第11章 关联规则分析

11.0 基本概念

- 频繁模式(frequent pattern)是指在数据集中频繁出现的模式。
- 现实生活中存在多种类型的频繁模式，包括频繁项集、频繁子序列（又称序列模式）和频繁子结构。

基本概念

编号	牛奶	果冻	啤酒	面包	花生酱
T_1	1	1	0	0	1
T_2	0	1	0	1	0
T_3	0	1	1	0	0
T_4	1	1	0	1	0
T_5	1	0	1	0	0
T_6	0	1	1	0	0
T_7	1	0	1	0	0
T_8	1	1	1	0	1
T_9	1	1	1	0	0

- 一个样本称为一个“事务”
- 每个事务由多个属性来确定，这里的属性我们称为“项”
- 多个项组成的集合称为“项集”

k-项集

- 由k个项构成的集合
  - {牛奶}、{啤酒}都是1-项集；
  - {牛奶，果冻}是2-项集；
  - {啤酒，面包，牛奶}是3-项集。
- 每个事务其实就是一个项集

编号	牛奶	果冻	啤酒	面包	花生酱
T_1	1	1	0	0	1
T_2	0	1	0	1	0
T_3	0	1	1	0	0
T_4	1	1	0	1	0
T_5	1	0	1	0	0
T_6	0	1	1	0	0
T_7	1	0	1	0	0
T_8	1	1	1	0	1
T_9	1	1	1	0	0

11.0 基本概念（续）

- 频繁项集一般是指频繁地在事务数据集中一起出现的商品的集合，如小卖部中被许多顾客频繁地一起购买的牛奶和面包。
- 频繁子序列，如顾客倾向于先购买便携机，再购买数码相机，然后再购买内存卡这样的模式就是一个（频繁）序列模式。

11.0 基本概念（续）

- 频繁子结构是指从图集合中挖掘频繁子图模式。子结构可能涉及不同的结构形式（例如，图、树或格），可以与项集或子序列结合在一起。如果一个子结构频繁地出现，则称它为（频繁）子结构模式。

11.0 基本概念（续）

- 频繁项集挖掘是频繁模式挖掘的基础。

11.1 频繁项集和关联规则

- 关联规则(Association Rule Mining)挖掘是数据挖掘中最活跃的研究方法之一。
- 关联规则挖掘的目的：找出数据库中不同数据项集之间隐藏的关联关系。

11.1 频繁项集和关联规则（续）

- 最早是由R.Agrawal等人在1993年提出的。
- 其目的是为了发现超市交易数据库中不同商品之间的关联关系。
- 一个典型的关联规则的例子是：70%购买了牛奶的顾客将倾向于同时购买面包。
- 经典的关联规则挖掘算法：Apriori算法和FP-growth算法 。

11.1 频繁项集合关联规则（续）

1. 购物篮分析—引发关联规则挖掘的例子

- 问题：“什么商品组或集合顾客多半会在一次购物中同时购买？”
- 购物篮分析：设全域为商店出售的商品的集合（即项目全集），一次购物购买（即事务）的商品为项目全集的子集，若每种商品用一个布尔变量表示该商品的有无，则每个购物篮可用一个布尔向量表示。通过对布尔向量的分析，得到反映商品频繁关联或同时购买的购买模式。这些模式可用关联规则描述。

11.1.1 问题描述

- **现实：**商店有很多商品，例如“面包”、“牛奶”、“啤酒”等。顾客将把他们需要的商品放入购物篮中。
- **研究的目的：**发现顾客通常会同时购买哪些商品。通过上述研究可以帮助零售商合理地摆放商品，引导销售。

11.1.1 问题描述（续）

- 举例：某一个时间段内顾客购物的记录形成一个交易数据库，每一条记录代表一次交易，包含一个交易标识符（TID）和本次交易所购买的商品。

一个简单交易数据库实例	
数据库D：	
TID	项
001	A、C、D
002	B、C、E
003	A、B、C、E
004	B、E

11.1.1 问题描述（续）

- 几个基本概念：
  - **数据项**：设I={i_1，i_2，...,i_m}是常数的集合，其中m是任意有限的正整数常量，每个常数i_k（k=1,2，...，m）称为一个数据项。
  - **项集**：由I中的数据项组成的集合，即X⊆I。
  - **K-项集**：一个大小为K的项集（包含有K项，如{A、B}为2-项集，{A、C、D}为3-项集）。
  - **一个交易T**：是由在I中的数据项所构成的集合，即T⊆I。

11.1.1 问题描述（续）

- 【定义1】以商场交易数据库为例，形式化地描述关联规则：
- 设I={i1，i2，…,im}是项的集合，表示各种商品的集合；D= {t1，t2，…,tn}为交易集，表示每笔交易的集合（是全体事务的集合）。其中每一个事务T都是项的集合，且有T⊆I。每个事务都有一个相关的唯一标识符和它对应，也就是事务标识符或TID。

11.1.1 问题描述（续）

- 设X为一个由多个项目构成的集合，称为项集，如001中的{A、C、D}，当且仅当X⊆T时我们说事务T包含X。

11.1.1 问题描述（续）

- 项集X在在事务数据库DB中出现的次数占总事务的百分比叫做项集的支持度。
- 如果项集的支持度超过用户给定的最小支持度阈值，就称该项集是频繁项集（或大项集）。

11.1.1 问题描述（续）

关联规则

- 关联规则是形如$X \Rightarrow Y$的蕴含式，其中$X \subseteq I$，$Y \subseteq I$且$X \cap Y = \varnothing$，则X称为规则的条件，Y称为规则的结果。
- 如果事务数据库D中有s%的事务包含$X \cup Y$，则称关联规则$X \Rightarrow Y$的支持度为s%。
- **支持度**是指项集X和Y在数据库D中同时出现的概率。

11.1.1 问题描述（续）

- 【定义2】关联规则 X⇒Y对事务集D的支持度（support）定义为D中包含有事务X和Y的百分比。关联规则X⇒Y对事务集合D的置信度（confidence）定义为D中包含有X的事务数与同时包含Y的百分比。即：

  - support(X⇒Y) = (包含X和Y的事务数/事务总数)×100%
  - confidence(X⇒Y) = (包含X和Y的事务数/包含X的事务数)×100%

11.1.1 问题描述（续）

- 【例11.1】某顾客购物的交易数据库总交易数为5。

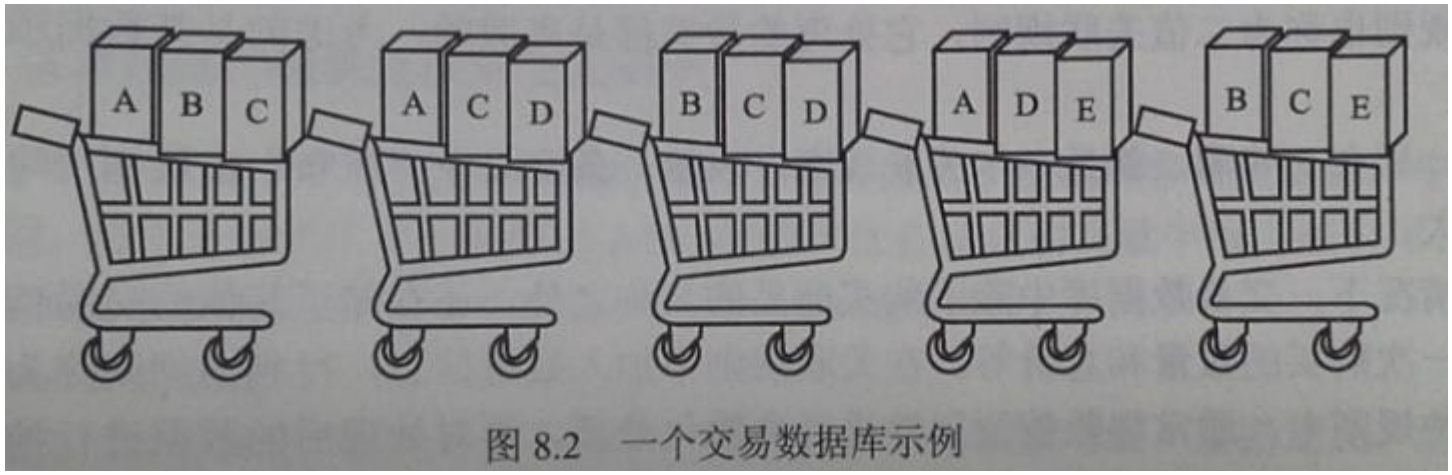


图 8.2　一个交易数据库示例

11.1.1 问题描述（续）

- 【例11.1】相关的支持度和置信度。

表 8.2 支持度和置信度示例

规则	支持度	置信度
$A \Rightarrow D$	2/5	2/3
$C \Rightarrow A$	2/5	2/4
$A \Rightarrow C$	2/5	2/3
$B\&C \Rightarrow D$	1/5	1/3

