}的2项子集是{A,B},{A,C},{B,C}，其中{A,B}不是L_2的元素，所以删除这个选项；
  - {A,C,E}的2项子集是{A,C},{A,E},{C,E}，其中{A,E} 不是L_2的元素，所以删除这个选项；
  - {B,C,E}的2项子集是{B,C},{B,E},{C,E}，它的所有2－项子集都是L_2的元素，因此保留这个选项。
- **这样，剪枝后得到C_3={{B,C,E}}**

Apriori 算法- 足够快吗?

- **Apriori算法核心:**
  - 使用 频繁$(k-1)$-项集 产生频繁 *k-项集的候选集*
  - 使用数据库扫描和模式匹配来计算候选集的出现频率
- ***Apriori算法的瓶颈*: 候选集的产生**
  - 庞大的候选集:
    - 10^4 频繁 1-项集将产生 10^7 候选 2-itemsets
    - 为发现长度为100的频繁模式, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, 需要生产 $2^{100} \approx 10^{30}$ 候选项.
  - 多次扫描数据库:
    - 需要$(n+1)$ 次扫描, n 为最大模式的长度

Apriori 算法-改进算法效率的方法

- 事务/记录压缩（压缩进一步迭代的事务数）
  - 不包含任何k-项集的事务不可能包含任何(k+1)-项集，这种事务在下一步的计算中可以加上标记或删除。
- 划分: DB中的任何频繁项集必须作为局部频繁项集至少出现在DB的一个部分中
- 基于hash表的项集计数: 对应哈希桶计数低于阈值的*k*-项集也不是频繁项集

Apriori 算法-改进算法效率的方法

- 采样:选择原始数据的一个样本，在这个样本上用Apriori算法挖掘频繁模式
- 动态项集计数:在扫描的不同点添加候选项集，这样，如果一个候选项集已经满足最少支持度，则在可以直接将它添加到频繁项集，而不必在这次扫描的以后对比中继续计算

Apriori 算法-划分

- **DB中的任何频繁项集必须作为局部频繁项集至少出现在DB的一个部分中。**
  - 第一次扫描：将数据划分为多个部分并找到局部频繁项集
  - 第二次扫描：评估每个候选项集的实际支持度，以确定全局频繁项集
- **A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association in large databases. In *VLDB'95***

Apriori 算法- DHP算法: 减少候选集的数目

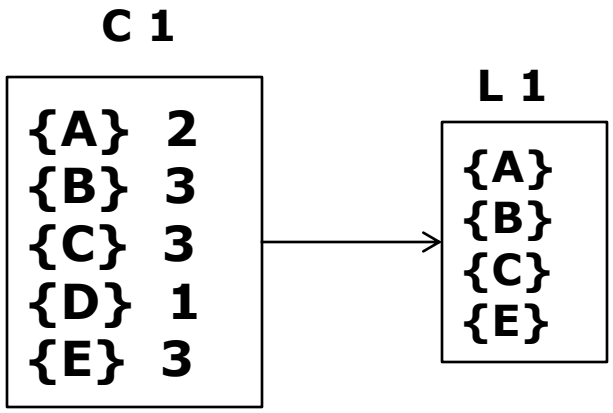
- **J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD'95***
- ***目标：*将每个项集通过相应的hash函数映射到hash表中的不同的桶中，这样可以通过将桶中的项集计数跟最小支持计数相比较先淘汰一部分项集.**

Apriori 算法- DHP算法: 减少候选集的数目

- ***DHP 和Apriori 的主要不同之处在于产生频繁K项集的过程, DHP 处理过程如下:***
  - Step1:产生每个事务的k项集, 将它们映射到一个哈希表中的桶内, 将这个桶的计数加1.
  - Step2:对应哈希桶计数低于阈值的k-项集不是频繁项集，将会从候选集中删除

Apriori 算法- DHP算法

- **Example:**
  - *Step1:*



Data Base

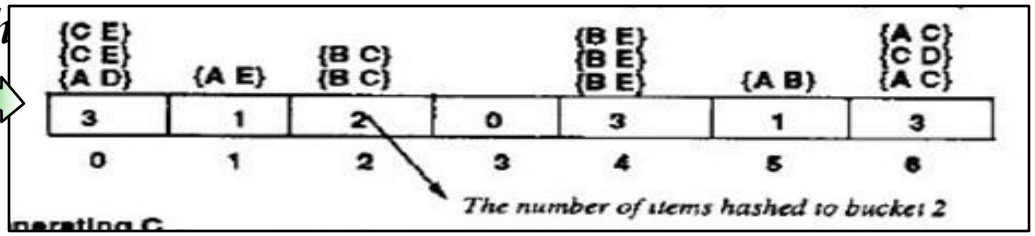
TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

Apriori 算法- DHP算法

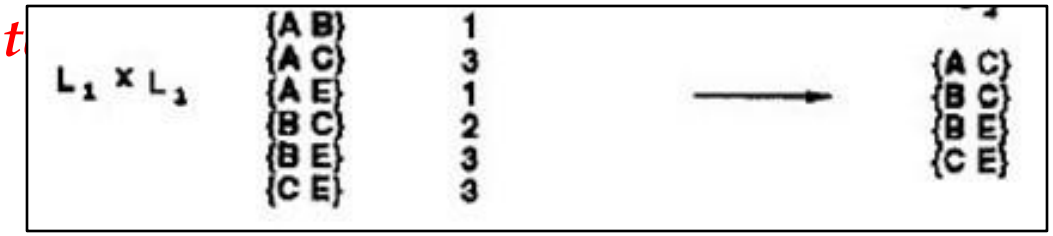
- *Making a hash table*

100{A C},{A D},{C D}
200{B C},{B E},{C E}
300{A B},{A C},{A E},{B C},{B E},{C E}
400{B E}

- *h*



St



TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

Apriori 算法-频繁模式抽样

- **基本思想：选择原始数据的一个样本，在这个样本上用Apriori算法挖掘频繁模式**
- **通过牺牲精确度来减少算法开销，为了提高效率，样本大小应该以可以放在内存中为宜，可以适当降低最小支持度来减少遗漏的频繁模式**
  - 可以通过一次全局扫描来验证从样本中发现的模式
  - 可以通过第二此全局扫描来找到遗漏的模式
- **H. Toivonen. Sampling large databases for association rules. In *VLDB'96***

Apriori 算法- DIC: 动态项集统计

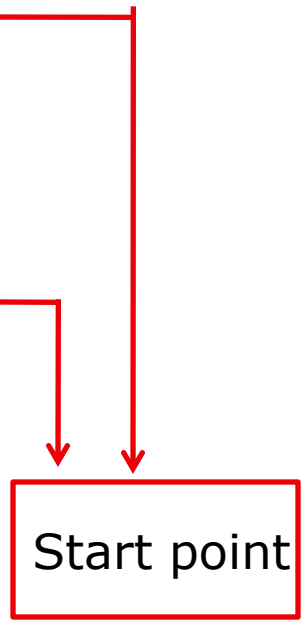
- **S. Brin R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD'97***
- **DIC:** 将数据库分成不同的块，并且用开始点来定位块。新的候选项集能在任何开始点之后被产生
- **Apriori:** 新的候选项及仅在扫描全库后产生
- **DIC 扫描数据库的次数少于 Apriori.**

Apriori 算法- DIC: 减少扫描次数

- Example

TID	Items
100	ABC
200	BCD
300	BCD
400	ABC
500	ABC
600	ABC
700	BCD
800	BCD

Block	TID	Items
B1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B2	500	ABC
	600	ABC
	700	BCD
	800	BCD



Min support = 2;

Apriori 算法- DIC: 减少扫描次数

- Example

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

A	1
D	
B	1
C	1

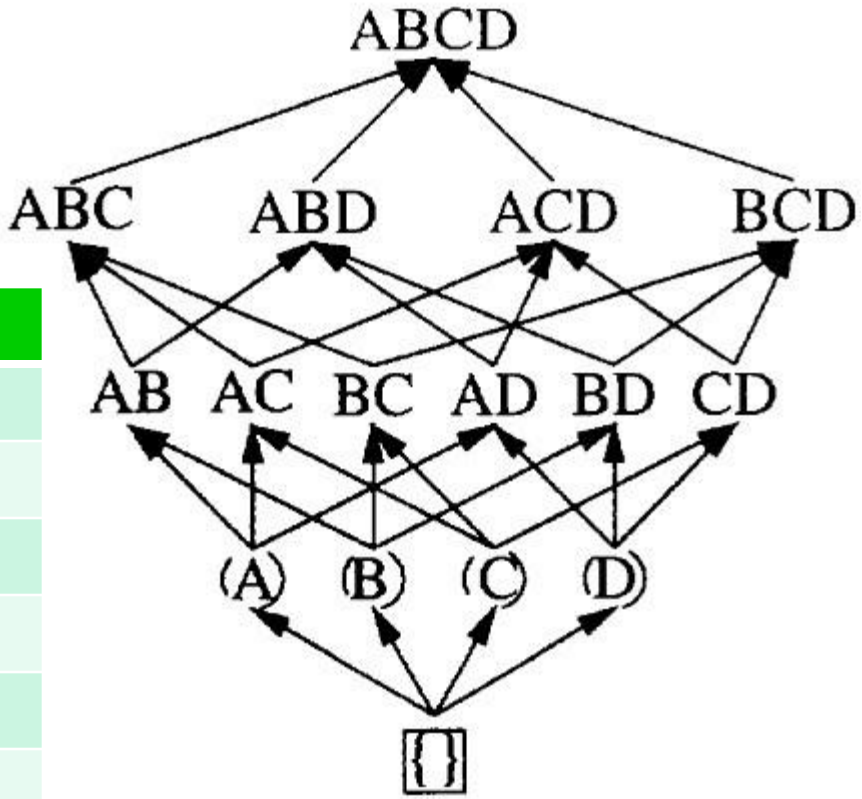


Figure 3: Start of DIC algorithm.

