关联规则

2021/11/29

目录

关联规则

- 就餐饮企业而言，经常会碰到这样的问题：

客户在餐厅点餐时，面对菜单中大量的菜品信息，往往无法迅速找到满意的菜品，既增加了点菜的时间，也降低了客户的就餐体验。

- 实际上，菜品的合理搭配是有规律可循的：顾客的饮食习惯、菜品的荤素和口味，有些菜品之间是相互关联的，而有些菜品之间是对立或竞争关系（负关联）。这些规律都隐藏在大量的历史菜单数据中，如果能够通过数据挖掘发现客户点餐的规则，就可以快速识别客户的口味，当他下了某个菜品的订单时推荐相关联的菜品，引导客户消费，提高顾客的就餐体验和餐饮企业的业绩水平。

关联规则

- 关联规则分析也成为购物篮分析，最早是为了发现超市销售数据库中不同的商品之间的关联关系。例如一个超市的经理想要更多地了解顾客的购物习惯，比如“哪组商品可能会在一次购物中同时购买？”或者“某顾客购买了个人电脑，那该顾客三个月后购买数码相机的概率有多大？”他可能会发现如果购买了面包的顾客同时非常有可能会购买牛奶，这就导出了一条关联规则“面包=>牛奶”，其中面包称为规则的前项，而牛奶称为后项。通过对面包降低售价进行促销，而适当提高牛奶的售价，关联销售出的牛奶就有可能增加超市整体的利润。
- 关联规则分析是数据挖掘中最活跃的研究方法之一，目的是在一个数据集中找出各项之间的关联关系，而这种关系并没有在数据中直接表示出来。

关联规则——常用关联规则算法

- 常用关联算法如下表：

算法名称	算法描述
Apriori	关联规则最常用也是最经典的挖掘频繁项集的算法，其核心思想是通过连接产生候选项及其支持度然后通过剪枝生成频繁项集。
FP-Tree	针对Apriori算法的固有的多次扫面事务数据集的缺陷，提出的不产生候选频繁项集的方法。Apriori和FP-Tree都是寻找频繁项集的算法。
Eclat算法	Eclat算法是一种深度优先算法，采用垂直数据表示形式，在概念格理论的基础上利用基于前缀的等价关系将搜索空间划分为较小的子空间。
灰色关联法	分析和确定各因素之间的影响程度或是若干个子因素（子序列）对主因素（母序列）的贡献度而进行的一种分析方法。

- 本节重点详细介绍Apriori算法。

定义：关联规则

- 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。
- 关联分析可以应用于生物信息学、医疗诊断、网页挖掘、科学数据分析等

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

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}

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

- 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

Example:

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

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

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

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
- 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
  - 从包含d个项的数据集提取的可能规则的总数$R=3^d-2^{d+1}+1$，如果d等于6，则R=602

挖掘关联规则（Mining Association Rules）

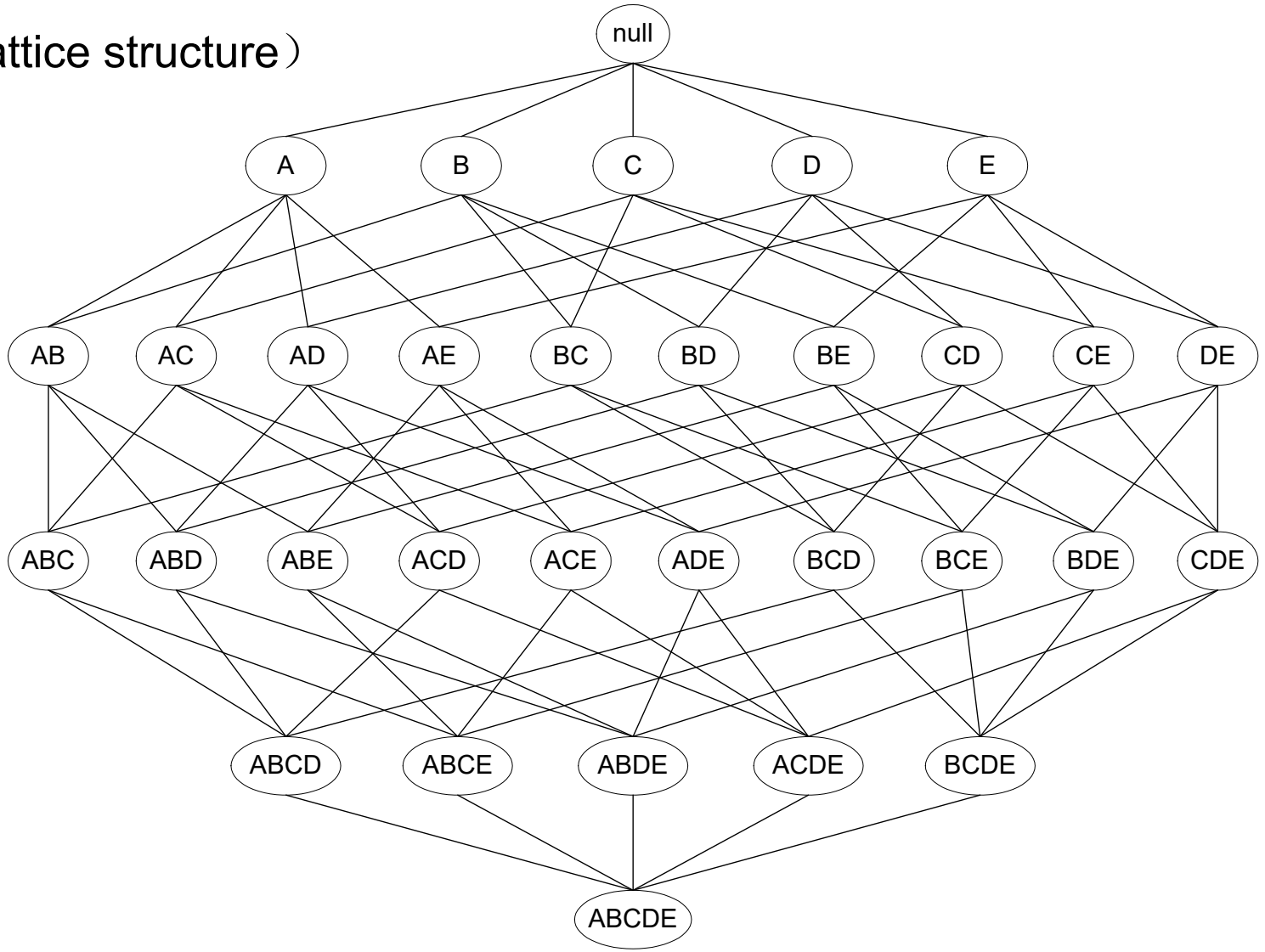
- 大多数关联规则挖掘算法通常采用的一种策略是，将关联规则挖掘任务分解为如下两个主要的子任务:

1. 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。

2. 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）。

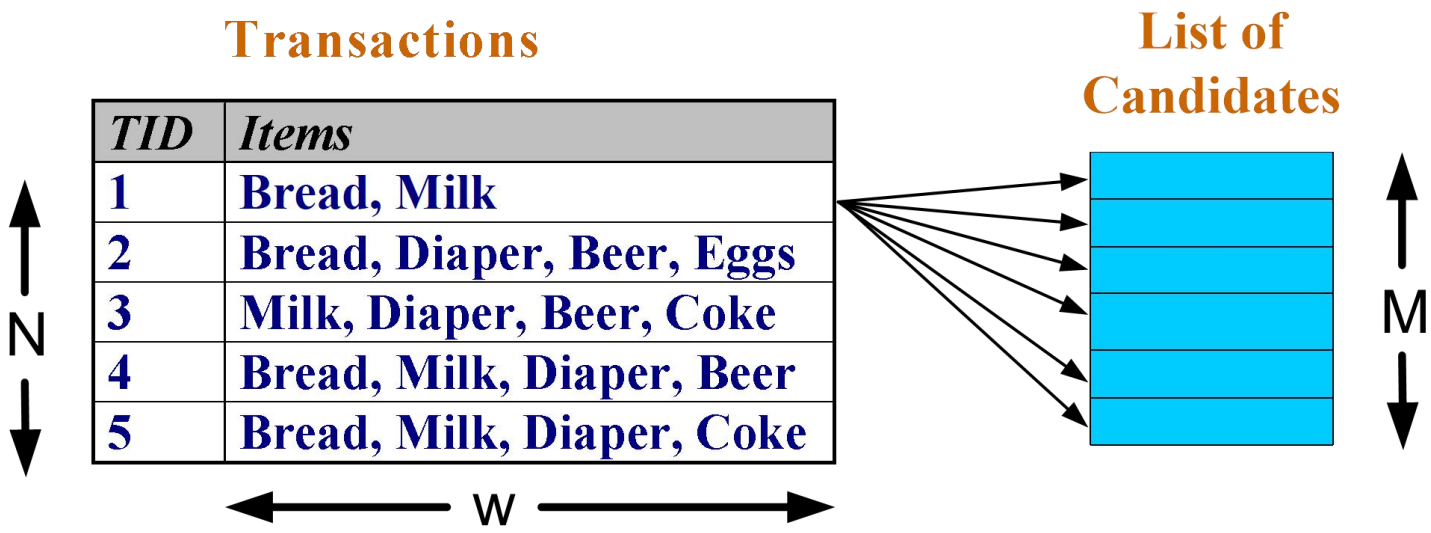
频繁项集产生（Frequent Itemset Generation）

格结构（lattice structure）



频繁项集产生（Frequent Itemset Generation）

- Brute-force 方法:
  - 把格结构中每个项集作为候选项集
  - 将每个候选项集和每个事务进行比较，确定每个候选项集的支持度计数。



  - 时间复杂度 ~ O(NMw)，这种方法的开销可能非常大。

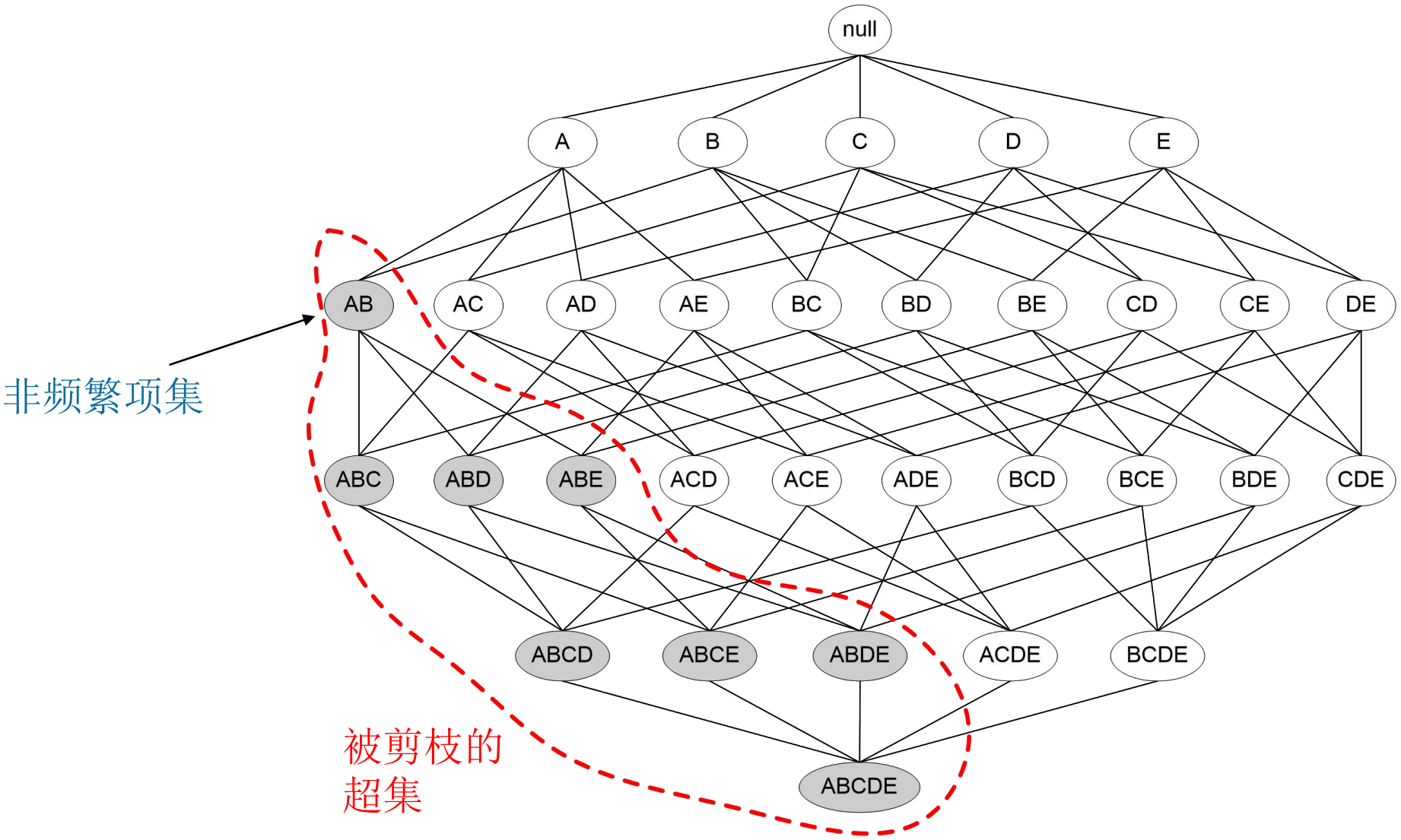
降低产生频繁项集计算复杂度的方法

- 减少候选项集的数量 (M)
  - 先验(apriori)原理
- 减少比较的次数 (NM)
  - 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

先验原理（Apriori principle）

- 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:
  - 这种基于支持度度量修剪指数搜索空间的策略称为基于支持度的剪枝（support-based pruning）
  - 这种剪枝策略依赖于支持度度量的一个关键性质，即一个项集的支持度决不会超过它的子集的支持度。这个性质也称为支持度度量的反单调性（anti-monotone）。

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算法的频繁项集产生

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

Apriori算法的频繁项集产生

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Pairs (2-itemsets)

支持度阈值=60%
最小支持度计数 = 3

枚举所有项集将产生
$C_6^1 + C_6^2 + C_6^3$ =41个候选

而使用先验原理，将减少为
$C_6^1 + C_4^2 + 1$ =13

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3

Apriori算法

算法 6.1 *Apriori* 算法的频繁项集产生

```
1:  k = 1
2:  F_k = {i | i∈I ∧ σ({i}) ≥ N × minsup}      {发现所有的频繁 1-项集}
3:  repeat
4:    k = k+1
5:    C_k = apriori-gen(F_{k-1})      {产生候选项集}
6:    for 每个事务 t∈T do
7:      C_t = subset(C_k, t)      {识别属于 t 的所有候选}
8:      for 每个候选项集 c∈C_t do
9:        σ(c) = σ(c) + 1      {支持度计数增值}
10:     end for
11:   end for
12:   F_k = {c | c∈C_k ∧ σ(c) ≥ N × minsup}      {提取频繁 k-项集}
13: until F_k = ∅
14: Result = ∪F_k
```

Apriori算法

- Apriori算法的频繁项集产生的部分有两个重要的特点：
  - 它是一个逐层算法。即从频繁1-项集到最长的频繁项集，它每次遍历项集格中的一层
  - 它使用产生-测试策略来发现频繁项集。在每次迭代，新的候选项集由前一次迭代发现的频繁项集产生，然后对每个候选的支持度进行计数，并与最小支持度阈值进行比较。
  - 该算法需要的总迭代次数是$k_{max}+1$，其中k_{max}是频繁项集的最大长度

Apriori算法

- 蛮力方法
  - 蛮力方法把所有的k-项集都看作可能的候选，然后使用候选剪枝除去不必要的候选
  - 第k层产生的候选项集的数目为 C_d^k
  - 虽然候选产生是相当简单的，但是候选剪枝的开销极大，因为必须考察的项集数量太大。
  - 设每一个候选项集所需的计算量为O（k），这种方法　的总复杂度为

$$O(\sum_{k=1}^{d} kC_d^k) = O(d \cdot 2^{d-1})$$

候选集的产生与剪枝

项

项
啤酒
面包
可乐
尿布
牛奶
鸡蛋

候选产生

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

候选剪枝

项集
{面包，尿布，牛奶}

图 6-6　产生候选 3-项集的蛮力方法

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Pairs (2-itemsets)

支持度阈值=60%
最小支持度计数 = 3

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3

枚举所有项集将产生
${}^6C_1 + {}^6C_2 + {}^6C_3 = 41$个候选
而使用先验原理，将较少为
6 + 6 + 1 = 13

候选集的产生与剪枝

- $F_{k-1} \times F_1$方法
  - 这种方法用其他频繁项来扩展每个频繁（k-1）-项集
  - 这种方法将产生$O(|F_{k-1}| \times |F_1|)$个候选k-项集，其中$|F_j|$表示频繁j-项集的个数。这种方法总复杂度是$O(\sum_k k|F_{k-1}||F_1|)$
  - 这种方法是完全的，因为每一个频繁k-项集都是由一个频繁（k-1）-项集和一个频繁1-项集组成的。因此，所有的频繁k-项集是这种方法所产生的候选k-项集的一部分。
  - 然而，这种方法很难避免重复地产生候选项集。
  - 如：{面包，尿布，牛奶}不仅可以由合并项集{面包，尿布}和{牛奶}得到，而且还可以由合并{面包，牛奶}和{尿布}得到，或由合并{尿布，牛奶}和{面包}得到。

