FP-growth算法

01 关联规则与Apriori算法

• 关联规则（Association Rules）反映一个事物与其他事物之间的相互依存性和关联性。

•步骤：①先从数据集中找出所有的频繁项集，它们的支持度均大于等于最小支持度阈值（min_sup）；②由这些频繁项集产生关联规则，计算它们的置信度，然后保留那些置信度大于等于最小置信度阈值（min_conf）的关联规则。

•Apriori算法通过连接步和剪枝步找出频繁项集。

01 关联规则与Apriori算法

编号	商品列表
1	ABC
2	ABD
3	ACD
4	ABCE
5	ACE
6	BDE
7	ABCD

min_sup=3/7

第1次扫描

C1	支持度
{A}	6/7
{B}	5/7
{C}	5/7
{D}	4/7
{E}	3/7

L1
{A}
{B}
{C}
{D}
{E}

连接
第2次扫描

C2	支持度
{A,B}	4/7
{A,C}	5/7
{A,D}	3/7
{A,E}	2/7
{B,C}	3/7
{B,D}	3/7
{B,E}	2/7
{C,D}	2/7
{C,E}	1/7
{D,E}	1/7

L2
{A,B}
{A,C}
{A,D}
{B,C}
{B,D}

连接

C3
{A,B,C}
{A,B,D}
{A,C,D}
{B,C,D}

剪枝
第3次扫描

C3	支持度
{A,B,C}	3/7
{A,B,D}	2/7
{B,C,D}	1/7

L3
{A,B,C}

01 关联规则与Apriori算法

- Apriori算法的缺点：

  多次扫描数据库，产生巨大数量的候选集，繁琐的支持度计算……

- 改进方法：

  FP-growth算法。

02 FP-growth算法

FP-growth算法不产生候选项集，而是采用分而治之的策略：

首先，压缩数据库，并将频繁项放入频繁模式树（FP-树），它仍保留项集的关联信息。

然后，将压缩数据库分成多个条件数据库，每个条件数据库关联到一个频繁项集，并分开来对其进行挖掘。

02 FP-growth算法

该算法只需要扫描两次数据库。

第一次扫描，得到所有频繁项及其支持度计数，并在每个事务中将频繁项依据支持度计数进行降序排列；

第二次扫描，将每个事务的项都合并进FP-树，对在不同事务中出现的公共项进行计数。

03 FP-growth算法实例

假设某商场的交易记录表为表1，并且假设最小支持度计数为3。挖掘频繁项集。

表1 交易记录表

编号	商品列表
1	f, a, c, d, g, i, m, p
2	a, b, c, f, l, m, o
3	b, f, h, j, o
4	b, c, k, s, p
5	a, f, c, e, l, p, m, n

03 FP-growth算法实例

冒号后的数字表示支持度计数

第一次扫描数据库，得到频繁项列表Flist={f:4,c:4,a:3,b:3,m:3,p:3}。

将频繁项按支持度计数降序排列。

表2 交易频繁项表

编号	商品列表	排序后的频繁项
1	f,a,c,d,g,i,m,p	f,c,a,m,p
2	a,b,c,f,l,m,o	f,c,a,b,m
3	b,f,h,j,o	f,b
4	b,c,k,s,p	c,b,p
5	a,f,c,e,l,p,m,n	f,c,a,m,p

