数据挖掘

关联规则挖掘

2025 年 9 月

南京大学
NANJING UNIVERSITY

目录

定义：关联分析（association analysis）

- 关联分析用于发现隐藏在大型数据集中的令人感兴趣的联系，所发现的模式通常用**关联规则或频繁项集**的形式表示。
- 关联规则反映一个**事物与其它事物**之间的相互依存性和关联性。如果**两个或者多个事物之间**存在一定的关联关系，那么，其中一个事物发生就能够预测与它相关联的其它事物的发生。

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

Rules Discovered:
{Diaper} --> {Beer}

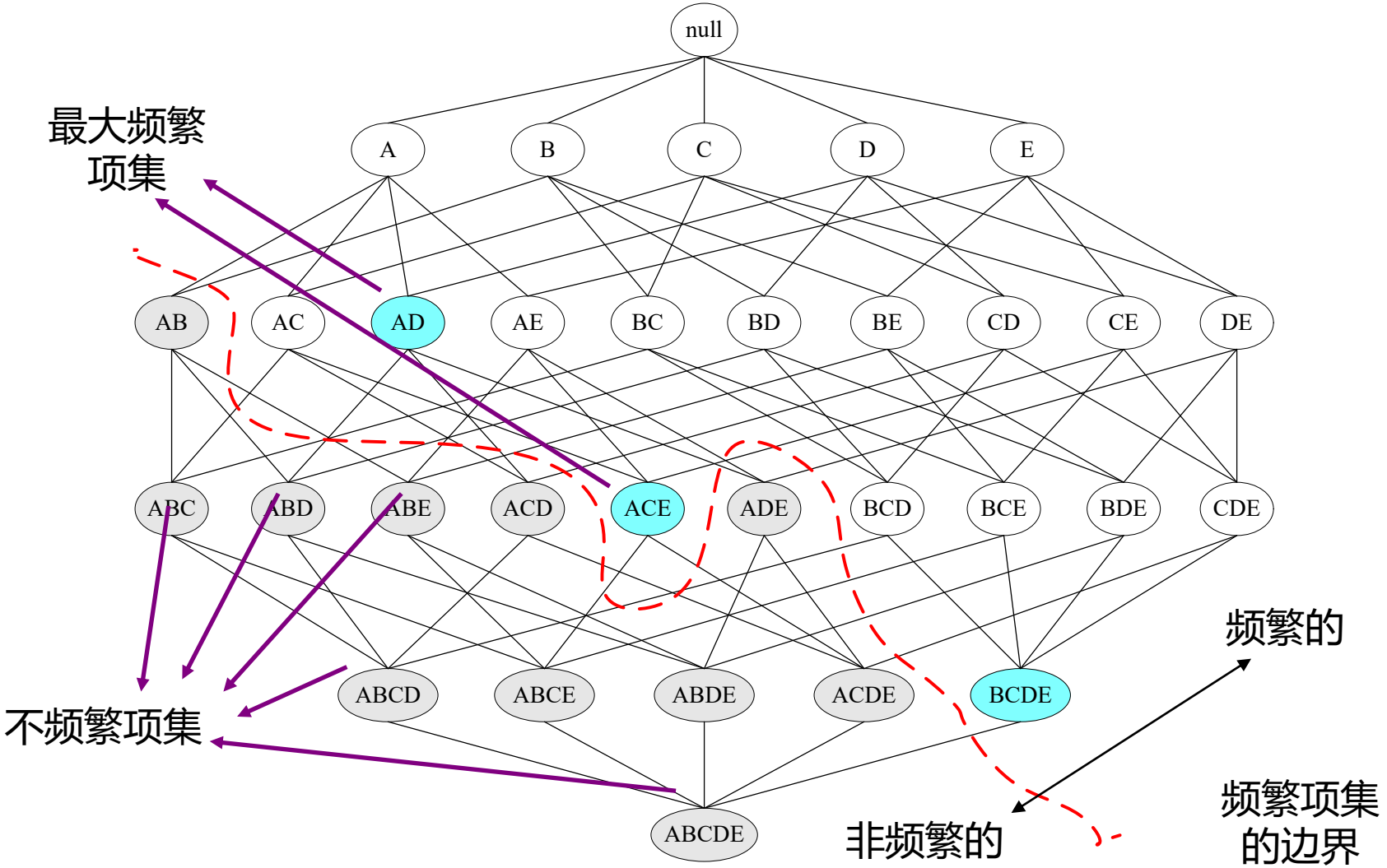
定义： 频繁项集（Frequent Itemset）

- **项集（Itemset）**
  - 包含0个或多个项的集合
    - 例子： {Milk, Bread, Diaper}
  - k-项集
    - 如果一个项集包含k个项
- **支持度计数（Support count ）（σ）**
  - 包含特定项集的事务个数
  - 例如： σ ({Milk, Bread,Diaper}) = 2
- **支持度（Support）**
  - 包含项集的事务数与总事务数的比值
  - 例如： s({Milk, Bread, Diaper}) = 2/5
- **频繁项集（Frequent Itemset）**
  - 满足最小支持度阈值（ minsup ）的所有项集

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

最大频繁项集（Maximal Frequent Itemset）

最大频繁项集是这样的频繁项集，它的直接超集都不是频繁的



定义：关联规则（Association Rule）

- 关联规则
  - 关联规则是形如 X → Y的蕴含表达式，其中 X 和 Y 是不相交的项集
  - 例子：

    {Milk, Diaper} → {Beer}
- 关联规则的强度
  - 支持度 Support (s)
    - 确定项集的频繁程度
  - 置信度 Confidence (c)
    - 确定Y在包含X的事务中出现的频繁程度

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：

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

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

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

关联规则挖掘问题

关联规则挖掘问题：

给定事务的集合 T，关联规则发现是指找出**支持度大于等于minsup**并且**置信度大于等于minconf**的所有规则，minsup和minconf是对应的支持度和置信度阈值

关联规则挖掘问题

关联规则挖掘问题：给定事务的集合 T，关联规则发现是指找出**支持度大于等于 minsup**并且**置信度大于等于minconf**的所有规则，minsup和minconf是对应的支持度和置信度阈值

事务Id	项集合
10	A, B, C
20	A, C
30	A, D
40	B, E, F

最小支持度 50%
最小置信度 50%

频繁模式	支持度
{A}	75%
{B}	50%
{C}	50%
{A, C}	50%

规则 $A \Rightarrow C$:

支持度 = support($\{A\} \cup \{C\}$) = **50%**

置信度 = support($\{A\} \cup \{C\}$)/support($\{A\}$) = **66.6%**

挖掘关联规则（Mining Association Rules）

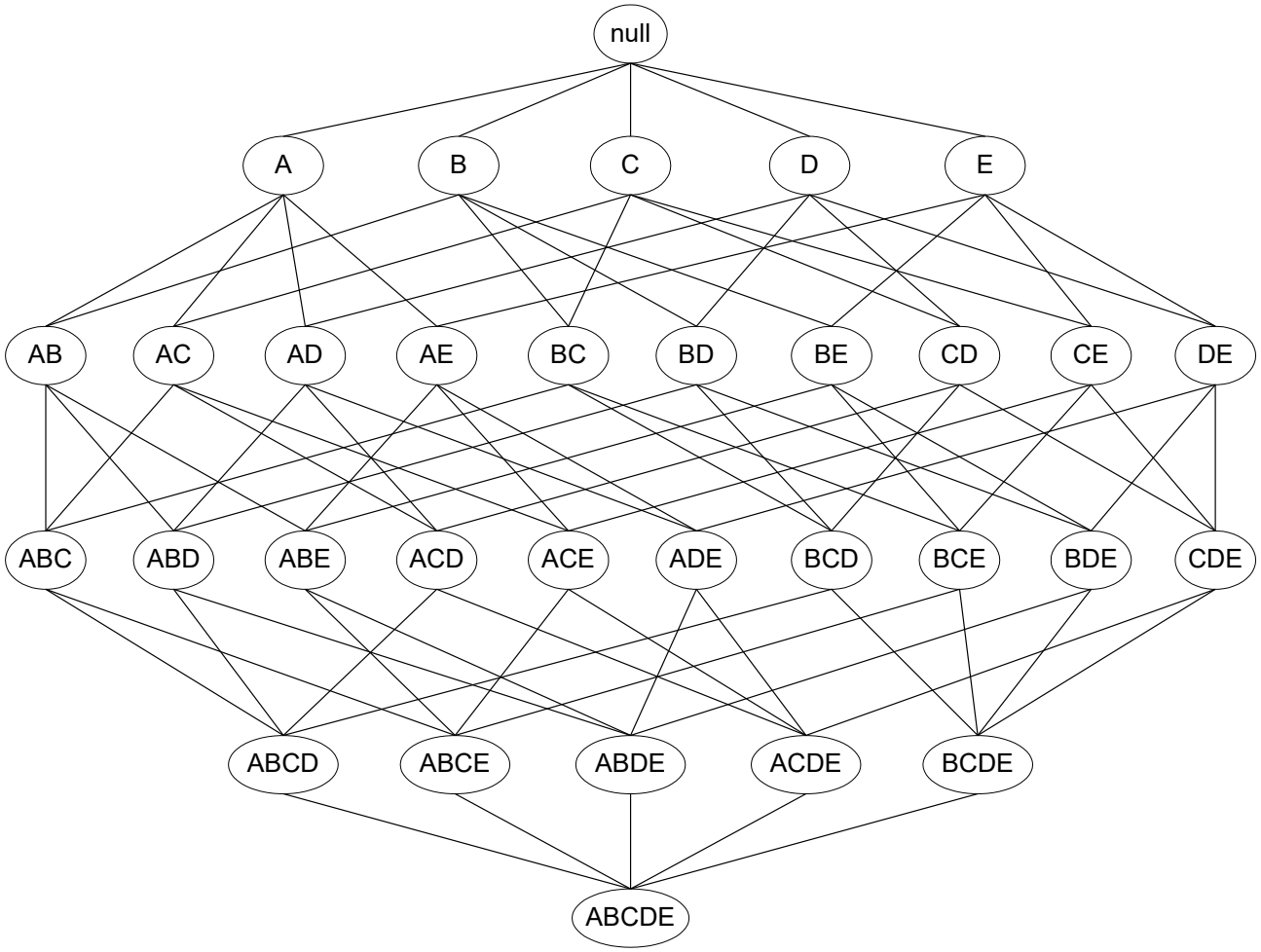
- 大多数关联规则挖掘算法通常采用的一种策略是，将关联规则挖掘任务分解为如下两个主要的子任务：
  - **频繁项集产生（Frequent Itemset Generation）**
    - 其目标是发现满足最小支持度阈值的所有项集，这些项集称作频繁项集。
  - **规则的产生（Rule Generation）**
    - 其目标是从上一步发现的频繁项集中提取所有高置信度的规则，这些规则称作强规则（strong rule）。