候选集的产生与剪枝

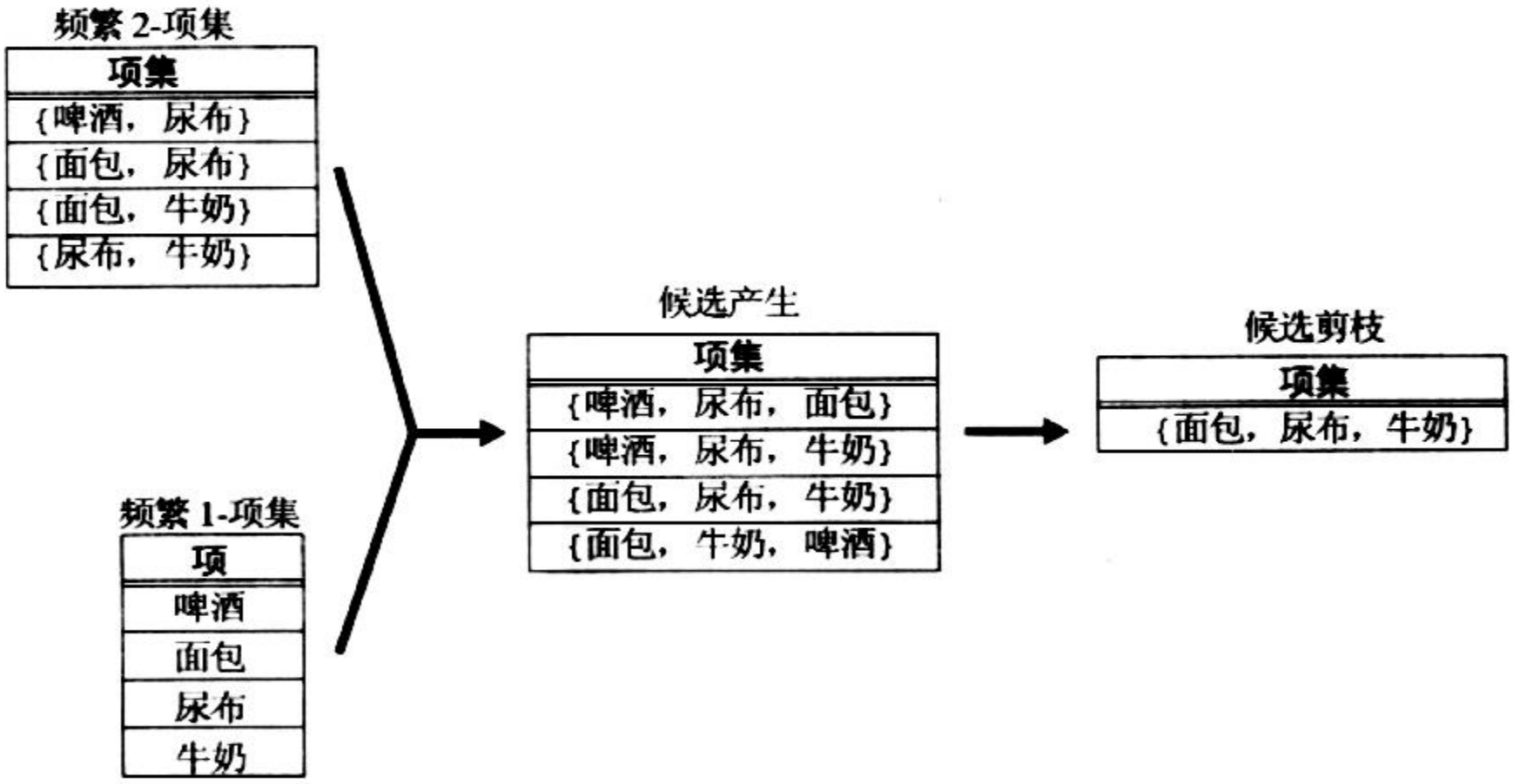


图 6-7　通过合并频繁(k–1)-项集和频繁 1-项集生成和剪枝候选 k-项集。注意：某些候选是不必要，因为它们的子集是非频繁的

候选集的产生与剪枝

- 避免产生重复的候选项集的一种方法是确保每个频繁项集中的项以字典序存储，每个频繁（k-1）-项集X只用字典序比X中所有的项都大的频繁项进行扩展

  如：项集{面包，尿布}可以用项集{牛奶}扩展，因为“牛奶”（milk）在字典序下比“面包”（Bread）和“尿布”（Diapers）都大。

- 尽管这种方法比蛮力方法有明显改进，但是仍然产生大量不必要的候选。

  例如，通过合并{啤酒，尿布}和{牛奶}而得到的候选是不必要的。因为它的子集{啤酒，牛奶}是非频繁的。

候选集的产生与剪枝

- $F_{k-1} \times F_{k-1}$方法
  - 这种方法合并一对频繁（k-1）-项集，仅当它们的前k-2个项都相同。

    如频繁项集{面包，尿布}和{面包，牛奶}合并，形成了候选3-项集{面包，尿布，牛奶}。算法不会合并项集{啤酒，尿布}和{尿布，牛奶}，因为它们的第一个项不相同。
  - 然而，由于每个候选都由一对频繁（k-1）-项集合并而成，因此，需要附加的候选剪枝步骤来确保该候选的其余k-2个子集是频繁的。

候选集的产生与剪枝

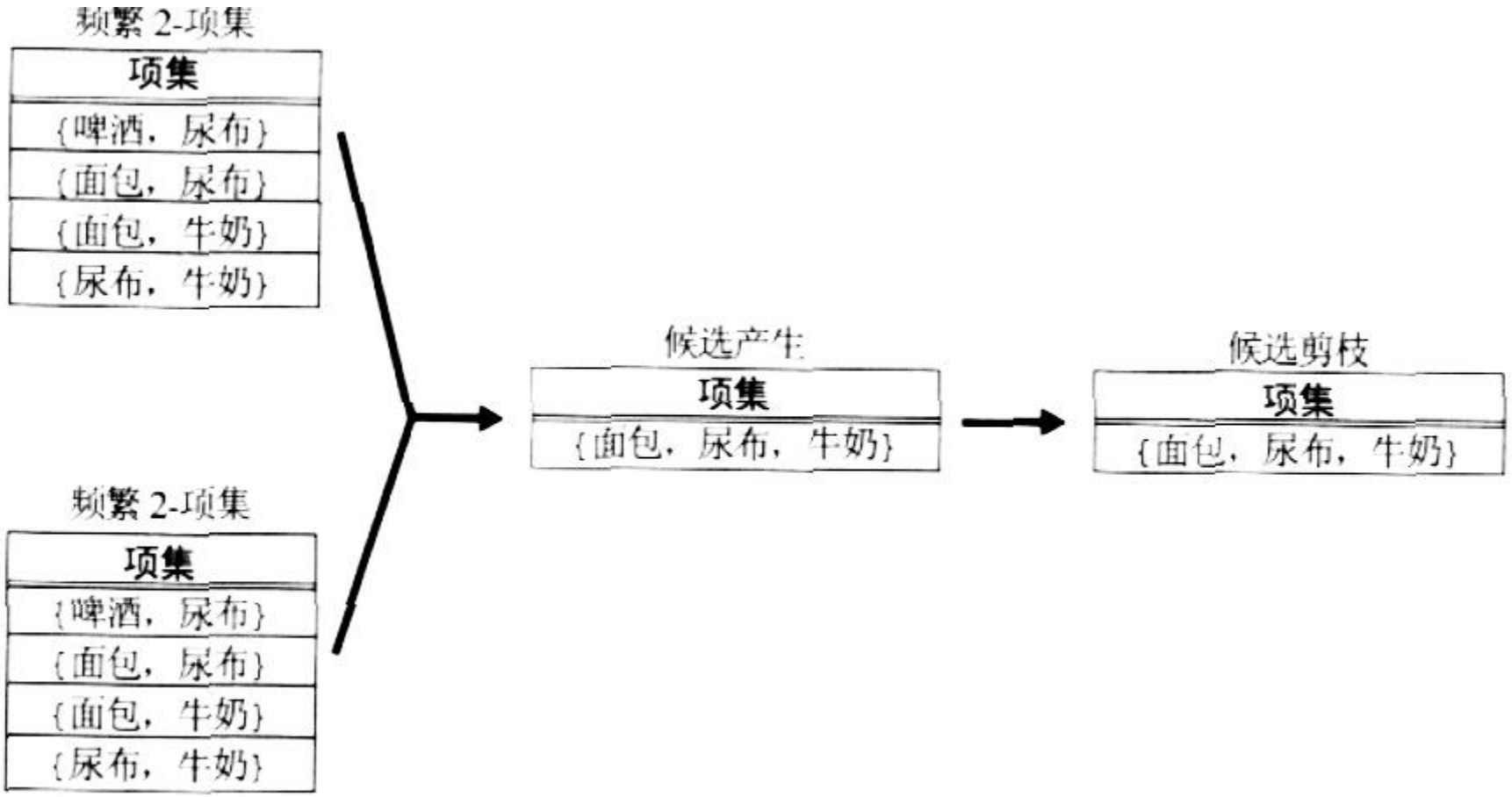
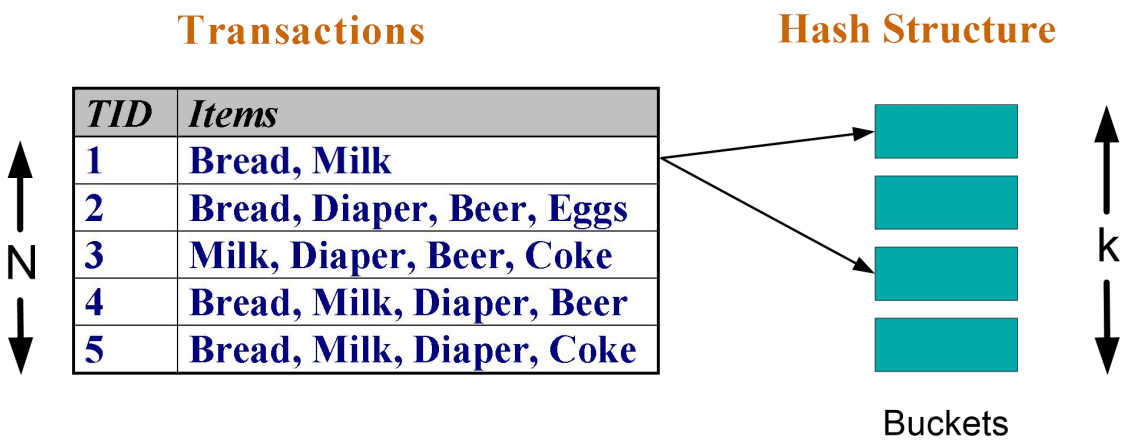


