数据挖掘
第六讲·频繁模式挖掘

提纲

1. 基本概念
2. 频繁项集挖掘方法
3. 哪些模式是有趣的：模式评估方法
4. 小结

有效的和可伸缩的频繁项集挖掘方法

- Apriori算法：使用候选产生发现频繁项集
  - 由频繁项集产生关联规则
- 提高Apriori算法的效率
- 不候选产生挖掘频繁项集
- 使用垂直数据格式挖掘频繁项集
- 挖掘闭频繁项集

Apriori缺陷

- Apriori算法的核心:
  - 用频繁的(k–1)-项集生成候选的k-项集
  - 用数据库扫描和模式匹配计算候选集的支持度

- Apriori 的瓶颈: 候选集生成
  - 巨大的候选集:
    - 10^4个频繁1-项集要生成10^7个候选2-项集
  - 多次扫描数据库
    - 如果最长的模式是n的话，则需要(n+1)次数据库扫描

频繁模式增长

频繁模式增长
frequent-pattern growth
FP- growth

http://hanj.cs.illinois.edu/

✓将频繁项集数据压缩到一棵**频繁模式树(FP-树)**，但仍须保留关联信息

✓将压缩后的数据库分成一组**条件FP-树**，每棵FP-树关联一个频繁项

✓分别挖掘每个条件FP-树

挖掘频繁项集的模式增长方法

- 频繁模式增长（frequent-pattern growth）
  - Step1:构建FP-树
    - 将频繁项集的数据压缩到一棵频繁模式树(FP-树)，该树保留项集的关联信息（从树中能得到关联规则）。
  - Step2:生成频繁项集
    - 将这种压缩后的数据库分成一组条件FP-树，每棵FP-树关联一个频繁项，并分别挖掘每个条件FP-树。

Step1：构造FP-树

- 第一次扫描数据库，与Apriori相同，导出频繁项（1项集）的集合和支持度计数。
- 设最小支持度计数为2。频繁项集按支持度计数递减序排序：

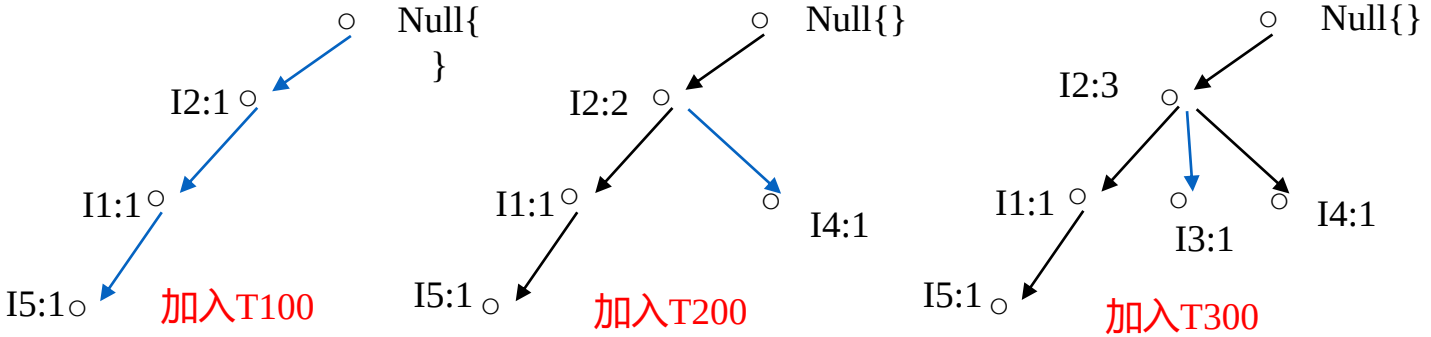
TID	项ID的列表
T100	I1,I2,I5
T200	I2,I4
T300	I2,I3
T400	I1,I2,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I1,I2,I3,I5
T900	I1,I2,I3

TID	项ID的列表
T100	I2,I1,I5
T200	I2,I4
T300	I2,I3
T400	I2,I1,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I2,I1,I3,I5
T900	I2,I1,I3