03 FP-growth算法实例

编号	排序后的频繁项
1	f,c,a,m,p
2	f,c,a,b,m
3	f,b
4	c,b,p
5	f,c,a,m,p

第二次扫描数据库，构建FP-树。

首先创建树的根结点，标记为null。

扫描第一个事务时，得到树的第一个分支：

<(f:1),(c:1),(a:1),(m:1),(p:1)>,该分支具有5个节点。

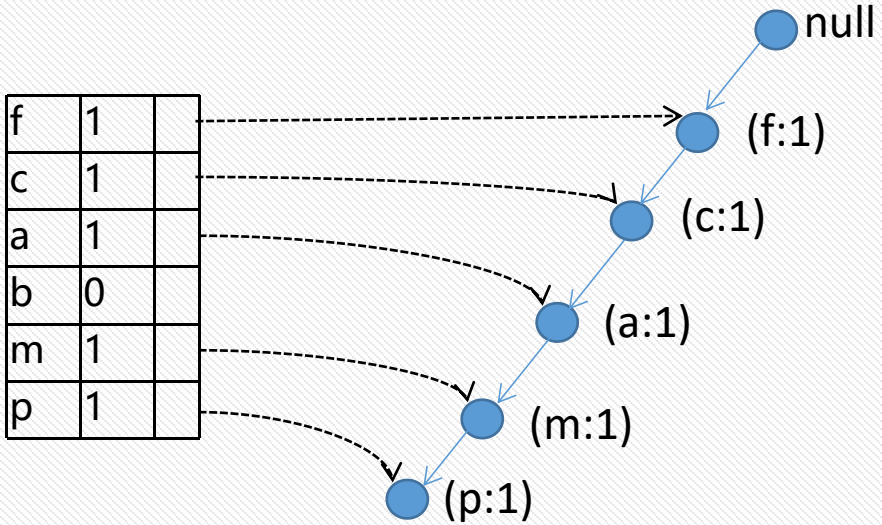


图1 读入第一个事务

03 FP-growth算法实例

编号	排序后的频繁项
1	f,c,a,m,p
2	f,c,a,b,m
3	f,b
4	c,b,p
5	f,c,a,m,p

扫描第二个事务时，得到树的第二个分支：<(f:1),(c:1),(a:1),(b:1),(m:1)>。该分支与已有的第一个分支共享一个共同的前缀<(f:1),(c:1),(a:1)>，因此，前缀中每个节点计数都增加1，并且还需要再创建两个新节点。

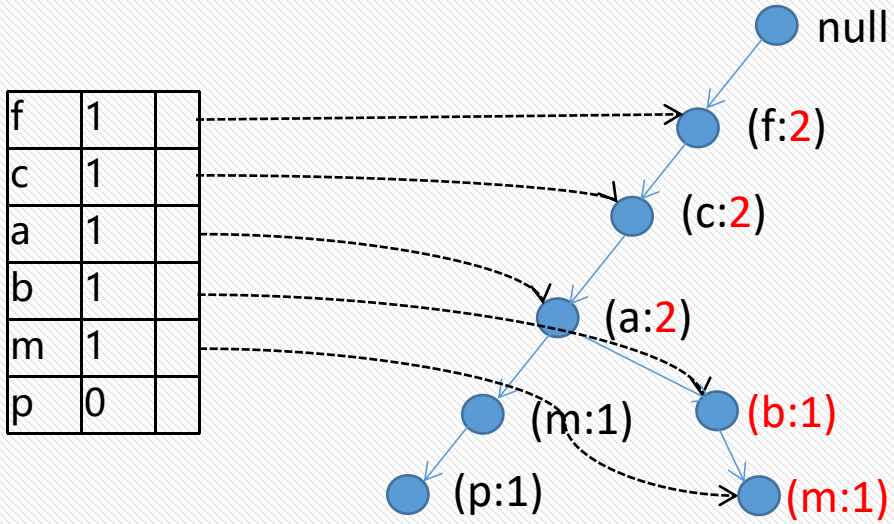
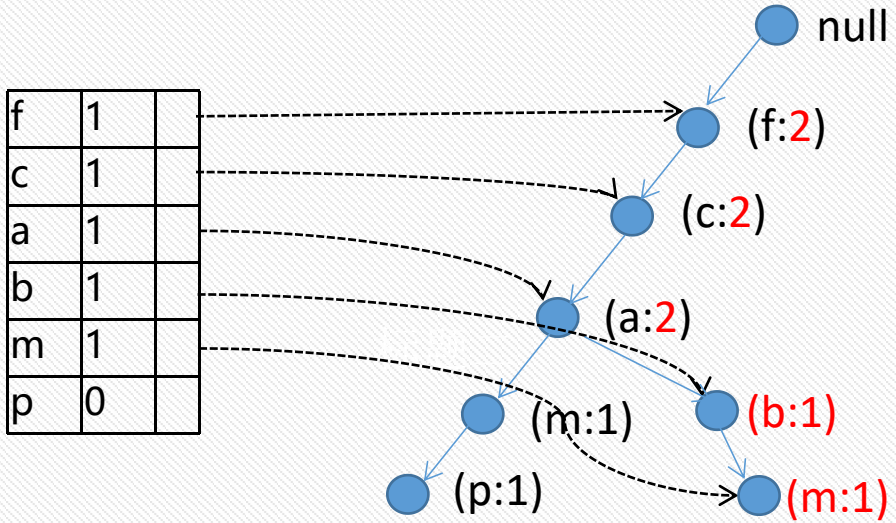


