Apriori Implementation

In this project we decided to use arrays for our implementation of the Apriori algorithm. To do this, we wrote two classes, Itemset, and SetHolder. Itemsets contain an array of integers while the SetHolder contains an array of Itemsets. Both classes have functions that we built into them that allow us to change the size of the array, check to see if the array is empty, and remove and add elements. This allows us to change the size of the array when we need to add more values as well as letting us add or remove items when we need to.

When running through the loop, we have 2 SetHolders, one that holds all frequent itemsets that have been discovered so far, and one that holds all of the candidate itemsets that we generate. Once all of the candidates have been generated for k, where k is the size of each Itemset, we check each Itemset’s support to see if it satisfies our minimum support threshold. If it does, then the Itemset is added to the frequent items, otherwise, it is pruned. When k becomes greater than 3, we begin applying the Apriori property.

To check the Apriori property we create a mask that goes over the array that we are checking, that contains k elements which are either a 0 or a 1. This allows us to generate all subsets and then to check and see if the all of the subsets are frequent. If all subsets are frequent, then the Itemset passes the Apriori check, otherwise it is pruned. This continues going until no frequent itemsets are generated, in which case the loop terminates.

We chose to do this data structure because it was easier to implement and understand when compared to other structures like trees and lists. It was much easier to visualize the algorithm by using arrays. It also gives us direct access to any of the items in the array, which means we don’t have to traverse all of the data when we want to check things in the middle. There are many disadvantages to our implementation over others, however. Our search times are only going to be O(n), which means that almost any other structure should be able to do it faster than ours. We also had to write our own classes so that the data could be held and accessed, which added a lot of extra work for us to do since we couldn’t use code that we had created at earlier points in the semester. Although this code may not be the fastest, and it is certainly not the prettiest, it does get the job done.