Apriori 算法- DIC: 减少扫描次数

- **Example**

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

A	1
D	1
B	2
C	2

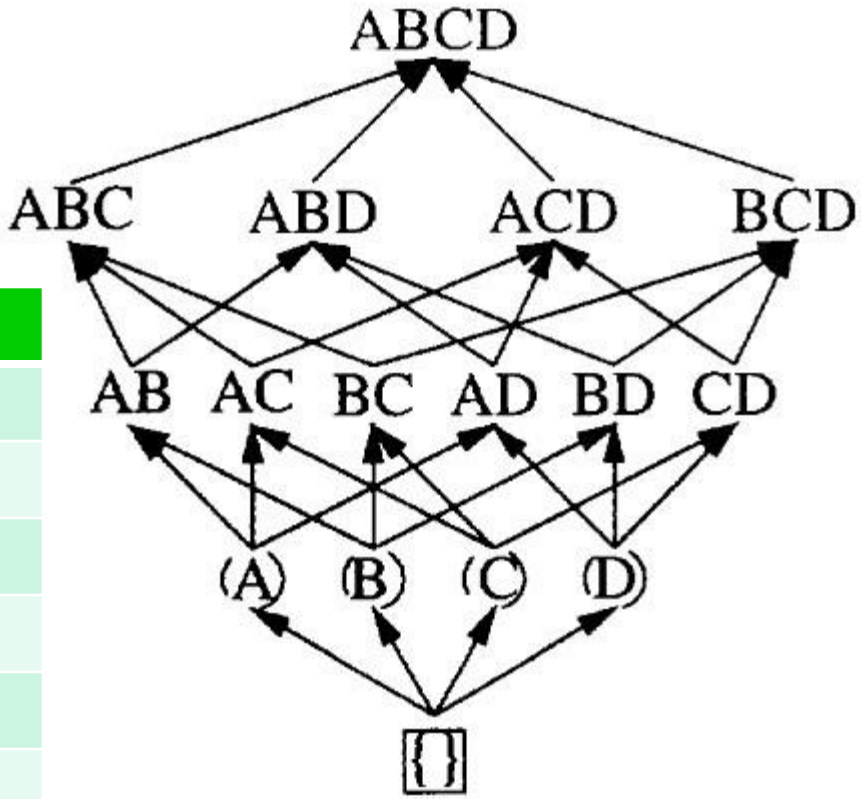


Figure 3: Start of DIC algorithm.

Apriori 算法- DIC: 减少扫描次数

- Example

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

A	1
D	2
B	3
C	3

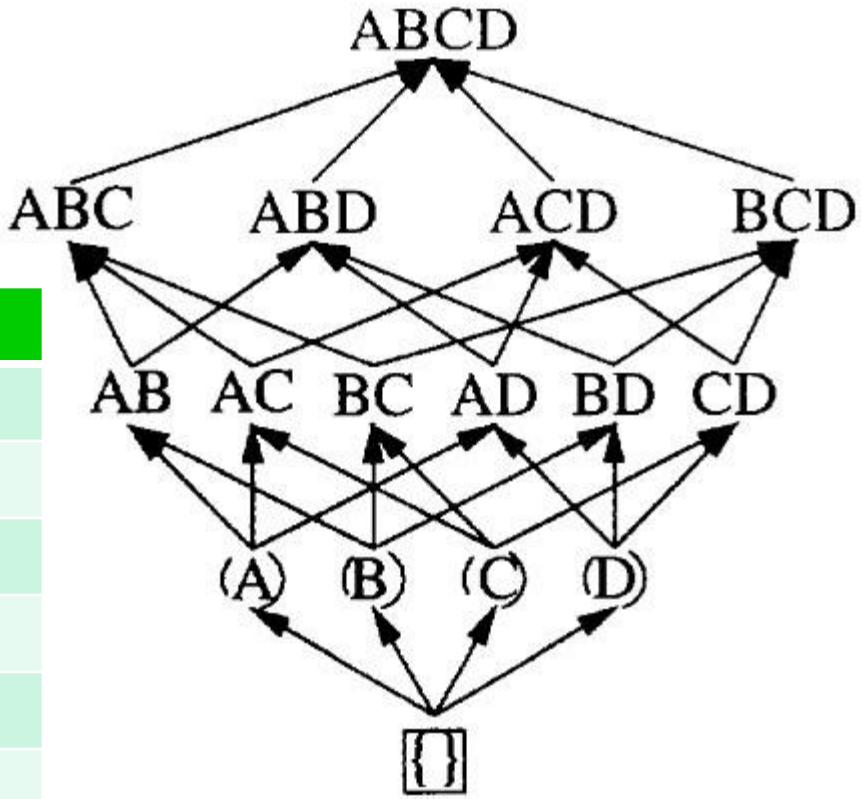


Figure 3: Start of DIC algorithm.

Apriori 算法- DIC: 减少扫描次数

- **Example**

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

A	2
D	2
B	4
C	4

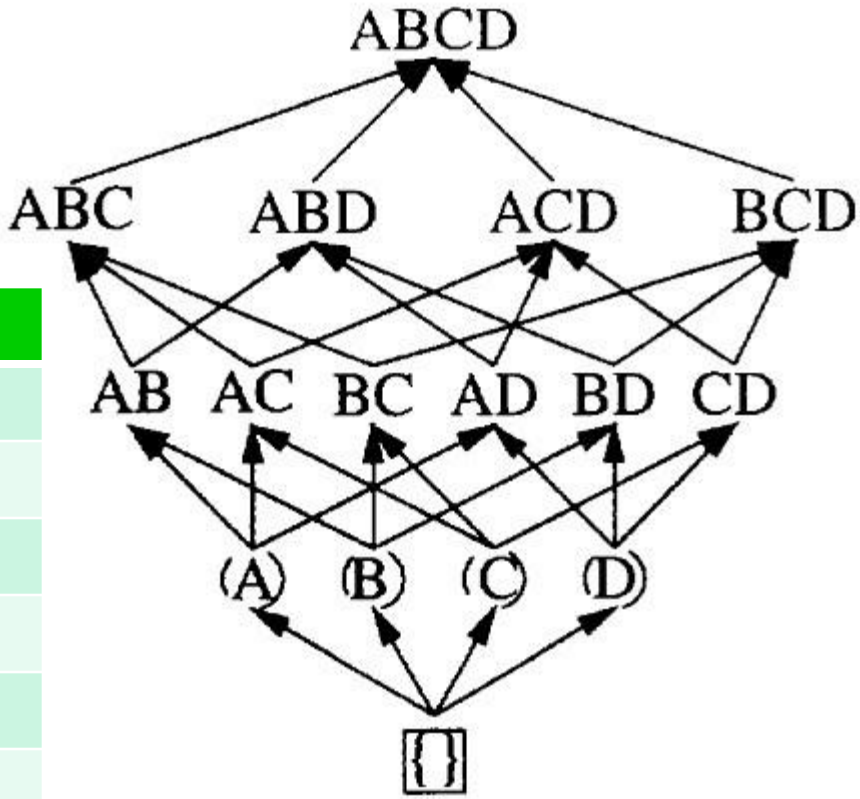


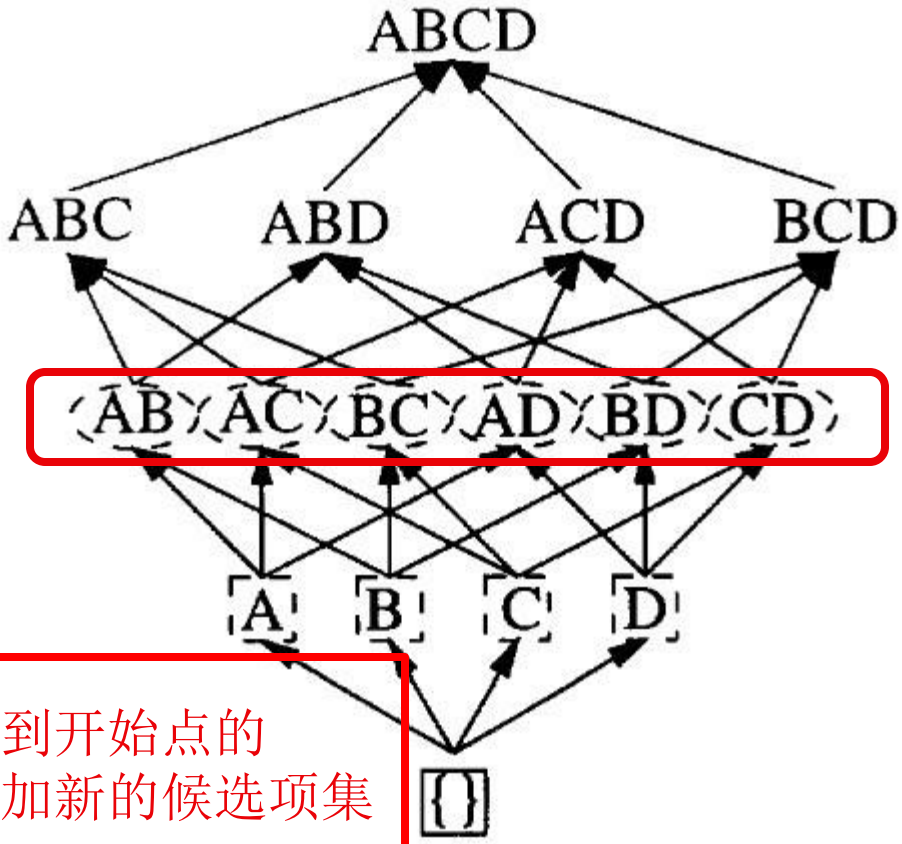
Figure 3: Start of DIC algorithm.

Apriori 算法- DIC: 减少扫描次数

- **Example**

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

A	2
D	2
B	4
C	4
AB	
AC	
BC	
AD	
BD	
CD	



当扫描到开始点的
时候增加新的候选项集

Figure 4: After M transactions.