图2 读入第二个事务

03 FP-growth算法实例

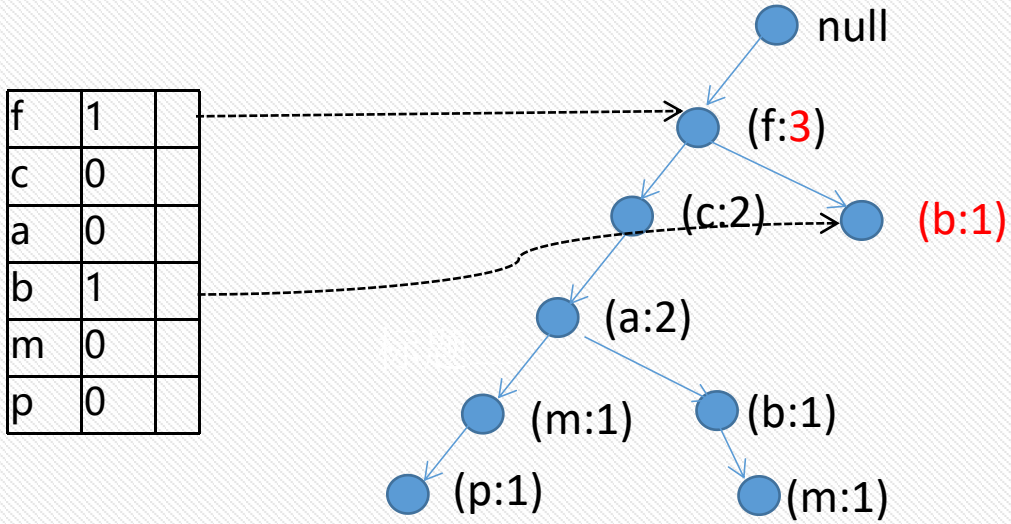
编号	排序后的频繁项
1	f,c,a,m,p
2	f,c,a,b,m
3	f,b
4	c,b,p
5	f,c,a,m,p



读入第二个事务

03 FP-growth算法实例

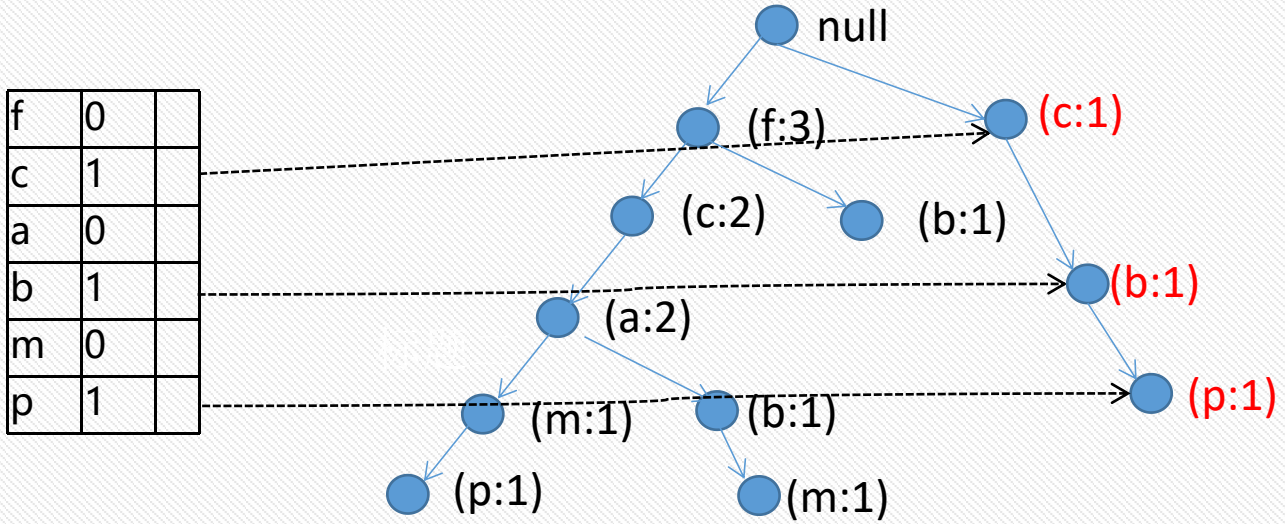
编号	排序后的频繁项
1	f,c,a,m,p
2	f,c,a,b,m
3	f,b
4	c,b,p
5	f,c,a,m,p



