FP-growth

前言

你用过搜索引擎挥发现这样一个功能：输入一个单词或者单词的一部分，搜索引擎酒会自动补全查询词项，用户甚至实现都不知道搜索引擎推荐的东西是否存在，反而会去查找推荐词项，比如在百度输入“为什么”开始查询时，会出现诸如“为什么我有了变身器却不能变身奥特曼”之类滑稽的推荐结果，为了给出这些推荐查询慈祥，搜索引擎公司的研究人员使用了本文要介绍的一个算法，他们通过查看互联网上的用词来找出经常在一块出现的词对，这需要一种高效发现频繁集的方法。该算法称作FP-growth,又称为FP-增长算法，它比Apriori算法要快，它基于Apriori构建，但在完成相同任务时采用了一些不同的技术。不同于Apriori算法的“产生 - 测试”，这里的任务是将数据集存储在一个特定的称做FP树的结构之后发现频繁项集或者频繁项对，即常在一块出现的元素项的集合FP树,这种做法是的算法的执行速度要快于apriori，通常性能要好两个数量级以上。

FP-growth算法(FP, Frequent Pattern)

FP-growth算法只需要对数据库进行两次扫描。而Apriori算法对于每个潜在的频繁项集都会扫描数据集判定给定的模式是否频繁，因此FP-growth算法要比Apriori算法快。

FP-growth算法只需要扫描两次数据集，第一遍对所有数据元素出现次数进行计数，第二遍只需考虑那些频繁的元素。发现频繁项集的基本过程分为两步，构建FP树和从FP树中挖掘频繁项集。

简单来说，算法的目的就是在多个出现的数据项中找到出现次数最多的数据项或者数据项集合，这里的最多指的是出现次数大于等于给定的阈值(最小支持度)。找到单个数据项的次数较为简单，只需要遍历计数即可，但是对于数据项的组合即数据项集的出现次数较难确定，比如，某个数据项A与数据项B的出现次数都是频繁的，但是他们的组合也就是说他们同时出现的次数却不频繁。数据集中出现较为频繁的数据项组合我们成为频繁项集，该频繁项集的大小大于等于1。FP-growth算法就是挖掘数据中的频繁项集算法中的一种。

在介绍FP-growth算法之前先介绍一些概念。

一、 数据挖掘算法中的数据是按照组(条)分的，一组数据是一个事务(这里解释不严谨，读者就理解为数据是一条一条输进算法里的，每条数据中包含一些数据项，一条数据就是一个事务)。

二、 FP树，FP树是FP-growth算法中构建的树形数据结构，用于组织数据。该树中每个叶结点到根节点的路径中所有数据项就是与该叶结点同时出现的数据们，不同叶结点到根节点的路径中共同的部分是大家共同的数据。

三、 CPB(Conditional pattern base)条件模式基，CPB是指FP树中叶结点到根节点这条路径中，不包含跟结点和叶结点的其他所有数据的集合(或者说是这些结点组成的路径，即前缀路径)。算法会对每个频繁项用其所有的CPB构建**条件FP树。**这棵树还是FP树，只不过所用的数据是某个频繁项的CPB。条件FP树是整个算法的难点，因为频繁项集通过该树挖掘，挖掘的过程是递归的。**这里简单理解就是对于频繁项T的条件FP树中所有的数据项是可以与频繁项T进行组合形成频繁项集的元素的集合。**

FP-growth算法运行过程

对于FP-growth算法的介绍，本文打算通过一个例子来解释。

假设某个超市想通过分析用户的购买习惯来改变进货数量以实现获利最大化。现在给出了某天的购物记录，为了理解方便假设所有商品的每个用户只买了一件，也就是对于一条交易记录只在乎商品的种类，数量不予考虑。再者，为了方便书写这里把商品名用字母表示。

记录ID	记录中的商品名
001	r,z,h,j,p
002	z,y,x,w,v,u,t,s
003	z
004	r,x,n,o,s
005	y,r,x,z,q,t,p
006	y,z,x,e,q,s,t,m

一、 遍历数据集，统计所有元素出现次数

r	z	h	j	p	y	x	w	v	u	t	s	n	o	q	e	m
3	5	1	1	2	3	4	1	1	1	3	3	1	1	2	1	1

此处设置最小支持度为3，也就是所出现次数大于等于3的即为频繁。在构建FP树时，为了使各个事务(记录)的公共部分在同一路径上，因此创建树时，各个事务应该一定顺序排列，此处按照商品出现次数的倒序排序。所以经过筛选得