Apriori 算法- DIC: 减少扫描次数

- Example

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

A	3
D	2
B	5
C	5
AB	1
AC	1
BC	1
AD	
BD	
CD	

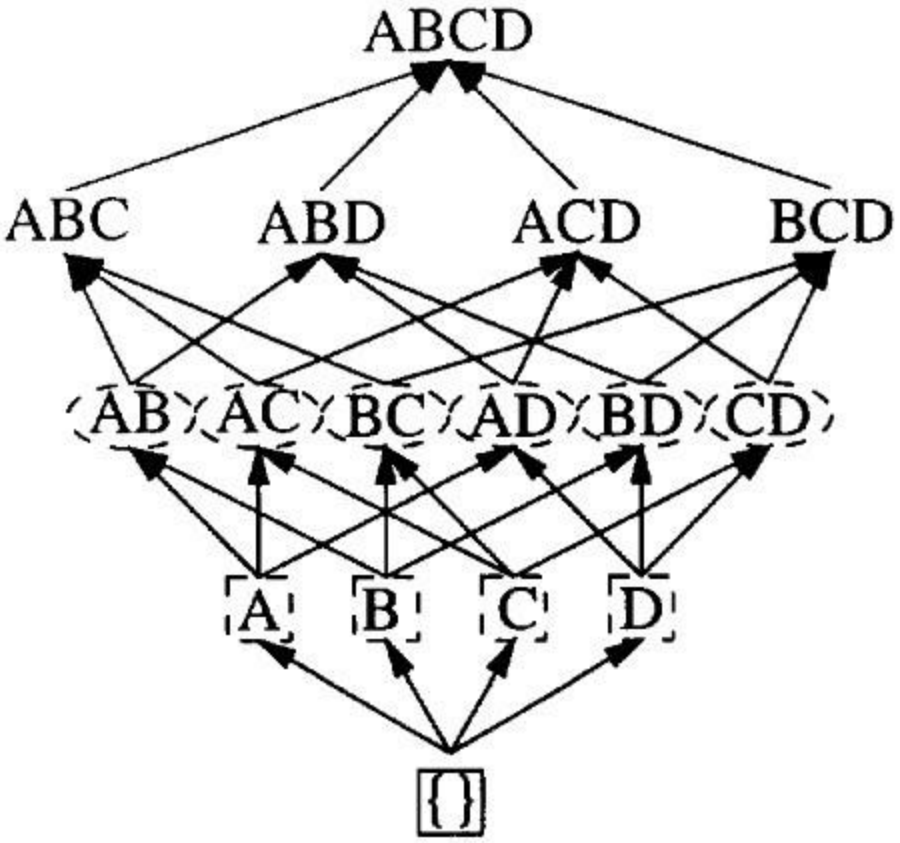


Figure 4: After M transactions.

Apriori 算法- DIC: 减少扫描次数

- Example

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

A	4
D	2
B	6
C	6
AB	2
AC	2
BC	2
AD	
BD	
CD	

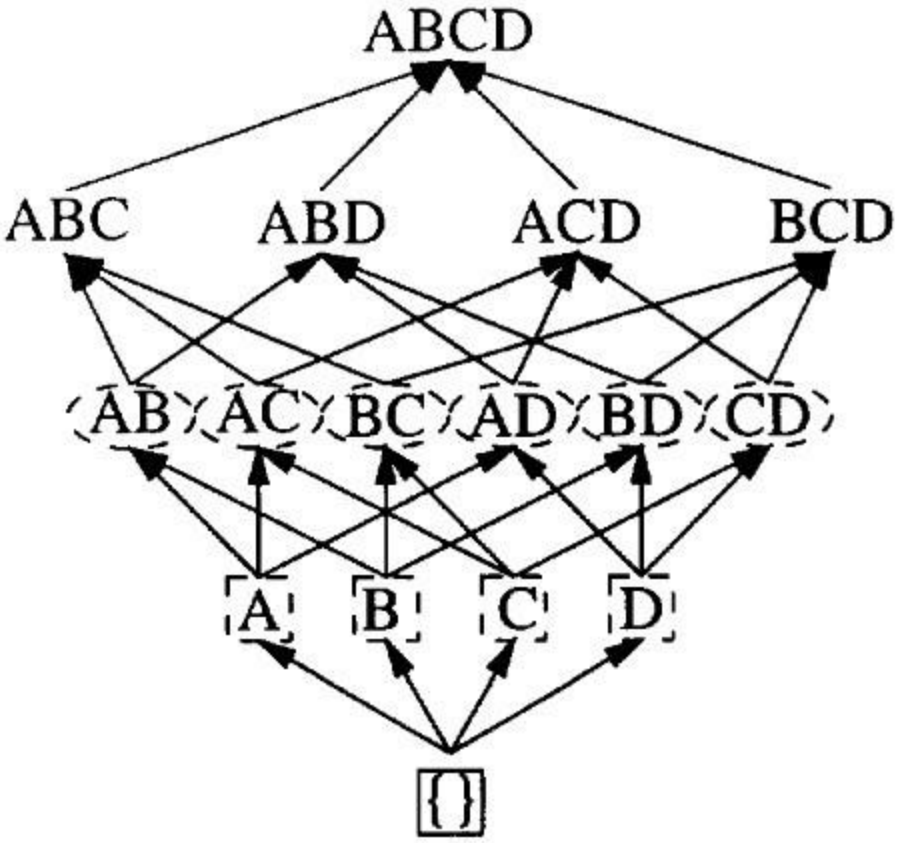


Figure 4: After M transactions.

Apriori 算法- DIC: 减少扫描次数

- Example

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

A	4
D	3
B	7
C	7
AB	2
AC	2
BC	3
AD	
BD	1
CD	1

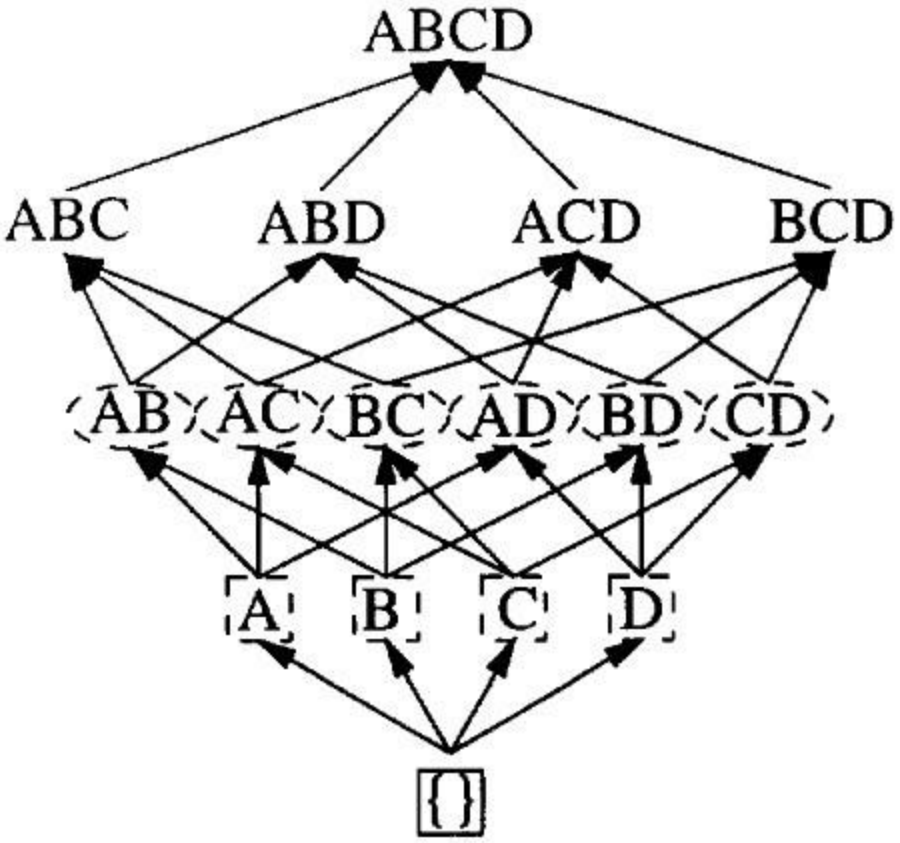


Figure 4: After M transactions.

Apriori 算法- DIC: 减少扫描次数

- Example

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

A	4
D	4
B	8
C	8
AB	2
AC	2
BC	4
AD	
BD	2
CD	2

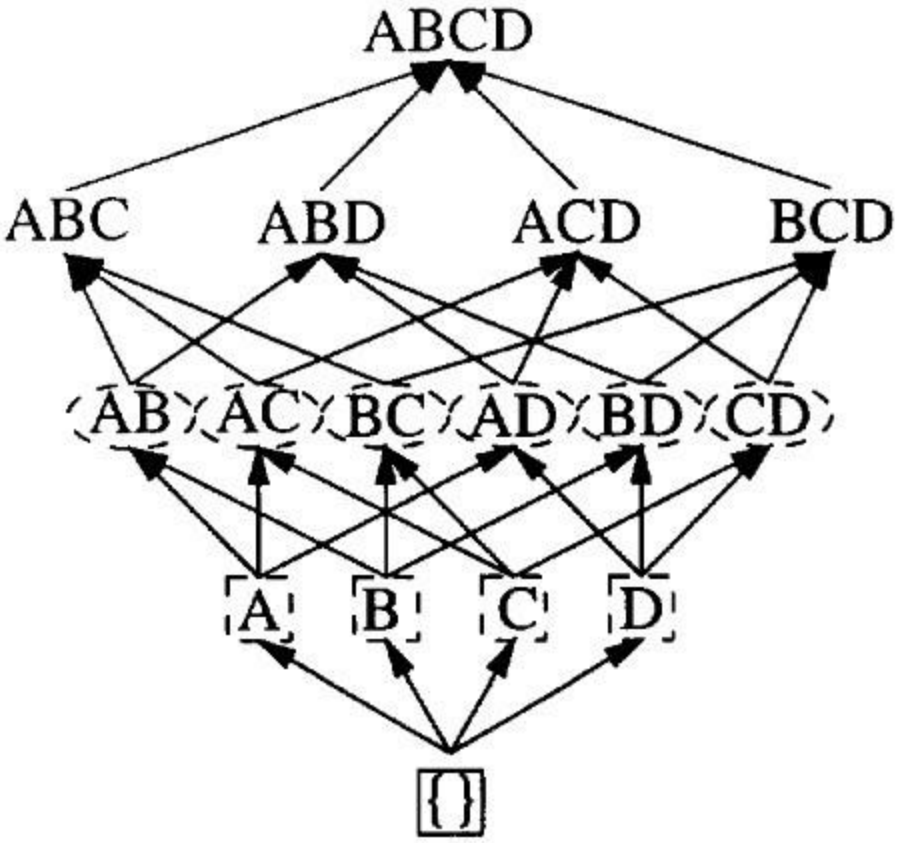


Figure 4: After M transactions.

