**Theoretical description of the algorithm:**

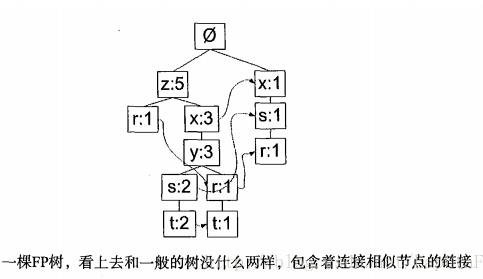
Advantages: Generally faster than Apriori.

Disadvantages: Implementation is more difficult and performance will degrade on some data sets.

Applicable data type: nominal data.

The FP-growth algorithm stores data in a compact data structure called a FP tree. FP stands for Frequent Pattern. An FP tree looks similar to other tree structures in computer science, but it connects similar elements through links, and the connected element items can be viewed as a linked list.

FP-tree is as follows:



Unlike the search tree, an element item can appear multiple times in a FP tree. The FP tree stores the frequency of occurrences of item sets, and each item set is stored in the tree as a path. A collection of similar elements will share a part of the tree. The tree will only fork when the sets are completely different. The tree element gives the individual elements in the set and their occurrences in the sequence, and the path gives the number of occurrences of the sequence.

The link between similar items is the node link, which is used to quickly find the location of similar items.

The workflow of the FP-growth algorithm is as follows. First build the FP tree and use it to mine frequent itemsets. To build an FP tree, you need to scan the original data set twice. The first pass counts the number of occurrences of all element items. That is, if an element is infrequent, the superset containing the element is infrequent, so there is no need to consider these supersets. The first pass of the database is used to count the frequency of occurrences, while only the frequent elements are considered in the second pass.

FP-growth general process

(1) Collect data: Use any method.

(2) Preparing data: Since the storage is a collection, discrete data is required. If you want to process continuous data, you need to quantize them to discrete values.

(3) Analyze data: use any method.

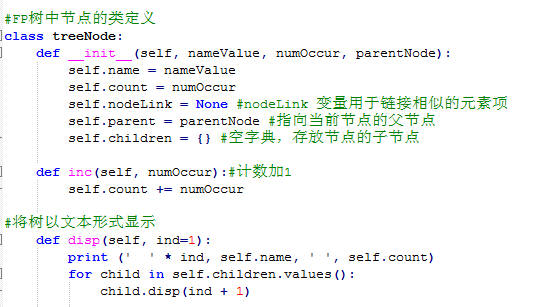
(4) Training algorithm: construct a FP tree and mine the tree.

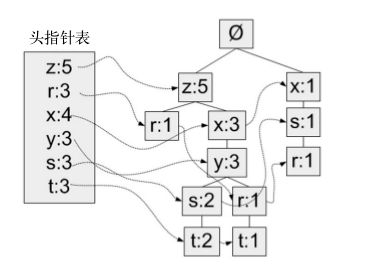
(5) Test algorithm: There is no test process.

(6) Use algorithm: It can be used to identify frequently occurring element items for use in making decisions, recommending elements, or making predictions.

**Key code for implementation:**

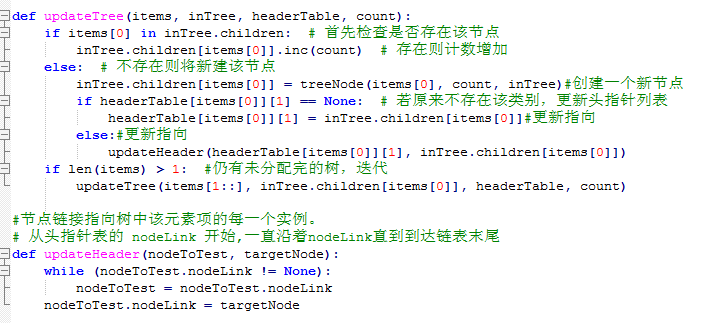
First, Create FP-tree



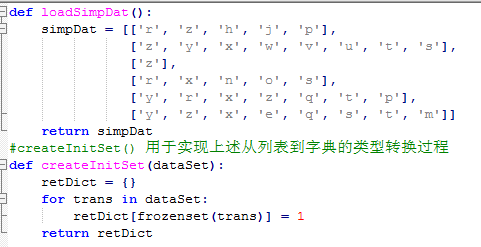
Before creating a true frequent set FP tree, you need to filter the data (not meeting frequent requirements) and sorting (sorted by frequency). With the head pointer table, you can quickly access all the elements of a given type in the FP tree.

Unlike the search tree, an element item can appear multiple times in a FP tree. The FP tree stores the frequency of occurrences of item sets, and each item set is stored in the tree as a path. A collection of similar elements will share a part of the tree. The tree will only fork when the sets are completely different. The tree element gives the individual elements in the set and their occurrences in the sequence, and the path gives the number of occurrences of the sequence.





The createTree() function is used later when building the tree, and the input data type of the function is not a list. What is needed is a dictionary in which the item set is the key in the dictionary and the frequency is the value corresponding to each key. createInitSet() is used to implement the above type conversion process from list to dictionary.



