1. （必填）自己提出的问题的理解（罗列全部）：
2. **提出的问题1：**

P41最后一行，经过算法计算之后得到的序列模式集合为包括{<{30}，{40}>,<{40}，{30}>}这样的序列，算不算是矛盾呢？需不需要考虑删除呢？

讨论后的理解：

尽管算法的最终目的是找到频繁序列集合，但是像{<{30}，{40}>,<{40}，{30}>}这样模式的序列时并不会矛盾。因为不管是先发生30后发生40，还是先发生40后发生30，这都是一种30和40之间关系的体现，不仅如此，当双向联系建立起之后，两个项目之间的联系会更紧密。即在产生规则时，看到其中一个项目出现，我们就可以预测另一个可能会跟着出现。

1. **提出的问题2：**

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呢讨论后的理解：

**GSP 算法**产生大量的候选序列，需要有足够的存贮空间。还需要对原始数据进行反复多次的扫描来计算支持度，需要占用很多的运行时间。当支持度下降的时候，频繁序列的数量呈指数上升，这样一来所花费的扫描时间也会呈指数级上升，并且候选序列的数量也会随着待挖掘序列的长度呈指数增长关系，算法的执行效率将会大大下降。GSP 由于采用了约束条件，减小了候选序列的数量，其执行效率比前者要高，但同时由于约束条件的使用，相应会使算法复杂一些，也会以相应的开销为代价。

**PrefixSpan 算法**属于模式增长方法，它的查找更加集中和有效。算法不生成大量的候选序列，而是以某种压缩的形式保留了原数据库基本的数据分组。算法每一次迭代不是扫描完整的原数据库来匹配相应的全部候选序列，而是限定在投影数据库中，大大节省了算法执行时间。在所需存贮空间，PrefixSpan 仅仅基于频繁前缀子序列投影，并通过

在其后添加后缀来实现序列的增长，在时间和空间上的执行性能比前者更优。

1. （必填）别人提出的问题的理解（选择几个问题罗列，并给出理解）：
2. **问题3**：

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是如果有一个k-1的子序列不频繁，那么剪枝。

自己的理解：

根据Apriori算法的要求，在剪枝时如果k-1序列中出现了一个序列不频繁，则会将该序列减掉。这也是GSP算法的思路，书中写道存在任意不频繁k-1序列则将其剪枝，所以两者表达的意思是一致的。

1. **问题4：**

GSP算法每次计算支持度的时候都要重新遍历数据，有什么办法可以优化？

自己的理解：

GSP算法很直白、简单，但缺点也很明显，即每次计算序列支持度时都学要扫描一遍数据集。SPADE算法则可以看成是GSP算法的提升，过程和GSP类似，只是在扫描的时候不是扫描整个数据库，而是扫描ID\_LIST。通过便利一次数据库得到的经验知识来降低多次对数据库的扫描。SPADE不仅通过减少对数据库的扫描降低I/O成本通过搜索ID\_list降低计算成本。

1. **问题5：**

Prefixspan算法思想？

自己的理解：

PrefixSpan算法的目标是挖掘出满足最小支持度的频繁序列。它从频繁1项集出发，一步步的挖掘2项集，直到最大的K项集。PrefixSpan算法也类似，它从长度为1的前缀开始挖掘序列模式，搜索对应的投影数据库得到长度为1的前缀对应的频繁序列，然后递归的挖掘长度为2的前缀所对应的频繁序列，以此类推，一直递归到不能挖掘到更长的前缀挖掘为止。