Apriori 算法- DIC: 减少扫描次数

- Example

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

A	4
D	4
B	8
C	8
AB	2
AC	2
BC	4
AD	
BD	2
CD	2
ABC	
BCD	

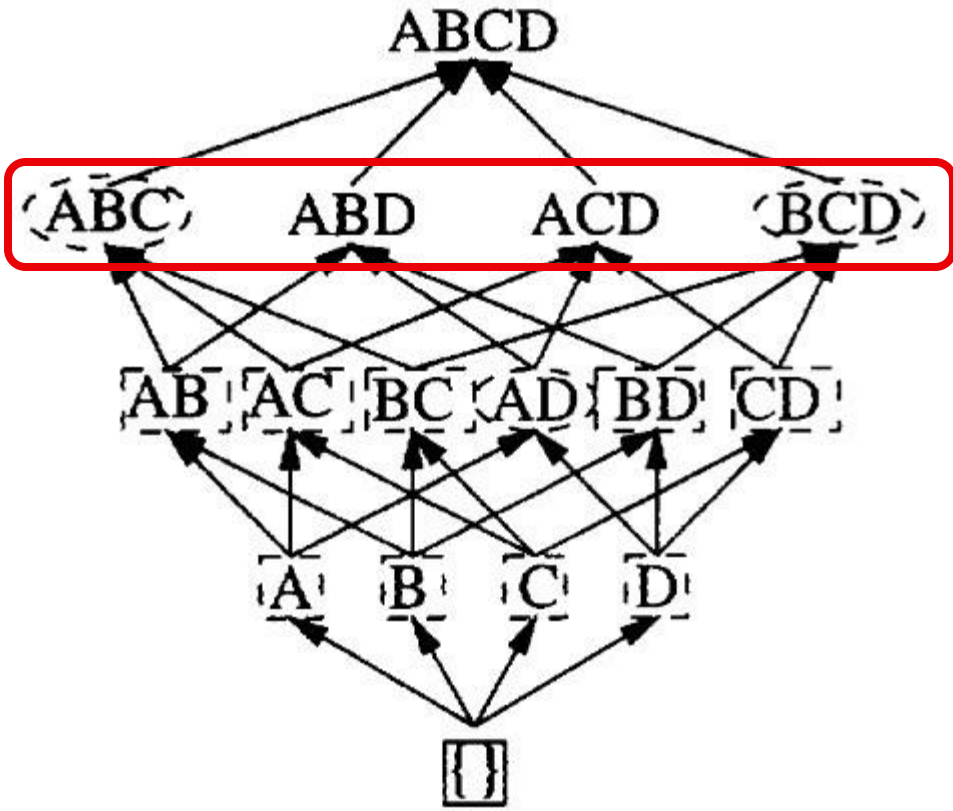


Figure 5: After $2M$ transactions.

Apriori 算法- DIC: 减少扫描次数

- **Example**

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

AB	2
AC	2
BC	4
AD	
BD	2
CD	2
ABC	
BCD	

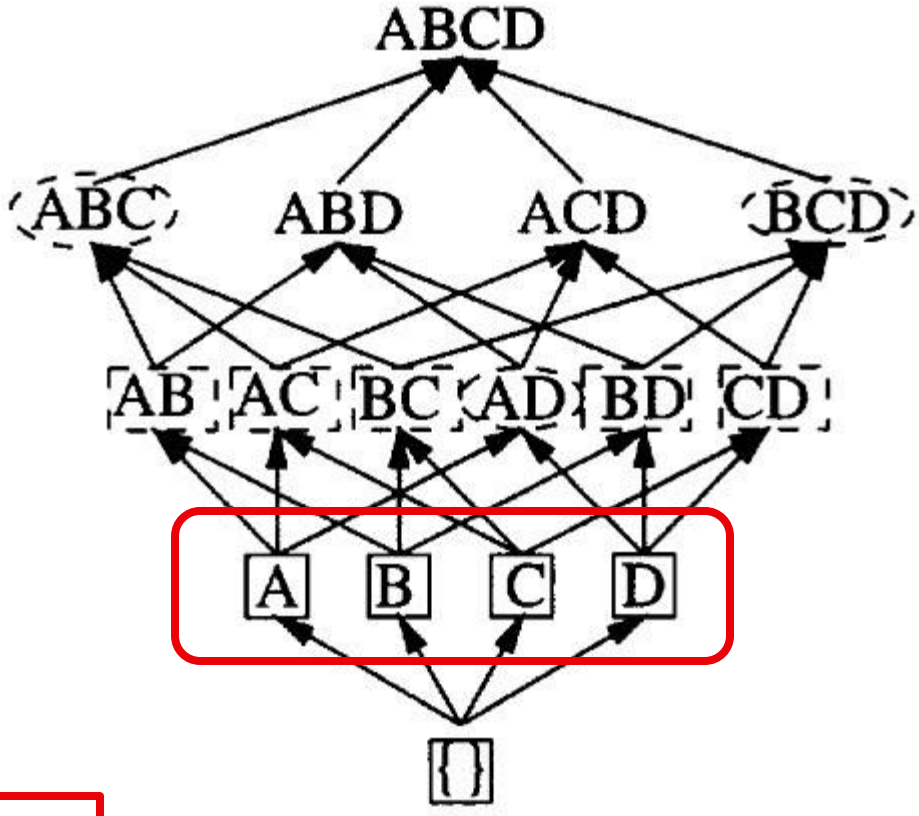


Figure 6: After one pass.

若一个候选项的计数已经超过了最小支持度，那么停止统计这个候选项，并将其加入频繁项集中.

Apriori 算法- DIC: 减少扫描次数

- Example

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

AB	3
AC	3
BC	5
AD	
BD	2
CD	2
ABC	1
BCD	

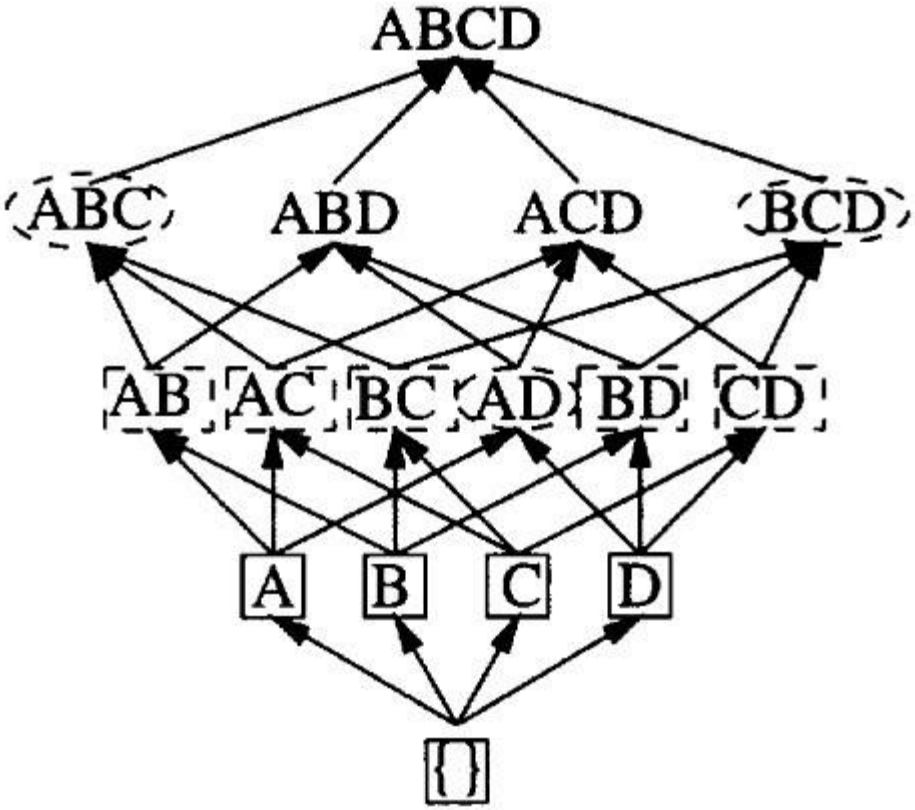


Figure 6: After one pass.

Apriori 算法- DIC: 减少扫描次数

- Example

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

AB	3
AC	3
BC	6
AD	
BD	3
CD	3
ABC	1
BCD	1

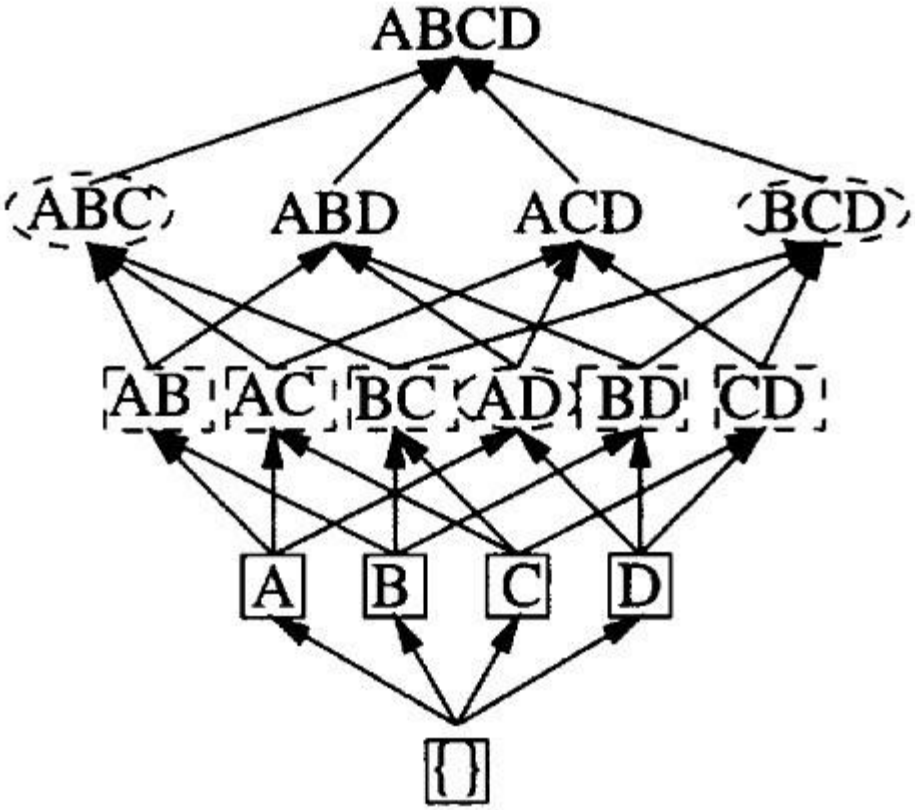


Figure 6: After one pass.

