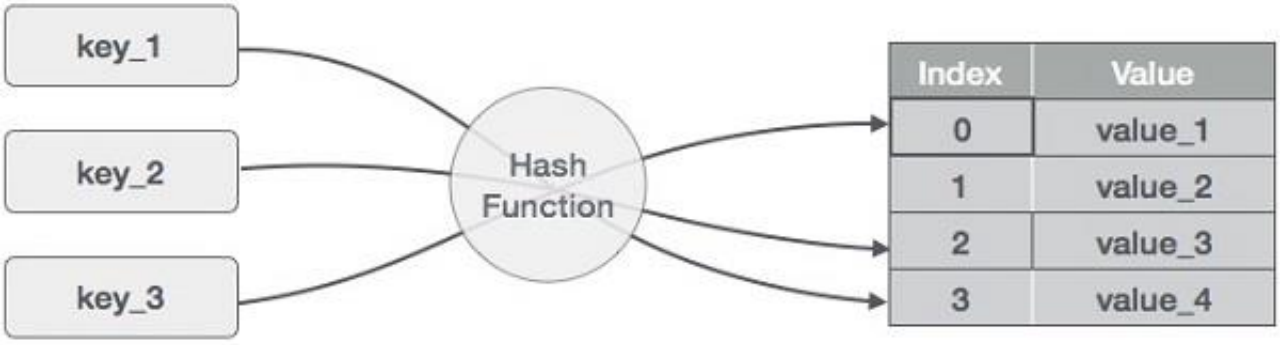
Hash Trees

Assignment 1

- 27 April 2017
  - VISHNU RAMESH(2015A7PS963H)
  - BHARGAV KANUPARTHI(2014A7PS527H)
  - KARTHIK MENON(2013B3A7487H)

HashTree

Introduction



What is a Hash Tree?

A hash function is any function that can be used to map data of arbitrary size to data of fixed size.Hash functions have a lot of efficiency when it comes to searches,data can be retrieved with much ease by using hash functions.

Hash Trees,a concept patented by Ralph Merkle in 1979,is a tree where every non-leaf node is assigned a location with respect to the value it gives out from a hashing function.

To generate frequent itemsets

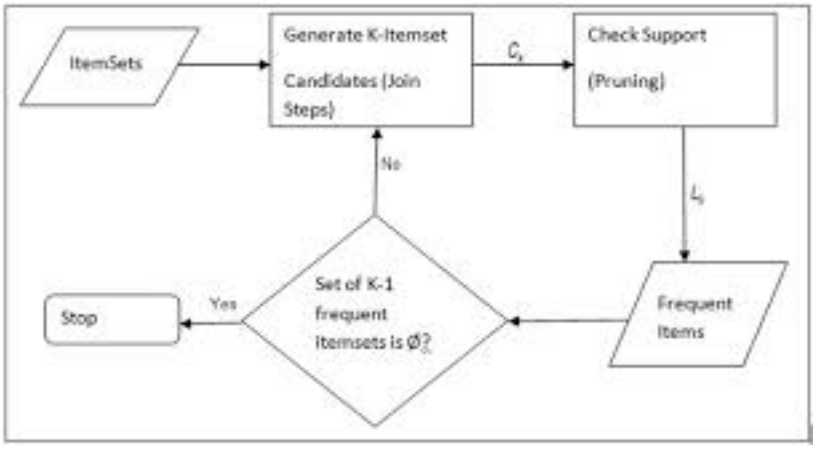
The Apriori Algorithm is an influential algorithm for mining frequent itemsets for boolean association rules. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis.

Apriori is designed to operate on databases containing transactions Each transaction is seen as a set of items (an itemset). Given a threshold ‘C’, the Apriori algorithm identifies the item sets which are subsets of at least ‘C’ transactions in the database.
Apriori uses breadth-first search and a Hash tree structure to count candidate item sets efficiently. It generates candidate item sets of length ‘k’ from item sets of length ‘k-1’. Then it prunes the candidates which have an infrequent sub pattern.

Limitations of Apriori Principle

The Apriori Principle,though useful,still suffers from few inefficiencies:

- Candidate generation generates large numbers of subsets as the algorithm attempts to load up the candidate set with as many as possible before each scan.
- Uses a uniform minimum support threshold.



Note on the raw data given

This gives the votes of 435 U.S. congressmen on16 key issues gathered in the mid-1980s, and also includes their party affiliation as a binary attribute. We are expected to apply association rule mining to seek out interesting associations within this data.
(Source: http://tunedit.org/repo/UCI/vote.arff)

Data Pre-processing:

There are 17 attributes and 435 unique transactions in the input file . Each attribute can take either of 2 values ‘y’ or ‘n’ . Hence there a possible 34 items in a particular itemsets

. For a given attribute if it is marked as ‘y’ , then attribute-y is marked as 1 else attribute-n is marked as 1 . If it ‘?’ then that attribute is ignored in the transaction .

Implementation:

In the given dataset there are 435 unique transactions . The hyper-parameters used for the following HashTree is as follows:

- Modval : 10
- BucketSize: 10
- SupportThreshold: 0.4
- ConfidenceThreshold: 0.9

Each node in the Hashtree is represented as a structure called **hashNode .**
The **hashNode** has 4 members:

- **int leaf :** Whether the current node is a leaf or not
- **struct hashNode *child[] :** An array of pointers for each of the node’s children
- **vector <nodeTrans> itemSet :** Which is a vector of Itemsets which are contained in that particular node . The structure nodeTrans has 2 members , one of them
  being the transaction itself and the other being the support count of the transaction itself.

We use the Apriori Algorithm in order to generate the Candidate Frequent Itemsets . In order to get the candidate frequent itemsets of size k+1 we cross frequent itemsets of size ‘k’ and frequent itemsets of size 1 . Now after the candidate frequent itemsets of size k has been generated in order to find the actual frequent itemsets we would insert each itemset into the HashTree .

Each itemset would then be inserted into the Hashtree based on the Mod-Value for that particular level . After all the itemsets have been inserted into the hashTree , in order to find the support count we would run all the transactions through the hashTree path and update the support count of each itemset whenever it is found to be a subset of the original set of transactions .

Finally after all the transactions have been run , we would pick those itemsets whose support is greater than the support threshold . This is then become the frequent itemsets for size k .

Similarly this is done recursively for all subsequent itemsets of greater size in order to find the frequent itemsets.

Once we have found the frequent itemsets , all are left to do is generate rules . Rule generation is generation is simple once we have found the frequent itemsets . For rule generation for a particular frequent itemset we find all possible rules that you can make using that frequent itemset and if it's confidence (support of whole itemsets / support of L.H.S) is greater than the Confidence Threshold we would accept the rule .

In our case with **support threshold as 0.4 and confidence threshold as 0.8** , the hashTree implementation has a generated a total of **383 rules** .