1. **自己提出的问题的理解：**

问题1：P34图2.13中合并步中，问什么说如果s1的第一个项目去掉后得到的余串和s1的最后一个项目去掉后的得到的余串相同就合并？

讨论后的理解：在使用图2.13的候选生成函数时，如果使用多最小支持度，则会产生<{1,2}{4}>（1具有最小MIS）是频繁序列，<{2}{4,5}>不是频繁序列，但是<{1,2}{3,4}>应该是频繁序列，却不能被生成的情况。这种情况只会产生在s1的最小MIS在开头项目和s2的最小MIS在最后项目的情况，因为只有在这种情况下，才会出现因为一个不频繁的序列，在增加了第一项或者最后一项MIS比较小的项目之后，整个序列的MIS被拉低（门槛降低），所以会出现本应该被合成的序列却没有被合成。

问题2：P38图2.15中剪枝部分为什么“出了唯一的不包含最小MIS值的项目的k-1子序列之外，其他子序列都是非频繁的，则这个时候序列被剪枝”？而在传统的Apriori算法中剪枝的时候的规则则是如果有一个k-1的子序列不频繁，那么剪枝。

讨论后的理解：翻译错误。那么之所以需要在剪枝的时候是判断的任意一个k-1子序列不频繁则剪枝，是因为向下封闭属性的存在，一个频繁序列的所有子序列必须是频繁序列，等价于如果子序列中有一个是不频繁的，那么该序列是不频繁的，所以在生成k长度的频繁序列的时候只要判断让所有的k-1子序列都是频繁的就好了。

1. **别人提出的问题的理解：**

问题1：prefixspan算法的核心思想是什么？

自己的理解：prefixspan算法首先找出所有的频繁项目，首先因为长度为1的序列模式即为这些值，而且因为长度大于1的所有序列模式必然是包含仅仅包含这些值的（和MS的算法不同，因为不会因为别的项目的MIS而拉低自己的MIS，所以必须所有的项目都满足MIS）

GSP相当于广搜，逐个k逐层的求解所有的频繁序列模式，而PrefixSpan相当于深搜，将前缀一点一点增加，从1-频繁序列出发，一个一个的增加，到2-频繁序列，直到最后增加到k-频繁序列。每增加一个就添加到相对应的k-序列中，直到找不到映射之后就返回。

2、问题2：P37页，为什么具有最小MIS的项目（设为x），不是s1中第一个项目，则s2中必然含有这个x? （例23？）

自己的理解：因为s1和s2在生成一个k频繁序列的过程中，需要s1和s2中所有的项目都满足最小支持度要求，如果s1的第一个项目不是最小的，那么假设最小的一个项目是s1m（m>1），因为在合并s1和s2生成k频繁序列的时候需要比较s1的除去第一项的序列和s2的除去最后一项的序列，此时两个子序列是相同的，因为s1中最小的一个项目是s1m，那么s1m必然也出现在s2(m-1) ，所以说如果具有 最小MIS值的项目不在s1的第一项，那么必然会在s2中出现。

3、问题3：P40写道PrefixSpan算法的主要优点使不生成候选序列，但在每一次对映射数据库进行匹配时，都需要遍历一次序列集，从时间复杂度问题上考虑，是不是并没有优于GSP呢？

自己的理解：和GSP算法相比，prefixspan算法不会产生候选序列，但是prefixspan算法和GSP算法本身思想并不相同，采用分治的思想从1频繁序列开始产生k频繁序列，因此并不需要每次都去遍历原始数据库，只需要使用递归的方法，在遍历其分治范围的一定长度的比较小的数据库，所以相对GSP算法，确实会在遍历数据上减少大部分的时间。

两种算法的比较：

|  |  |  |
| --- | --- | --- |
| 属性 | GSP | PrefixSpan |
| 候选序列 | 产生 | 不产生 |
| 数据结构 | Hash Tree | Wap tree |
| 数据库分割 | 否 | 是 |
| 对原始数据的扫描次数 | 反复多次 | 2次 |
| 算法执行 | 循环 | 递归 |

从表格中可以看出，对于遍历GSP也采用了改变数据结构使用Hash Tree的方式存储数据来减小时间复杂度。

1. 读书计划

1、本周完成的内容章节：2.5-2.9

2、下周计划：第五章

1. 读书摘要及理解或伪代码的具体实现

**(1)读书摘要：**

## 2.7 基于GSP挖掘序列模式

### 2.7.1 GSP算法

GSP算法需要注意的一点是在合并的时候，书上写s1去掉第一个元素和s2去掉最后一个元素剩下的余串相同，意思是需要序列中的项集格式也相同，比如<{1,2}>和<{1}{2}>是不同的。

### 2.7.2 多最小支持度挖掘

（1）序列模式挖掘和普通的关联规则挖掘不同的一点是因为项目是有固定顺序的，不能按照传统的关联规则挖掘去使用MIS对项目进行排序，这个时候MIS最小的项目就有可能在序列的任意位置。

（2）能够将两个序列s1和s2进行合并的条件：

s1[1]是最小MIS（这里容易忘记）：

①删除s1[2]和s2[-1]之后剩下的余串相同的

②MIS[s2[-1]]>MIS[s1[-1]]

操作：将s2[-1]添加到s1末尾，此时又有两种情况：

①将s2[-1]作为独立元素加入到s1末尾，可能的情况：

s2[-1]本来就是一个独立的元素

②将s2[-1]作为s1最后一个元素的末尾加入到s1：除了上述情况的其余情况（感觉这里书上写的有点怪）

s2[-1]是最小MIS：和上面相反

其余情况

（4）注意一点，传统的剪枝都是只要存在任一k-1子序列不满足最小支持度要求，就需要剪枝，而这里需要遍历除了不包含第一个项目的那个子序列外的所有k-1子序列，需要所有的都是非频繁才需要剪枝。

## 2.8 基于PrefixSpan算法的序列模式挖掘

### 2.8.2 多最小支持度挖掘

（1）最小支持计数是MIS的百分制形式乘以n之后向上取整

（2）在算法第三部的r-PrefixSpan函数中生成的序列模式必须包括ik

（3）注意每一轮之后删除的ik是在S中删除而不是Sk

## 2.9 从序列模式中产生规则

（1）2.9.2中书上说“另外要注意一点，由于使用了通配符，仅基于频繁序列是不足以计算规则的置信度的，还需要扫描数据”，这里应该说的是频繁序列中是具体的某些支持度足够的序列，但是因为目的是匹配标签，需要知道某个具体的标签序列在所有的可能序列中出现的次数，所以这里需要计算的是序列数据库而不是频繁序列。

（2）分类序列规则中的覆盖是指序列匹配，标签可以不同，此时序列数据库中的序列覆盖了分类序列规则中的序列。满足是指标签也需要相同。