Apriori 算法- DIC: 减少扫描次数

- Example

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

AB	3
AC	3
BC	7
AD	
BD	4
CD	4
ABC	1
BCD	2

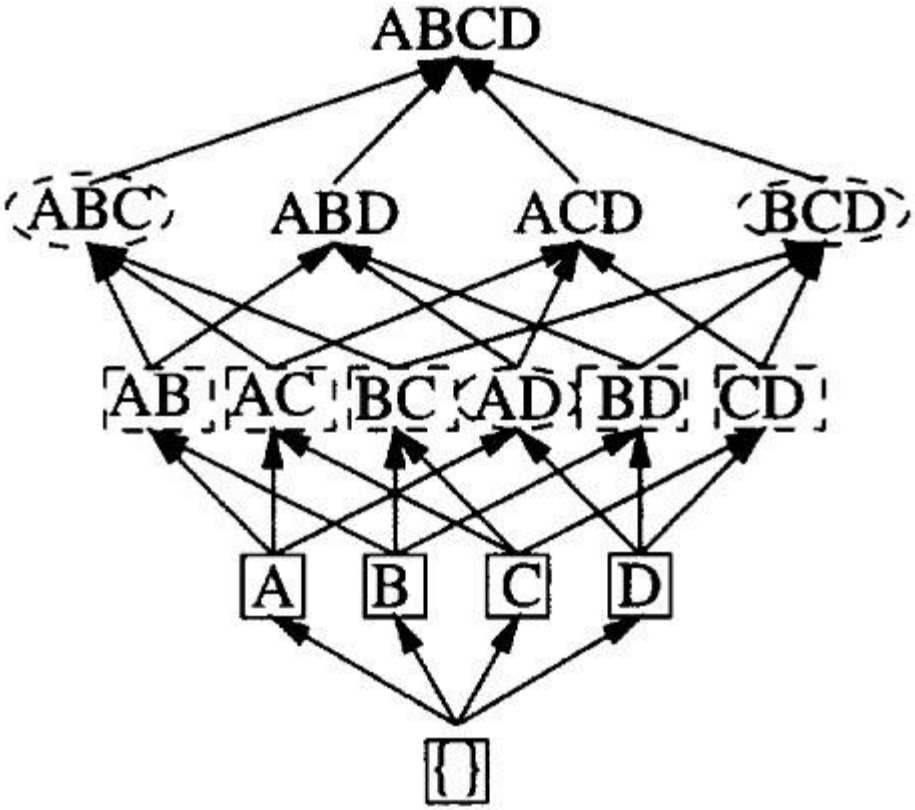


Figure 6: After one pass.

Apriori 算法- DIC: 减少扫描次数

- Example

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

AB	4
AC	4
BC	8
AD	
BD	4
CD	4
ABC	2
BCD	2

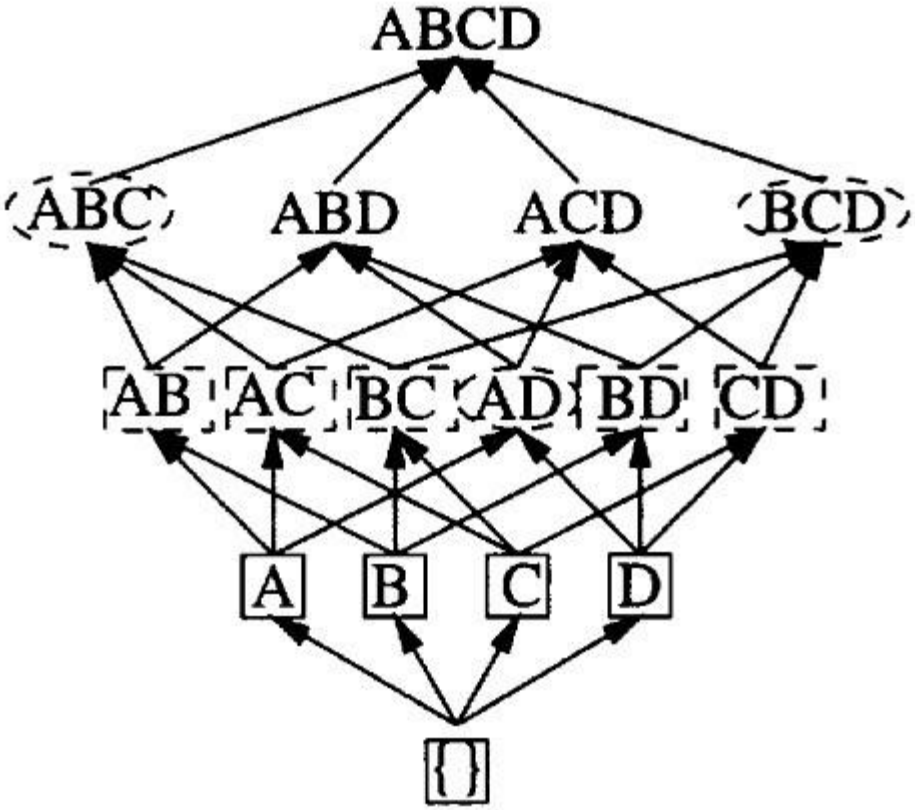


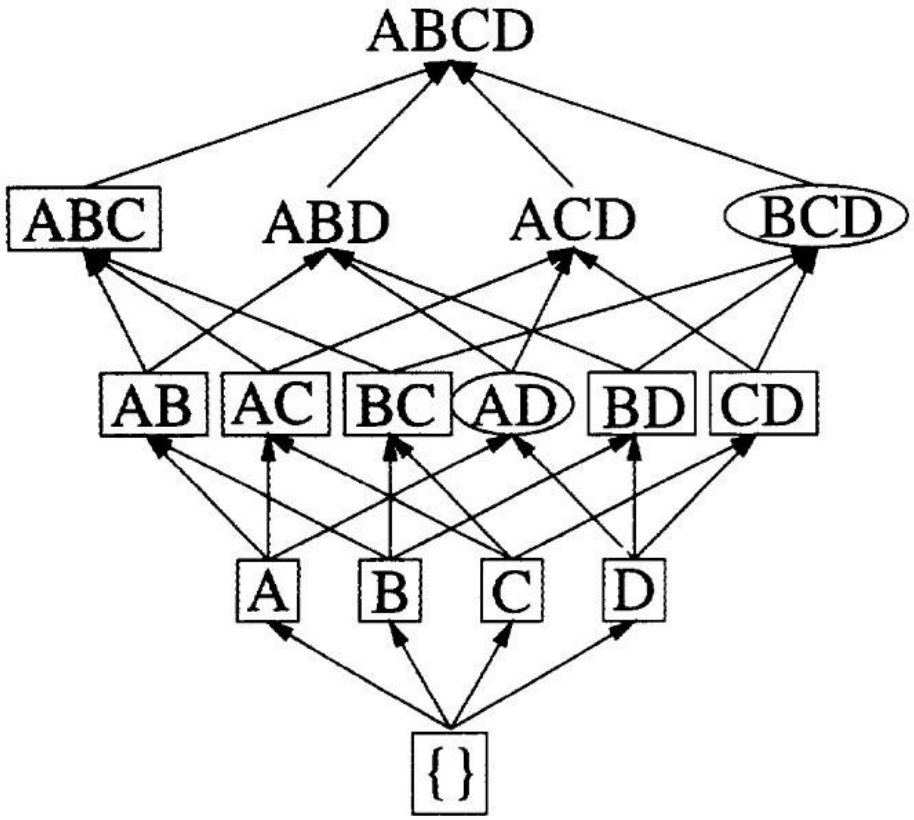
Figure 6: After one pass.

Apriori 算法- DIC: 减少扫描次数

- **Example**

B 1	100	ABC
	200	BCD
	300	BCD
	400	ABC
B 2	500	ABC
	600	ABC
	700	BCD
	800	BCD

AB	4
AC	4
BC	6
AD	
BD	4
CD	4
ABC	2
BCD	2



若一个候选项的计数已经超过了最小支持度，那么停止统计这个候选项，并将其加入频繁项集中

Finish!

Apriori 算法- DIC: 减少扫描次数

- **Example**

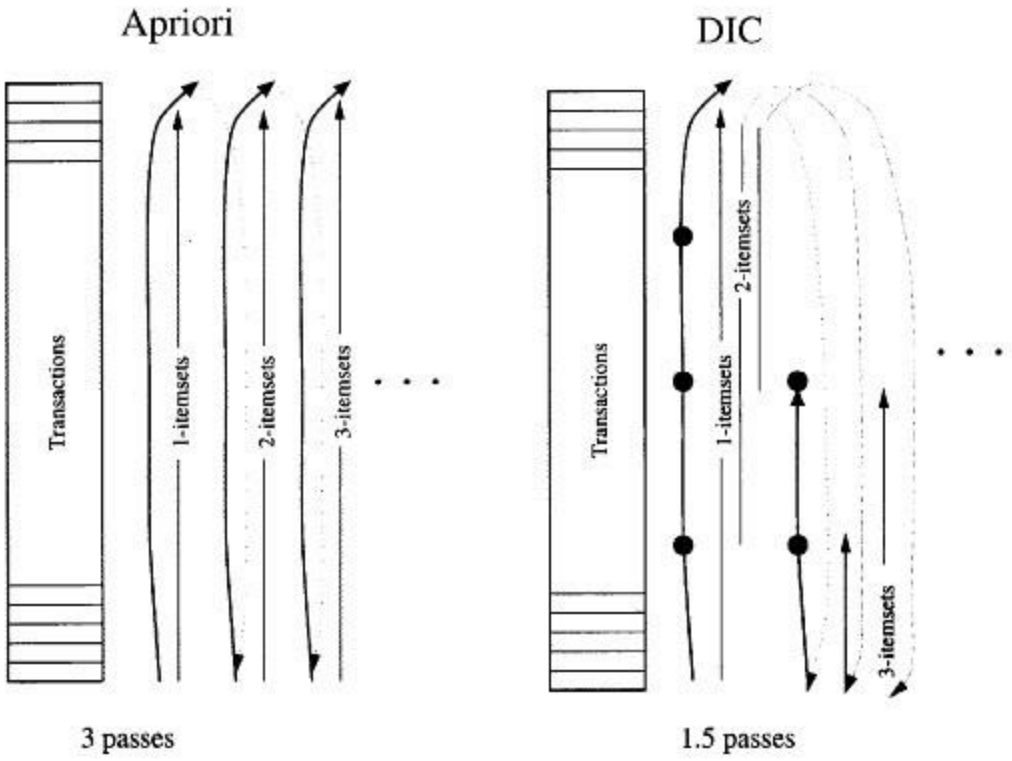
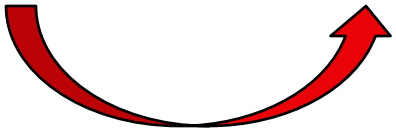