- support(X⇒Y) = (包含X和Y的事务数/事务总数)×100%
- confidence(X⇒Y) = (包含X和Y的事务数/包含X的事务数)×100%

11.1.1 问题描述（续）

- 频度：由于分母相同，有时仅用分子表示，即项集在数据库中出现的次数来代表支持度。
- 通过支持度和置信度作为评分函数，给出了对模式进行评价的一个量化标准。

11.1.1 问题描述（续）

- 进行关联规则挖掘时，要求用户给出两个阈值：
  - 最小支持度（频度）s；
  - 最小置信度c。
- 表示：

$$support(X \Rightarrow Y) >= min\_sup$$

$$confidence(X \Rightarrow Y) >= min\_conf$$

的关联规则称为强规则；否则称为弱规则。

- 数据挖掘主要就是对强规则的挖掘。通过设置最小支持度和最小置信度可以了解某些数据之间的关联程度。

11.1.1 问题描述（续）

- 关联规则挖掘就是要从大量的潜在的规则库中寻找出满足支持度（频度）和置信度阈值的所有规则。

11.1.1 问题描述（续）

- 举例：一个食品连锁店保留着每周的事务记录，其中每一条事务表示在一项收款机业务中卖出的项目。连锁店的管理会收到一个事务汇总报告，报告表明了每种项目的销售量是多少。此外，他们要定期了解哪些项目经常被顾客一起购买。他们发现顾客购买了花生酱后，100%地会购买面包。而且，顾客购买了花生酱后，有33%也购买果冻。不过，所有事务中大约只有50%包含花生酱。

11.1.1 问题描述（续）

- 被用于在其中寻找关联规则的数据库可以看作为一个元组集合，每个元组包含一组项目。一个元组可能是：

  {花生酱、面包、果冻}

- 包含三个项目：花生酱、面包、果冻
- 每个项目表示购买的一种产品
- 一个元组是一次购买的产品列表

11.1.1 问题描述（续）

- 样本数据库

演示关联规则的样本数据	
事务	项目
t_1	面包、果冻、花生酱
t_2	面包、花生酱
t_3	面包、牛奶、花生酱
t_4	啤酒、面包
t_5	啤酒、牛奶

11.1.1 问题描述（续）

找出的所有项目集合的支持度

集合	支持度	集合	支持度
啤酒	40	啤酒、面包、牛奶	0
面包	80	啤酒、面包、花生酱	0
果冻	20	啤酒、果冻、牛奶	0
牛奶	40	啤酒、果冻、花生酱	0
花生酱	60	啤酒、牛奶、花生酱	0
啤酒、面包	20	面包、果冻、牛奶	0
啤酒、果冻	0	面包、果冻、花生酱	20
啤酒、牛奶	20	面包、牛奶、花生酱	20
啤酒、花生酱	0	果冻、牛奶、花生酱	0
面包、果冻、	20	啤酒、面包、果冻、牛奶	0
面包、果冻	20	啤酒、面包、果冻、花生酱	0
面包、花生酱	60	啤酒、面包、牛奶、花生酱	0
果冻、牛奶	0	啤酒、果冻、牛奶、花生酱	0
果冻、花生酱	20	面包、果冻、牛奶、花生酱	0
牛奶、花生酱	20	啤酒、面包、果冻、牛奶、花生酱	0
啤酒、面包、果冻	0		

11.1.1 问题描述（续）

- 问题发现：项目的个数成指数增长：从5个项目的集合得到31个项目集合（忽略空集）
- 关联规则挖掘过程：
  - 第一步：寻找频繁项集。根据定义，这些项集出现的频度不小于预先定义的最小额度。---较难
  - 第二步：由频繁项集产生关联规则。根据定义，这些规则必须满足最小支持度和最小置信度。--较易

11.1.2 关联规则分类

- 购物篮分析只是关联规则挖掘的一种形式。
- 根据不同的分类标准，关联规则有多种分类方法：
  - 根据规则中所处理的数据类型分类
  - 根据规则中涉及的数据维数分类
  - 根据规则中数据的抽象层次分类
  - 其它

1. 根据规则中所处理的数据类型分类

- 根据规则中所处理的数据类型，可以分为：
  - **布尔关联规则**，也称为二值关联规则，处理的数据都是离散的。如：尿布⇒啤酒。
  - **量化关联规则**：在关联规则中加入数量信息得到的规则。如：职业＝“学生”⇒收入＝“0...1000”。

数值类型

2. 根据规则中涉及的数据维数分类

- 根据规则中涉及的数据维数，可以分为：
  - 单维关联规则，只涉及数据表的一个字段。如：尿布⇒啤酒。
  - 多维关联规则：涉及数据表的多个字段。如：性别=“女”⇒职业=“护士”，是二维关联规则；又如：年龄=“20...30”∧职业=“学生”⇒购买=“电脑”，是三维关联规则。

3. 根据规则中数据的抽象层次分类

- 根据规则中数据的抽象层次，可以分为：
  - 单层关联规则，所有的变量都是细节数据，没有层次之分，如：IBM台式机⇒HP打印机。
  - 多层关联规则：发生关联的数据可能位于同一层次，也可能位于不同的层次。如：台式机⇒HP打印机。

4. 其它

- 可以对关联规则施加语义约束，以便限制规则左部或者右部必须包含某些字段。

- **后续章节将着重介绍布尔关联规则挖掘的两类具有代表性的算法。**

11.1.3 关联规则挖掘的经典算法Apriori

- R.Agrawal等人于1993年首先提出了挖掘顾客交易数据库中项集间的关联规则问题，给出了形式化定义和算法AIS，但该算法影响不大。
- R.Agrawal等人又于1994年提出了著名的Apriori算法。

11.1.3 关联规则挖掘的经典算法Apriori（续）

- Apriori算法是一种最有影响的挖掘布尔关联规则大（频繁）项目集的算法。它使用一种称作逐层搜索的迭代算法，通过k-项集用于探索（k+1）-项集。已经为大部分商业产品所使用。

1. Apriori算法描述

- 关联规则挖掘过程：
  - **第一步：寻找频繁项集。**根据定义，这些项集出现的频度不小于预先定义的最小额度。**---较难**

找出满足定义的大项目集

  - **第二步：由频繁项集产生关联规则。**根据定义，这些规则必须满足最小支持度和最小置信度。**--较易**

从大项目集（频繁项目集）生成关联规则

1. Apriori算法描述（续）

- 上述两步工作中第二步比较容易。
- 目前主要研究重点：如何快速地找出所有频繁项集。--核心

（1） 寻找频繁项集

- 找出大项目集的算法可以很简单，但代价很高。
- 简单的方法是：对出现在事务中的所有项目集进行计数。
- 给定一个大小为m的项目集合，共有2^m个子集，去掉空集，则潜在的大项目集数为$2^m - 1$。随着项目数的增多，潜在的大项目集数成爆炸性增长。（当m=5，为31个；当m=30，变成1073741823个）
- 解决问题的难点：如何高效确定所有大项目集。