关联规则原始方法

- **挖掘关联规则的一种原始方法是：**

  Brute-force approach：

  - 计算每个可能规则的支持度和置信度
  - 这种方法计算代价过高，因为可以从数据集提取的规则的数量达指数级

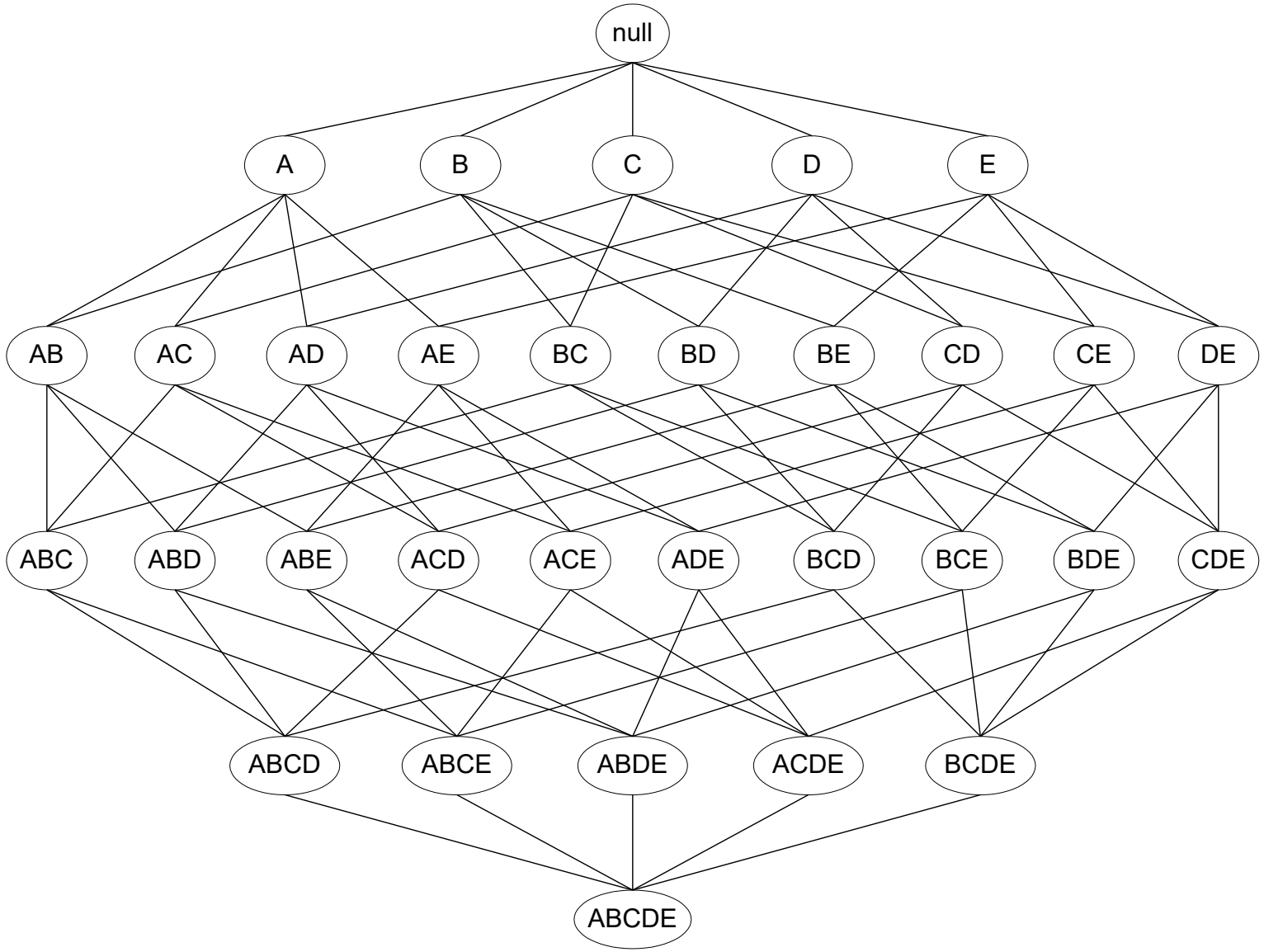


南京大学
NANJING UNIVERSITY

目录

频繁项集产生（Frequent Itemset Generation）

- Brute-force 方法：
  - 计算开销大



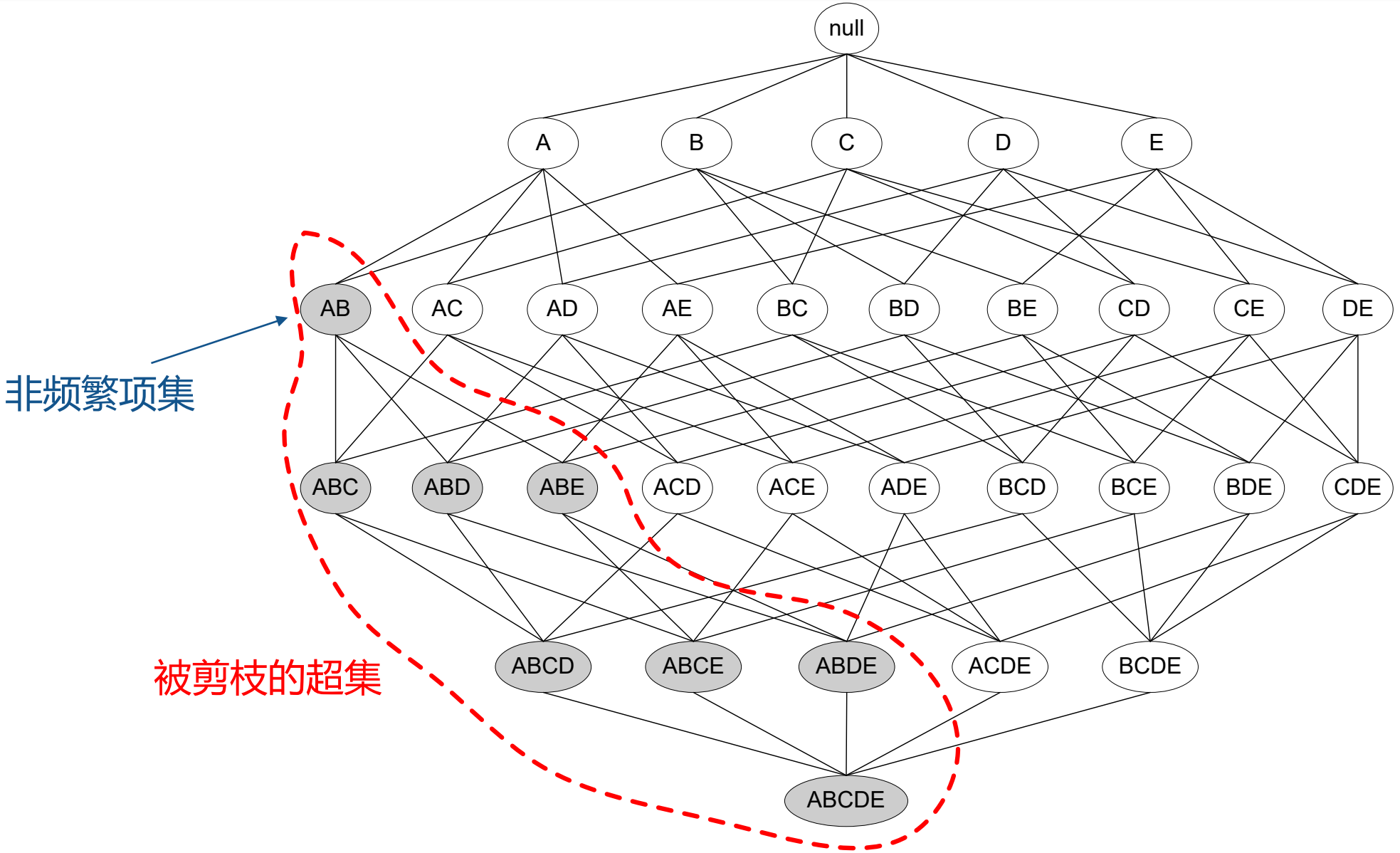
降低产生频繁项集计算复杂度的方法

- 减少候选项集的数量
  - 先验原理：(**Apriori**)
- 减少比较的次数
  - 替代将每个候选项集与每个事务相匹配，可以使用更高级的数据结构，或存储候选项集或压缩数据集，来减少比较次数(**FPGrowth**)

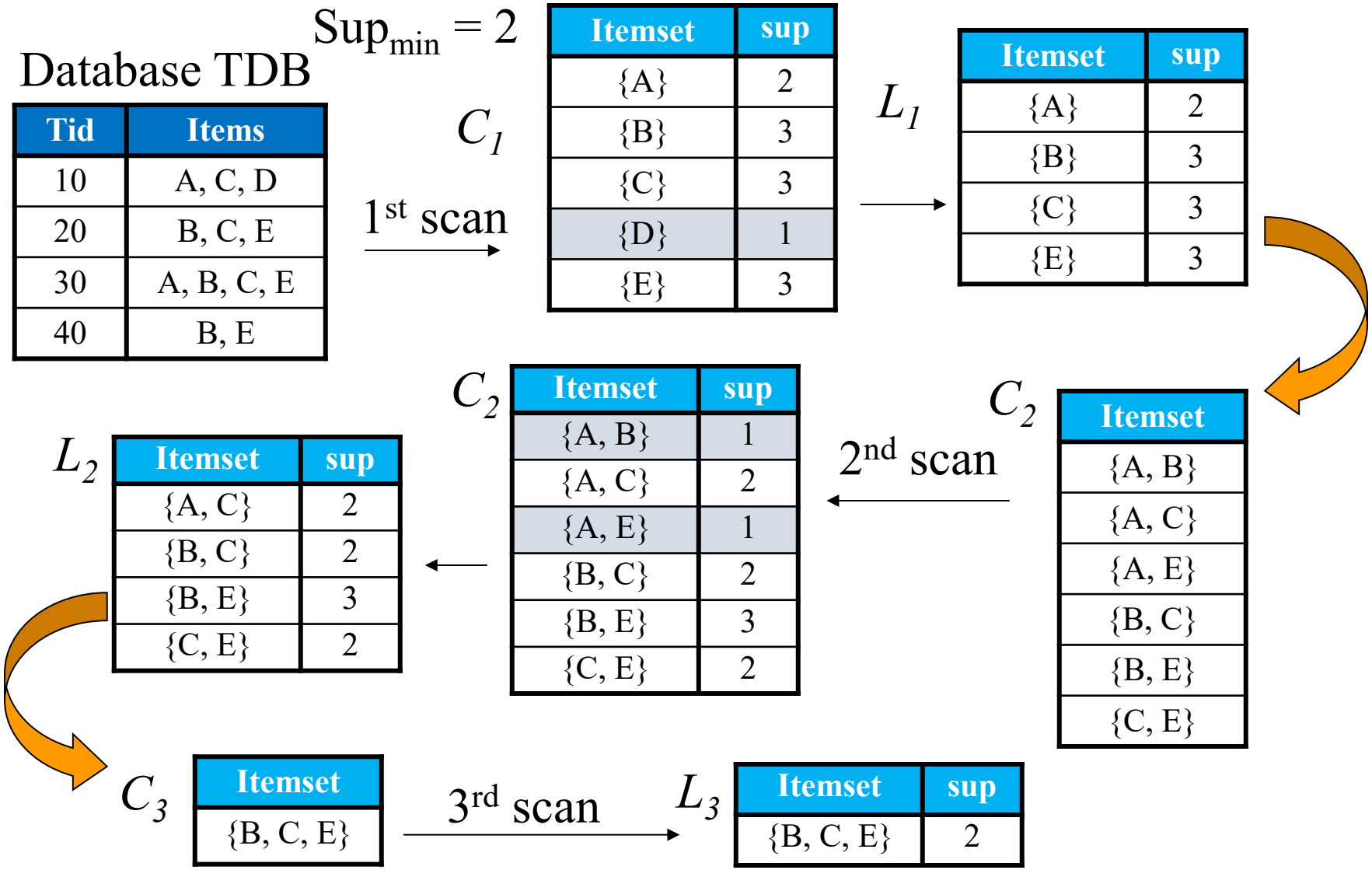
Apriori

- **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
  - 相反，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