读入第三个事务

03 FP-growth算法实例

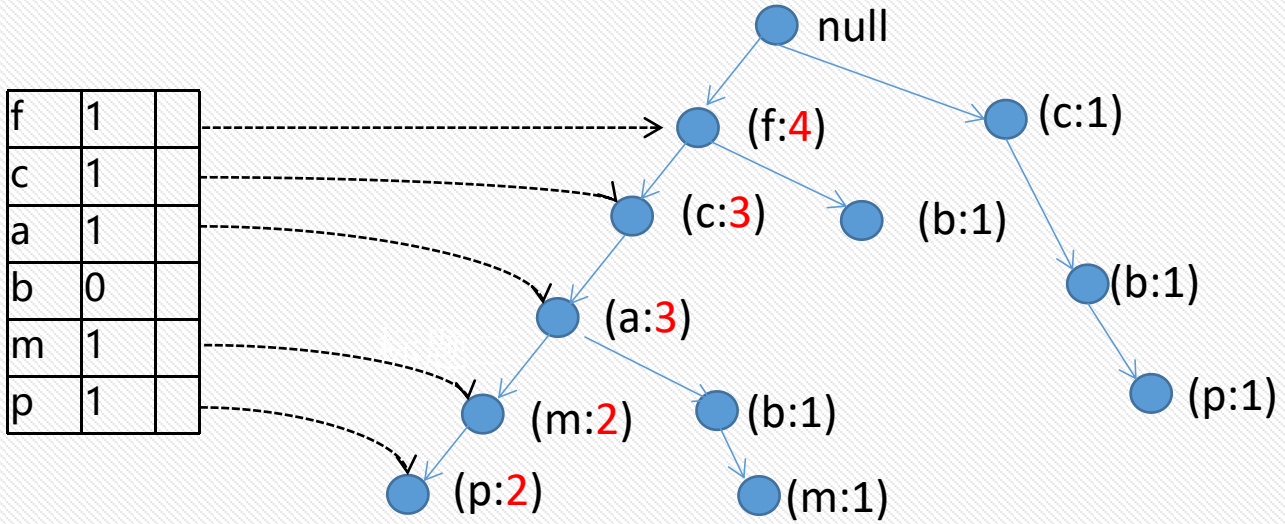
编号	排序后的频繁项
1	f,c,a,m,p
2	f,c,a,b,m
3	f,b
4	c,b,p
5	f,c,a,m,p