大部分关联规则算法都利用巧妙的方法来减少要计数的项目集。

（1） 寻找频繁项集（续）

- 【公理1】：如果一个项集S是频繁的（项集S的出现频度大于最小频度），那么S的任意非空子集也是频繁的。反之，如果一个项集S的某个非空子集不是频繁的，则这个项集也不可能是频繁的。
- 举例：如果一个交易包含{A、B}，则它必然也包含{A、B}的所有子集；反过来，如果{A}或{B}不是频繁项集，即{A}或{B}的出现频度小于最小频度，则{A、B}的出现频度也一定小于最小频度，因此{A、B}也不可能是频繁项集。

（1） 寻找频繁项集（续）

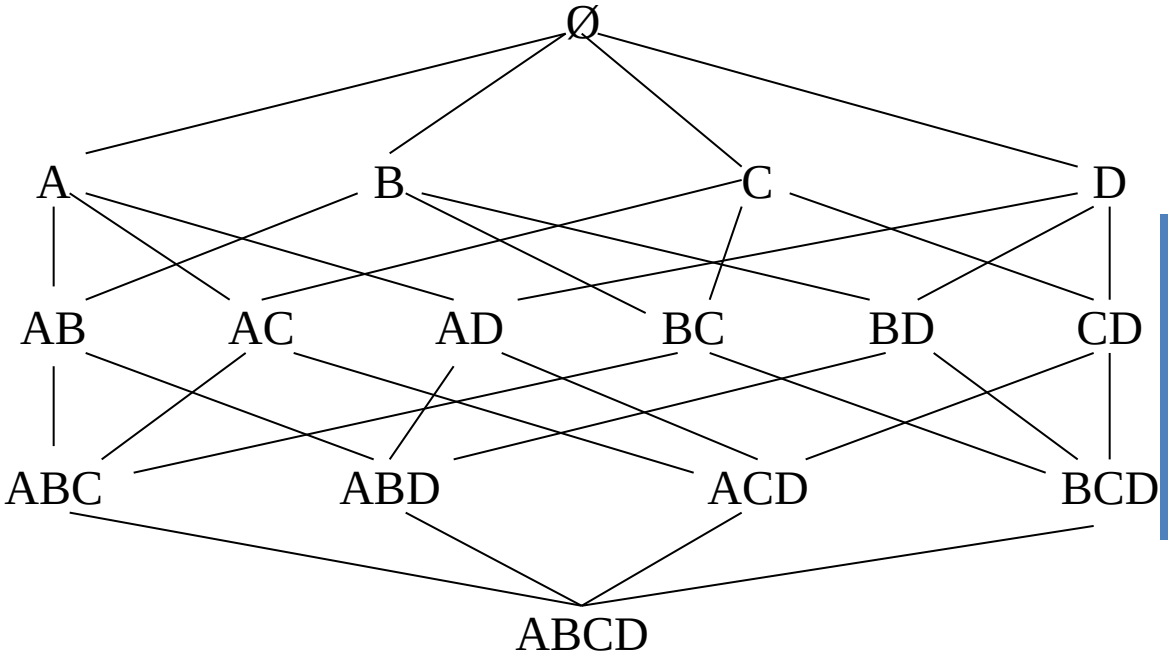
- 【结论一】：假设项集{A、B}具有一个非频繁子集{A}，则根据【公理1】可知，{A、B}不可能是频繁项集。
- 【频繁项集（大项目集）的性质】：
  - 大项目集的任一子集也一定是大的。
  - 大项目集也称作是向下封闭的，如果一个项目集满足最小支持度的要求，其所有的子集也满足这一要求。

（1） 寻找频繁项集（续）

- **其逆命题**：如果知道一个项目集是小的，就不需要生成它的任何超集来作为它的候选集，因为它们也一定是小的。
- **Apriori性质基于如下事实**：根据定义，如果项集I不满足最小支持度阈值min_sup，则I不是频繁的，即sup（I）< min_sup。如果将项A添加到I，则结果项集（即I∪A）不可能比I更频繁出现。因此，I∪A也不是频繁的，即sup（I∪A）< min_sup。
- 频繁项集的Apriori性质用于压缩搜索空间（剪枝），以提高逐层产生频繁项集的效率。

（1） 寻找频繁项集（续）

- 用图表示上述性质，例子中有四个项目{A，B，C，D}，格中的线表示子集关系，大项目集的性质表明：如果原来的项目集是大的，则在路径中位于其上的任何集合也一定是大的。

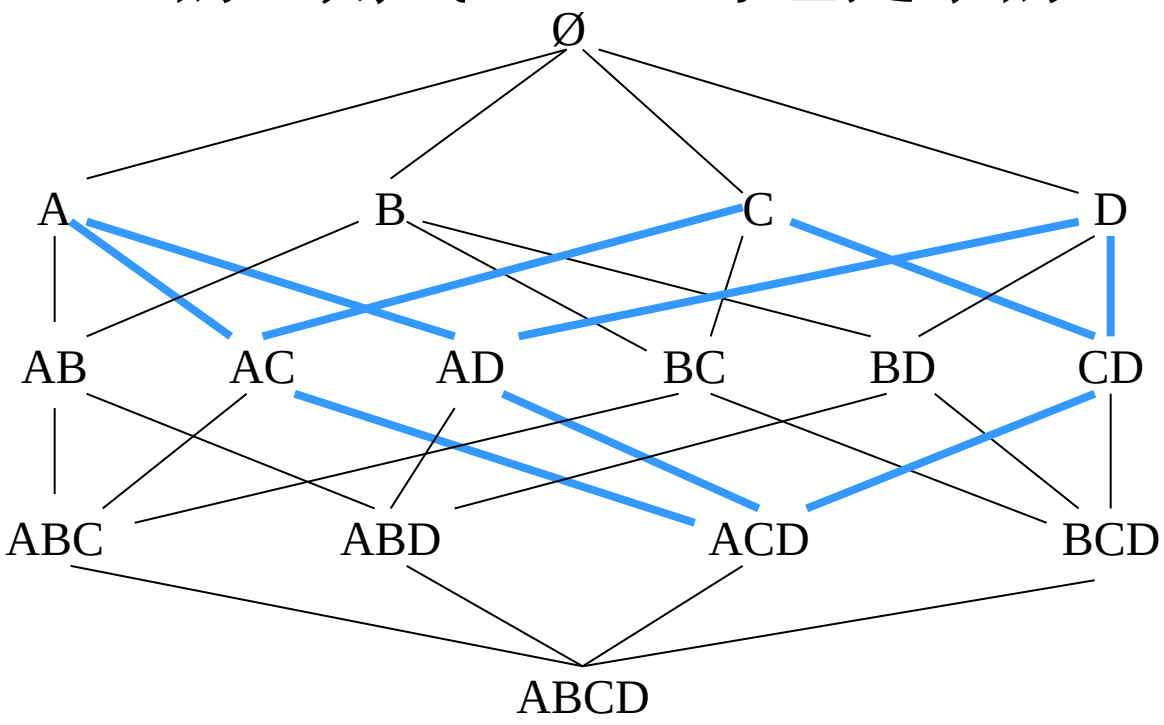


项目{ACD}的非空子集是：
{AC，AD，CD，A，C，D}

{A，B，C，D}项目集的格结构

（1） 寻找频繁项集（续）

- 如果{A，C，D}是大（频繁）的，则其每一个子集也是大的，如果其任何一个子集是小的，则 {A，C，D}也是小的。



项目{ACD}的非空子集是：
{AC，AD，CD，A，C，D}

{A，C，D}的子集

（1） 寻找频繁项集（续）

- 【思路】：先找出所有的频繁1-项集，以此为基础，由它们来产生候选的2-项集，通过观察数据（扫描数据库）来计算它们的频度，从而找出真正的频繁2-项集。以此类推，得到其它K-项集。