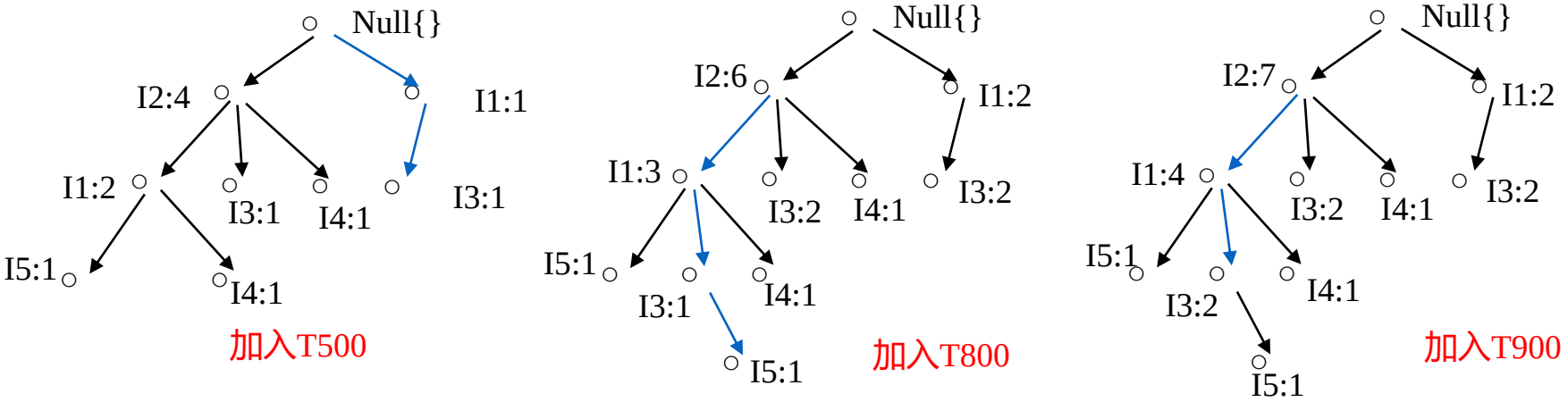
Step1：构造FP-树

TID	项ID的列表
T100	I2,I1,I5
T200	I2,I4
T300	I2,I3
T400	I2,I1,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I2,I1,I3,I5
T900	I2,I1,I3

第二次扫描数据库

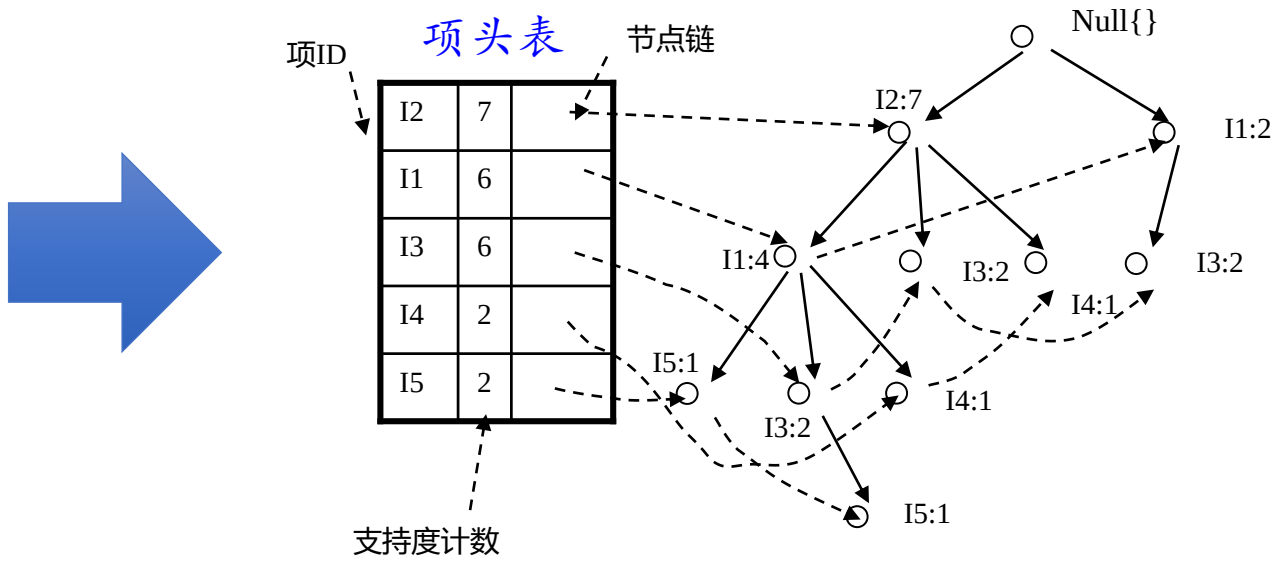


当为一个事务考虑增加分支时，沿共同前缀上的每个结点的计数增加1，为前缀之后的项创建结点和链接。



Step1：构造FP-树

TID	项ID的列表
T100	I2,I1,I5
T200	I2,I4
T300	I2,I3
T400	I2,I1,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I2,I1,I3,I5
T900	I2,I1,I3