读入第四个事务

03 FP-growth算法实例

编号	排序后的频繁项
1	f,c,a,m,p
2	f,c,a,b,m
3	f,b
4	c,b,p
5	f,c,a,m,p



读入第五个事务

03 FP-growth算法实例

编号	排序后的频繁项
1	f,c,a,m,p
2	f,c,a,b,m
3	f,b
4	c,b,p
5	f,c,a,m,p

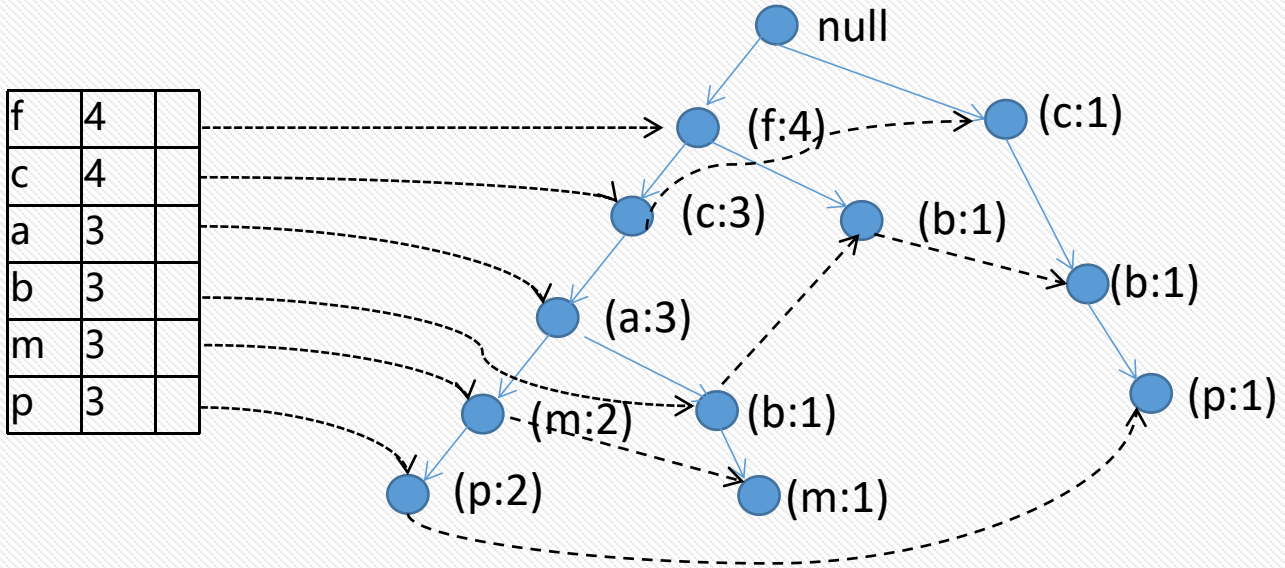


图3 由表生成的FP-树

挖掘数据库中的频繁项集→挖掘FP-树

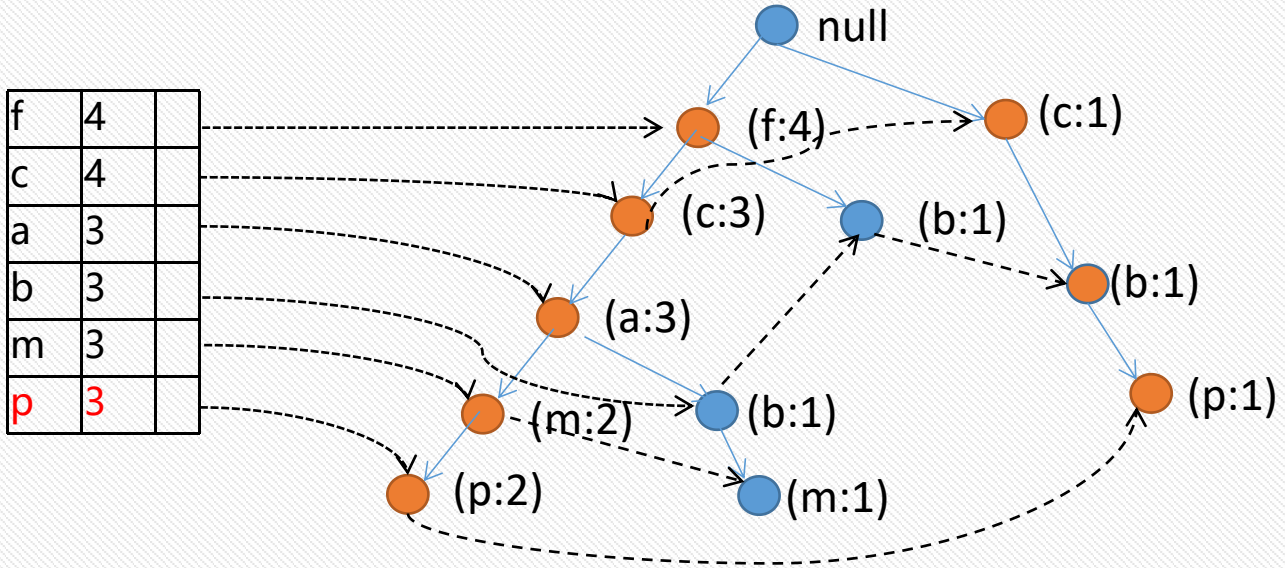
03 FP-growth算法实例

FP-树的挖掘过程为：