（1） 寻找频繁项集（续）

- 【Apriori算法的基本思想】：它使用一种称作逐层搜索的迭代算法，通过k-项集用于探索（k+1）-项集。

（1） 寻找频繁项集（续）

- 【Apriori算法描述】：
  - 首先，通过扫描数据集，产生一个大的候选数据项集，并计算每个候选数据项C发生的次数，然后基于预先给定的最小支持度生成频繁1-项集的集合，该集合记作L_1；
  - 然后基于L_1 和数据集中的数据，产生频繁2-项L_2集 ；
  - 用同样的方法，直到生成频繁n-项集，其中已不再可能生成满足最小支持度的（N+1）－项集L_n。
  - 最后，从大数据项集中导出规则。
- 在第一次迭代的第一步中，产生的候选集包含所有1-项集，实为数据库中所有的项，再计算各自的支持度。

（1） 寻找频繁项集（续）

- 【Apriori算法】：
  - 在第i趟扫描的过程中，对C_i进行计数，**通过对数据库的一次扫描得到每个候选项集的频度**，只有那些大的候选集被用于生成下一趟扫描的候选集，即用L_i生成C_{i+1}。
  - 为了生成大小为i+1的候选，要对前一趟扫描发现的大项目集进行连接运算。
  - 表示：$L_k * L_k = \{X \cup Y$ 其中 $X, Y \in L_k, |X \cap Y| = k - 1\}$

（1） 寻找频繁项集（续）

- Apriori算法中的关键步骤是由L_{k-1}找L_k，该步骤可分为两步：
  - 第1步（连接）：为找L_k，通过L_{k-1}与自己连接产生候选K-项集的集合。将该候选项集的集合记作C_k。设l_1和l_2是L_{k-1}中的项集，记号$l_i[j]$表示l_i的第j项。执行连接l_1和l_2，其中L_{k-1}的元素是**可连接**，如果它们前（k-2）个项相同而且第（k-2）项不同（为简单计，设$l_1[k-1]<l_2[k-1]$），即：

$$l_1[1]=l_2[1]\wedge l_1[2]=l_2[2]\wedge\cdots\cdots\wedge l_1[k-2]=l_2[k-2]\wedge l_1[k-1]<l_2[k-1]$$

则L_{k-1}的元素l_1和l_2是可连接的。连接l_1和l_2产生的结果的项集是$l_1[1]l_1[2]\cdots\cdots l_1[k-1]l_2[k-1]$。

（1） 寻找频繁项集（续）

- 第2步（剪枝）：C_k是L_k的超集，即它的成员可以是也可以不是频繁的，但所有的频繁k-项集都包含在C_k中。扫描数据库，确定C_k中每个候选的计数，从而确定L_k。然而，C_k可能很大，这样所涉及的计算量就很大。为压缩C_k，可以用以下办法使用Apriori性质：任何非频繁的（k-1）-项集都不可能是频繁k-项集的子集。因此，如果一个候选k-项集的（k-1）-子集不在L_{k-1}中，则该候选也不可能是频繁的，从而可以由C_k中删除。