Second, mining frequent itemsets from a FP tree

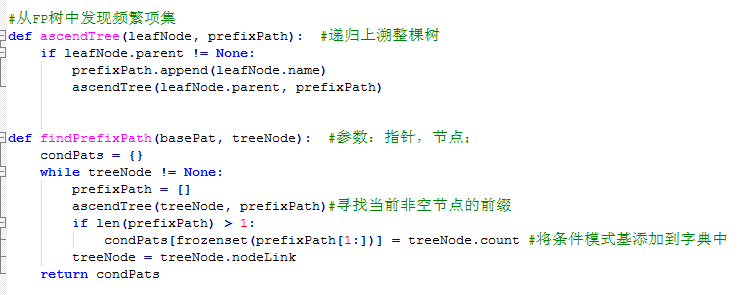
The three basic steps for extracting frequent itemsets from the FP tree are as follows:

1. Obtaining a conditional pattern base from the FP tree;

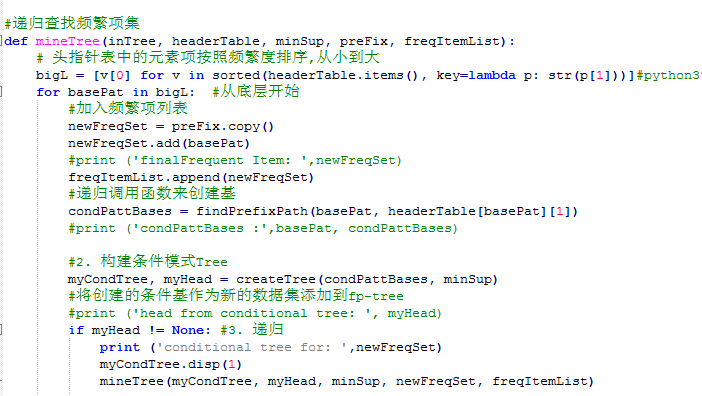
2. Construct a conditional FP tree using the conditional pattern base;

3. Iteratively repeat step 1 and step 2 until the tree contains an element item.

Next, first look for the process of the conditional pattern base. A corresponding conditional FP tree is then created for each conditional pattern base. Finally, you need to construct a little code to encapsulate the above two functions, and get frequent itemsets from the FP tree.

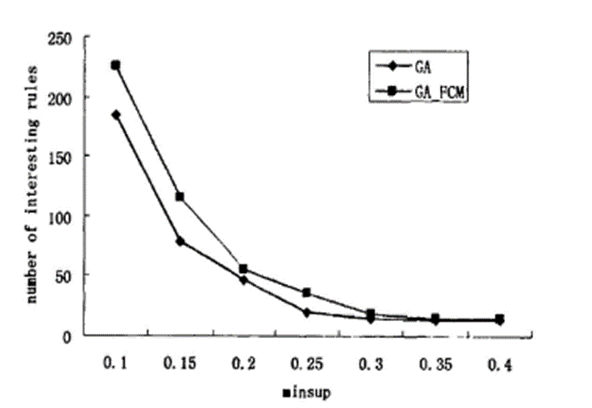
Each path is actually a prefix path. In short, a prefix path is everything between the element item being looked up and the root node of the tree.

For each frequent item, a conditional FP tree is created. Using the conditional pattern base as the input data, the condition tree is constructed with the same tree building code, then the frequent items are found recursively, the condition pattern base is found, and the condition tree is continuously constructed until There are no elements in the condition tree.

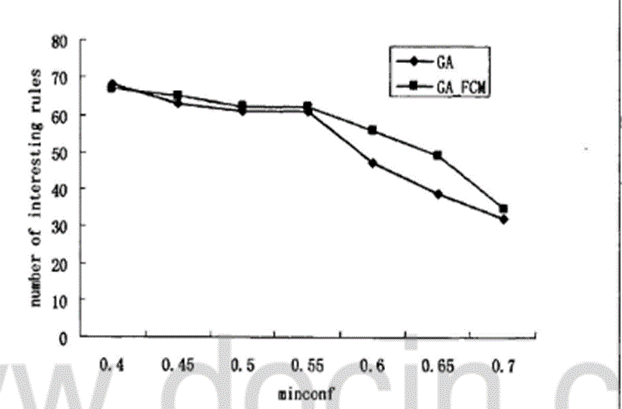
The function mineTree performs frequent item set mining on the FP tree represented by the parameter inTree. First, the single elements appearing in the headerTable are sorted from small to large according to the frequency of occurrence. Then, the conditional pattern base of each element is used as input data to establish a conditional tree for the current element. If the generated conditional tree still has elements, it is This conditional tree looks for frequent itemsets because the prefix parameter is passed down continuously during the recursion process, so all frequent itemsets derived from an element x in the original headerTable have x.

**Test description:**

1. 给定置信度为80%，关联规则数目随支持度变化的曲线图



2）给定支持度为30%，关联规则数目随置信度变化的曲线图



3）给定置信度为80%，请确定某个支持度s使得获取的关联规则数目正好大于20，请输出s和所获取的关联规则数目

S=0.08

Number of association rules =25