- 对应于项的节点通过项头表中的节点链给出其指针
- 构建FP-树的过程是逐条读取交易记录并将其映射到树中的一条路径
- 项在交易记录中出现的顺序是固定的，因此共享某些项的交易记录对应的路劲是有重合的部分
- 路径重合的越多，FP-树对数据的压缩效果越好
- 频繁项集是从FP-树中挖掘获得

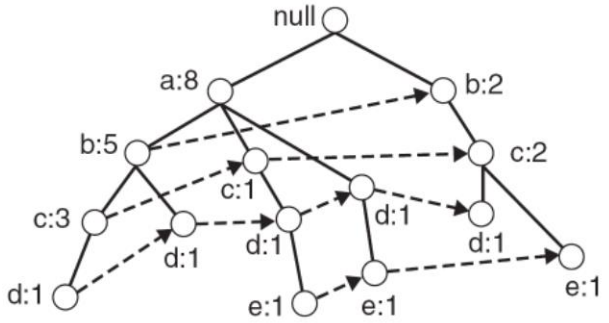
- ✓ **初始后缀模式**：长度为1的频繁模式
- ✓ **条件模式基**：在FP-树中与后缀模式一起出现的前缀路径集组成
- ✓ **条件FP-树**

FP树结构的特点

- 完整性(Completeness)
  - 保存了频繁模式相关的所有信息
  - 没有破坏事务数据中的任何长模式
- 紧致性(Compactness)
  - 去除了无关信息，即非频繁项集
  - 项在树中按支持度降序排列；出现频繁的项更可能被共享，从而有效地节省算法运行所需要的空间
  - 构造出的FP树大小不会超过原始数据库

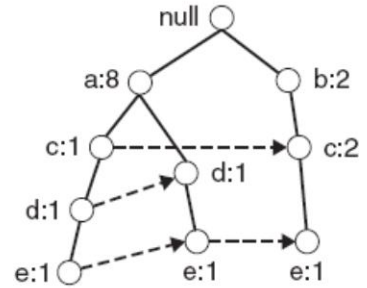
Step2:生成频繁项集

- FP-Growth算法从FP-树中挖掘频繁项集
- 自底向上算法：从FP-树的叶节点开始向根节点进行
  - 首先查看以e结尾的项集，然后de,…; d, cd…