（1） 寻找频繁项集（续）

- 【Apriori算法举例】：假设事务数据库D中有4个事务，最小频度是2，则算法的主要步骤如图所示。

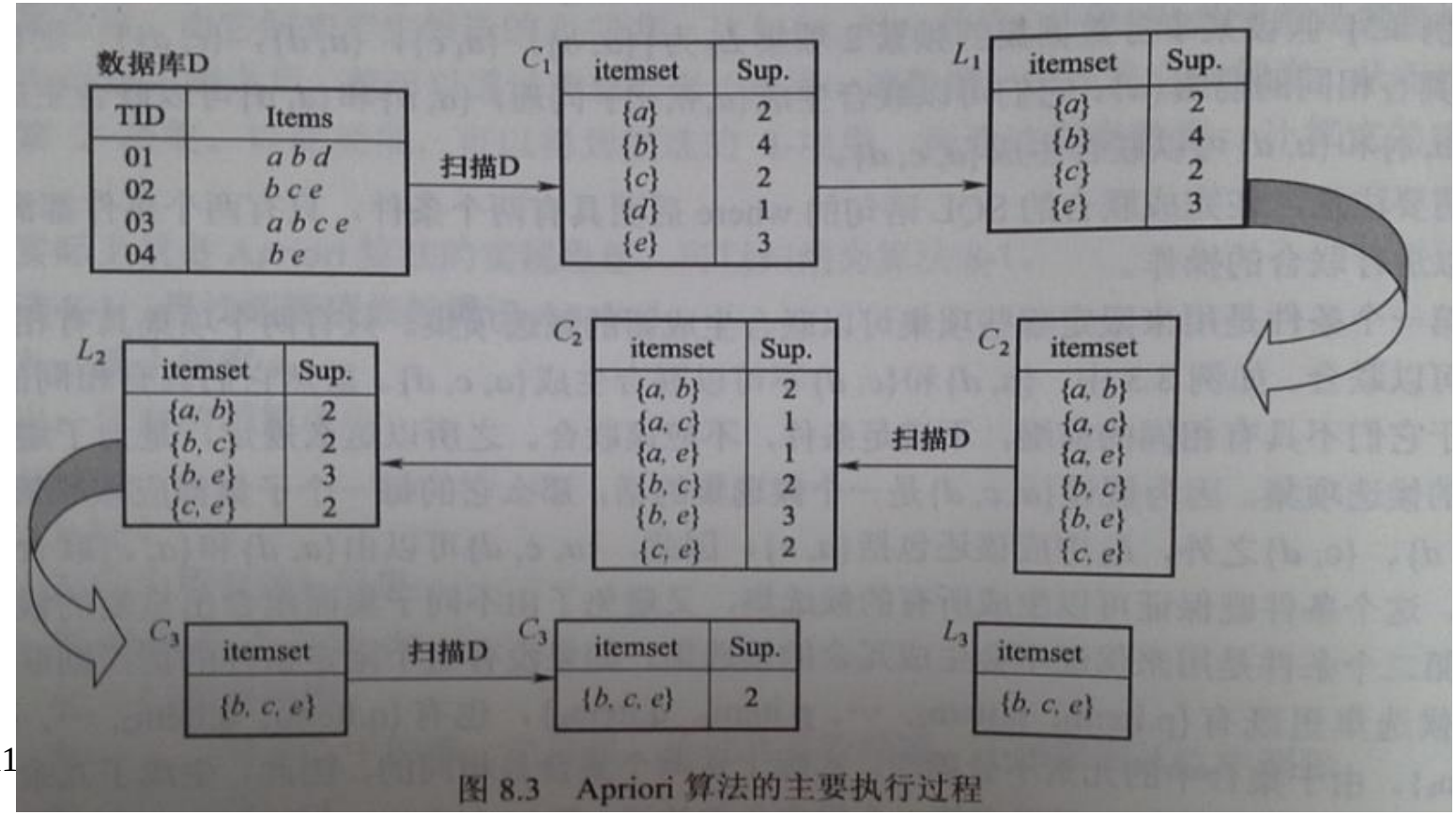


图 8.3 Apriori 算法的主要执行过程

Apriori算法——示例

最小支持计数：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

（1） 寻找频繁项集（续）

- Apriori算法是一种自底向上宽度优先搜索过程。

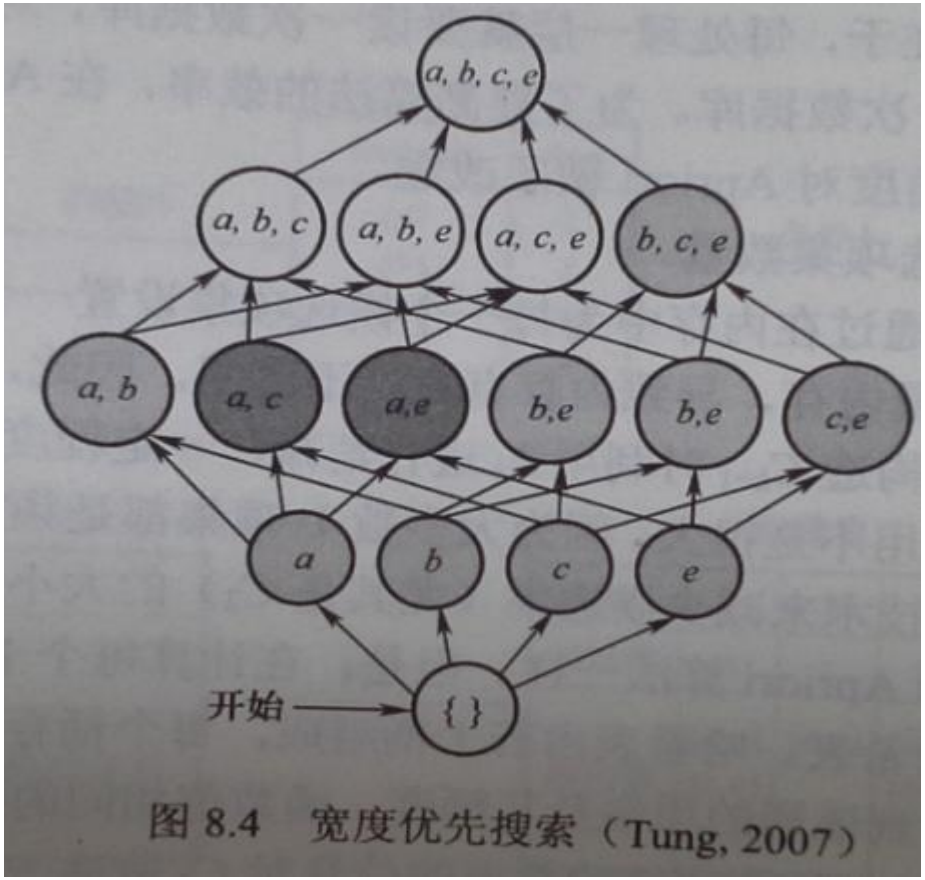


图 8.4 宽度优先搜索（Tung, 2007）

（2） 由频繁项集产生关联规则

- 一旦找出所有的频繁项集，就可以由它们来产生关联规则。
- 关联规则产生的步骤：
  - 对于每个频繁项集r，产生r的所有非空子集。
  - 对于r的每个非空子集s，如果support_count(r)/support_count(s)≧min_conf，则输出规则s⇒（r-s）。其中，support_count为支持度，min_conf为置信度。

（2） 由频繁项集产生关联规则（续）

- 结合下图数据库举例，产生关联规则方法。

演示关联规则的样本数据	
TID	**项目**
01	**a、b、d**
02	**b、c、e**
03	**a、b、c、e**
04	**b、e**

- 根据前述计算得到的频繁项集为{b、c、e}。
- 获得所有非空子集{b、c}、{b、e}、{c、e}、{b}、{c}、{e}。

（2） 由频繁项集产生关联规则（续）

- 产生的关联规则：

规则	置信度
$b \wedge c \Rightarrow e$	2/2=100%
$b \wedge e \Rightarrow c$	2/3=66%
$c \wedge e \Rightarrow b$	2/2=100%
$b \Rightarrow c \wedge e$	2/4=50%
$c \Rightarrow b \wedge e$	2/2=100%
$e \Rightarrow b \wedge c$	2/3=66%

2. 对Apriori算法的改进

- Apriori算法的主要缺点：
  - 每处理一层就要读一次数据库。
  - 对于一个有n个项目的数据集，最坏的情况需要读n次数据库。

- 为了提高算法效率，需要对Apriori算法改进。
- 人们相继提出了一些方法，从不同角度对Apriori算法进行改进。

（1）减少必须分析的候选项集数量

- Apriori算法通过在内存中为每一个候选项集设置一个计数器来计算频度。
- 当候选项集很多时将占据大量内存，导致内存不够用，需要尽量减少候选项集数量。
- Apriori算法在构造C_{k+1}时利用L_k进行消减，一定程度降低了候选项集数量，但是对C_2作用不大。

（1）减少必须分析的候选项集数量（续）

- PCY算法：通过一种基于哈希的技术来减少候选集（尤其是C_2）的大小。
- PCY算法思路：整体流程和Apriori算法一样，但是在计算每个1-项集频度生成L_1时，PCY算法顺便生成一个哈希表。
- 哈希表由若干桶组成，每个桶存放一组项集和一个计数器，用来记录通过哈希函数映射到该桶的项集及其频度，函数值相同的项集存放在同一个桶中。
- 在计算C_2时，PCY算法利用该哈希表的信息对C_2做进一步消减。

（1）减少必须分析的候选项集数量（续）

- PCY算法步骤实现过程：
  - **第一趟扫描数据库**时计算所有1-项集的频度。
  - 对每个交易，将其中的数据项进行两两组合，然后哈希到一个桶中，桶计数加1.
  - 扫描结束时，将频度大于最小频度的1-项集放入L_1中。
  - **进行第二趟扫描数据库**，完成由L_1生成C_2。每一个候选2-项集（i, j）**必须满足两个条件**：第一，i在L_1中，j在L_1中；第二，2-项集（i, j）必须哈希到一个计数值大于最小频度的桶中。

（1）减少必须分析的候选项集数量（续）

- PCY算法举例：假设哈希是h(*i*，*j*)=((order of *i*)*10+（order of *j*））mod 7。数据项a、b、c、d、e的次序（order）分别设为1、2、3、4、5。