例子


null
A
B
C
D
E
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
ABC
ABD
ABE
ACD
ACE
ADE
BCD
BCE
BDE
CDE
ABCD
ABCE
ABDE
ACDE
BCDE
ABCDE
非频繁项集
被剪枝的超集

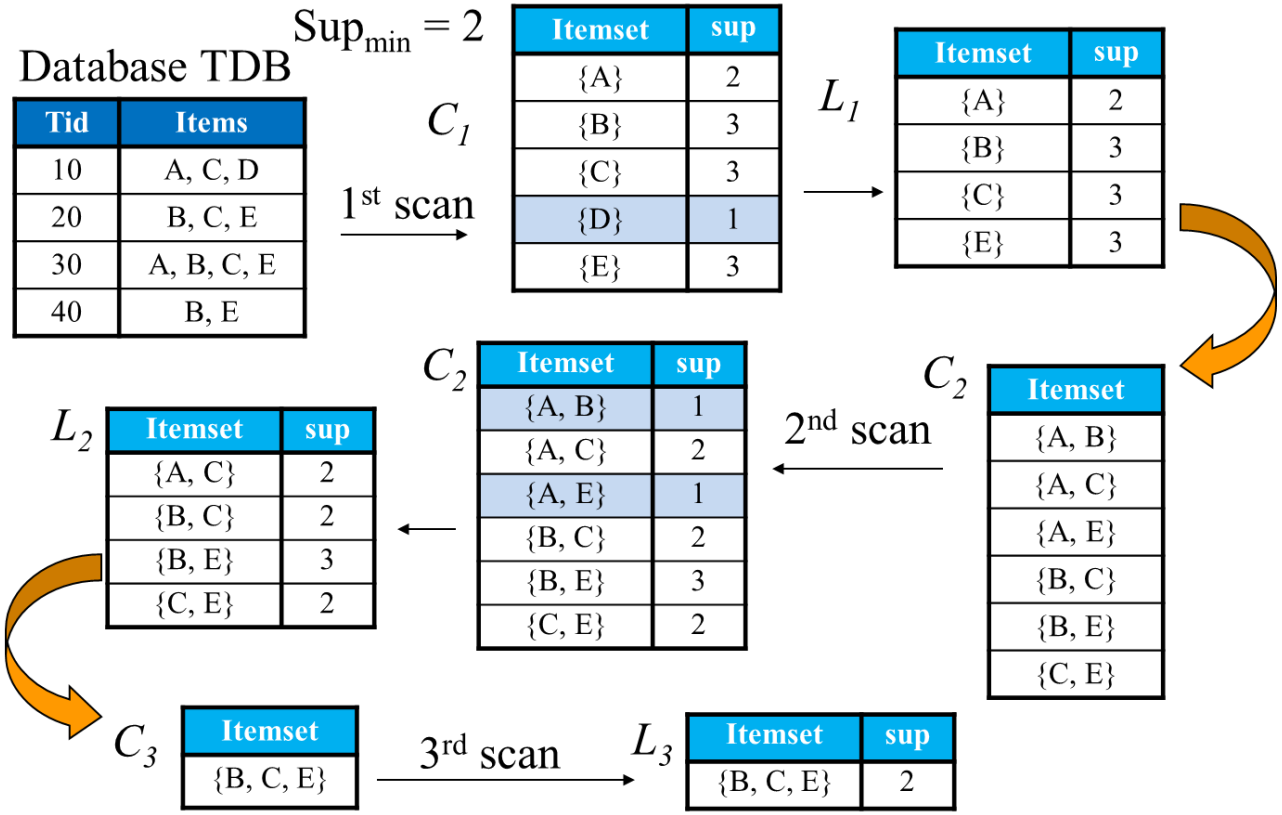
Apriori算法过程：最小支持度计数=2



Apriori算法注意问题 —— 项的字典序

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

尽管集合具有无序性，但为了快速连接操作，**通常对所有商品做一个默认的排序**（类似于建立一个字典索引）



Apriori算法注意问题 —— 项的连接

对于任何2个需要连接的项集

A,B,C

A,B,E

Apriori算法注意问题 —— 项的连接

对于任何2个需要连接的项集

A,B,~~C~~ **去掉第1个项集的尾项**

A,B,E

Apriori算法注意问题 —— 项的连接

对于任何2个需要连接的项集

A,B,~~C~~ **去掉第1个项集的尾项**

A,B,~~E~~ **去掉第2个项集的尾项**

Apriori算法注意问题 —— 项的连接

对于任何2个需要连接的项集

A,B,~~C~~ **去掉第1个项集的尾项**

A,B,~~E~~ **去掉第2个项集的尾项**

若剩下的项一样 →

A,B

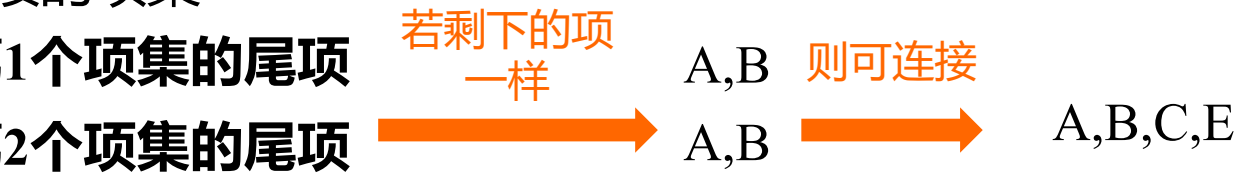
A,B

Apriori算法注意问题 —— 项的连接

对于任何2个需要连接的项集

A,B,~~C~~ **去掉第1个项集的尾项**

A,B,~~E~~ **去掉第2个项集的尾项**



Apriori算法注意问题 —— 项的连接

对于任何2个需要连接的项集

A,B,~~C~~ **去掉第1个项集的尾项**

A,B,~~E~~ **去掉第2个项集的尾项**

对于任何2个需要连接的项集

A,B,~~D~~ **去掉第1个项集的尾项**

B,C,~~E~~ **去掉第2个项集的尾项**

Apriori算法注意问题 —— 项的连接

对于任何2个需要连接的项集

A,B,~~C~~ **去掉第1个项集的尾项**

A,B,~~E~~ **去掉第2个项集的尾项**

若剩下的项一样 → A,B / A,B → 则可连接 → A,B,C,E

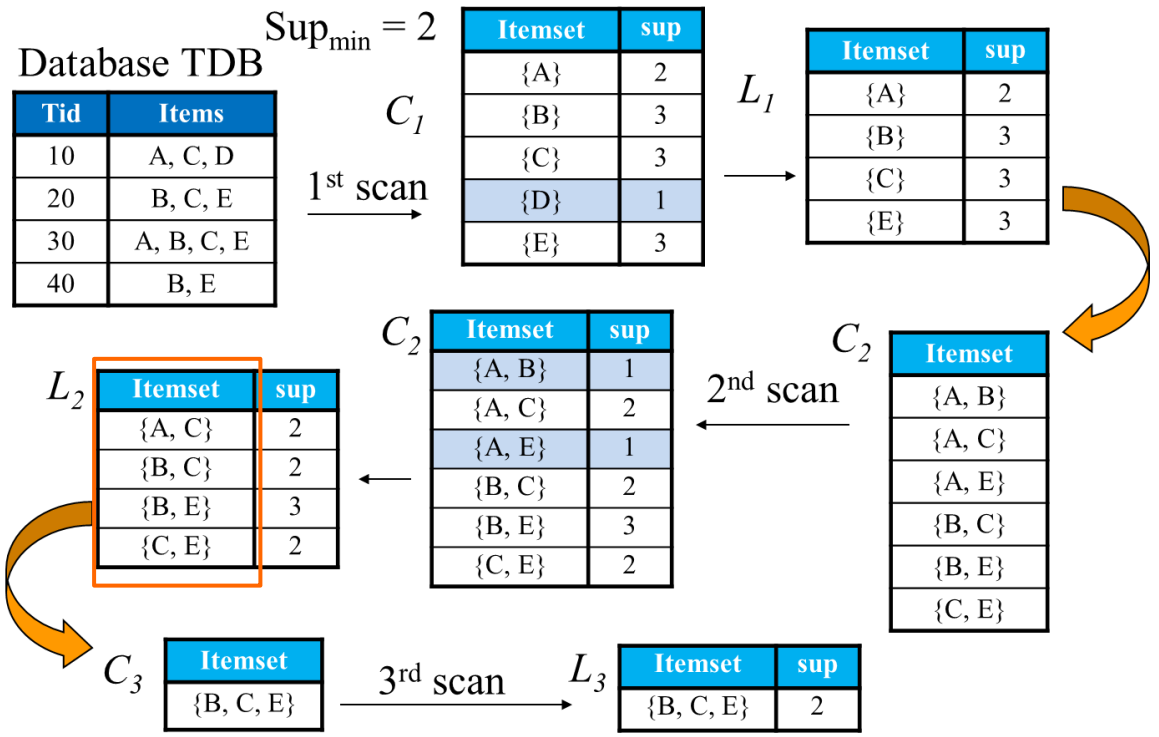
对于任何2个需要连接的项集

A,B,~~D~~ **去掉第1个项集的尾项**

B,C,~~E~~ **去掉第2个项集的尾项**

若剩下的项不一样 → A,B / B,C → 则不可连接

频繁2项集生成的候选3项集是什么?