由长度为1的频繁模式（初始后缀模式）开始，构造它的条件模式基。条件模式基是一个子数据库，由FP-树中与该后缀模式一起出现的前缀路径集组成。然后由此构造频繁模式的条件FP-树，并递归地在该树上进行挖掘。

03 FP-growth算法实例

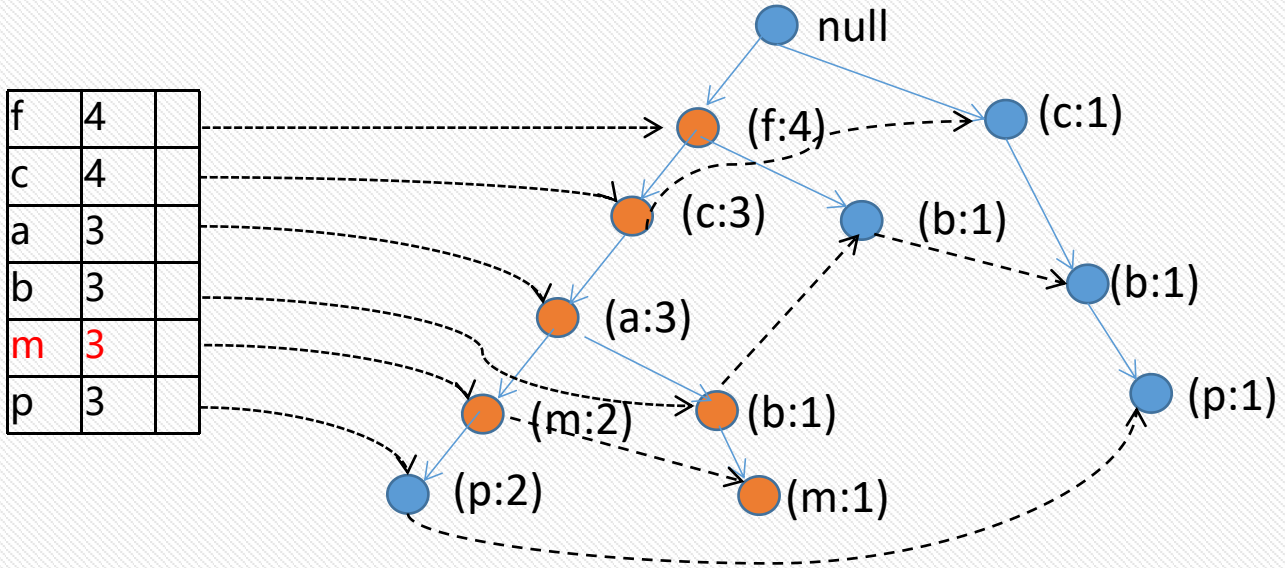
编号	排序后的频繁项
1	f,c,a,m,p
2	f,c,a,b,m
3	f,b
4	c,b,p
5	f,c,a,m,p



对于p，它出现在FP-树的两个分支中。考虑以p为后缀，它的两个对应前缀路径就是{f:2,c:2,a:2,m:2}和{c:1,b:1}，它们形成p的条件模式基。对于这样一个子数据库，建立条件FP-树。条件FP-树只有一个分支{c:3},因为其他支持度计数均小于最小支持度计数3。所以，与p有关的频繁模式为：**{p:3}和{cp:3}**。