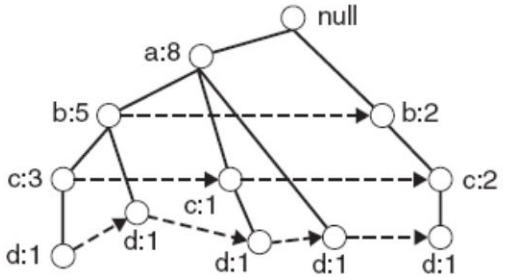


″Complete FP-tree

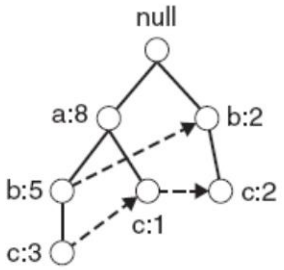
! **Example:** prefix path sub-trees



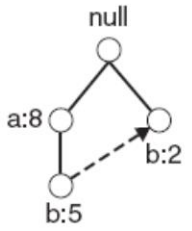
(a) Paths containing node e



(b) Paths containing node d



(c) Paths containing node c

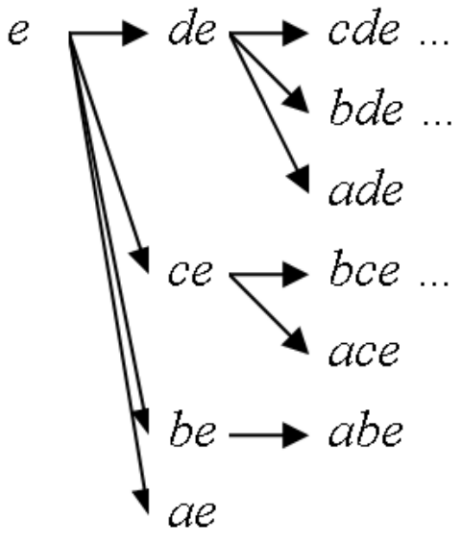


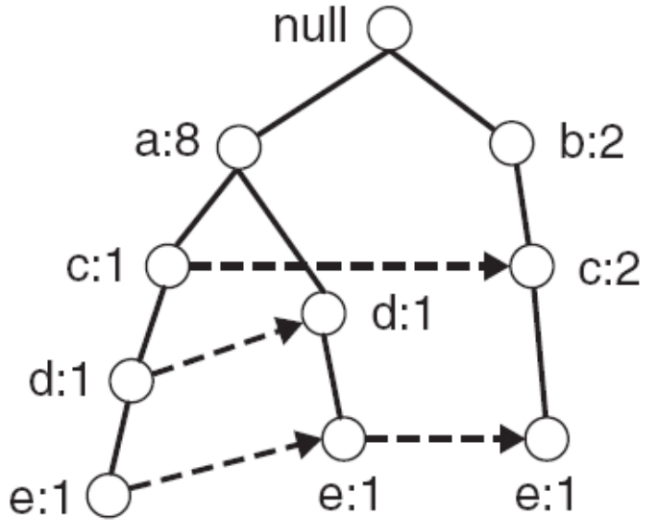
(d) Paths containing node b

(e) Paths containing node a

Step2:生成频繁项集

- 递归处理每个前缀路径子树从而抽取频繁项集
- 例如:

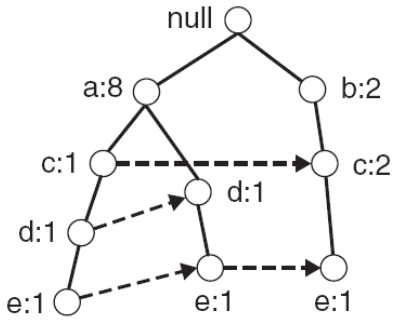




Prefix path sub-tree ending in *e*.

Step2:生成频繁项集

- 例： minSup=2,抽取所有包含e的频繁项集
  - （1）获得e的前缀路径子树



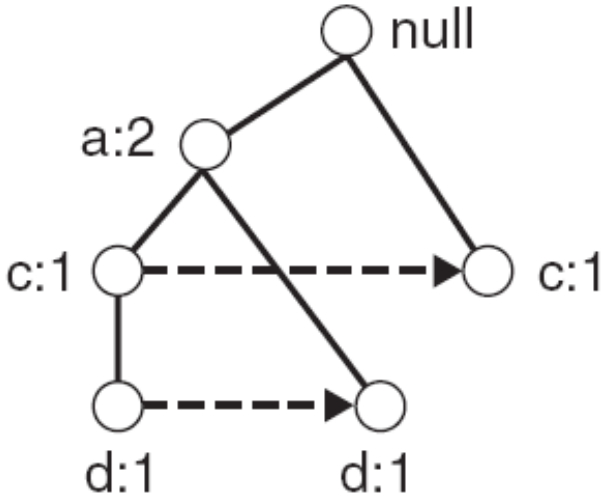
  - （2）检查e是否是频繁项
  - （3）如果e频繁，则寻找以e为后缀的频繁项集，即de,ce,be,ae (为了实现这一步，需要获得e的条件FP-树)

条件FP-树

- 给定一个项集I，基于数据集中所有包含该项集的记录构建的FP-树称为I的条件FP-树
- 例：e的条件FP-树

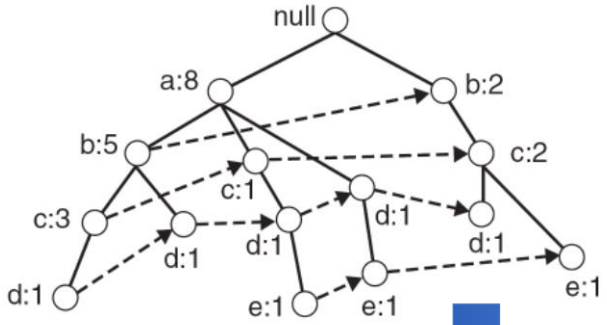
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