Apriori算法注意问题 —— 项的连接

对于任何2个需要连接的项集

A,B,~~C~~ **去掉第1个项集的尾项**

A,B,~~E~~ **去掉第2个项集的尾项**

若剩下的项一样 → A,B / A,B → 则可连接 → A,B,C,E

对于任何2个需要连接的项集

A,B,~~D~~ **去掉第1个项集的尾项**

B,C,~~E~~ **去掉第2个项集的尾项**

若剩下的项不一样 → A,B / B,C → 则不可连接

~~A,B,C~~

~~A,C,E~~

B,C,E

Database TDB

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

$Sup_{min} = 2$

C_1 (1st scan)

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}

C_2 (2nd scan)

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 (3rd scan)

Itemset
{B, C, E}

L_3

Itemset	sup
{B, C, E}	2

Apriori算法注意问题 —— 项的连接

对于任何2个需要连接的项集

A,B,~~C~~ **去掉第1个项集的尾项**

A,B,~~E~~ **去掉第2个项集的尾项**

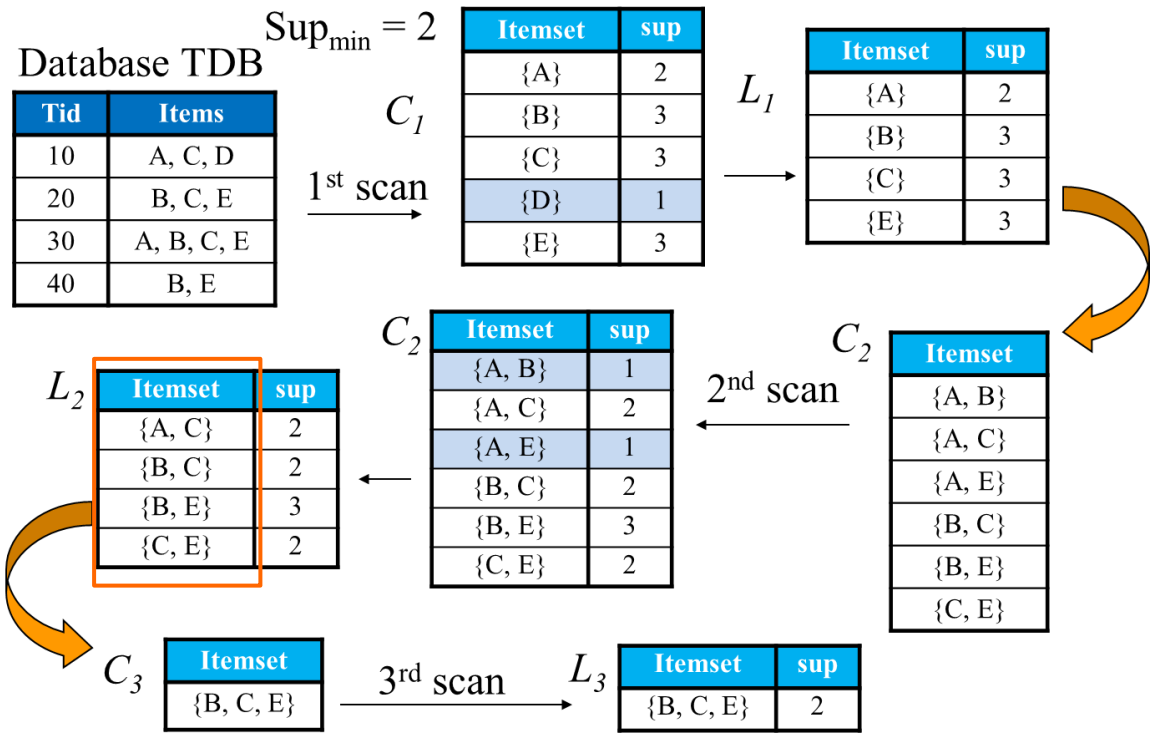
若剩下的项一样 → A,B / A,B → 则可连接 → A,B,C,E

对于任何2个需要连接的项集

A,B,~~D~~ **去掉第1个项集的尾项**

B,C,~~E~~ **去掉第2个项集的尾项**

A,B / B,C → 则不可连接



~~A,B,C~~

~~A,C,E~~

B,C,E

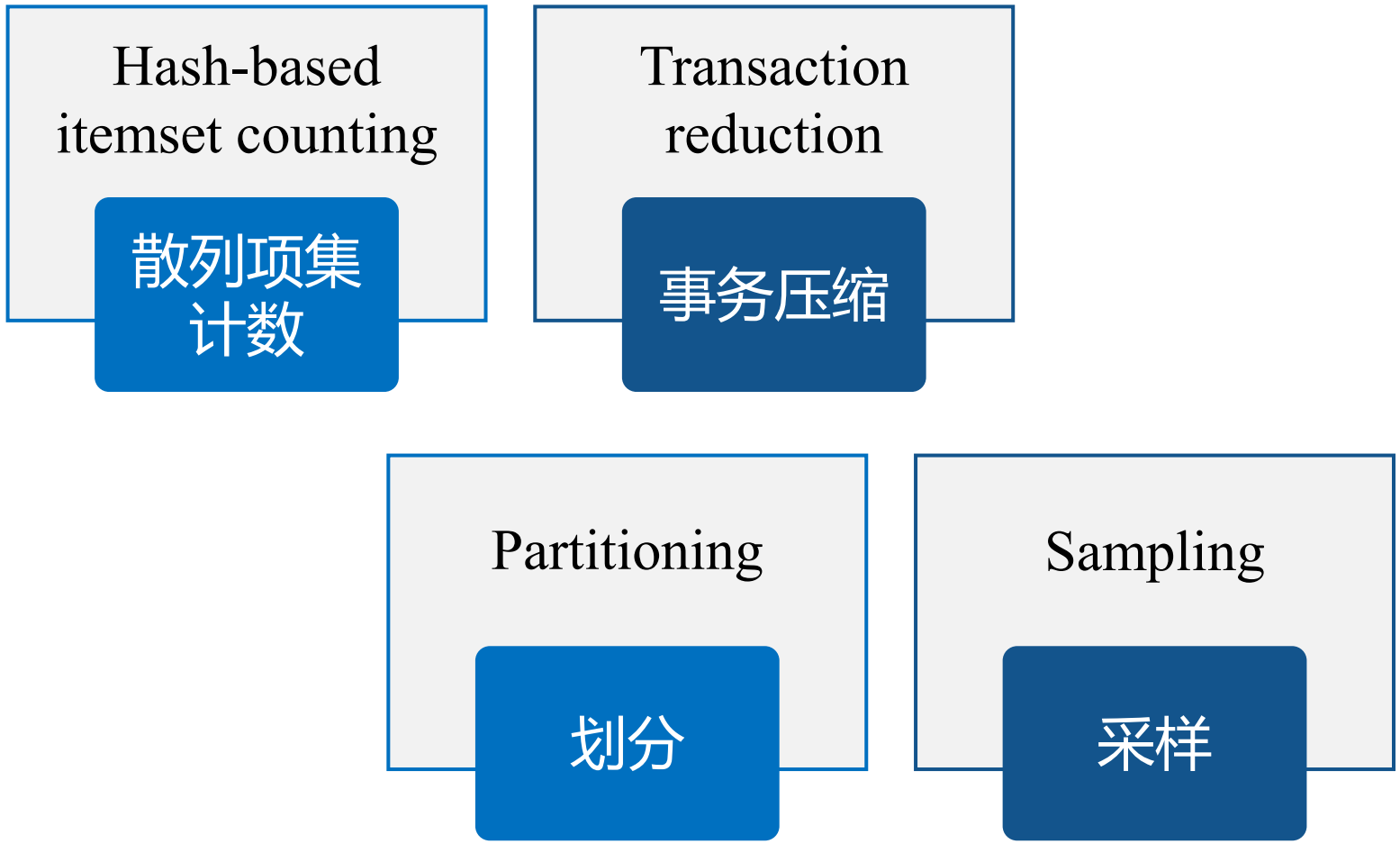
项的连接准则的优点是降低候选项的生成

Apriori算法特点

- **优点**
  - 使用先验性质，大大提高了频繁项集逐层产生的效率
  - 简单易理解；数据集要求低
- **缺点**
  - 多次扫描数据库
  - 候选项规模庞大
  - 计算支持度开销大

Apriori算法需要反复的生成候选项，如果项的数目比较大，候选项的数目将达到**组合爆炸式的增长**

提高Apriori算法性能的方法



FPGrowth算法

- **背景**
  - 韩家炜等人，2000年
- **基本思想**
  - 只扫描数据库两遍，构造频繁模式树（FP-Tree）
  - 自底向上递归产生频繁项集
  - FP树是一种输入数据的压缩表示，它通过逐个读入事务，并把**每个事务映射到FP树中的一条路径来构造**。

构造FP树：第一遍扫描

原始事务

TID	Items
1	{B,F,A}
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}
11	{G}

删除支持度小于2的项

Item	Count
A	8
B	7
C	6
D	5
E	3
F	1
G	1

假设事先指定最小支持度计数为2

Items按照出现次数降序排列

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}

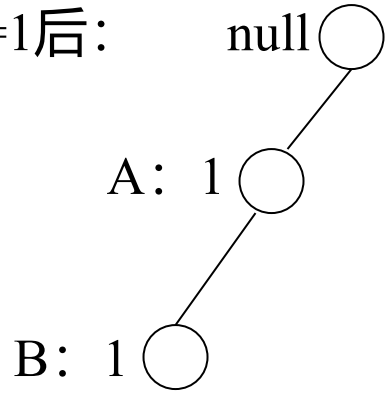
构造FP树：第二遍扫描

- 开始时FP树没有数据，建立FP树时我们一条条的读入排序后的数据集插入FP树，排序靠前的节点是祖先节点，靠后的是子孙节点。
- 如果有共用的祖先，则对应的公用祖先节点计数加1。
- 插入后，如果有新节点出现，则项头表对应的节点会通过节点链表链接上新节点。
- 直到所有的数据都插入到FP树后，FP树的建立完成。