03 FP-growth算法实例

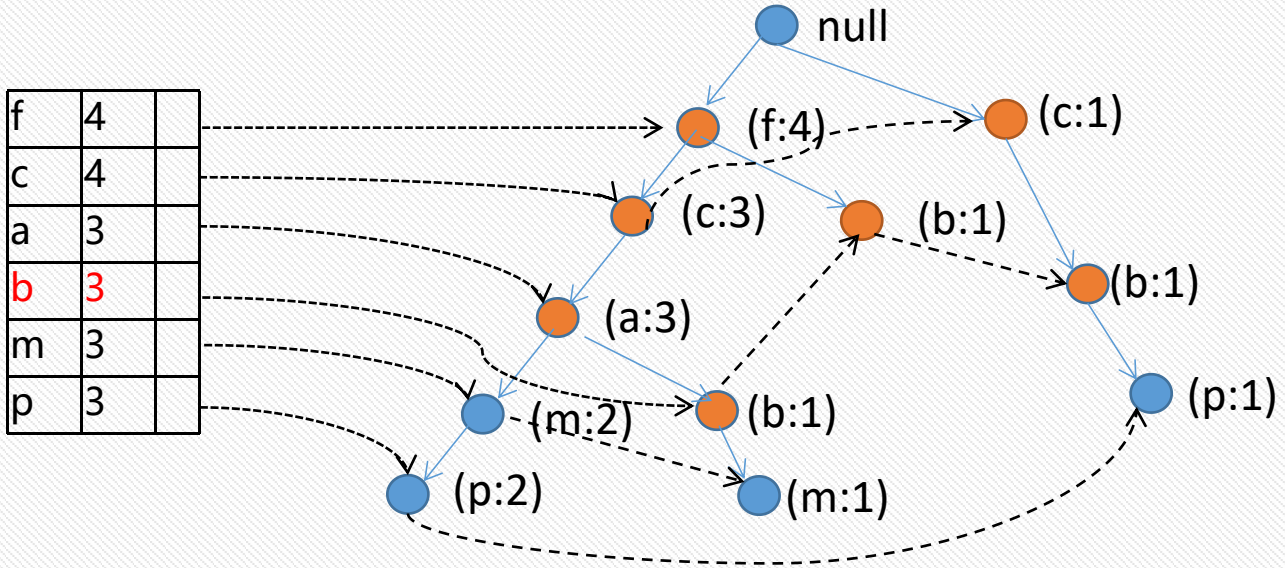
编号	排序后的频繁项
1	f,c,a,m,p
2	f,c,a,b,m
3	f,b
4	c,b,p
5	f,c,a,m,p



对于m，它出现在FP-树的两个分支中。考虑以m为后缀，它的两个对应前缀路径就是{f:2,c:2,a:2}和{f:1,c:1,a:1,b:1}，它们形成m的条件模式基。对于这样一个子数据库，建立条件FP-树。条件FP-树只有一个分支{f:3,c:3,a:3}。递归挖掘得与m有关的频繁模式为：**{m:3},{am:3},{cm:3},{fm:3},{cam:3},{fam:3},{fcm:3},{fcam:3}**。

03 FP-growth算法实例

编号	排序后的频繁项
1	f,c,a,m,p
2	f,c,a,b,m
3	f,b
4	c,b,p
5	f,c,a,m,p



对于b，它出现在FP-树的三个分支中。它的对应前缀路径{f:1,c:1,a:1},{f:1}和{c:1}，它们形成b的条件模式基。对于这样一个子数据库，建立条件FP-树。因为所有支持度计数均小于最小支持度计数，所以没有频繁项。所以与b有关的频繁模式为：**{b:3}**。

03 FP-growth算法实例

表3 创建条件模式基挖掘FP-树

项	条件模式基	条件FP-树	产生的频繁模式
p	{fcam:2},{cb:1}	{c:3}	{p:3},{cp:3}
m	{fca:2},{fcab:1}	{f:3,c:3,a:3}	{m:3},{am:3},{cm:3},{fm:3}, {cam:3},{fam:3},{fcm:3},{fcam:3}
b	{fca:1},{f:1},{c:1}	null	{b:3}
a	{fc:3}	{f:3,c:3}	{a:3},{ca:3},{fa:3},{fca:3}
c	{f:3}	{f:3}	{c:3},{fc:3}

04 FP-growth算法的优点

1、FP-growth算法仅仅遍历了2次数据库，大大节省了扫描数据库的时间。

2、选用了分治策略，把挖掘的长频繁模式转换成递归挖掘短模式问题，再与后缀相连。

谢谢！