Figure 1: Apriori and DIC

Apriori	DIC
3 round	1.5 round



FP-Tree 算法

FP-Tree 算法-在不产生候选项集的情况下挖掘频繁模式

- **使用局部频繁项，在短模式的基础上生成新的长模式**
  - “abc” 是频繁模式
  - 找到所有包含“abc”的交易记录: DB|abc
  - “d” 是DB|abc中的频繁模式 → abcd 是频繁模式

FP-trees 算法

- **思路: 频繁模式增长**
  - 通过模式及数据库划分，递归的增加频繁模式
- **方法**
  - 对于每个频繁项集构建它的条件数据库, 并构建 它的条件FP-tree
  - 重复处理每个新出现的条件FP-tree
  - 直到FP-tree为空, 或者该tree仅包含一条路径—单路径将产生其子路径的所有组合，其中每一个都是频繁模式

FP-trees 算法-从事务数据库中创建FP-tree

TID	*Items bought*	*(ordered) frequent items*
100	{*f, a, c, d, g, i, m, p*}	{*f, c, a, m, p*}
200	{*a, b, c, f, l, m, o*}	{*f, c, a, b, m*}
300	{*b, f, h, j, o, w*}	{*f, b*}
400	{*b, c, k, s, p*}	{*c, b, p*}
500	{*a, f, c, e, l, p, m, n*}	{*f, c, a, m, p*}

min_support = 3

1. 扫描数据库，找到频繁1项集
2. 降序排列频繁项得到 f-list
3. 再次扫描数据库, 建立FP-tree

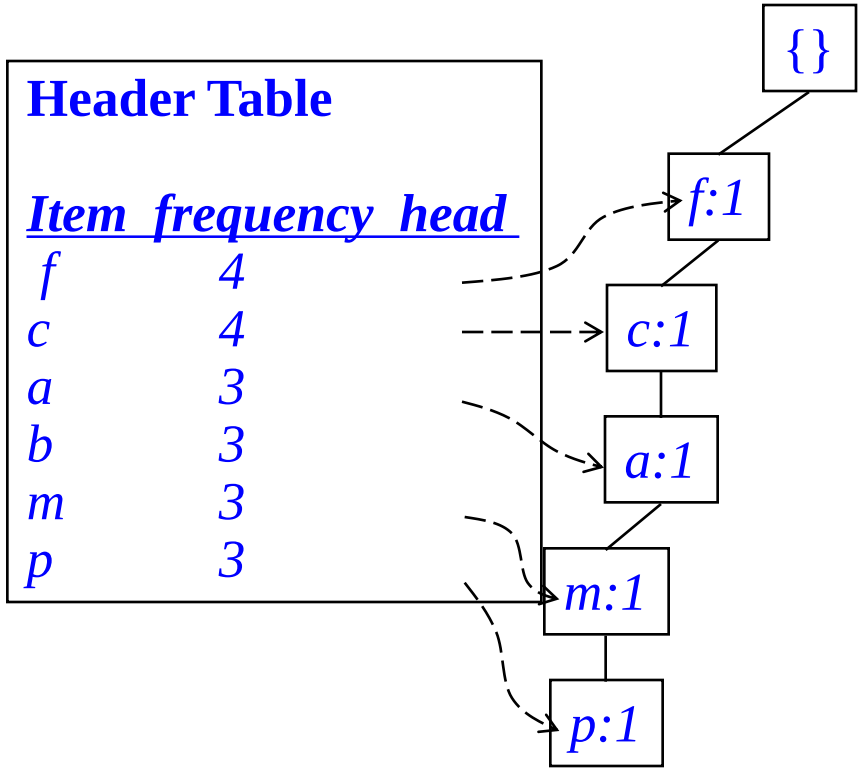
Header Table

Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	

F-list=f-c-a-b-m-p

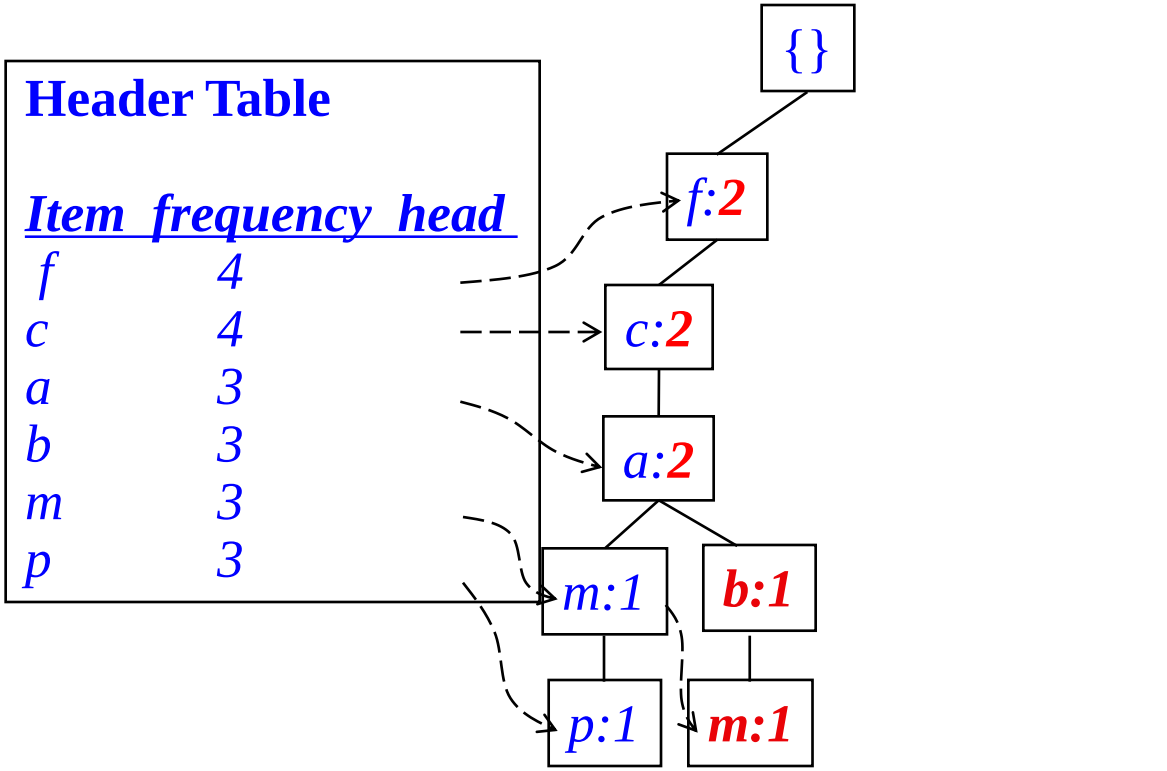
FP-trees 算法-在事务数据库中建立FP-tree

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}



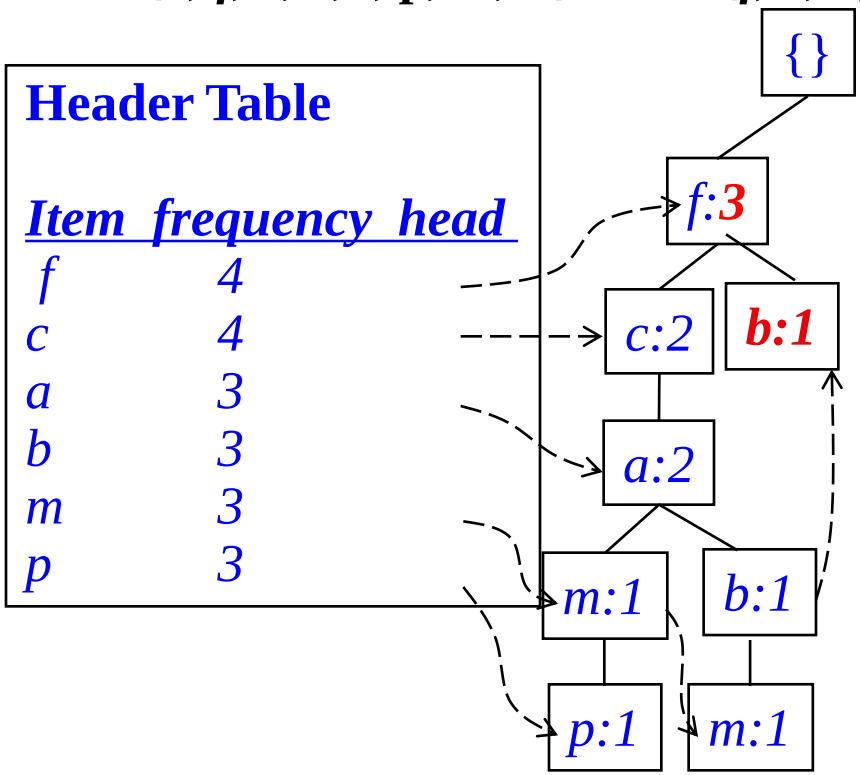
FP-trees 算法-在事务数据库中建立FP-tree

TID	*Items bought*	*(ordered) frequent items*
100	{*f, a, c, d, g, i, m, p*}	{*f, c, a, m, p*}
200	{*a, b, c, f, l, m, o*}	{*f, c, a, b, m*}
300	{*b, f, h, j, o, w*}	{*f, b*}
400	{*b, c, k, s, p*}	{*c, b, p*}
500	{*a, f, c, e, l, p, m, n*}	{*f, c, a, m, p*}