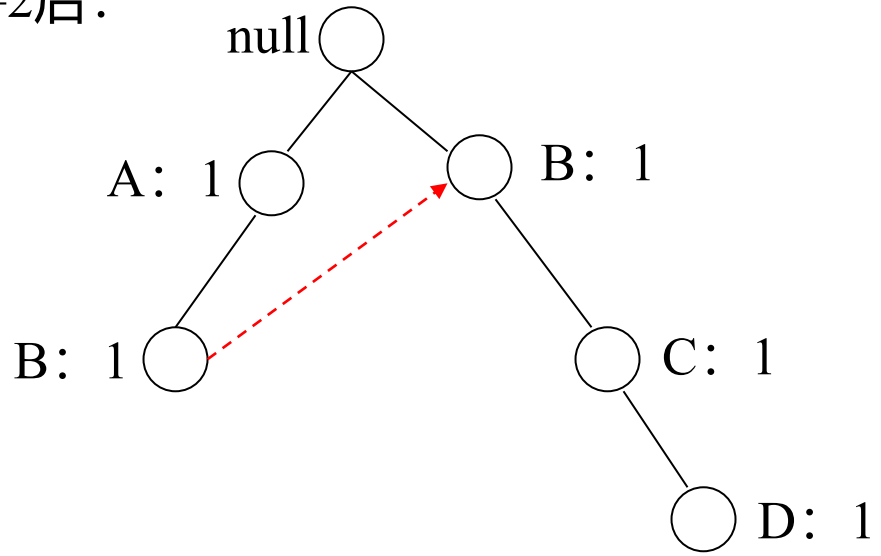
构造FP树：第二遍扫描

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：



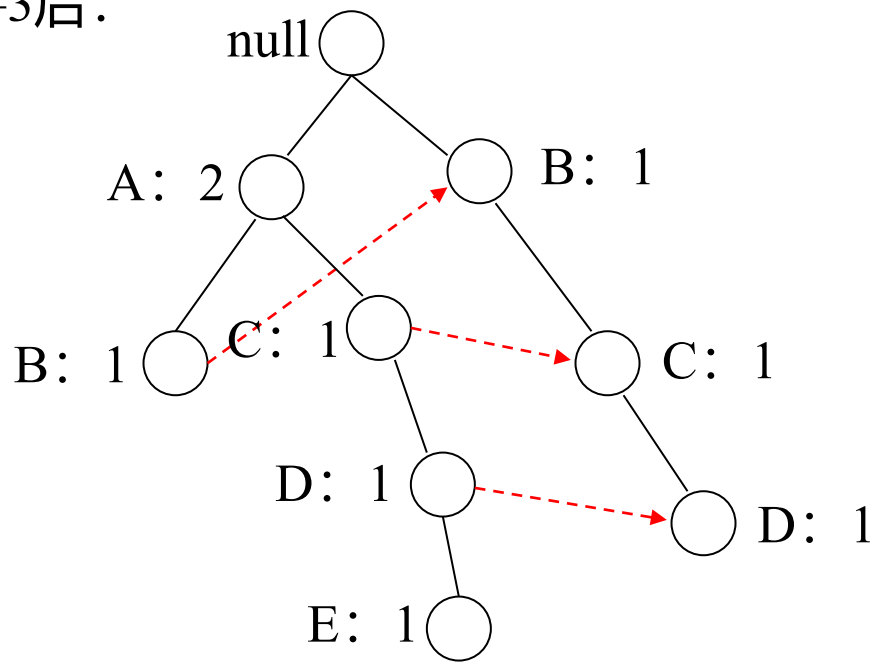
读入事务 TID=2后：



构造FP树：第二遍扫描

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=3后：

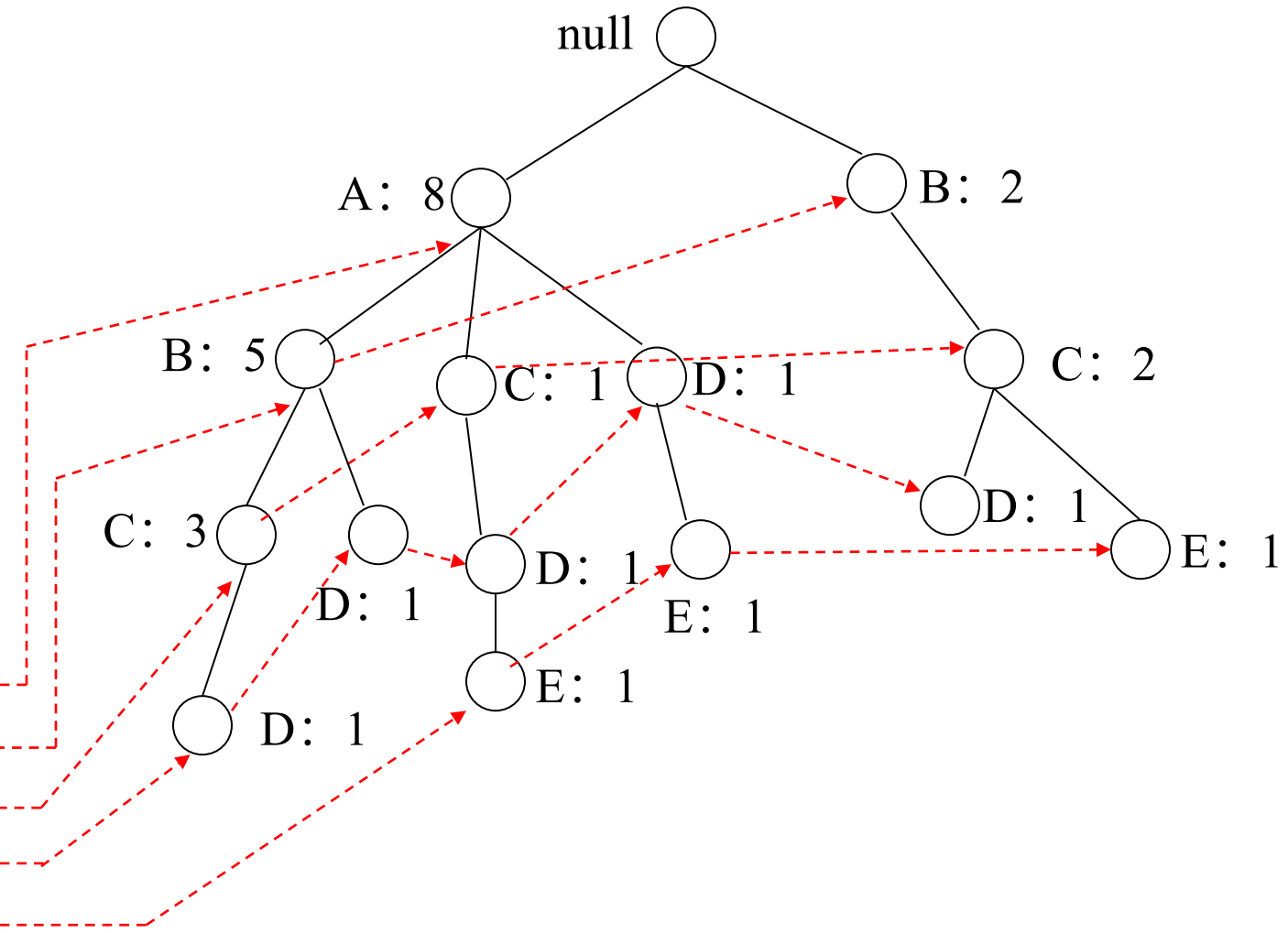


构造FP树：第二遍扫描

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	Pointer
A	
B	
C	
D	
E	

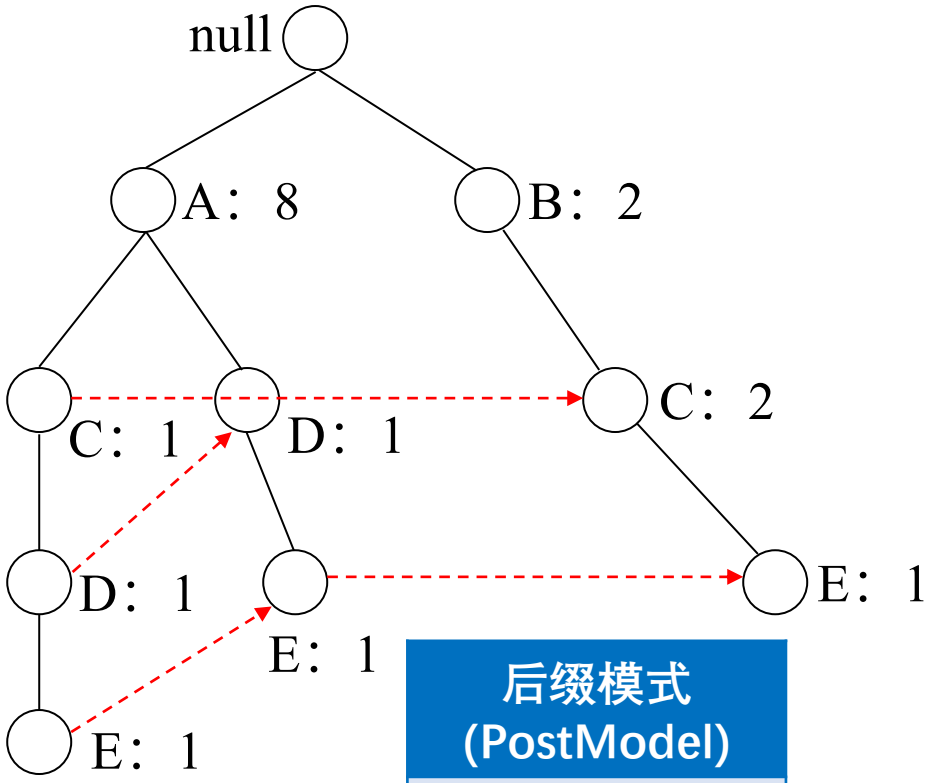


构造FP树：用FP-tree挖掘频繁集

- 基本思想 (分治)
  - 用FP-tree递归增长频繁集
- 方法
  - 对每个项，生成它的**条件模式基**, 然后生成它的**条件 FP-tree**
  - 对每个新生成的条件FP-tree，重复这个步骤
  - 直到结果FP-tree为**空**， 或只含**唯一的一个路径** (此路径的每个子路径对应的项集都是频繁集)

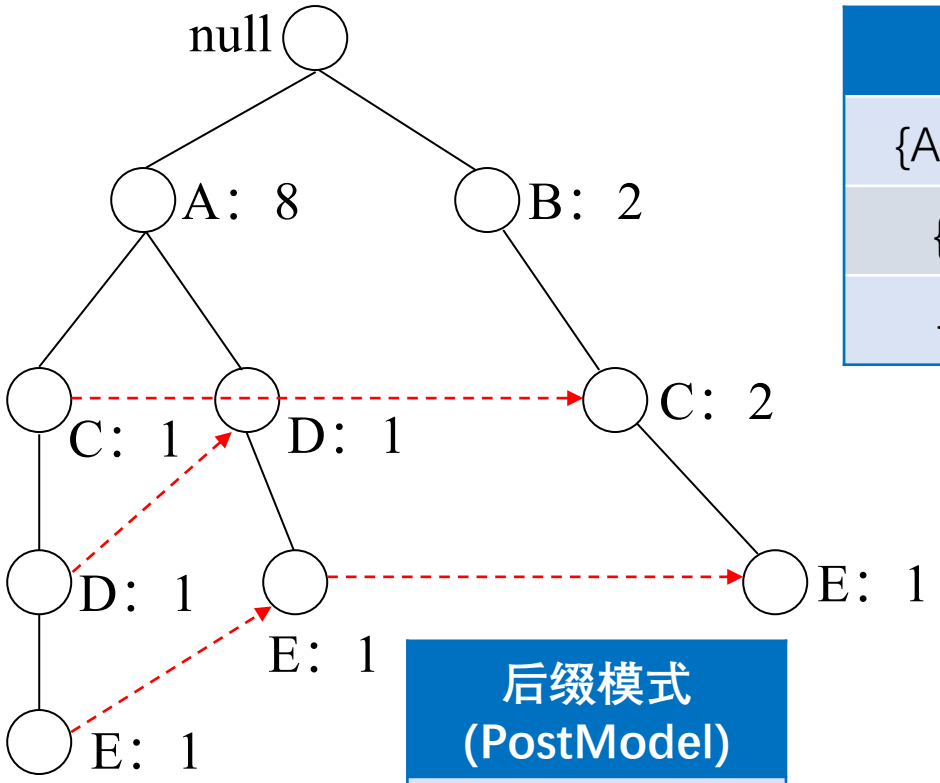
举例：节点E

- 基本思想 (分治)
  - 用FP-tree递归增长频繁集
- 方法
  - 对每个项，生成它的**条件模式基**, 然后生成它的**条件 FP-tree**
  - 对每个新生成的条件FP-tree，重复这个步骤
  - 直到结果FP-tree为**空**， 或只含**唯一的一个路径** (此路径的每个子路径对应的项集都是频繁集)