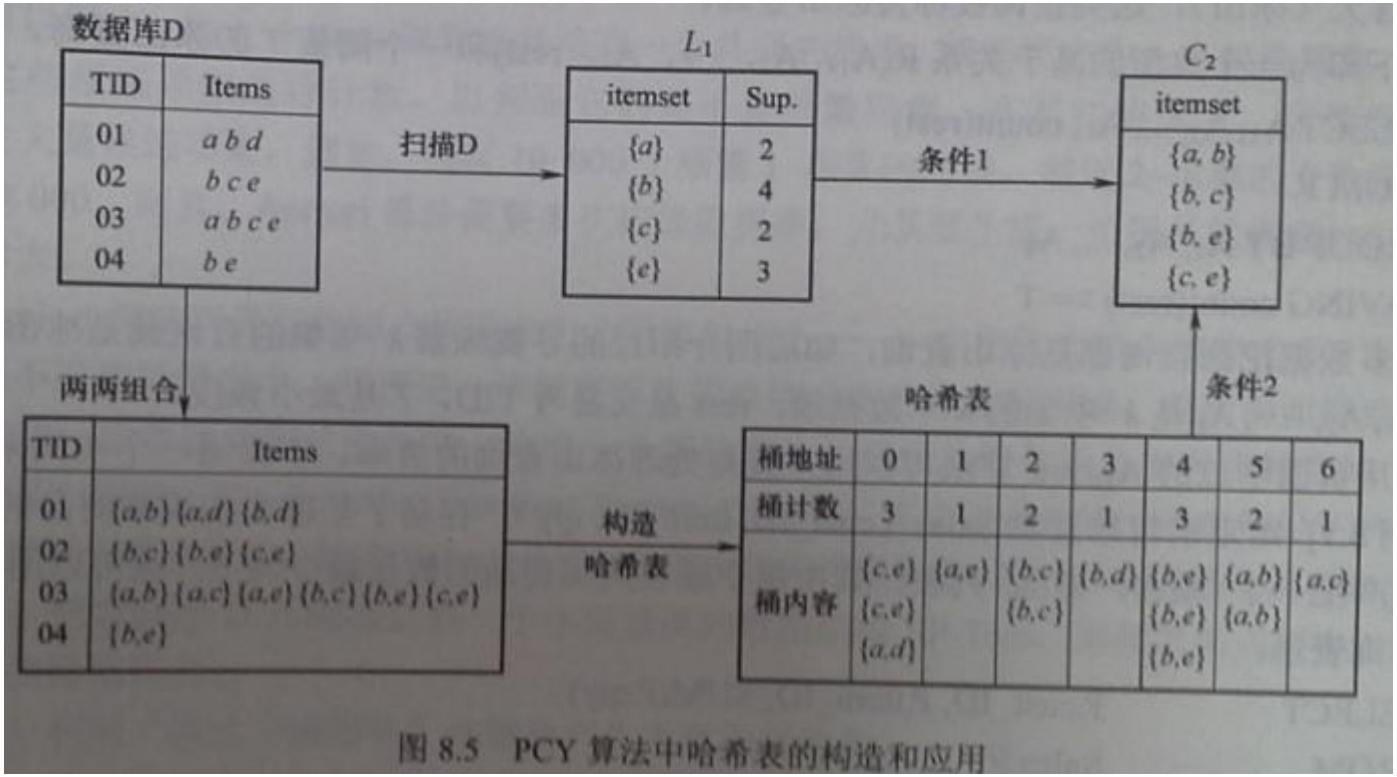


图 8.5　PCY 算法中哈希表的构造和应用

（1）减少必须分析的候选项集数量（续）

- PCY算法优点：减少了候选集C_2的大小。

（2）减少数据库扫描的次数

- Apriori算法要求多次扫描数据库。如果大项集的最大长度是k，则需要最多扫描k+1遍数据库。
- 人们提出了多种方法，通过两次或一次扫描数据库来获得所有的频繁项目集。
- 有关方法：
  - 基于采样的方法
  - 基于划分的方法

（2）减少数据库扫描的次数（续）

- 基于采样的方法
  - 取主存大小的一个数据库样本，运用Apriori算法，并且按比例伸缩最小支持度（频度）s。
  - 再对数据库进行一次完整扫描，对由样本数据库求得的频繁项集进行验证。

（2）减少数据库扫描的次数（续）

- 基于划分的方法
  - 将交易数据库D划分为n块不想交的部分D^1，D^2，...D^n（要求每一块都能够放在内存中），用Apriori算法求出每一块D^i中的所有频繁项集L^i；然后合并所有的L^i。
  - 再次完整扫描一遍数据库，对L中的每一项集进行验证。

11.1.4 关联规则挖掘的重要算法FP-Growth

- Apriori算法的特点是要产生候选项集。然后对候选项集进行计数，以判断它们是不是频繁项集。
- 在某些情况下，这类算法可能会产生大量候选项集，代价非常大。
- Apriori算法的变形虽然使其得到一定程度的改善，但并未根本改观。
- **迫切需要寻找新的算法。**

11.1.4 关联规则挖掘的重要算法FP-Growth（续）

- Han等人引入“频繁模式增长”（简称FP-增长）的概念，可以不产生候选就能够找出所有的频繁项集。
- 韩家炜现为美国伊利诺伊大学计算机系正教授。韩教授于2003年获选美国计算机协会院士（ACM Fellow）（Citation: “For contributions in knowledge discovery and data mining”， 汉译：“对知识发现和数据挖掘做出贡献”）。
- 韩教授1978毕业于郑州大学计算机科学系，同年考入中科院研究生，1985年美国威斯康辛大学计算机系博士毕业。

11.1.4 关联规则挖掘的重要算法FP-Growth（续）

- FP-Growth算法的特点
  - 把数据D压缩映射到一个小而紧凑的数据结构FP-Tree，即频繁模式树中，避免了多次扫描数据库D。
  - 利用“模式分段增长”法避免产生大量的候选集。
  - 采用分而治之的方法将数据挖掘任务分解成许多小任务，从而极大地缩小了搜索空间。

11.1.4 关联规则挖掘的重要算法FP-Growth（续）

- 【举例】使用FP-Growth算法重新对前面例子中的事务数据库进行关联规则挖掘，具体步骤分为：
  - 构造FP-Tree
  - 挖掘FP-Tree

1. 构造FP-Tree

- 对数据库的第一次扫描与Apriori算法相同，扫描结束后得到一个频繁项（1-项集）集合，以及频度。
- 设最小频度为2。将所有的频繁1-项集按频度降序排序，结果集记作L。则L={b:4，e:3，a:2，c:2}。
- 构造FP-Tree：
  - 首先创建树的根节点，用“null”标记。
  - 对数据库做第二次扫描。数据库中每条交易中的数据项按L中的次序依次处理（即根据递减频度排序），并对每个交易创建一个分枝。

1. 构造FP-Tree（续）

- 所生成的FP-Tree为：

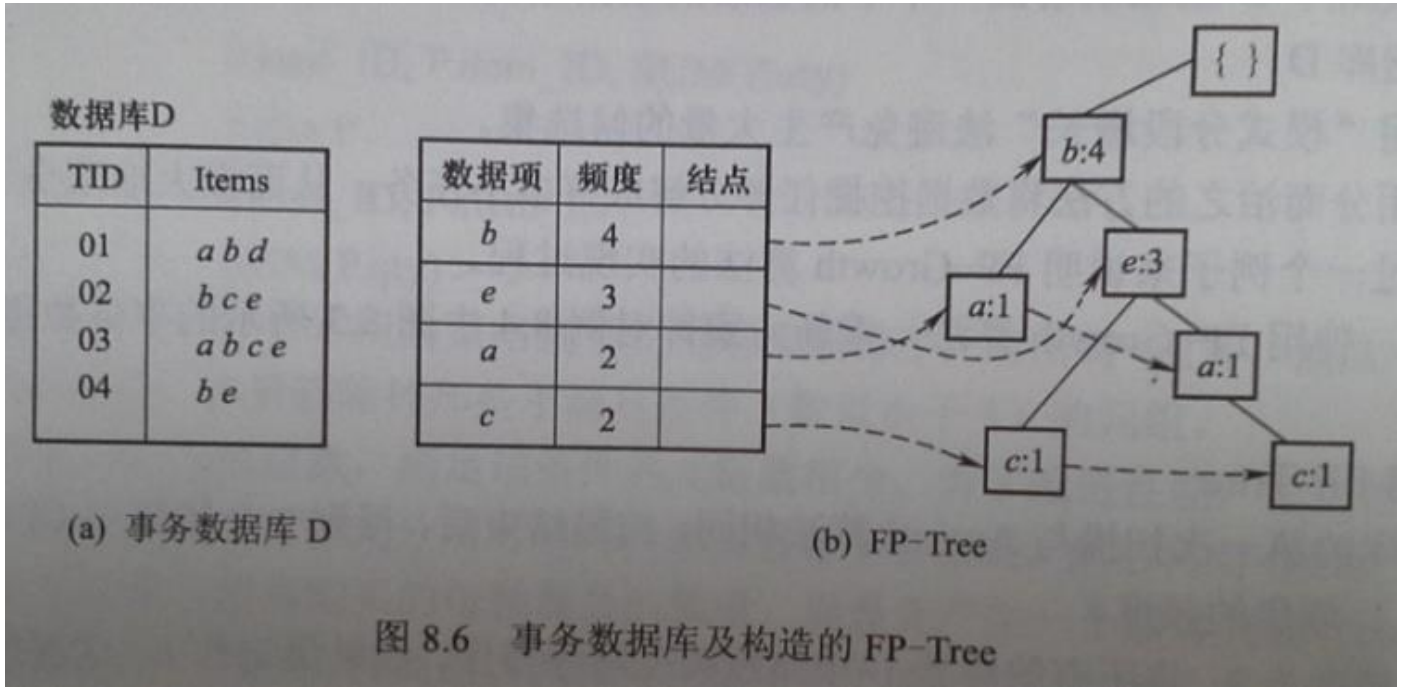
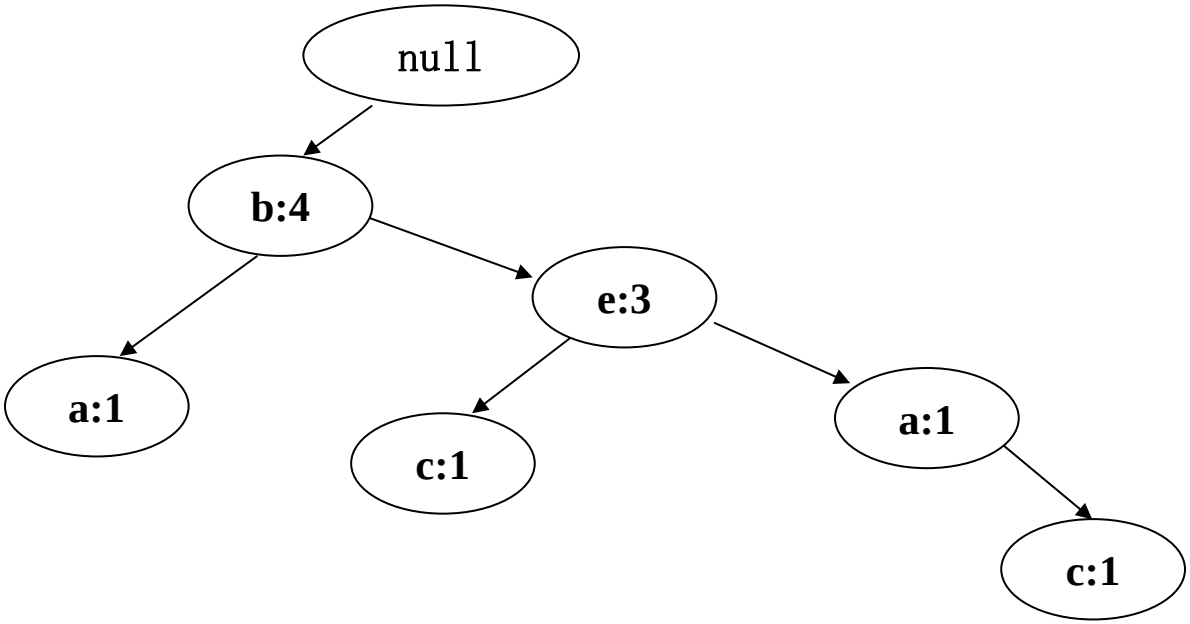