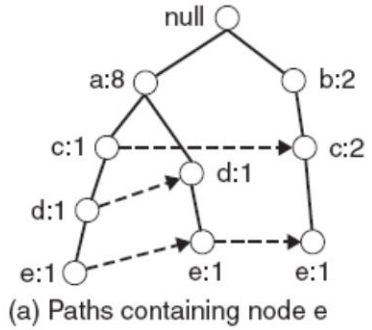
TID	Items
~~1~~	~~{a,b}~~
~~2~~	~~{b,c,d}~~
3	{a,c,d,~~e~~}
4	{a,d,~~e~~}
~~5~~	~~{a,b,c}~~
~~6~~	~~{a,b,c,d}~~
~~7~~	~~{a}~~
~~8~~	~~{a,b,c}~~
~~9~~	~~{a,b,d}~~
10	{b,c,~~e~~}



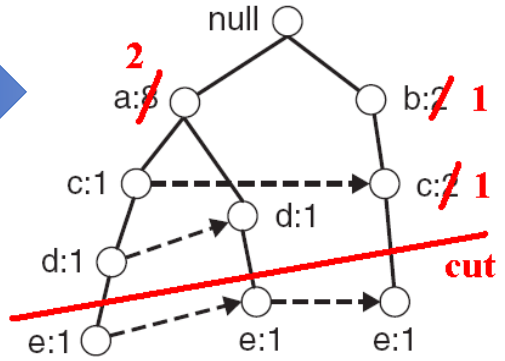
条件FP-树

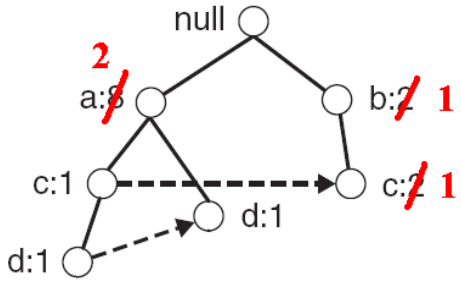
- 获取e的条件FP-树
  - 在e的前缀路径子树中更新节点的支持度计数
  - 删除e节点
  - 删除非频繁节点