举例：节点E

- 基本思想 (分治)
  - 用FP-tree递归增长频繁集
- 方法
  - 对每个项，生成它的**条件模式基**, 然后生成它的**条件 FP-tree**
  - 对每个新生成的条件FP-tree，重复这个步骤
  - 直到结果FP-tree为**空**， 或只含**唯一的一个路径** (此路径的每个子路径对应的项集都是频繁集)

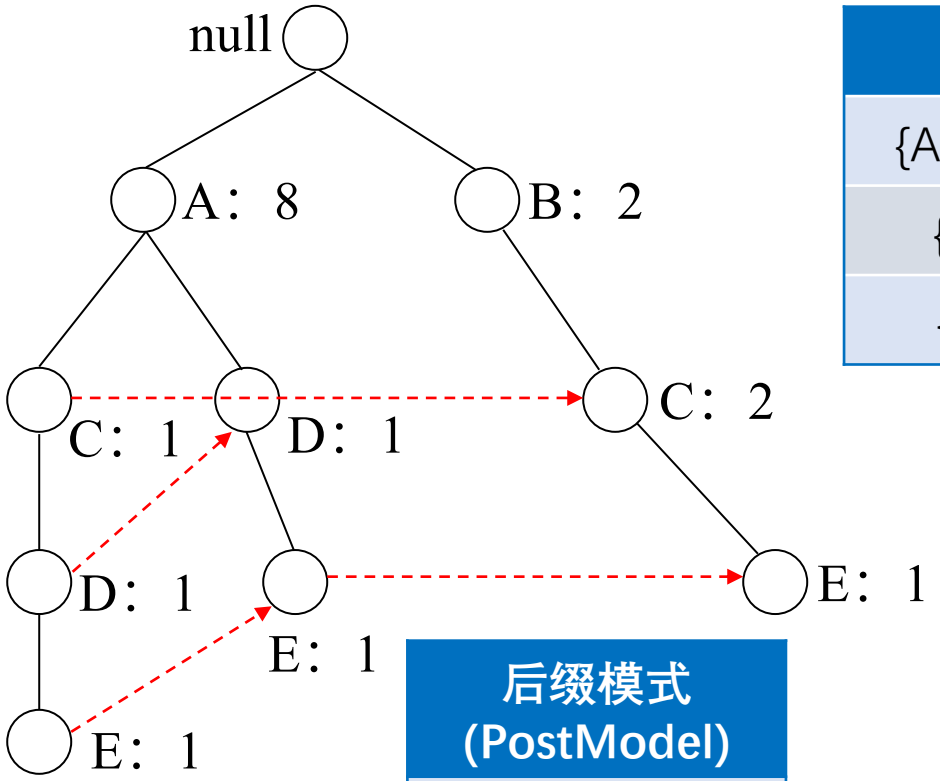


Items
{A:8,C:1,D:1,E:1}
{A:8,D:1,E:1}
{B:2,C:2,E:1}

后缀模式 (PostModel)
E

举例：节点E

- 基本思想 (分治)
  - 用FP-tree递归增长频繁集
- 方法
  - 对每个项，生成它的**条件模式基**, 然后生成它的**条件 FP-tree**
  - 对每个新生成的条件FP-tree，重复这个步骤
  - 直到结果FP-tree为**空**， 或只含**唯一的一个路径** (此路径的每个子路径对应的项集都是频繁集)



Items
{A:8,C:1,D:1,E:1}
{A:8,D:1,E:1}
{B:2,C:2,E:1}

Items
{A:1,C:1,D:1,E:1}
{A:1,D:1,E:1}
{B:1,C:1,E:1}

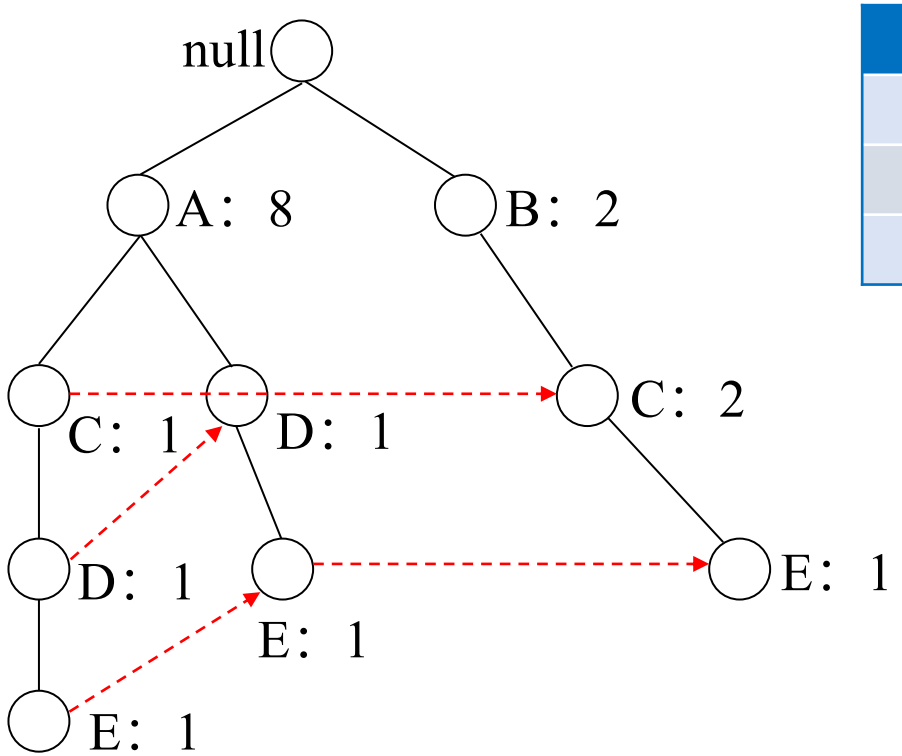
后缀模式 (PostModel)
E

条件模式基 (Conditional Pattern Base,CPB)
{A:1,C:1,D:1}
{A:1,D:1}
{B:1,C:1}

举例：节点E

- 基本思想 (分治)
  - 用FP-tree递归增长频繁集
- 方法
  - 对每个项，生成它的**条件模式基**, 然后生成它的**条件 FP-tree**
  - 对每个新生成的条件FP-tree，重复这个步骤
  - 直到结果FP-tree为**空**， 或只含**唯一的一个路径** (此路径的每个子路径对应的项集都是频繁集)

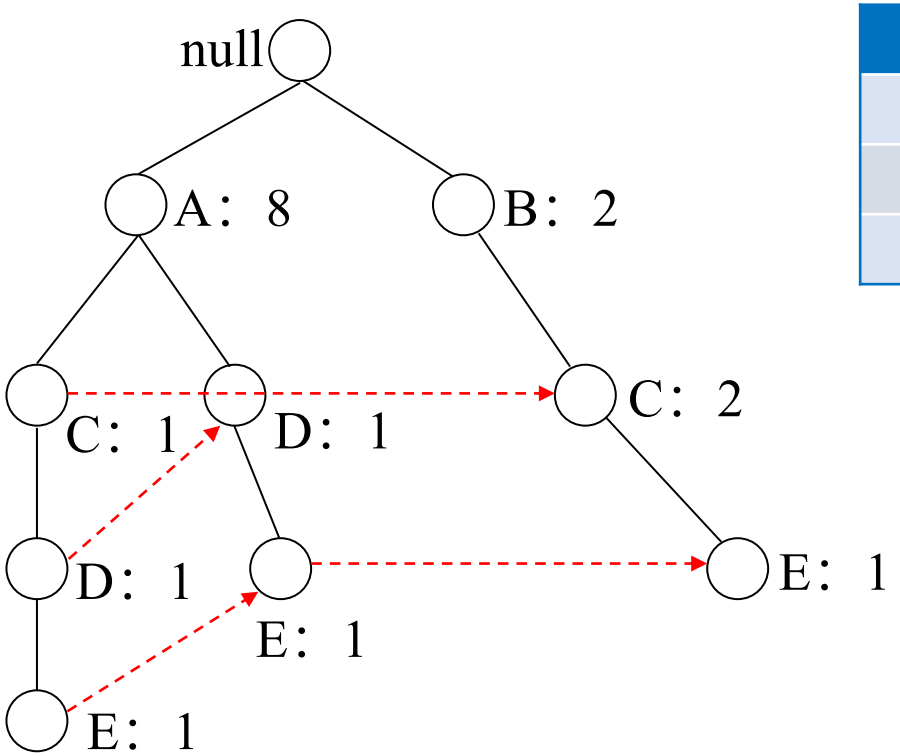
条件模式基
{A:1,C:1,D:1}
{A:1,D:1}
{B:1,C:1}



表头项
A:2
C:2
D:2

举例：节点E

- 基本思想 (分治)
  - 用FP-tree递归增长频繁集
- 方法
  - 对每个项，生成它的**条件模式基**, 然后生成它的**条件 FP-tree**
  - 对每个新生成的条件FP-tree，重复这个步骤
  - 直到结果FP-tree为**空**， 或只含**唯一的一个路径** (此路径的每个子路径对应的项集都是频繁集)