图 8.6　事务数据库及构造的 FP-Tree

1. 构造FP-Tree（续）

- 所生成的FP-Tree为：



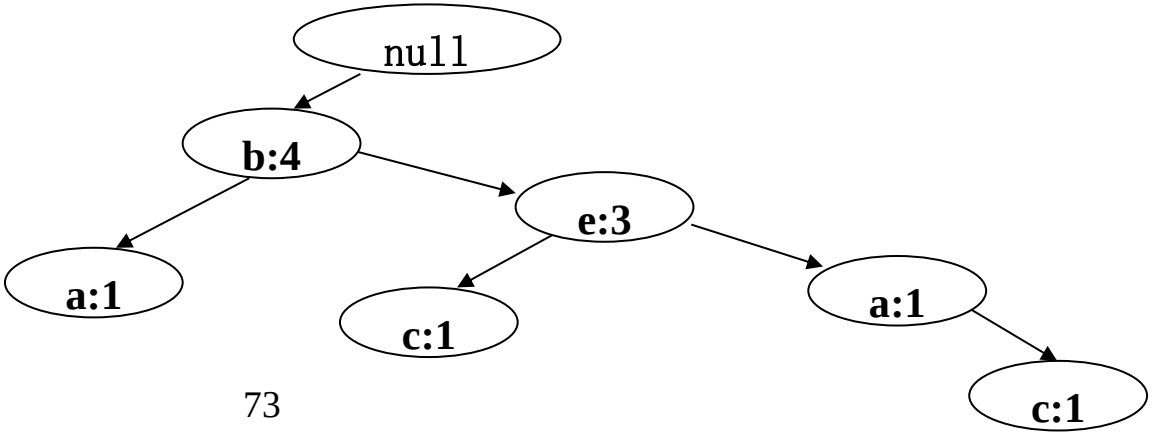
- 将对交易数据库的频繁模式挖掘问题转换成针对该FP-Tree进行挖掘的问题。

2. 挖掘FP-Tree

- 构造FP-Tree时是按照1-项集频度的降序进行的，对构造后的FP-Tree进行挖掘时，需要**按照1-项集频度的升序进行**。
- 对于每一个1-项集，首先构造它的条件数据库。
- 所谓条件数据库，是一个“子数据库”，由FP-Tree中与该1-项集一起出现的前缀路径组成。
- 具体实现：从数据项头表中首先找到该1-项集，然后顺着链表找到它在树中出现的位置，每找到一个位置，则得到从树根到该位置的一条路径，该路径就构成了条件数据库中的一部分。

2. 挖掘FP-Tree（续）

- 针对图11.6构造的FP-Tree树进行挖掘过程：
  - 先从L中的最后一个数据项c（按频度的升序）开始，沿着c的节点链表，首先发现C出现在FP-Tree的一条分枝<b:1 e:1 c:1>上，则将该路径的前缀<b:1 e:1>放到c的条件数据库中；
  - 再顺着c的链表走下去，发现c出现在FP-Tree的另一条分枝<b:1 e:1 a:1 c:1>上，则将该路径前缀<b:1 e:1 a:1>放到c的条件数据库中；
  - 得到c的条件数据库为{<b:1 e:1>，<b:1 e:1 a:1>}，构造出的FP-Tree有两个节点<b:2 e:2>，b和e的频度均不小于2，是频繁的。



2. 挖掘FP-Tree（续）

- 得到该子数据库生成的频繁模式{b，e，be}。
- 将其与生成该子数据库的项目c连接后（称为增长模式），生成所有包含c的频繁模式，即{bc：2}，{ec：2}，{bec：2}。
- 依次类推...

	条件数据库	条件FP-Tree	产生的频繁模式
e	<*b*:3>	<*b*:3>	{*b e*:3}
a	<*b*:1>, <*b*:1 *e*:1>	<*b*:2>	{*b a*:2}
c	<*b*:1 *e*:1>, <*b*:1 *e*:1 *a*:1>	<*b*:2 *e*:2>	{*b c*:2},{*e c*:2},{*b e c*:2}

图 8.7　挖掘 FP-Tree 的过程

11.1.5 其它关联规则挖掘方法

- 前面介绍的关联规则没有考虑数据对象的概念层次和蕴含多个谓词，实际生活中往往并非如此。如：惠普牌打印机->打印机->电子产品；或者数据库中不但记录了顾客购买商品的名称，而且还记录了数量、单价等，需要体现多种维度的关联关系。
- 多层关联规则
- 多维关联规则

1. 多层关联规则

- **挖掘方法：**一般采用自顶向下的策略，从最一般的概念层（第0层）开始，到较具体的某特定概念层，在每个概念层上寻找频繁项集，直到不能找到频繁项集为止。
- **最小支持度的设置：**采用逐层递减的支持度设置策略。

2. 多维关联规则

- 涉及数据表的多个字段。
- **二维关联规则**：如：性别＝“女”⇒职业＝“护士”。
- **三维关联规则**：如：年龄＝“20...30”∧职业＝“学生”⇒购买＝“电脑”。

11.1.6 关联规则的兴趣度

- 按照前述方法产生的关联规则并非都有用。
- 举例：如下是从一个有5000名学生的学校的调查结果中进行挖掘的实例。提供早餐的零售商对这些学生每天早上所从事的活动进行了一次调查。数据表明：60%的学生（3000名学生）打篮球，75%的学生（3750名学生）吃这种早餐，40%的学生（2000名学生）既打篮球，也吃这种早餐。那么如果设minsup为40%，minconf为60%挖掘关联规则，我们可以得到如下的关联规则：