1. （必填）读书计划

1、本周完成的内容章节：第三章（4.5-4.9）

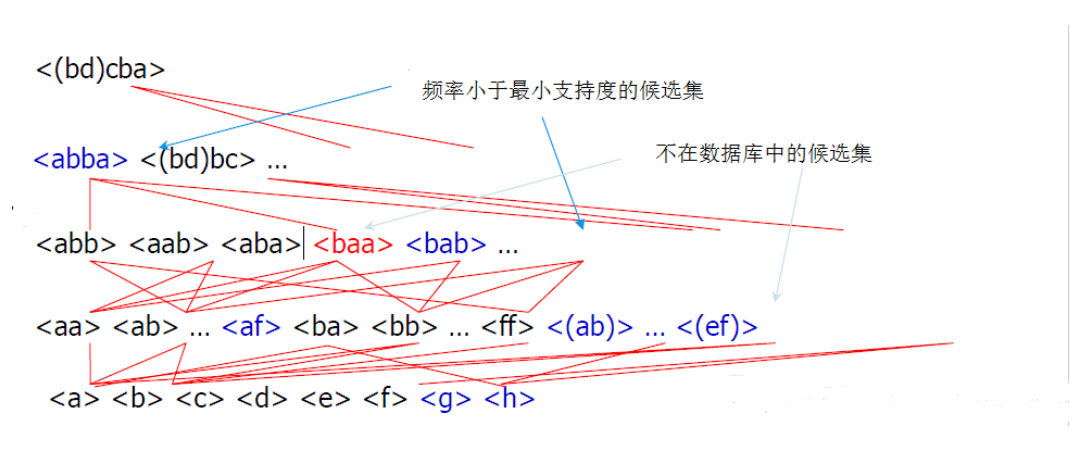
2、下周计划：第四章（5.1）

四、（选做）读书摘要及理解或伪代码的具体实现（读书摘要、伪代码的具体实现代码等可以写到这个部分）

1、读书摘要及理解（选做）

摘要：本周完成了关联规则相关内容，包括频繁项目集生成的Apriori算法和关联规则生成算法，以及多最小支持度的关联规则挖掘。

**GSP算法**是AprioriAll算法的扩展算法，而AprioriAll算法为Apriori类算法，故GSP算法也是一个Apriori类算法。GSP算法是一种非常有效的序列模式挖掘算法，该算法使用一种称作为逐层搜索的迭代方法，首先找出频繁1-序列模式的集合F1，F1用于寻找频繁2-序列模式F2，F2用于寻找频繁3-序列模式、F3...，如此下去，直到不能找到频繁序列模式为止。。算法过程如下：



GSP算法的基本思想和Apriori算法一样，其主要步骤包括三个步骤：

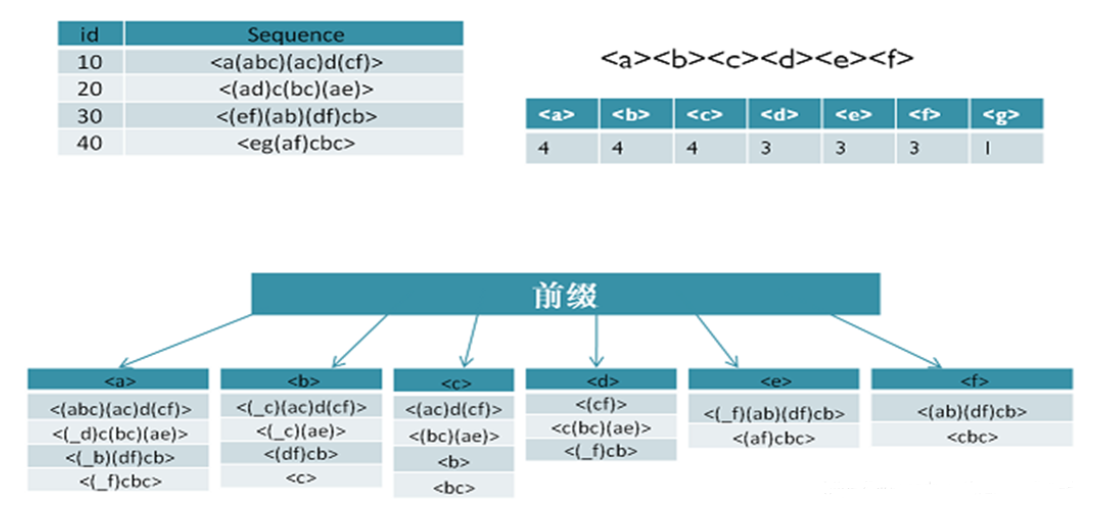
1）扫描序列数据库，得到长度为1的序列模式L1，作为初始的种子集；

2）根据长度为i的种子集Li通过连接操作和剪切操作生成长度为i+1的候选序列模式Ci+1；然后扫描序列数据库，计算每个候选序列模式的支持数，产生长度为i+1的序列模式Li+1，并将Li+1作为新的种子集；

3）重复第二步，直到没有新的序列模式或新的候选序列模式产生为止。

**PrefixSpan算法**的全称是Prefix-Projected Pattern Growth，即前缀投影的模式挖掘。其采用一种类深度优先搜索遍历的思想去遍历序列，在每一轮遍历中，将序列分为前缀和后缀，然后以后缀作为模式对序列进行映射，构建映射数据库。

算法过程如图所示：



算法过程如下：

1）找出所有长度为1的前缀和对应的投影数据库

2）对长度为1的前缀进行计数，将支持度低于阈值α的前缀对应的项从数据集S删除，同时得到所有的频繁1项序列，i=1.

3）对于每个长度为i满足支持度要求的前缀进行递归挖掘：

a) 找出前缀所对应的投影数据库。如果投影数据库为空，则递归返回。

b) 统计对应投影数据库中各项的支持度计数。如果所有项的支持度计数都低于阈值α，则递归返回。

c)将满足支持度计数的各个单项和当前的前缀进行合并，得到若干新的前缀。

d) 令i=i+1，前缀为合并单项后的各个前缀，分别递归执行第3步。

2、伪代码的具体实现(选做)

实现了GSP算法：

算法准备：



算法实现：