条件模式基
{A:1,C:1,D:1}
{A:1,D:1}
{B:1,C:1}

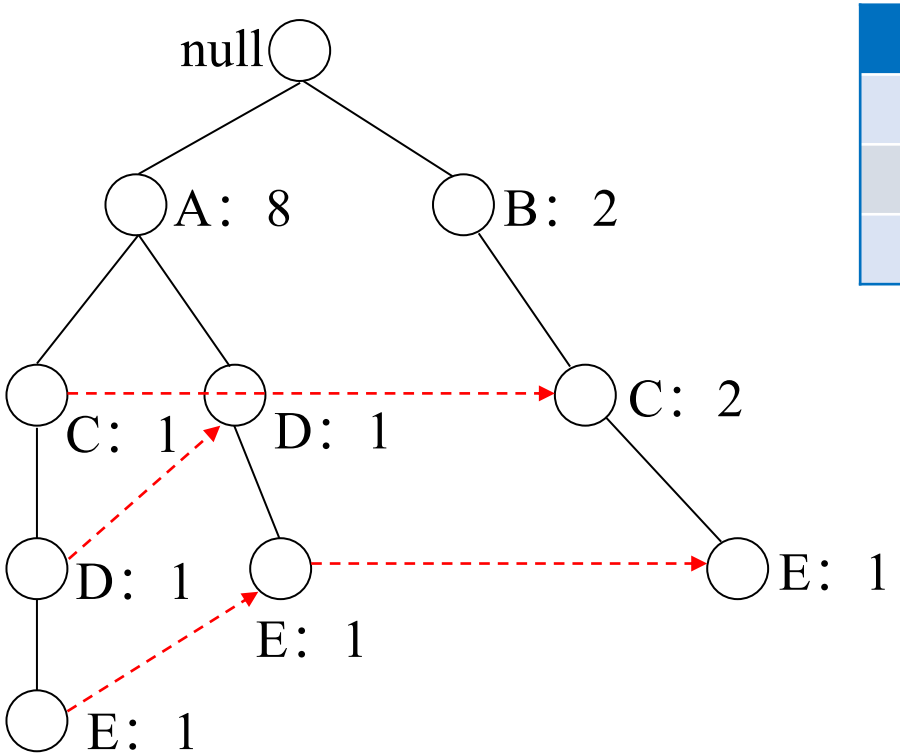
表头项
A:2
C:2
D:2

后缀模式 (PostModel)
E

频繁项集
AE:2
CE:2
DE:2
E:3

举例：节点E

- 基本思想 (分治)
  - 用FP-tree递归增长频繁集
- 方法
  - 对每个项，生成它的**条件模式基**, 然后生成它的**条件 FP-tree**
  - 对每个新生成的条件FP-tree，重复这个步骤
  - 直到结果FP-tree为**空**， 或只含**唯一的一个路径** (此路径的每个子路径对应的项集都是频繁集)



条件模式基
{A:1,C:1,D:1}
{A:1,D:1}
{B:1,C:1}

表头项
A:2
C:2
D:2

后缀模式 (PostModel)
E

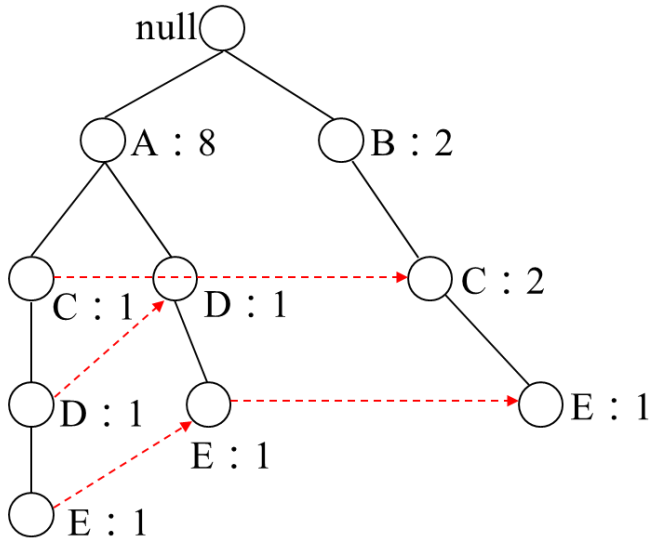
频繁项集
AE:2
CE:2
DE:2
E:3

新的后缀模式
AE
CE
DE

举例：节点E

- 基本思想 (分治)
  - 用FP-tree递归增长频繁集
- 方法
  - 对每个项，生成它的**条件模式基**, 然后生成它的**条件 FP-tree**
  - 对每个新生成的条件FP-tree，重复这个步骤
  - 直到结果FP-tree为**空**， 或只含**唯一的一个路径** (此路径的每个子路径对应的项集都是频繁集)

新的后缀模式
AE
CE
DE

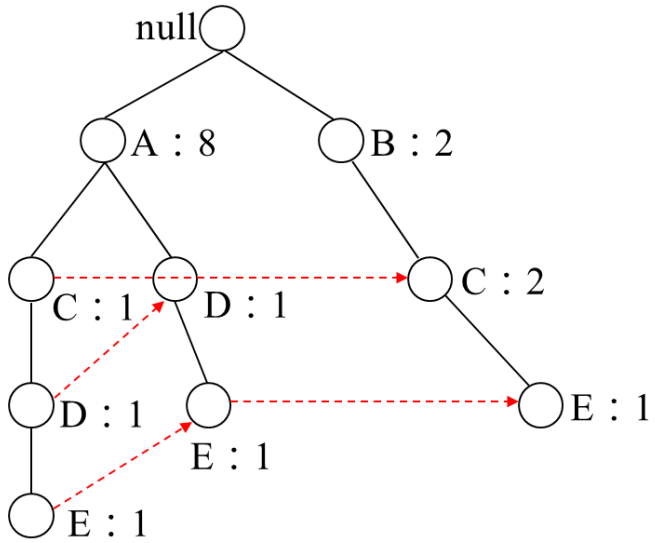


举例：节点E

- 基本思想 (分治)
  - 用FP-tree递归增长频繁集
- 方法
  - 对每个项，生成它的**条件模式基**, 然后生成它的**条件 FP-tree**
  - 对每个新生成的条件FP-tree，重复这个步骤
  - 直到结果FP-tree为**空**， 或只含**唯一的一个路径** (此路径的每个子路径对应的项集都是频繁集)

新的后缀模式
AE
CE
DE

Items
{A:1,C:1,D:1,E:1}
{A:1,D:1,E:1}
{B:1,C:1,E:1}

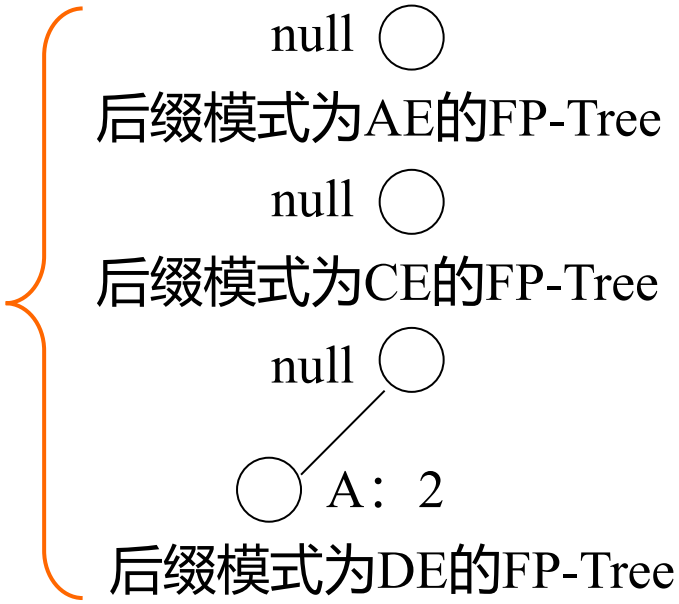


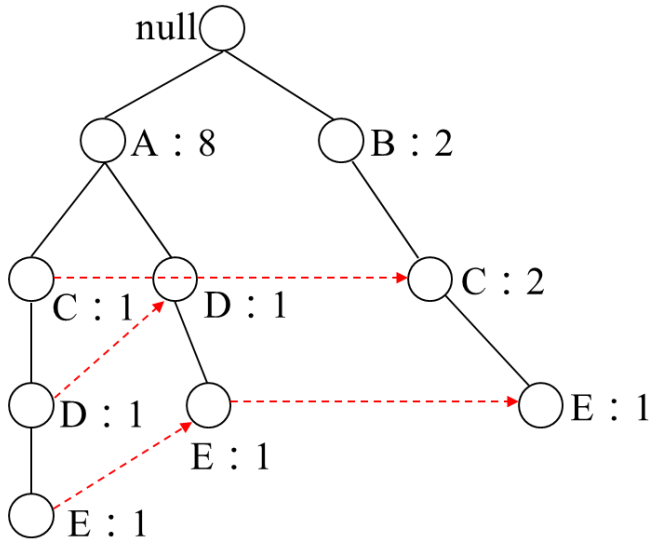
举例：节点E

- 基本思想 (分治)
  - 用FP-tree递归增长频繁集
- 方法
  - 对每个项，生成它的**条件模式基**, 然后生成它的**条件 FP-tree**
  - 对每个新生成的条件FP-tree，重复这个步骤
  - 直到结果FP-tree为**空**， 或只含**唯一的一个路径** (此路径的每个子路径对应的项集都是频繁集)

新的后缀模式
AE
CE
DE

Items
{A:1,C:1,D:1,E:1}
{A:1,D:1,E:1}
{B:1,C:1,E:1}



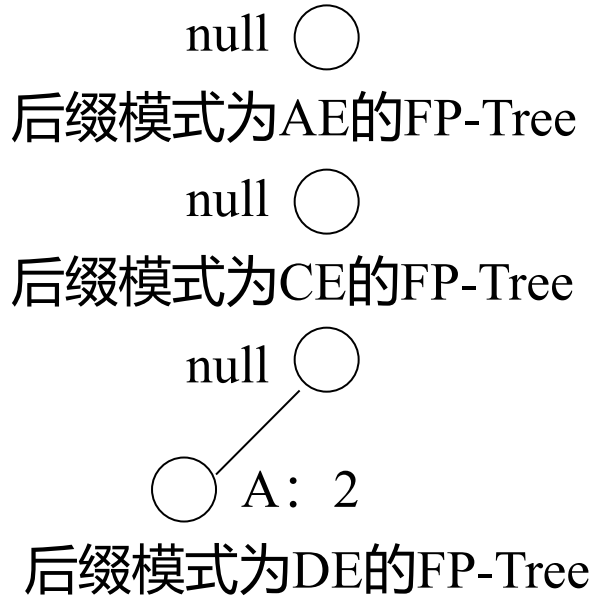


举例：节点E

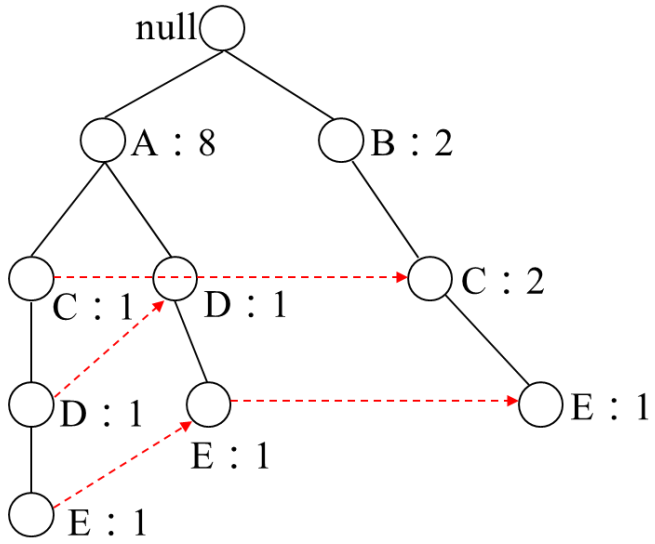
- 基本思想 (分治)
  - 用FP-tree递归增长频繁集
- 方法
  - 对每个项，生成它的**条件模式基**, 然后生成它的**条件 FP-tree**
  - 对每个新生成的条件FP-tree，重复这个步骤
  - 直到结果FP-tree为**空**， 或只含**唯一的一个路径** (此路径的每个子路径对应的项集都是频繁集)