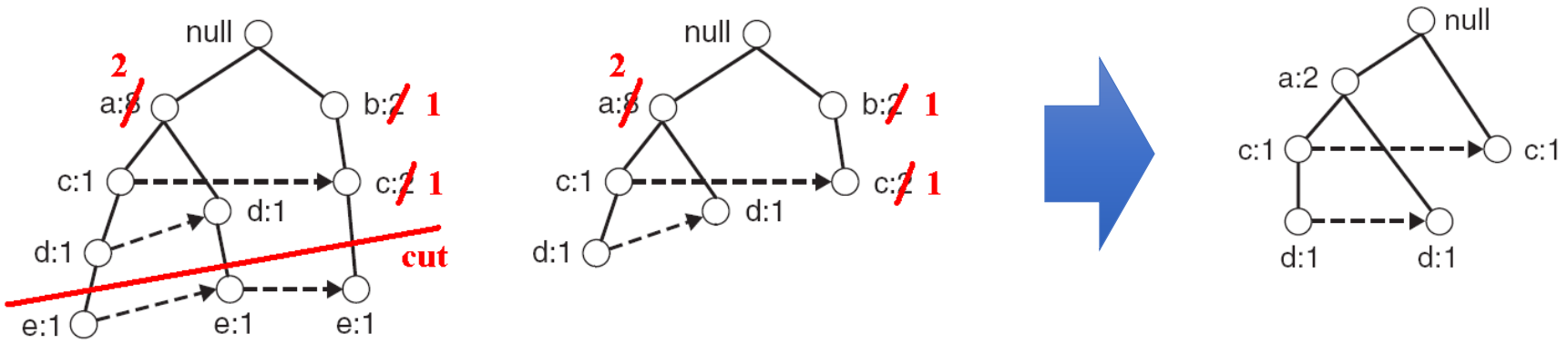
(a) Paths containing node e





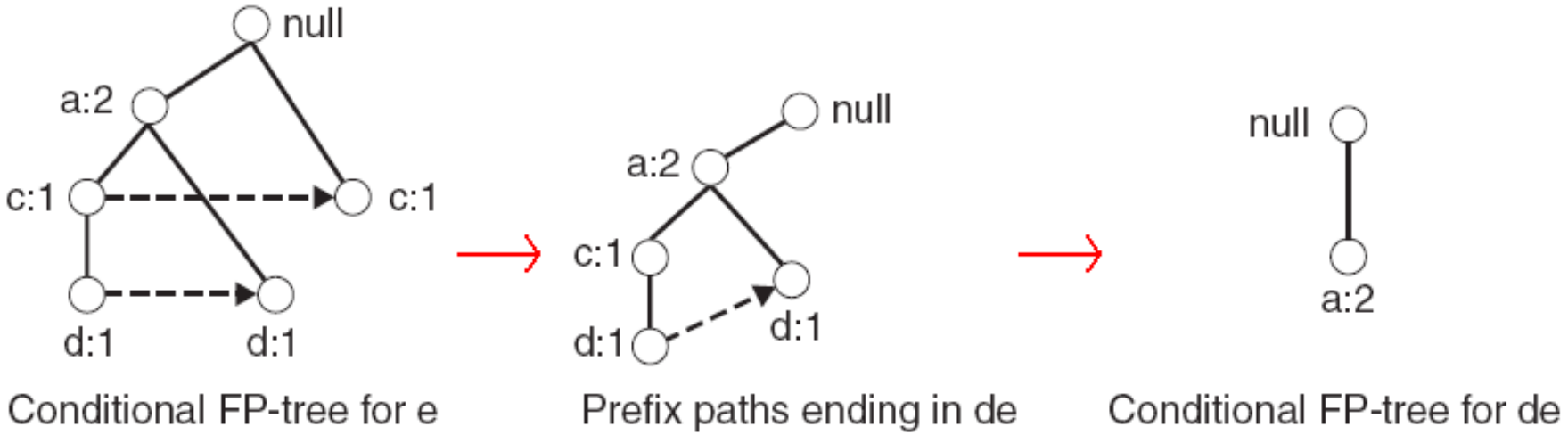
条件FP-树

- 获取e的条件FP-树
  - 在e的前缀路径子树中更新节点的支持度计数
  - 删除e节点
  - 删除非频繁节点



条件FP-树

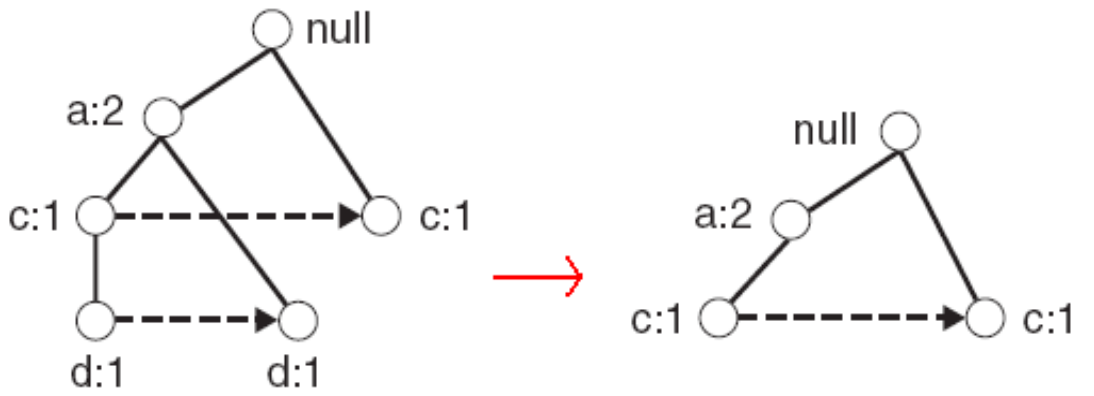
- 使用e的条件FP-树寻找以de,ce,ae为后缀的频繁项集



Conditional FP-tree for e　　Prefix paths ending in de　　Conditional FP-tree for de

条件FP-树

- 使用e的条件FP-树寻找以de,ce,ae为后缀的频繁项集



Conditional FP-tree for e　　Prefix paths ending in ce

条件FP-树

- 频繁项集结果

Suffix	Frequent Itemsets
e	{e}, {d,e}, {a,d,e}, {c,e},{a,e}
d	{d}, {c,d}, {b,c,d}, {a,c,d}, {b,d}, {a,b,d}, {a,d}
c	{c}, {b,c}, {a,b,c}, {a,c}
b	{b}, {a,b}
a	{a}