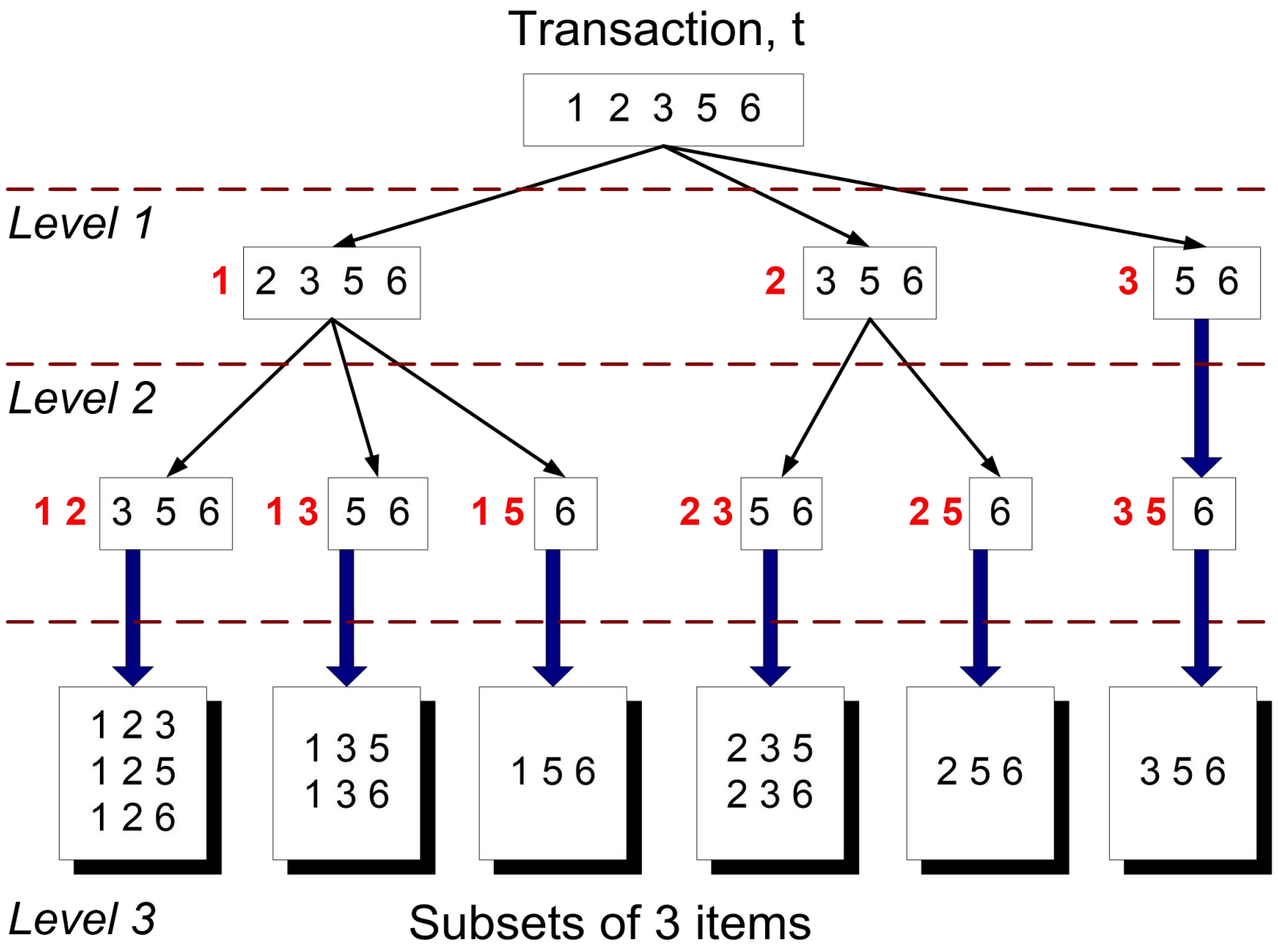
图 6-8　通过合并一对频繁$(k-1)$-项集生成和剪枝候选 k-项集

支持度计数

- 支持度计数过程确定在apriori-gen函数的候选项剪枝步骤保留下来的每个候选项集出现的频繁程度。计算支持度的主要方法:
  - 一种方法是将每个事务与所有的候选项集进行比较，并且更新包含在事务中的候选项集的支持度计数。这种方法是计算昂贵的，尤其当事务和候选项集的数目都很大时。
  - 另一种方法是枚举每个事务所包含的项集，并且利用它们更新对应的候选项集的支持度。



枚举事务t的所有包含3个项的子集

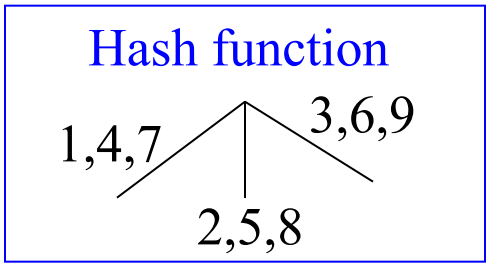


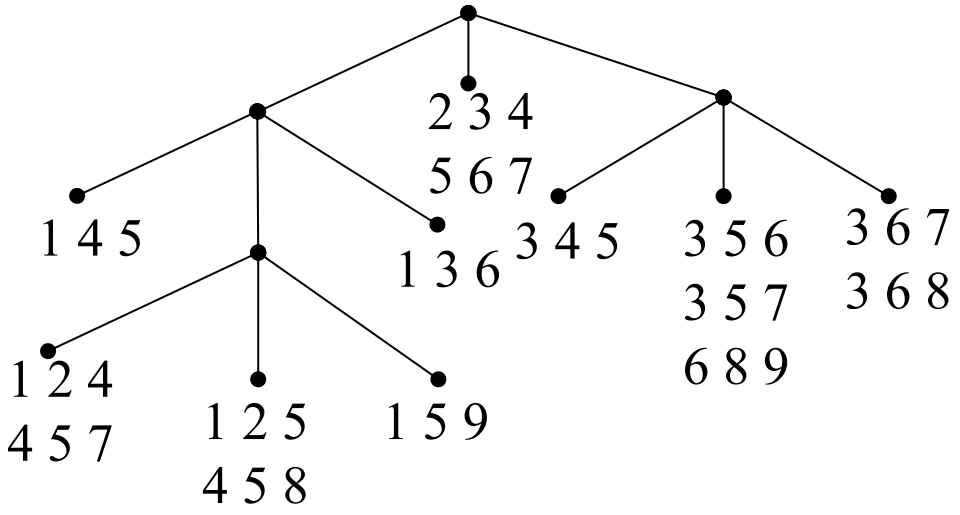
产生Hash树

Hash函数h(p)=p mod 3

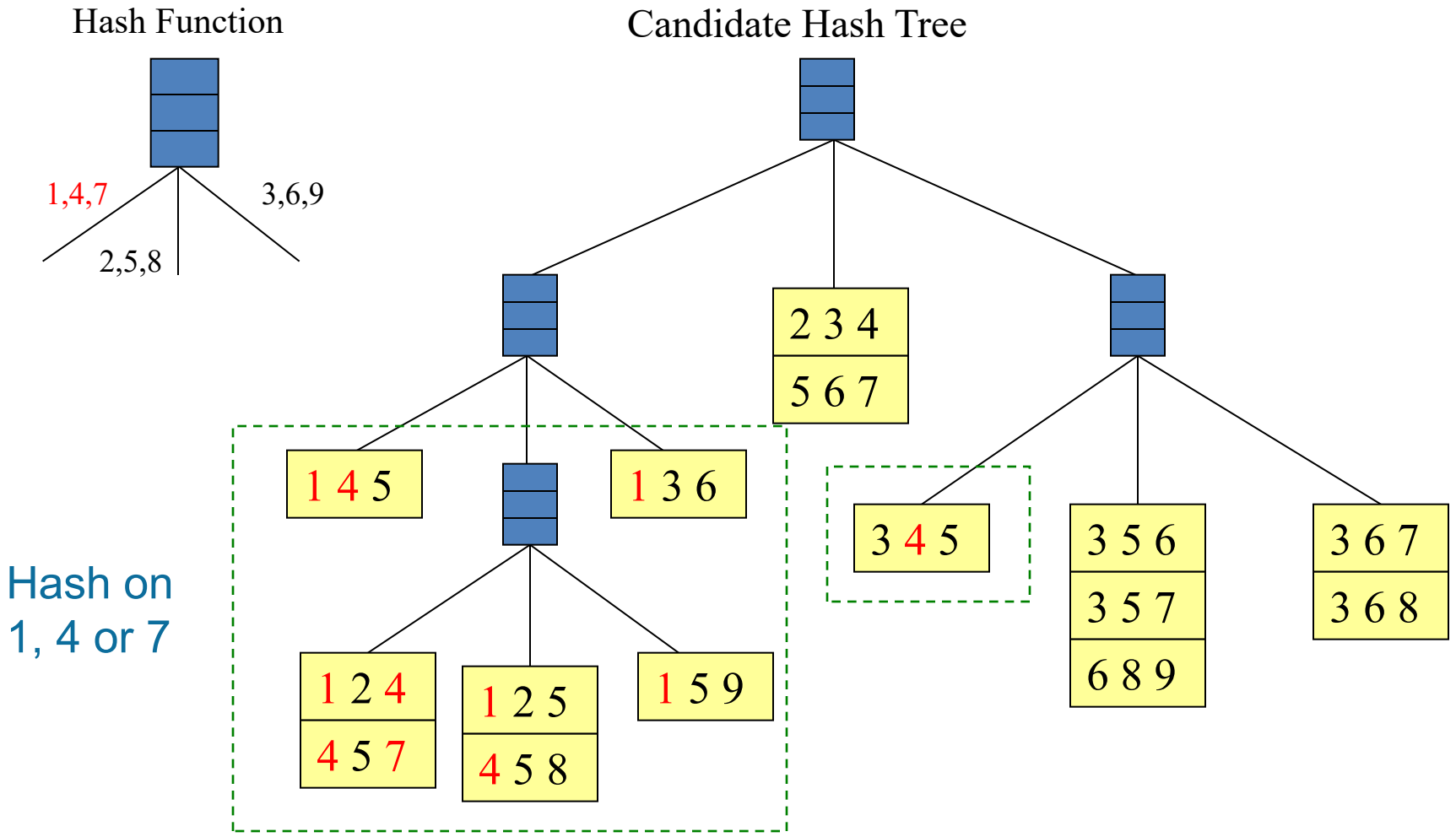
假设有15个候选3-项集:

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

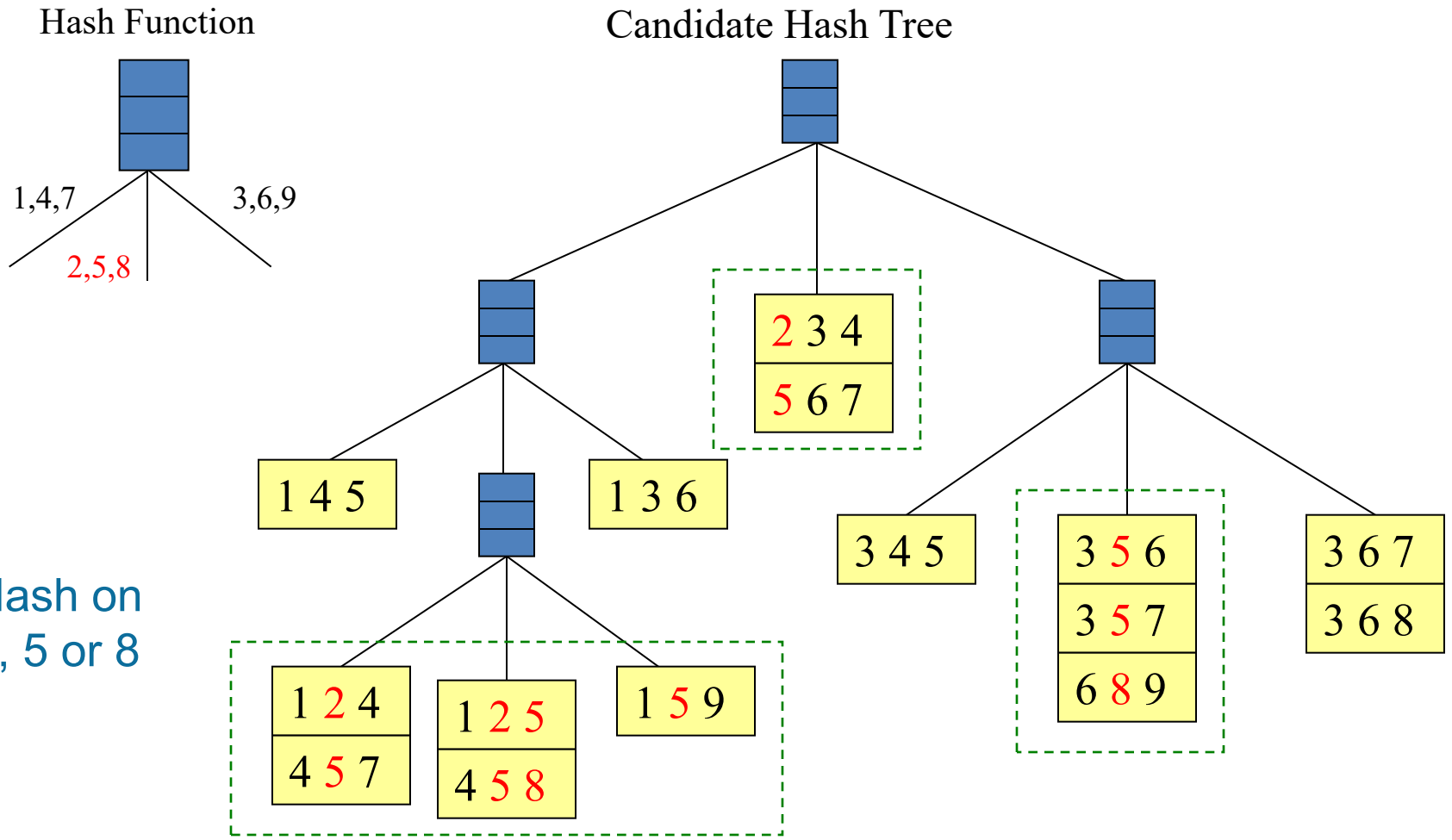




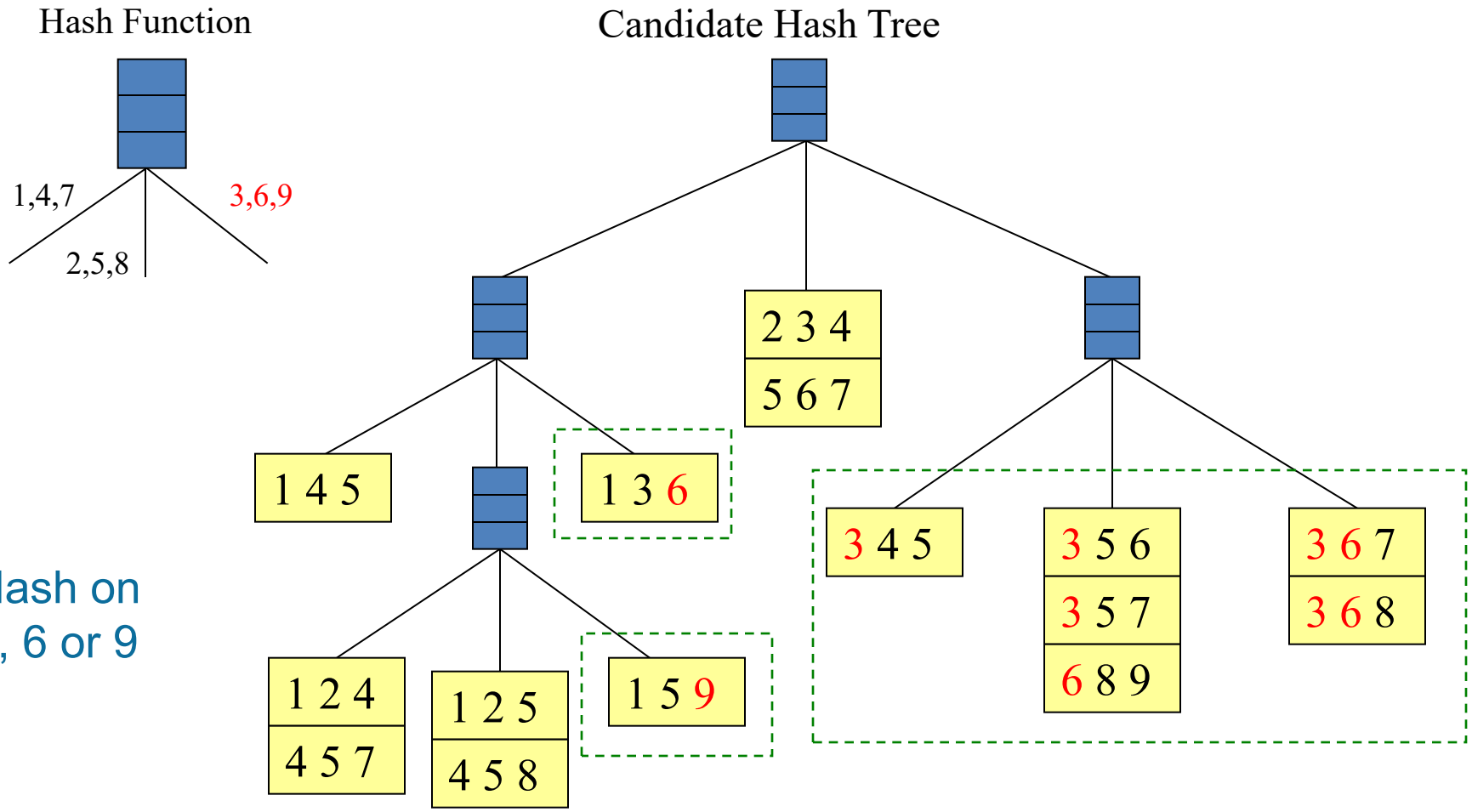
Hash树结构


Hash Function
1,4,7
2,5,8
3,6,9
Candidate Hash Tree
1 4 5
1 3 6
2 3 4
5 6 7
3 4 5
3 5 6
3 5 7
6 8 9
3 6 7
3 6 8
1 2 4
4 5 7
1 2 5
4 5 8
1 5 9
Hash on
1, 4 or 7

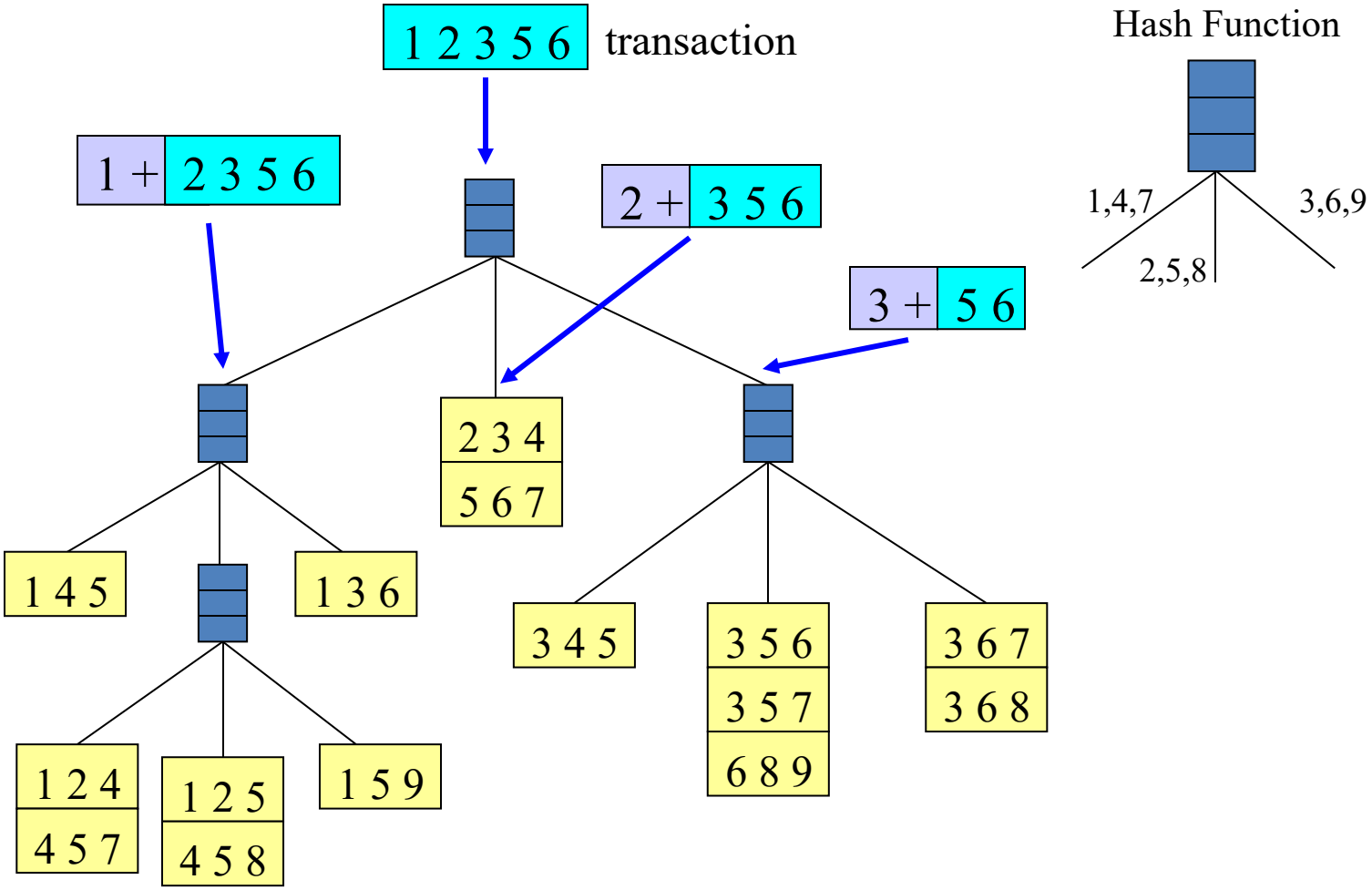
Hash树结构


Hash Function
1,4,7
2,5,8
3,6,9
Candidate Hash Tree
1 4 5
1 3 6
2 3 4
5 6 7
3 4 5
3 5 6
3 5 7
6 8 9
3 6 7
3 6 8
1 2 4
4 5 7
1 2 5
4 5 8
1 5 9
Hash on
2, 5 or 8