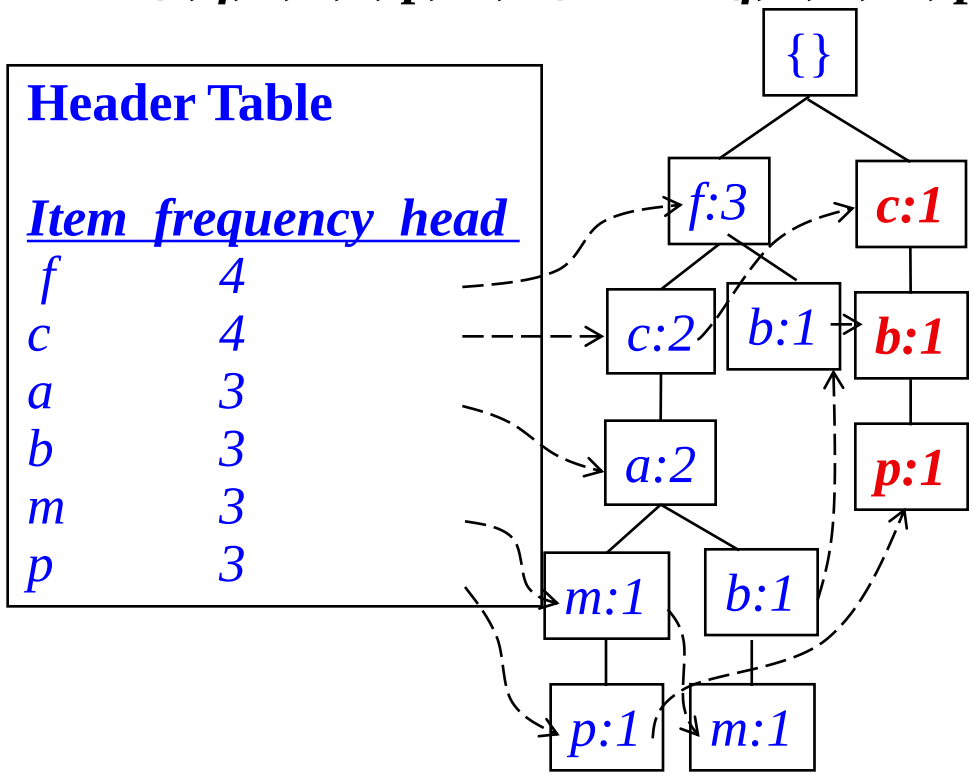
FP-trees 算法-在事务数据库中建立FP-tree

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}



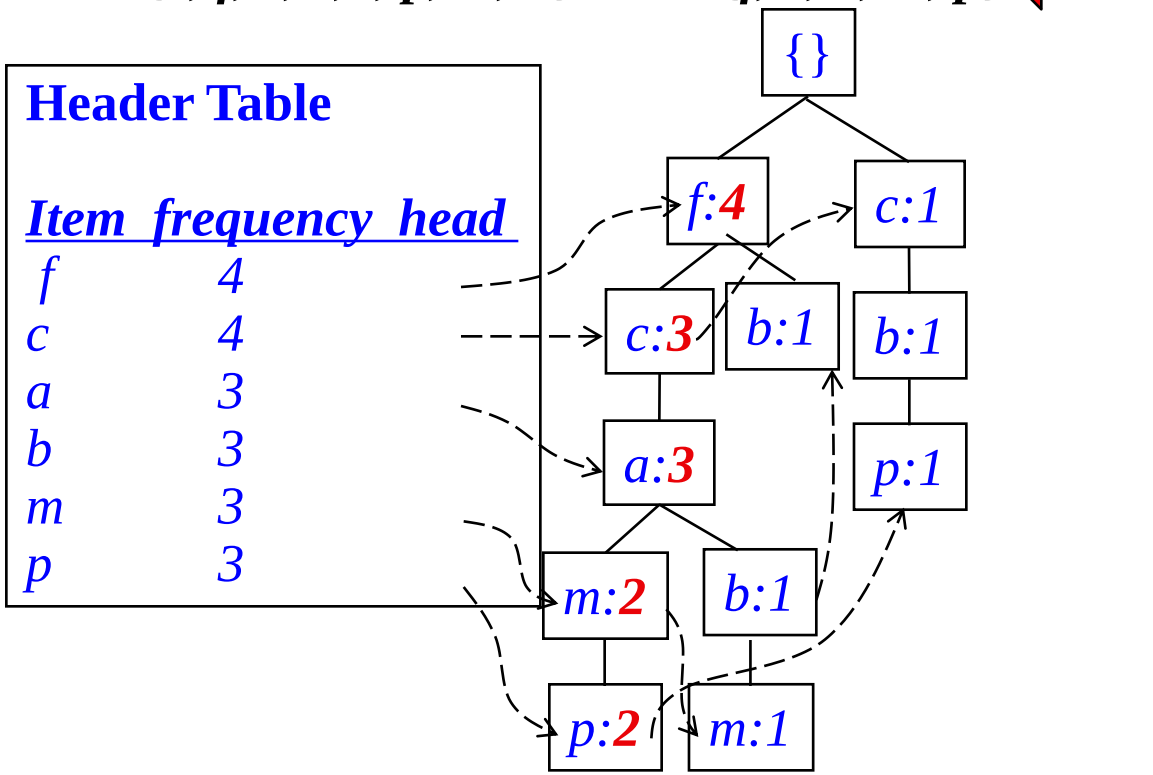
FP-trees 算法-在事务数据库中建立FP-tree

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}



FP-trees 算法-在事务数据库中建立FP-tree

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

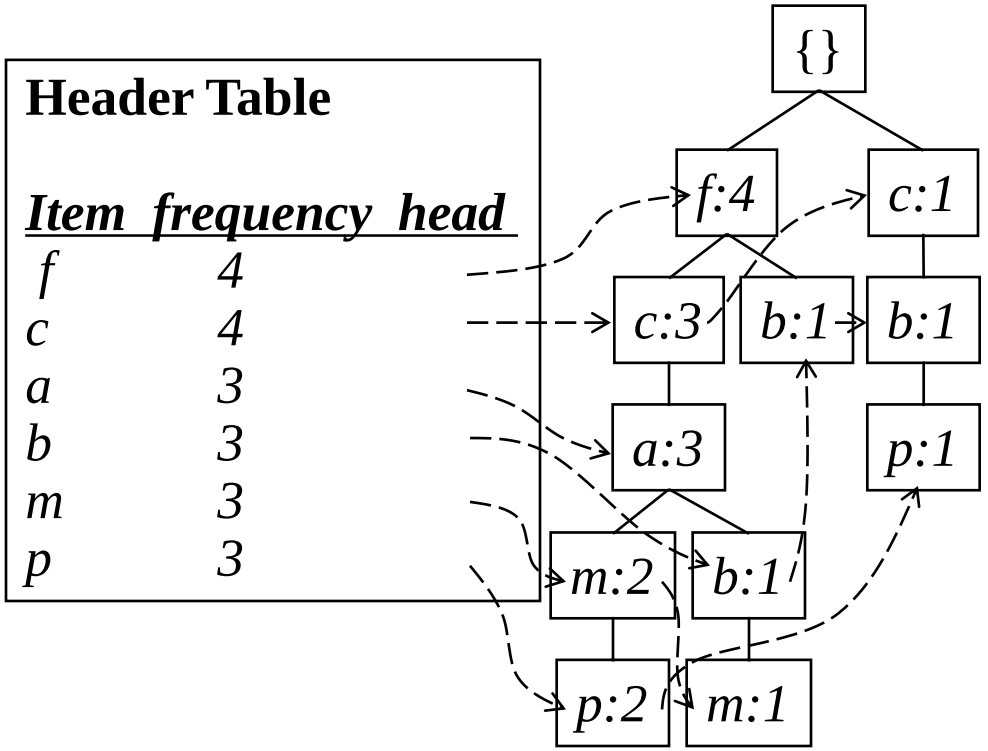


FP-trees 算法结构的优势

- **完全性**
  - 保存完整的频繁模式挖掘信息
  - 能发现所有交易记录中的长模式
- **紧凑型**
  - 减少了不相关信息—不频繁的项很快被删去
  - Items以频繁度降序排列: 出现频率越高, 越可能被共享
  - tree不会比原始数据库大 (不计算节点连接和计数字段)
  - 对于4次连续的DB, 压缩比超过100

FP-trees 算法-从P条件数据库中找到含有P的频繁模式

- 从FP-tree的各个频繁项表处开始
- 跟着频繁项P的链接遍历 FP-tree(P条件数据库)
- 聚集p的所有的转换前缀路径来产生 *p的条件模式库*

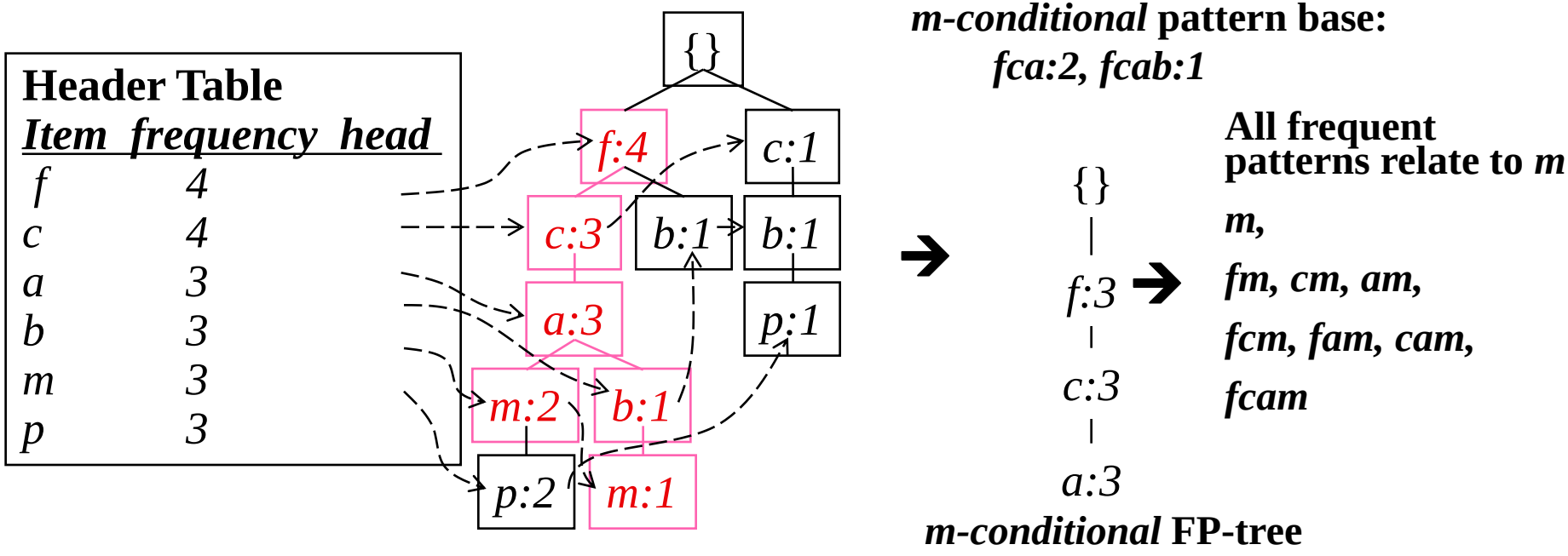


条件模式库

item	*cond. pattern base*
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

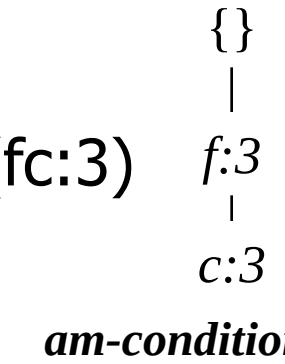
4.3 FP-trees 算法-从条件模式库到条件FP-tree

- 对于每一个模式库
  - 计算模式库中的每个项的数目
  - 构建模式库中每个频繁项的FP-tree



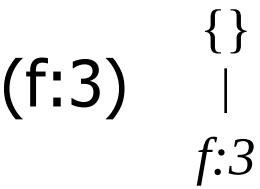
FP-trees 算法-递归: 分析每个 FP-tree

Cond. pattern base of "am": (fc:3)