FP-Growth算法

算法：**FP-增长**。使用 FP-树，通过模式段增长，挖掘频繁模式。

输入：事务数据库 D；最小支持度阈值 *min_sup*。

输出：频繁模式的完全集。

方法：

1．按以下步骤构造 FP-树：

(a) 扫描事务数据库 *D* 一次。收集频繁项的集合 *F* 和它们的支持度。对 *F* 按支持度降序排序，结果为频繁项表 *L*。

(b) 创建 FP-树的根结点，以“null”标记它。对于 D 中每个事务 *Trans*，执行：

选择 *Trans* 中的频繁项，并按 *L* 中的次序排序。设排序后的频繁项表为[*p* | *P*]，其中，*p* 是第一个元素，而 *P* 是剩余元素的表。调用 **insert_tree**([*p* | *P*], *T*)。该过程执行情况如下。如果 *T* 有子女 *N* 使得 *N.item-name* = *p.item-name*，则 *N* 的计数增加 1；否则创建一个新结点 *N*，将其计数设置为 1，链接到它的父结点 *T*，并且通过结点链结构将其链接到具有相同 *item-name* 的结点。如果 *P* 非空，递归地调用 **insert_tree**(*P*, *N*)。

2．FP-树的挖掘通过调用 **FP_growth**(FP_tree, null)实现。该过程实现如下：

procedure **FP_growth**(*Tree*, α)
(1) **if** *Tree* 含单个路径 *P* **then**
(2) **for** 路径 *P* 中结点的每个组合（记作β）
(3) 产生模式$\beta \cup \alpha$，其支持度 *support* = β中结点的最小支持度；
(4) **else for each** a_i 在 *Tree* 的头部 {
(5) 产生一个模式$\beta = a_i \cup \alpha$，其支持度 *support* = a_i.*support*；
(6) 构造β的条件模式基，然后构造β的条件 FP-树 *Tree*β；
(7) **if** *Tree*β ≠ ∅ then
(8) 调用 FP_growth (*Tree*β, β)；}

FP- growth.py

FP增长方法小结

- FP增长方法将发现长频繁模式的问题转换成递归地搜索一些较短模式，然后连接后缀。
  - 使用最不频繁的项作后缀，提供了较好的选择性。大大降低了搜索开销。
- 对于挖掘长和短的频繁模式，它都是有效的和可规模化的，并且大约比Apriori算法快一个数量级.

FP增长方法在哪些步骤上降低了搜索开销?

当数据库很大时，无法构造基于内存的FP树时怎么办?

划分数据集，在每个集合上构造FP树，分别挖掘。

有效的和可伸缩的频繁项集挖掘方法

- Apriori算法：使用候选产生发现频繁项集
- 由频繁项集产生关联规则
- 提高Apriori算法的效率
- 不候选产生挖掘频繁项集
- 使用垂直数据格式挖掘频繁项集
- 挖掘闭频繁项集

使用垂直数据格式挖掘频繁项集(1/3)

- 首先，通过扫描一次数据集将水平格式({ TID: item_set })的数据转换成垂直格式({item: TID_set})。

TID	项ID的列表
T100	I1,I2,I5
T200	I2,I4
T300	I2,I3
T400	I1,I2,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I1,I2,I3,I5
T900	I1,I2,I3

垂直数据格式的1项集

项集	TID集
I1	{T100,T400,T500,T700,T800,T900}
I2	{T100,T200,T300,T400,T600,T800,T900}
I3	{T300,T500,T600,T700,T800,T900}
I4	{T200,T400}
I5	{T100,T800}

- 取每对频繁单项的TID集的交。设最小支持度计数为2。总共进行10次交运算，得到8个非空2项集，得到下表。

使用垂直数据格式挖掘频繁项集(2/3)