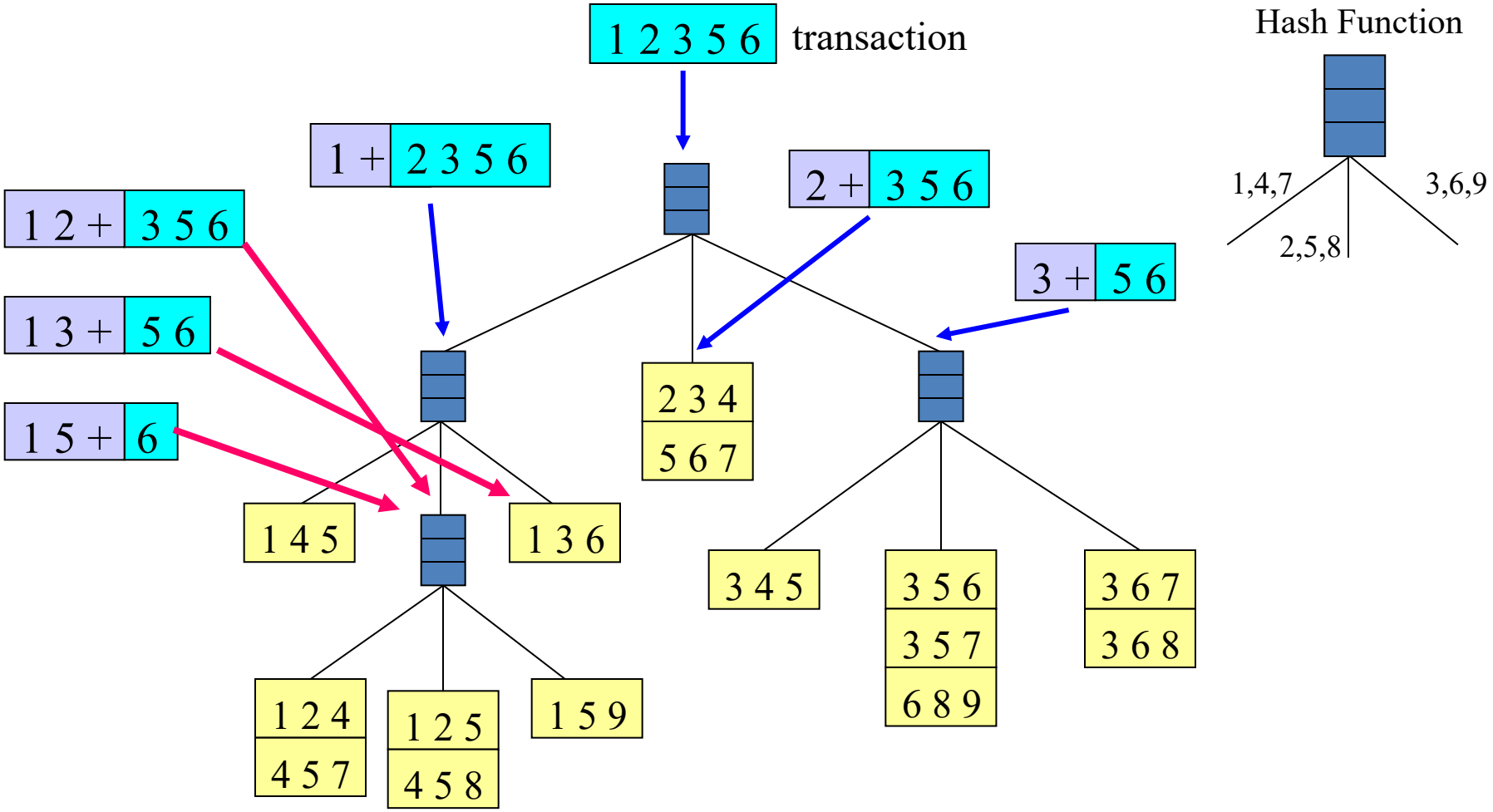
Hash树结构


Hash Function
1,4,7
2,5,8
3,6,9
Candidate Hash Tree
1 4 5
1 3 6
2 3 4
5 6 7
3 4 5
3 5 6
3 5 7
6 8 9
3 6 7
3 6 8
1 2 4
4 5 7
1 2 5
4 5 8
1 5 9
Hash on
3, 6 or 9

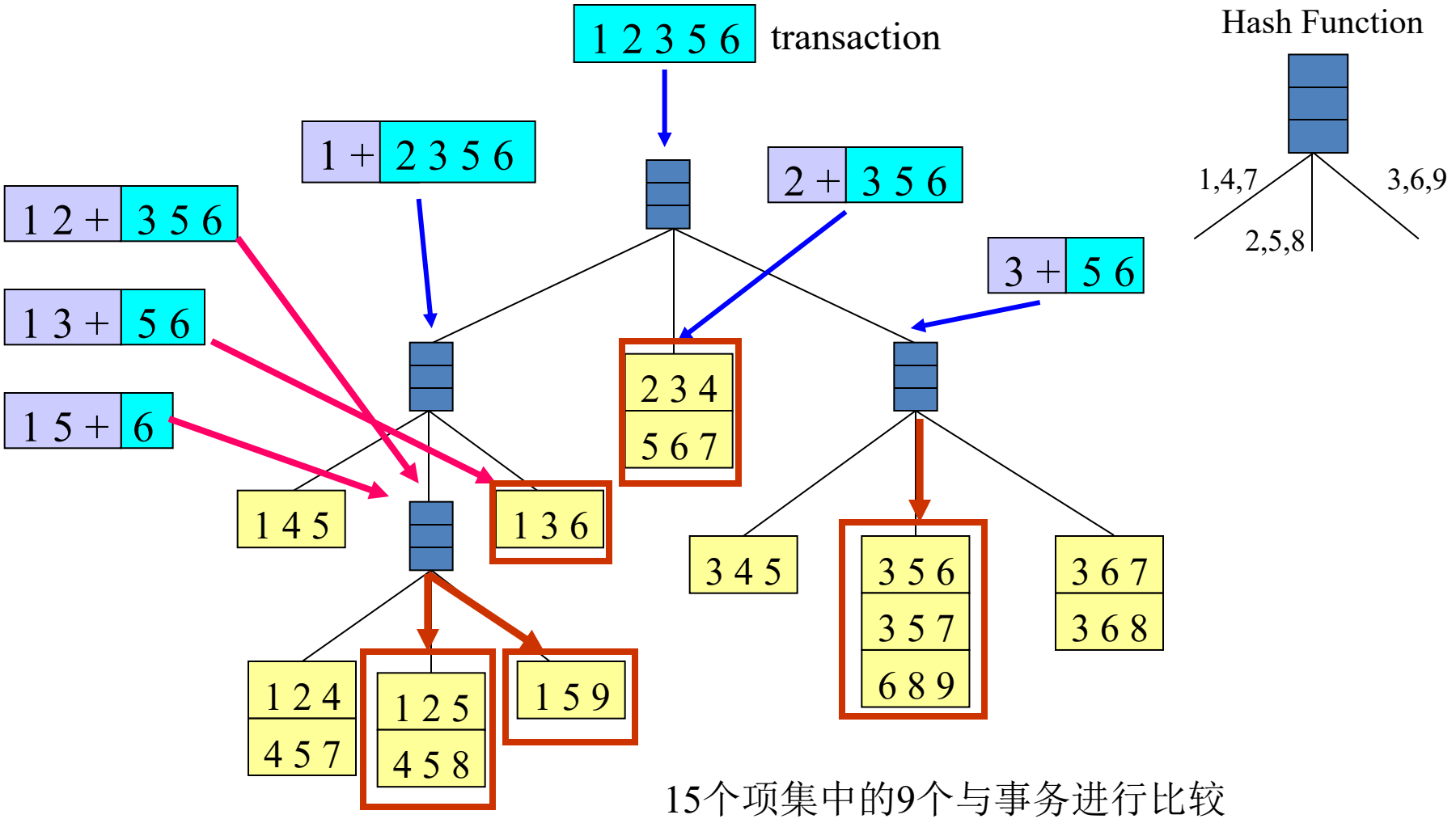
使用Hash树进行支持度计数



使用Hash树进行支持度计数



使用Hash树进行支持度计数

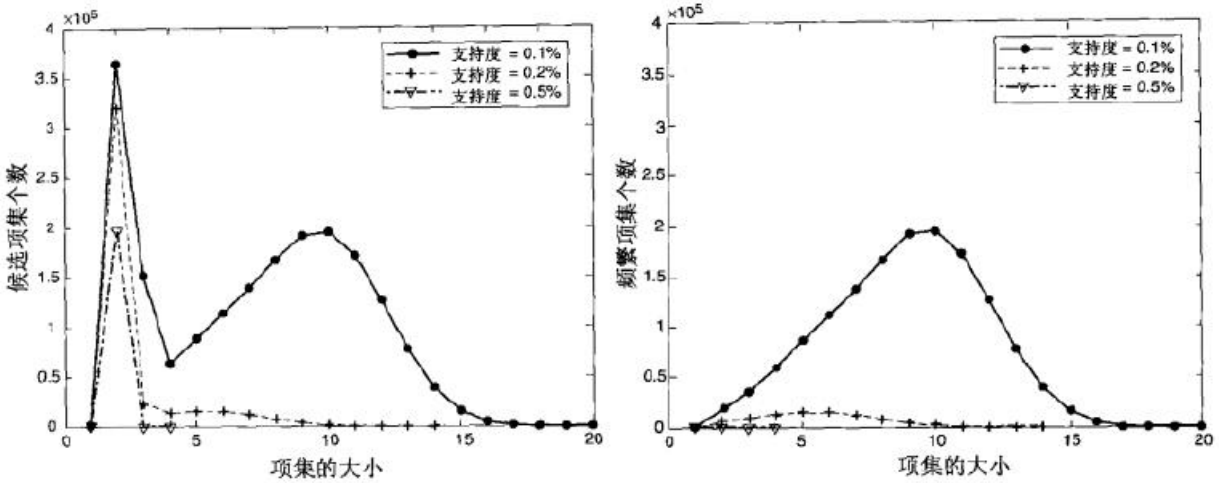


15个项集中的9个与事务进行比较

- 存放在被访问的叶结点中的候选项集与事务进行比较，如果候选项集是该事务的子集，则增加它的支持度计数。
- 在该例子中，访问了9个叶子结点中的5个。
- 15个项集中的9个与事务进行比较