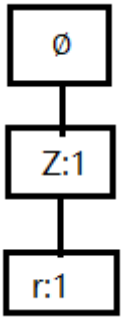
记录ID	记录中的商品名	筛选且排序后的商品
001	r,z,h,j,p	z,r
002	z,y,x,w,v,u,t,s	z,x,y,s,t
003	z	z
004	r,x,n,o,s	x,s,r
005	y,r,x,z,q,t,p	z,x,y,r,t
006	y,z,x,e,q,s,t,m	z,x,y,s,t

二、 构建FP树

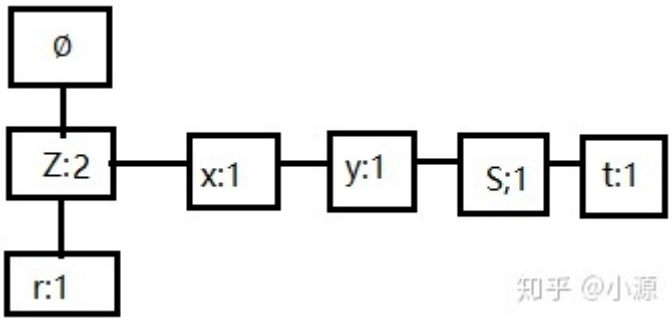
FP树的根节点是空结点，逐条将排序筛选后的记录插入树中。如果记录中的结点在树的该路径中有则该结点计数加一，否则分支。