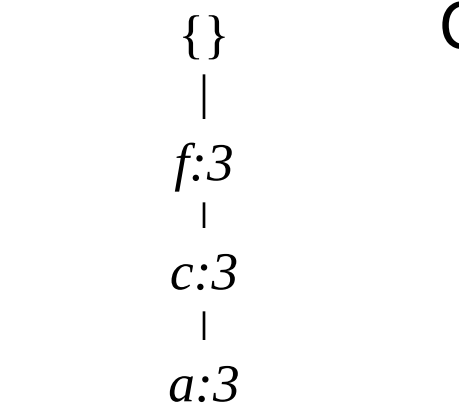


***am-conditional* FP-tree**

Cond. pattern base of "cm": (f:3)



***cm-conditional* FP-tree**



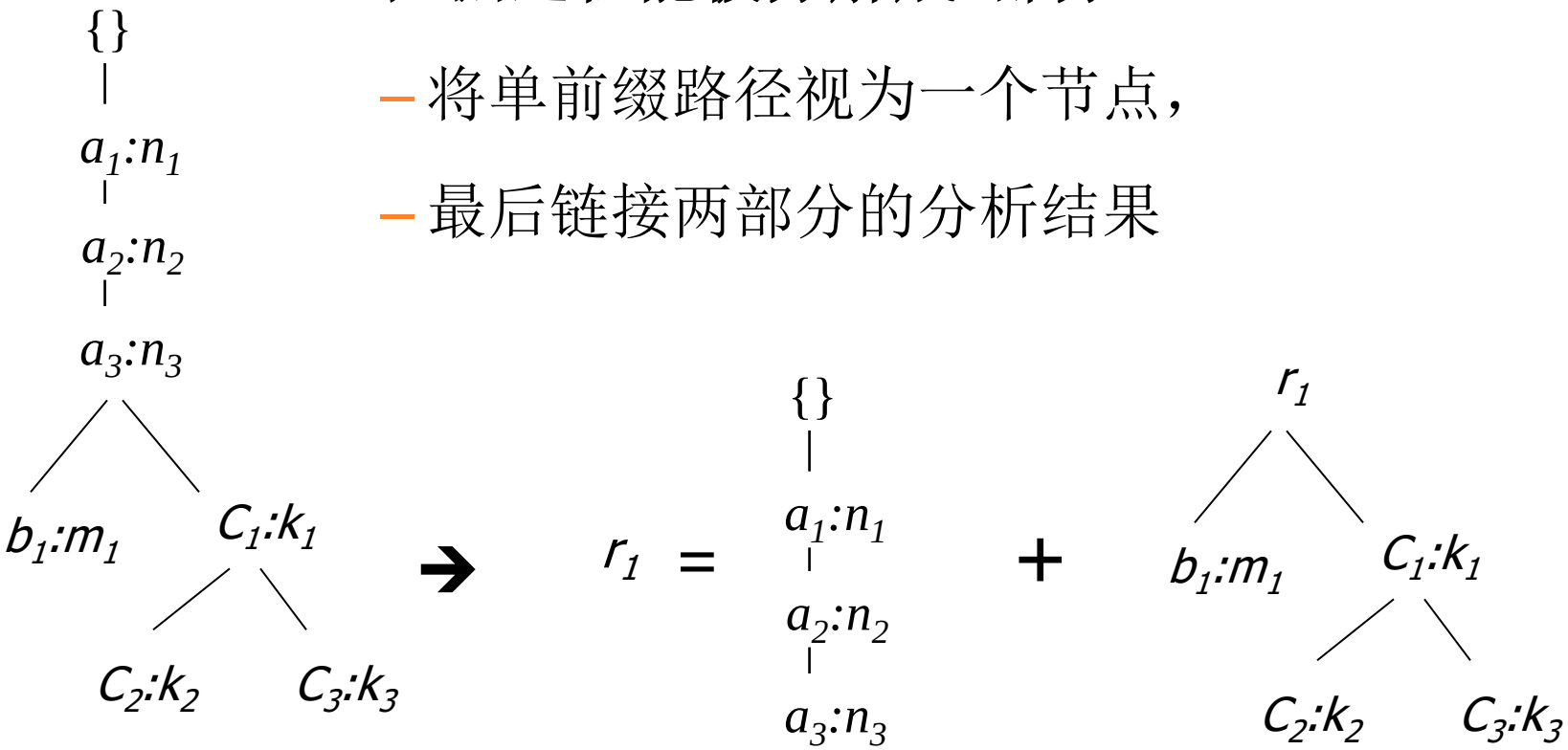
***m-conditional* FP-tree**

Cond. pattern base of "cam": (f:3)

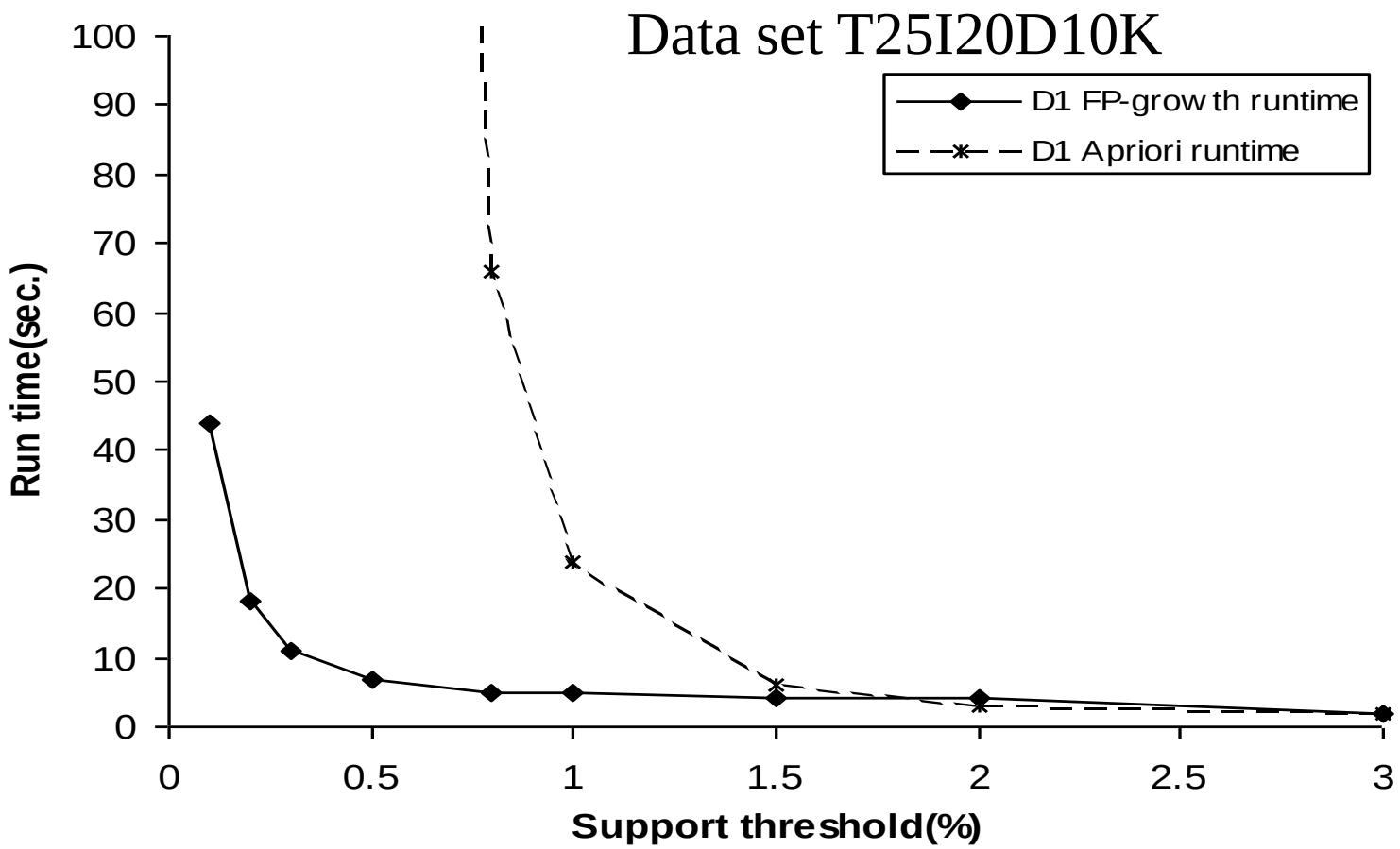
***cam-conditional* FP-tree**

FP-trees 算法-例子：FP-tree中的单前缀路径

- **假设 (条件) FP-tree T 拥有一条单前缀路径 P**
- **挖掘过程能被分解成2部分**
  - 将单前缀路径视为一个节点，
  - 最后链接两部分的分析结果



FP-trees 算法-FP tree vs. Apriori: 随着支持度阈值的增长



FP-trees 算法-为什么FP-Growth运行获胜?

- **分治策略:**
  - 依据已有的频繁模式来分解挖掘工作及DB空间
  - 导致在小数据库内进行搜索
- **其他因素**
  - 无候选项产生, 无候选项的测试
  - 压缩的数据库空间: FP-tree structure
  - 不对整个数据库进行扫描
  - 基本操作—统计局部频繁项 建立子 FP-tree, 无模式搜索和匹配

小结

- **关联规则挖掘：**
  - 从事务数据库，关系数据库和其他信息存储中的大量数据的项集之间发现有趣的、频繁出现的模式、关联和相关性。
- **Apriori算法： 利用频繁项集性质的先验知识，通过逐层搜的迭索代方法来穷尽数据集中的所有频繁项.**
- **FP-Tree算法：没有频繁项集的生成过程，通过模式及数据库划分，递归的增加频繁模式**

结束

2020年10月20日