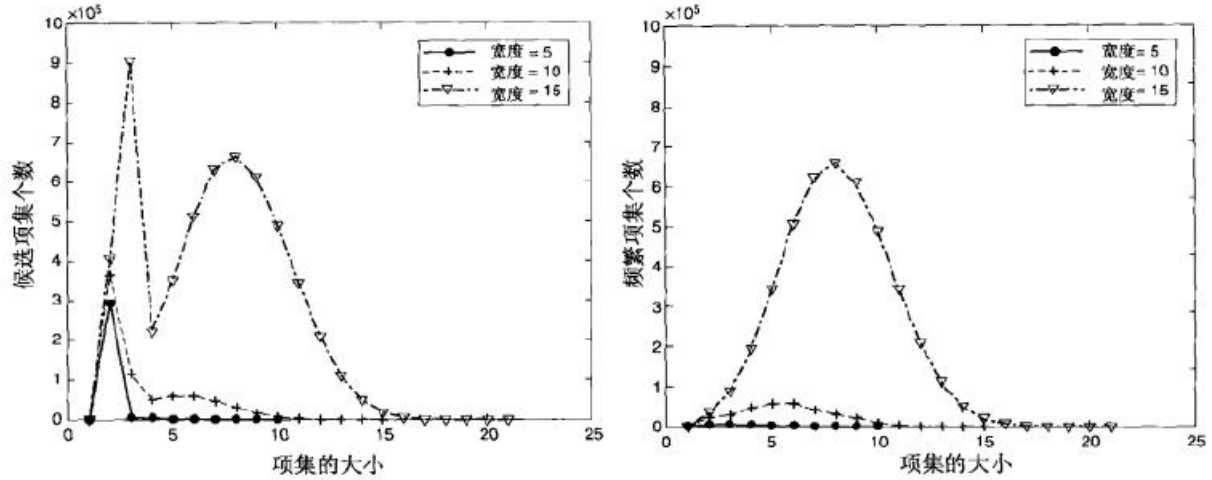
计算复杂性

- 支持度阈值
  - 降低支持度阈值通常将导致更多的项集是频繁的。计算复杂度增加
  - 随着支持度阈值的降低，频繁项集的最大长度将增加，导致算法需要扫描数据集的次数也将增多
- 项数
  - 随着项数的增加，需要更多的空间来存储项的支持度计数。如果频繁项集的数目也随着数据项数增加而增长，则由于算法产生的候选项集更多，计算量和I/O开销将增加
- 事务数
  - 由于Apriori算法反复扫描数据集，因此它的运行时间随着事务数增加而增加
- 事务的平均宽度
  - 频繁项集的最大长度随事务平均宽度增加而增加
  - 随着事务宽度的增加，事务中将包含更多的项集，这将增加支持度计数时Hash树的遍历次数



(a) 候选项集的个数　　(b) 频繁项集的个数

图 6-13　支持度阈值对候选项集和频繁项集的数量的影响



(a) 候选项集的个数　　(b) 频繁项集的个数

图 6-14　事务的平均宽度对候选项集和频繁项集的数量的影响

规则产生

- 忽略那些前件或后件为空的规则，每个频繁k-项集能够产生多达2^k-2个关联规则
- 关联规则的提取：将一个项集 Y划分成两个非空的子集 X 和Y-X，使得$X \rightarrow Y - X$满足置信度阈值。
  - 如果 {A,B,C,D} 是频繁项集, 候选项集为:

ABC →D,	ABD →C,	ACD →B,	BCD →A,
A →BCD,	B →ACD,	C →ABD,	D →ABC
AB →CD,	AC → BD,	AD → BC,	BC →AD,
BD →AC,	CD →AB,		

- 这样的规则必然已经满足支持度阈值，因为它们是由频繁项集产生的。

规则产生

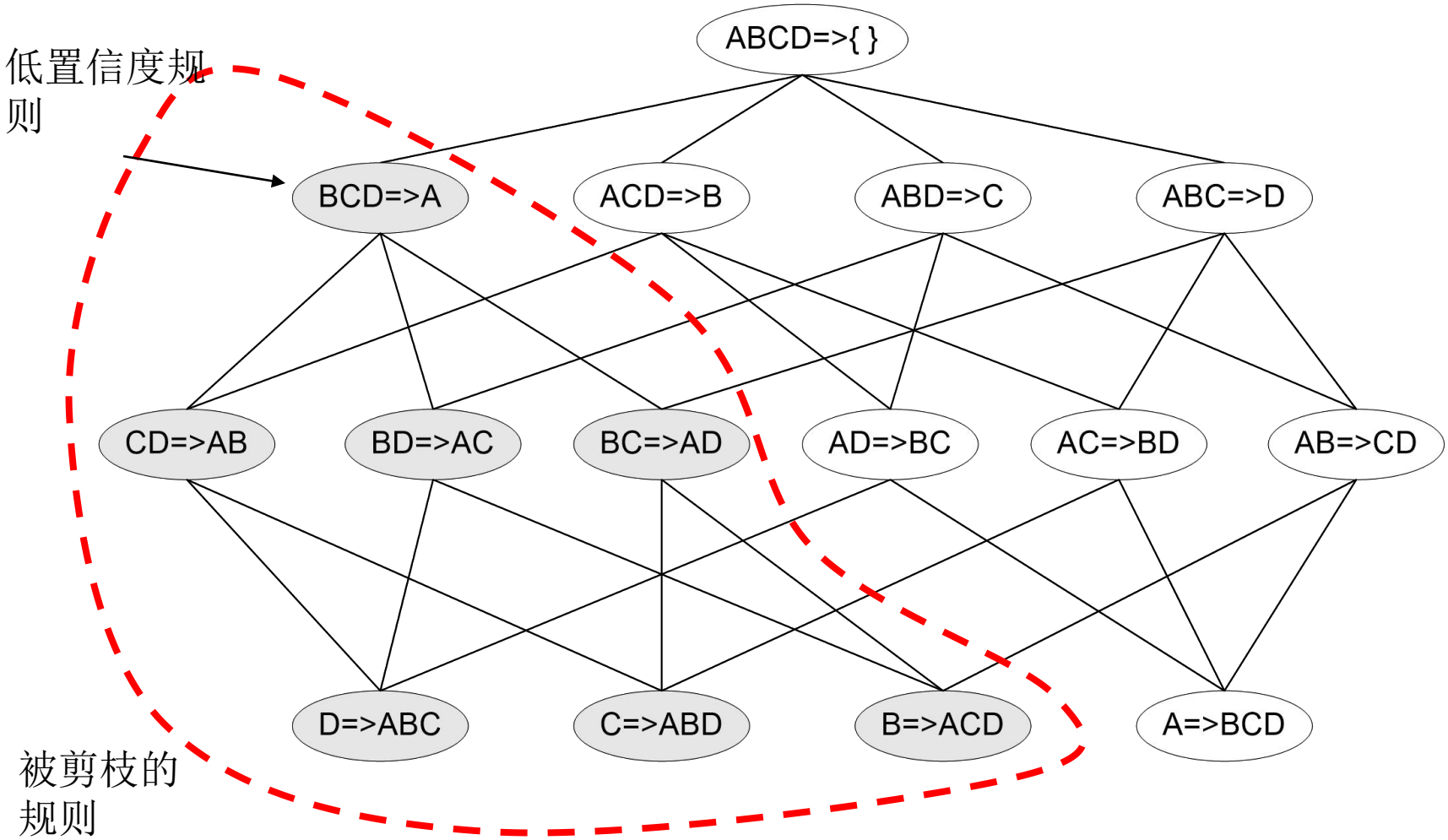
- 怎样有效的从频繁项集中产生关联规则?
  - 一般，计算关联规则的置信度并不需要再次扫描事务数据集。规则{A,B,C} →{D}的置信度为σ(ABCD)/ σ(ABC)。因为这两个项集的支持度计数已经在频繁项集产生时得到，因此不必再扫描整个数据集

  - 如果规则X →Y-X不满足置信度阈值，则形如X'→Y-X'的规则一定也不满足置信度阈值，其中X'是X的子集。

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

    因为σ(AB) ≥σ(ABC)，则σ(ABCD)/ σ(ABC) ≥ σ(ABCD)/ σ(AB)，则$c(ABC \rightarrow D) \geq c(AB \rightarrow CD)$

Apriori 算法中规则的产生



Apriori 算法中规则的产生

算法 6.2 *Apriori* 算法中的规则产生

```
1: for 每一个频繁 k-项集 f_k , k≥2 do
2:     H_1={i | i ∈ f_k }                    {规则的 1-项后件}
3:     call ap-genrules(f_k, H_1)
4: end for
```