新的后缀模式
AE
CE
DE

Items
{A:1,C:1,D:1,E:1}
{A:1,D:1,E:1}
{B:1,C:1,E:1}



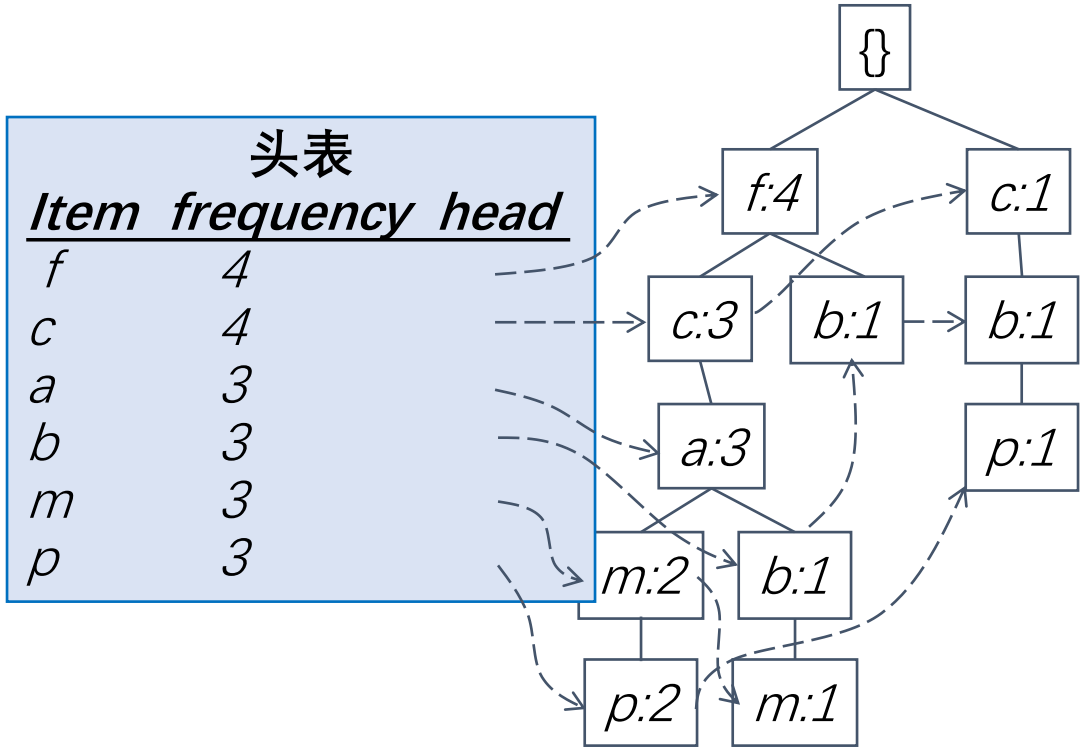
频繁项集
ADE:2



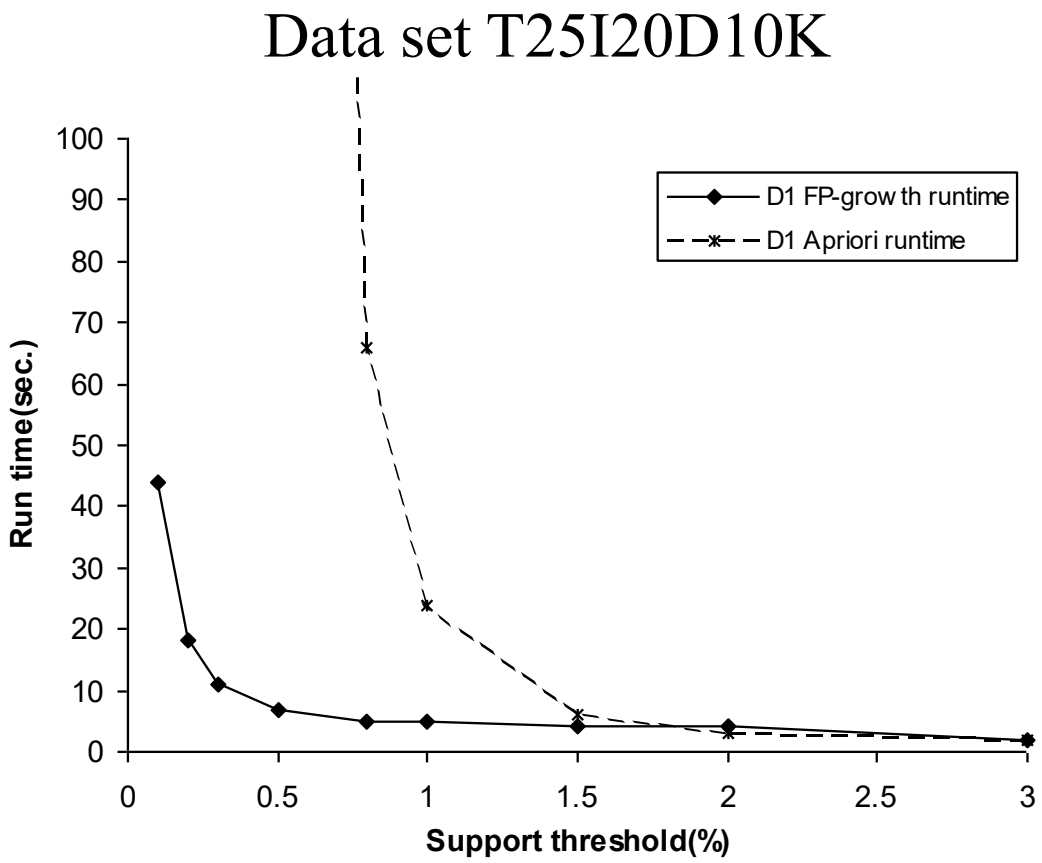
FP-tree 结构的优点

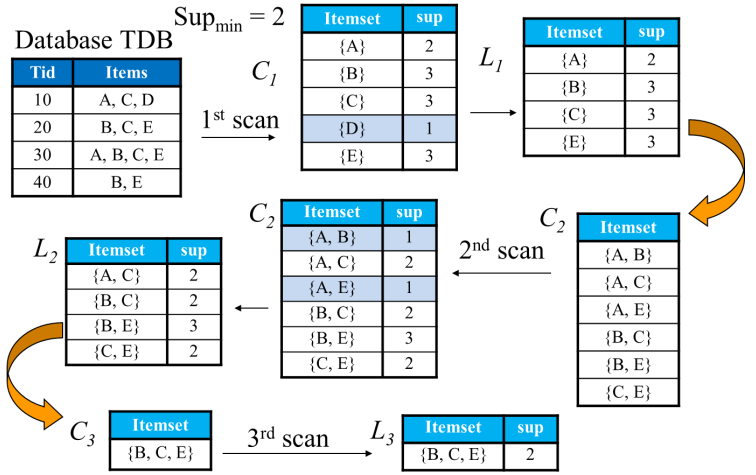
- 完备：
  - 不会打破交易中的任何模式
  - 包含了频繁模式挖掘所需的全部信息
- 紧密
  - 支持度降序排列： 支持度高的项在FP-tree中共享的机会也高
  - 决不会比原数据库大

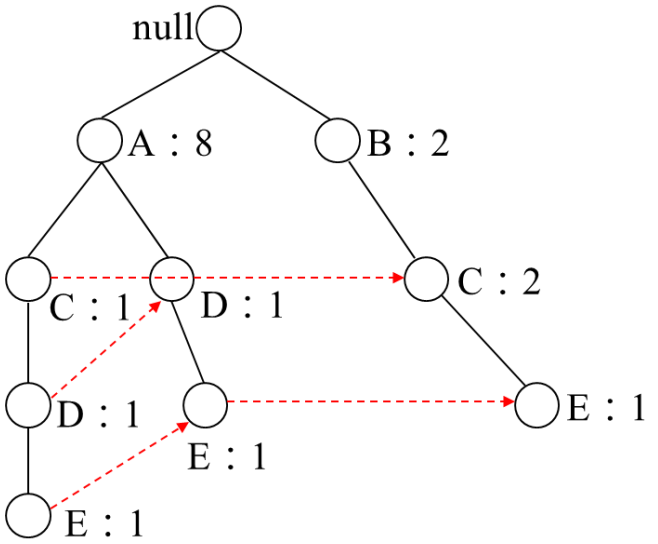
https//www.cnblogs.com/itbuyixiaogong/p/9077428.html



FP-tree 结构的优点 —— 性能对比







挖掘关联规则（Mining Association Rules）

- 大多数关联规则挖掘算法通常采用的一种策略是，将关联规则挖掘任务分解为如下两个主要的子任务：
  - **频繁项集产生（Frequent Itemset Generation）**

    —— 其目标是发现满足最小支持度阈值的所有项集，这些项集称作频繁项集。
  - **规则的产生（Rule Generation）**

    —— 其目标是从上一步发现的频繁项集中提取所有高置信度的规则，这些规则称作强规则（strong rule）。

产生关联规则

- 任务描述：给定频繁项集Y, 查找Y的所有非空真子集$X \subset Y$，使得 $X \rightarrow Y - X$ 的置信度超过最小置信度阈值minconf
  - 例子：If {A,B,C} is a frequent itemset, 候选规则如下：

$$AB \rightarrow C, \quad AC \rightarrow B, \quad BC \rightarrow A$$

$$A \rightarrow BC, \quad B \rightarrow AC, \quad C \rightarrow AB$$

- 如果 $|Y| = k$, 那么会有 $2^k - 2$ 个候选关联规则 (不包括 $Y \rightarrow \varnothing$ and $\varnothing \rightarrow Y$)

产生关联规则

- **如何高效地从频繁项集中产生关联规则?**
  - 通常置信度不满足反单调性（anti-monotone property），例如:
  - $c(ABC \rightarrow D)$ 可能大于也可能小于 $c(AB \rightarrow D)$
  - 但是，针对同一个频繁项集的关联规则，如果规则的**后件满足子集关系**，那么这些规则的**置信度间满足反单调性**
  - e.g., Y= {A,B,C,D}:

    $$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

产生关联规则: Apriori算法