打篮球⇒吃早餐　　（1）

11.1.6 关联规则的兴趣度（续）

- 这条规则相应的置信度为2000/3000=0.66，是错误的关联规则，因为吃早餐的学生的比例是75%，大于66%。打篮球和吃早餐实际上是负关联的。

- **只凭支持度和置信度阈值未必总能找出符合实际的规则。**

11.1.6 关联规则的兴趣度（续）

- 为了消除这种误导的规则，应该在关联规则A⇒B的置信度超过某个特定的度量标准时，定义它为有意义的。
- 因此有如下关联规则S（A，B）/S（A） - S（B）＞d或者：S（A，B）- S（A）*S（B） ＞k
- 式中d和k是适当的量。
- **从而提出了兴趣度的概念**

11.1.6 关联规则的兴趣度（续）

- 兴趣度
  - 为了删掉一些无趣的规则，即避免生成“错觉”的关联规则，人们定义了兴趣度的度量值，通过兴趣度来修剪无趣的规则 。
  - 今后确定关联规则可以采用三个度量值：支持度、置信度、兴趣度。

11.1.6 关联规则的兴趣度（续）

- 兴趣度定义的方法
  - 客观兴趣度
  - 主观兴趣度

11.2 序列模式挖掘

- **序列数据库：**是指包含了一序列有序事件的数据库，这些事件发生的具体时间并不重要，但事件发生的先后次序却非常关键。
- **序列模式挖掘：**是指从序列数据库中找出频繁发生的子序列的过程。
- **应用举例：**在购买电脑的人们中，60%（？）的人会在3（？）个月内购买打印机。

11.2.1 问题描述

- 序列模式挖掘最早是在1995年由Agrawal和Srikant提出的。
- 和关联规则挖掘类似，可仍然采用交易数据库作为数据源。每个交易由顾客号、交易时间、交易中购买的商品组成。

11.2.1 问题描述（续）

- 交易数据库示例

表 8.4　交易数据库示例

顾客号	交易时间	所购商品
0001	2010 年 6 月 10 日	面包、酸奶、打印纸
0002	2010 年 8 月 10 日	香蕉
0002	2010 年 9 月 15 日	面包、打印纸
0003	2010 年 3 月 19 日	香蕉、酸奶
0004	2010 年 5 月 10 日	口香糖、苹果
0001	2010 年 9 月 11 日	香皂、酸奶、方便面
0002	2010 年 9 月 17 日	香蕉、圆珠笔、饼干
0004	2010 年 8 月 23 日	面包、酸奶、啤酒、橙汁

11.2.1 问题描述（续）

- 几个概念：
  - 一个序列S是一个事件的有序列表，通常表示为$<e_1e_2e_3...e_m>$，其中e_1在e_2之前发生，e_2在e_3之前发生，...。
  - 一个事件e_i被称为序列S的一个元素。在交易数据库中e_i代表某个顾客到某个商店的一次购物行为。
  - 一个序列中的项集的个数，称为该序列的长度。
  - 一个长度为k的序列通常被称为k-序列。

11.2.1 问题描述（续）

- 按照顾客号和交易时间排序后的交易数据库

表 8.5　交易数据库（按顾客号和交易时间排序）

顾客号	交易时间	购物事件
0001	2010 年 6 月 10 日	面包、酸奶、打印纸
	2010 年 9 月 11 日	香皂、酸奶、方便面
0002	2010 年 8 月 10 日	香蕉
	2010 年 9 月 15 日	面包、打印纸
	2010 年 9 月 17 日	香蕉、圆珠笔、饼干
0003	2010 年 3 月 19 日	香蕉、酸奶
0004	2010 年 5 月 10 日	口香糖、苹果
	2010 年 8 月 23 日	面包、酸奶、啤酒、橙汁

11.2.1 问题描述（续）

- 根据上述说明：
  - 顾客0002的3次购物事件构成的序列表示为<(香蕉)(面包、打印纸)(香蕉、圆珠笔、饼干)>。
  - 该序列的长度为3。
  - 该序列是一个3-序列。

表 8.5 交易数据库（按顾客号和交易时间排序）

顾客号	交易时间	购物事件
0001	2010 年 6 月 10 日	面包、酸奶、打印纸
	2010 年 9 月 11 日	香皂、酸奶、方便面
0002	2010 年 8 月 10 日	香蕉
	2010 年 9 月 15 日	面包、打印纸
	2010 年 9 月 17 日	香蕉、圆珠笔、饼干
0003	2010 年 3 月 19 日	香蕉、酸奶
0004	2010 年 5 月 10 日	口香糖、苹果
	2010 年 8 月 23 日	面包、酸奶、啤酒、橙汁

11.2.1 问题描述（续）

- 定义：
  - 给定两个序列$a=<a_1a_2...a_n>$和$b=<b_1b_2...b_m>$，如果存在整数$1 \leqq i_1<i_2<...<i_n \leqq m$，且$a_1$包含于$b_{i1}$，$a_2$包含于$b_{i2}$，$a_n$包含于$b_{in}$，则称序列a包含于序列b，又称序列a是序列b的子序列。
- 举例：序列<（a）（bc）（f）>包含于序列<（e）（af）（g）（bcd）（f）>。

11.2.1 问题描述（续）

- 序列模式挖掘和关联规则挖掘类似，用序列S在数据库D中出现的频度（支持度）来度量S是否频繁。
- 序列S在数据库中的频度被定义为该数据库中包含S的元组数。
- 序列S在数据库中的支持度为频度与D中元组总数之比。

11.2.1 问题描述（续）

- 举例

序列号	序列
1	<a(abc)(ac)>
2	<a(bc)(ae)>q
3	<(ef)(ab)(df)cb>
4	<ebc>

- 序列<（ab）c>的频度是多少？支持度是多少？

11.2.1 问题描述（续）

- 给定一个序列数据库D，和一个最小频度min-sup，如果某序列s在D中的频度大于min-sup，则说序列s是频繁的。
- 给定一个序列集合，如果序列s不包含于任何一个其它的序列中，则称s是最大的（maximal Sequence）。
- 序列模式挖掘的任务就是要找出所有最大的频繁序列。

11.2.1 问题描述（续）

- 序列模式挖掘的大多数算法和关联规则的挖掘算法类似。
- 比较典型的算法有：
  - AprioriAll算法
  - GSP算法
  - PrefixSpan算法

AprioriAll 算法

AprioriAll 算法与*Apriori* 类似，首先遍历数据库生产候选序列并利用 *Apriori* 的特性进行剪枝来得到频繁序列。每次遍历时通过连接上一次得到的频繁序列来生长度加1的候选序列，然后对每个候选序列进行扫描，按照最小支持度来确定哪些频繁序列。

AprioriAll 算法

AprioriAll 算法的不足在于容易生成数量庞大的候选序列，同时还需要多次扫描数据库。

AprioriSome 与 *AprioriAll* 只是在序列阶段有所不同，*AprioriAll* 是首先生成所有频繁序列，然后在极大序列阶段删除那些非极大的序列。

AprioriSome 将序列分成两部分分别计数，前半部分只对一定长度的序列计数，后半部分跳过已经计数的序列。在实际过程中两个部分是混合在一起的，以减少候选序列占用的资源。

AprioriAll 算法

GSP 算法是*AprioriAll* 的扩展算法，其算法的执行过程和 *AprioriAl*类似，最大的不同就在于*GSP* 引入了时间约束、滑动窗口和分类层次技术，增加了扫描的约束条件，有效地减少了需要扫描的候选序列的数量，同时还克服了基木序列模型的局限性，更切合实际，减少多余的无用模式的产生。

另外*GSP*利用哈希树来存储候选序列，减小了需要扫描的序列数量。

AprioriAll 算法

在*Rakesh Agrawal* 关于序列模式挖掘的论述中，将序列模式挖掘的一般步骤分为5个阶段，即：

- 排序阶段
- 频繁项集阶段
- 转换阶段
- 序列阶段
- 选极大序列阶段