算法 6.3 过程 ap-genrules(f_k, H_m)

```
1:  k = |f_k|          {频繁项集的大小}
2:  m = |H_m|          {规则后件的大小}
3:  if k > m + 1 then
4:     H_{m+1} = apriori-gen(H_m)
5:     for 每个 h_{m+1} ∈ H_{m+1} do
6:        conf = σ(f_k)/ σ(f_k − h_{m+1})
7:        if conf≥minconf then
8:           output: 规则 (f_k − h_{m+1}) → h_{m+1}
9:        else
10:          从 H_{m+1} delete h_{m+1}
11:       end if
12:    end for
13:    call ap-genrules(f_k, H_{m+1})
14: end if
```

关联规则——Apriori算法案例

下面将结合餐饮行业的实例来讲解Apriori关联规则算法挖掘的实现过程。数据库中部分点餐数据下表：

序列	时间	订单号	菜品id	菜品名称
1	2014/8/21	101	18491	健康麦香包
2	2014/8/21	101	8693	香煎葱油饼
3	2014/8/21	101	8705	翡翠蒸香茜饺
4	2014/8/21	102	8842	菜心粒咸骨粥
5	2014/8/21	102	7794	养颜红枣糕
6	2014/8/21	103	8842	金丝燕麦包
7	2014/8/21	103	8693	三丝炒河粉
…	…	…	…	…

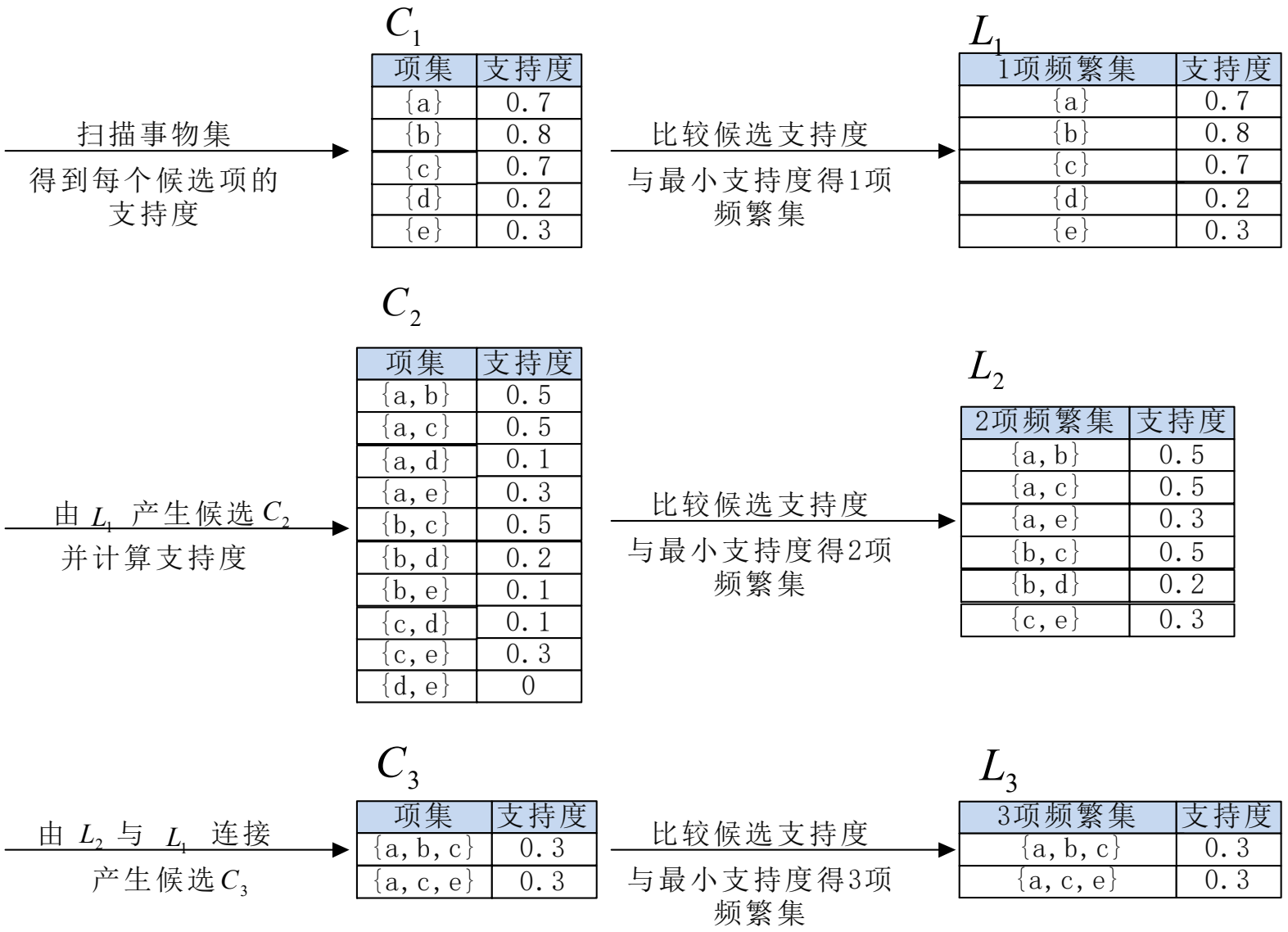
关联规则——Apriori算法案例

- 首先将上表中的事务数据（一种特殊类型的记录数据）整理成关联规则模型所需的数据结构。从中抽取10个点餐订单作为事务数据集为方便起见将菜品{18491，8842，8693，7794，8705}分别简记为{a，b，c，d，e}），如：

订单号	菜品id	菜品id
1	18491, 8693，8705	a，c，e
2	8842,7794	b，d
3	8842，8693	b，c
4	18491，8842，8693，7794	a，b，c，d
5	18491，8842	a，b
6	8842，8693	b，c
7	18491，8842	a，b
8	18491，8842,8693,8705	a，b，c，e
9	18491，8842,8693	a，b，c
10	18491，8693	a，c，e

关联规则——Apriori算法案例

- 设支持度为0.2，即支持度计数为2，算法过程如下图：



关联规则——Apriori算法案例

- 过程一：找最大k项频繁集

1. 算法简单扫描所有的事务，事务中的每一项都是候选 1 项集的集合 C_1 的成员，计算每一项的支持度。比如

$$P(\{a\}) = \frac{\text{项集}\{a\}\text{的支持度计数}}{\text{所有事务个数}} = \frac{7}{10} = 0.7$$

2. 对 C_1 中各项集的支持度与预先设定的最小支持度阈值作比较，保留大于或等于该阈值的项，得1项频繁集 L_1;
3. 扫面所有事务，L_1与 L_1 连接得候选2项集 C_2 并计算每一项的支持度。如

$$P(\{a,b\}) = \frac{\text{项集}\{a,b\}\text{的支持度计数}}{\text{所有事务个数}} = \frac{5}{10} = 0.5$$

接下来是剪枝步，由于 C_2 的每个子集（即 L_1）都是频繁集，所以没有项集从 C_2 中剔除;

关联规则——Apriori算法案例

- 过程一：找最大k项频繁集

4. 对C_2中各项集的支持度与预先设定的最小支持度阈值作比较，保留大于或等于该阈值的项，得 2 项频繁集 L_2;

5. 扫描所有事务，L_2与 L_1 连接得候选 3 项集C_3，并计算每一项的支持度，如：

$$P(\{a,b,c\}) = \frac{\text{项集}\{a,b,c\}\text{的支持度计数}}{\text{总的事务计数}} = \frac{3}{10} = 0.3$$

接下来是剪枝步，L_2 与 L_1 连接的所有项集为：{ a，b，c}，{ a，b，d}，{ a，b，e}，{ a，c，d}，{ a，c，e}，{ b，c，d}，{ b，c，e}。

根据Apriori算法，频繁集的所有非空子集也必须是频繁集，因为{b，d}，{b，e}，{c，d}不包含在b项频繁集 L_2中，即不是频繁集，应剔除，最后的 C_3中的项集只有{ a，b，c}和{ a，c，e};

关联规则——Apriori算法案例

- 过程一：找最大k项频繁集

6. 对 C_3 中各项集的支持度与预先设定的最小支持度阈值作比较，保留大于或等于该阈值的项，得3项频繁集 L_3；

7. L_3 与 L_1 连接得候选 4 项集，易得剪枝后为空集。最后得到最大3项频繁集{ a，b，c}和{ a，c，e}。

- 由以上过程可知 L_1, L_2, L_3 都是频繁项集，L_3 是最大频繁项集。

关联规则——Apriori算法案例

- 过程二：由频繁集产生关联规则

置信度的计算公式为：

$$Confidence(A \Rightarrow B) = P(A|B) = \frac{Support(A \cup B)}{Support(A)} = \frac{Support\_count(A \cup B)}{Support\_count(A)}$$

其中 $Support\_count(A \cup B)$ 和 $Support\_count(A)$ 分别是包含项集 $A \cup B$ 和A的事务数，根据该公式，尝试基于该例产生关联规则。

最终输出的关联规则如下：

关联规则——Apriori算法案例

- 过程二：由频繁集产生关联规则

Rule	(Support, Confidence)
a -> b	(50%, 71.4286%)
b -> *a*	(50%, 62.5%)
a -> c	(50%, 71.4286%)
c -> *a*	(50%, 71.4286%)
b -> c	(50%, 62.5%)
c -> b	(50%, 71.4286%)
e -> *a*	(30%, 100%)
e -> c	(30%, 100%)
a,b -> c	(30%, 60%)
a,c -> b	(30%, 60%)
b,c -> *a*	(30%, 60%)
e -> *a*,c	(30%, 100%)
a,c -> e	(30%, 60%)
a,e -> c	(30%, 100%)
c,e -> *a*	(30%, 100%)
d -> b	(20%, 100%)

- 就第一条输出结果进行解释：客户同时点菜品a和b的概率是50%，点了菜品，再点菜品 的概率是71.4286%。知道了这些，就可以对顾客进行智能推荐，增加销量同时满足客户需求。

Thank You!