- 项集{I1,I4}和{I3,I5}均只包含一个事务，不属于频繁2项集。
- 根据Apriori性质，一个给定的3项集是频繁的，仅当它的每一个2项集 子集都是频繁的。
- 候选过程仅产生两个3项集：{I1,I2,I3}和{I1,I2,I5}。
- 取这些候选3项集任意两个对应的2项集的TID集的交，得到表2，其中只有2个频繁3项集：{I1,I2,I3：2}和{I1,I2,I5：2}。

表1：垂直数据格式的2项集

项集	TID集
{I1,I2}	{T100,T400,T800,T900}
{I1,I3}	{T500,T700,T800,T900}
{I1,I4}	{T400}
{I1,I5}	{T100,T800}
{I2,I3}	{T300,T600,T800,T900}
{I2,I4}	{T200,T400}
{I2,I5}	{T100,T800}
{I3,I5}	{T900}

表2：垂直数据格式的3项集

项集	TID集
{I1,I2,I3}	{T800,T900}
{I1,I2,I5}	{T100,T800}

使用垂直数据格式挖掘频繁项集(3/3)

- 扫描一次数据集，将水平格式({ TID: item_set })的数据转换成垂直格式({item: TID_set})。项集的支持度计数=项集的TID集的长度。
- 从 k=1 开始，根据Apriori性质，使用频繁k项集来构造候选(k+1)项集。通过频繁k项集的TID集的交运算，计算对应的(k+1)项集的TID集。重复该过程，每次k增加1，直到不能再找到频繁集或候选项集。
- 该方法的优点
  - 由频繁 k项集产生候选 (k+1) 项集时利用了Apriori性质.
  - 不需要扫描数据库来确定(k+1) 项集的支持度。每个k项集的TID集携带了计算该支持度所需的完整信息。

有效的和可伸缩的频繁项集挖掘方法

- Apriori算法：使用候选产生发现频繁项集
- 由频繁项集产生关联规则
- 提高Apriori算法的效率
- 不候选产生挖掘频繁项集
- 使用垂直数据格式挖掘频繁项集
- 挖掘闭频繁项集

闭频繁项集和极大频繁项集

- 一个长项集将包含组合个数较短的频繁子项集
  - 例如，$\{a_1,a_2,\ldots,a_{100}\}$包含$C_{100}^{1}=100$个1项集；$C_{100}^{2}$个2项集，…
  - 长度为100的项集所有子项集数量

    $C_{100}^{1}+C_{100}^{2}+\ldots+C_{100}^{100}=2^{100}-1\approx 1.27\times 10^{30}$
- 在挖掘过程中，项集个数太大，将导致计算机无法计算和存储
- 为了克服这个问题，引入闭频繁项集和极大频繁项集的概念
- An itemset X is closed if there exists no super-pattern Y ⊃X, with the same support as X (proposed by Pasquier, et al. @ ICDT'99)
- An itemset X is a maximal pattern if X is frequent and there exists no frequent super-pattern Y ⊃X (proposed by Bayardo @ SIGMOD'98)

闭频繁项集和极大频繁项集

- 如果不存在X的真超项集Y在S中有与X相同的支持度计数，则称项集X在数据集S中是闭的。
- 如果X在S中是闭的和频繁的，则称项集X是数据集S中的闭频繁项集。
- 如果X是频繁的，并且不存在超项集Y使得X真包含于Y并且Y在S中是频繁的，则称项集X是S中的极大频繁项集。

闭频繁项集与极大频繁项集的区别是什么?

- ✓ 闭频繁项集包含了频繁项集的完整信息
- ✓ 极大频繁项集不包含对应频繁项集的完整支持度信息。

举例

- 假定事务数据库只有两个事务:

  {<a1,a2,…a100>;<a1,a2,…,a50>}。设min_sup = 1

  - 两个闭频繁项集:

    C = { {a1,a2,…,a100}:1;{a1,a2,…,a50}:2 }

  - 一个极大频繁项集:

    M={{a1,a2,…,a100}:1}

  - 可以从C推出所有频繁项集及其支持度:
    - {a2,a45:2},因为{a2,a45}是{a1,a2,…,a50:2 }的子集;
    - {a8,a55:1},因为{a8,a55}不是{a1,a2,…,a50:2} 的子集，而是{a1,a2,…,a100:1}的子集。

提纲

1 基本概念

2 频繁项集挖掘方法

3 哪些模式是有趣的：模式评估方法

4 小结