\1. 初始时，FP树只有一个结点即空结点

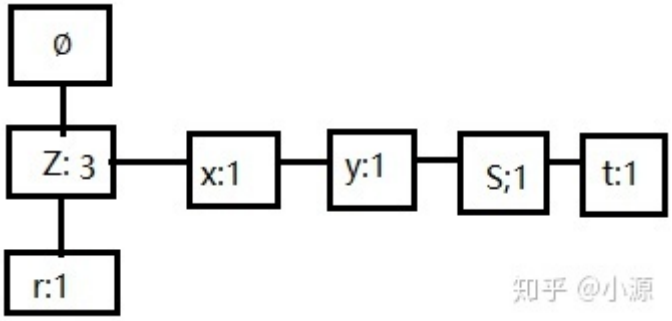
\2. 将{z,r}记录插入后得



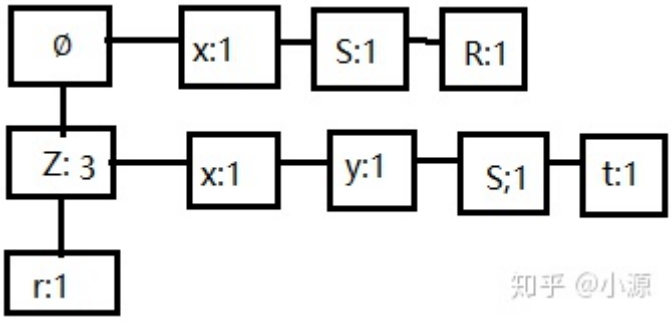
\3. 将{z,x,y,s,t}记录插入后得



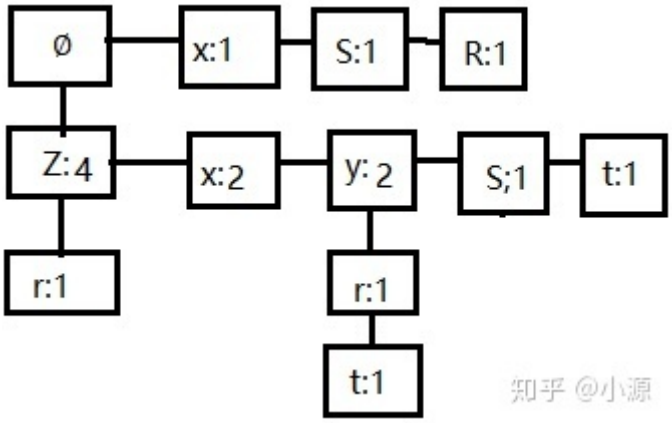
\4. 将{z}将入



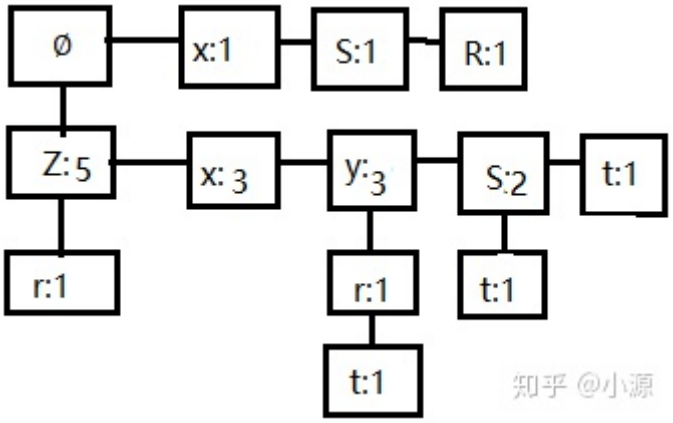
\5. 将{x,s,r}



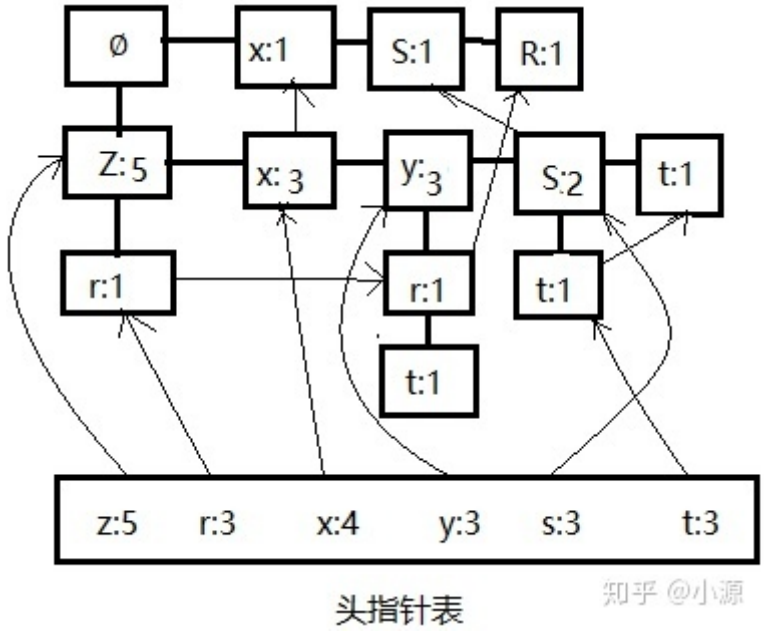
\6. 将{z,x,y,r,t}



\7. 将{z,x,y,s,t}



此时FP树已经构建的差不多了，为了在找频繁项集时方便找到树中相同的数据，需要把相同的数据项连接起来(结点链接)



三、 从FP树中挖掘频繁项集

创建了FP树之后，就可以抽取频繁项集了。与Apriori算法类似，首先从单个元素集合开始，然后在此基础上逐步构建更大的集合。FP-growth算法中的频繁项集是递归挖掘的，其过程简单解释就是在集合中每次加入一个新元素a后得频繁项集S，在此基础上继续创建a的条件FP树把对应不满足的元素删除得到对应的头表H，然后在以此递归创建H中所有元素的条件FP树得其头表，直到某递归层H为空。有些复杂，具体看示例演示过程。

首先，查找每个频繁项的条件模式基

频繁项	条件模式基
z	{}5
r	{x, s}1, {z, x, y}1, {z}1
x	{z}3, {}1
y	{z, x}3
s	{z, x, y}2, {x}1
t	{z, x, y, s}2, {z, x, y, r}1

找到所有频繁项的CPB后就是创建条件FP树，递归的挖掘频繁项集。

计算过程如下：

找到所有元素的CPB之后(编程时不用把所有元素CPB都找出再进行下一步，直接递归操作)，创建对应的条件FP树，将各个CPB中的元素的出现次数进行统计，剔除不满足最小支持度的集合，得到的就是该元素对应条件FP树。比如对于元素t，经过计数后得{z:3, x:3, y:3, s:2, r:1}。将s与r剔除因为不满足最小支持度3。对应的含义是{t, s}, {t，r}两个项集不频繁，注意此处元素t是频繁的，但是与s，r的组合不频繁。

频繁项	筛选后的条件模式基
z	{}5
r	{}0
x	{z}3, {}1
y	{z, x}3
s	{x}2，{x}1
t	{z, x, y,}3

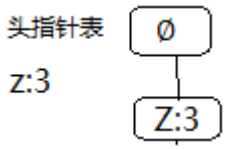
然后开始创建频繁项集。频繁项集的产生过程其实就是不断组合元素得到不同长度的集合，找到出现频数高的集合。接下来对每个元素分别创建包含它们且长度不同的集合。我们为了方便介绍将各个元素条件FP树产生的头指针表记为H。

这里挑选元素的顺序是按照每个条件FP树中频繁项频数的逆序选择的。

\1. 对于z来说，z本身是频繁的，所以包含单个元素的项集频繁即{z},又因为H为空所以最终Z的频繁项集为{z}

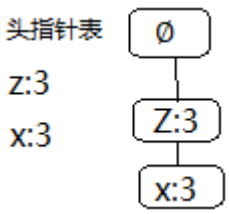
\2. 对于r来说，r本身是频繁的，所以包含单个元素的项集频繁即{r},又因为H为空所以最终r的频繁项集为{r}

\3. 对于x来说，x本身是频繁的，所以包含单个元素的项集频繁即{x},又因为H为{{z}3, {}}，对应的条件FP树如图

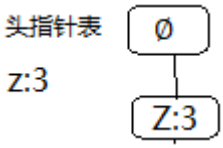


所以x与z的组合频繁，而z的H为空，也就是z对应的条件FP树为空，所以含x的频繁项集为{{x}，{x，z}}

\4. 对于y来说，y本身是频繁的，所以包含单个元素的项集频繁即{y},又因为H为{{z, x}3}，对应的条件FP树如图

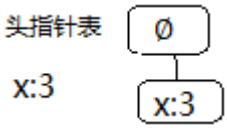


所以z与y组合的集合频繁。而z的H为空即z的条件FP树为空，所以y与z的组合只有{y, z}。



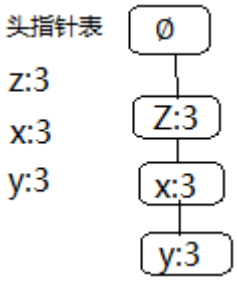
而y与x的组合也频繁，所以{y，x}频繁，然后x的条件FP树如上图。注意此处，x的条件FP树是在y的FP树的基础上创建的，也就是说，该条件FP树中与x频繁的元素也与y频繁，因为该元素也出现在y的FP树中。x的H包含z，z的FP树为空，所以不在递归下去，故{y，x，z}也频繁。所以包含y的频繁项集为{{y}，{y, z}，{y，x，z}，{y，x}}。

\5. 对于s来说，s本身是频繁的，所以包含单个元素的项集频繁即{s},又因为H为{{x}2，{x}1}，条件FP树为



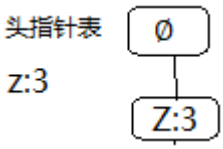
同理{s，x}频繁，将x加入后继续生成x的条件FP树为空，不在继续寻找。得包含s的频繁项集为{{s}，{s，x}}

\6. 对于t来说，t本身是频繁的，所以包含单个元素的项集频繁即{t},又因为H为{{z, x, y,}3}，条件FP树为



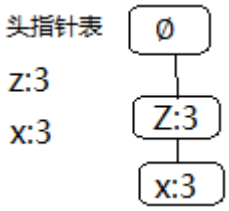
然后，将z加入集合{t}，z的条件FP树为空不在继续遍历得频繁项集{t，z}。

然后，将x加入集合{t}，因为x的条件FP 树为下图，故在{t，x}中加入z得



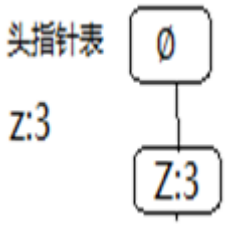
{t，x，z}，由于z的条件FP树为空，不在继续遍历。故此次过程得到的频繁项集为{{t，x}，{t，x，z}}。

然后，将y加入集合{t}，y对应的条件FP树为下图，故在{t，y}中加入z，由于z的条件FP树为空，不在继续遍历。得{t，y，z}。故此次递归得到的频繁项集为{{t，y，z}，{t，y}}



然后，将x加入{t，y}，生成x的条件FP树为下图，故在{t，y，x}中加入z得{t，y，x，z}。由于z的条件FP树为空，不在继续遍历。故此次递归得到的频繁项集为

{{t，y，x，z}，{t，y，x} }



最终结果为

频繁项	频繁项集
z	{z}
r	{r}
x	{{x}，{x，z}}
y	{{y}，{y, z}，{y，x，z}，{y，x}}
s	{{s}，{s，x}}
t	{{t}，{t，z}，{t，x，z}，{t，x}，{t，y，z}，{t，y，x，z}，{ t，y，x }，{ t，y}}

总结一下过程，对于每个元素a的FP树的H中的每个元素b都与a的组合频繁，而**在a元素对应的H中**，有与b频繁的元素c，那么{a，b，c}也频繁。同理可以处理c及其他元素。之所以可以这样做是应为元素a与其对应的H中的所有元素都频繁出现，而在该H中，b又与c频繁出现，c在该H中，所以a，b，c频繁。**每一个元素对应不同的H，每一次递归时，对应的频繁项集的长度就会增加，H范围会在上一递归层H_old的范围的基础上缩小。**