Project Report on FP Growth Algorithm

Subject Code- ESE589 (Learning Systems)

By:Aditya Gogia(SBU ID:114964012)
Vaibhav Tyagi (SBU ID: 114912842)

Abstract

In this report we have written about our FP-Growth algorithm implementation in java. The FP-growth algorithm is one the fastest algorithms to find the frequent itemset mining. We have described code implementation in java. Initially we did preprocessing where we stored the number of frequencies of individual items in a hashmap and then removed those items from our hashmap which are occurring less than the minimum threshold frequencies. After removing the threshold frequencies we have finally made a sorted item sets list. This list is sorted lexicographically which also makes sure that we are respecting the order of the items in transactions based upon their frequencies. Using this list we created our FP Tree using recursion. With the help of the FP tree finally the FP growth function is generating the frequent itemsets of the input data.

Introduction

Fp-Growth algorithm follows divide and conquer strategy for finding out frequent item sets. Basically, FP-Growth is mainly used for mining of the frequent item sets. Following are the steps in the FP-Growth Algorithm:

Step1: A FP tree is built using the database showing the frequent item sets. The FP-Tree is basically made using 2 passes over the dataset.

Step2: In the algorithm the FP Tree is divided into a set of conditional data and mining of each database is done and thus extraction of frequent item sets from FP-Tree is done directly. The FP tree consists of one root which is labeled as null and a set of items prefix sub-trees which are children of the root along with a frequent item header table. Each node in the item prefix sub tree consists mainly of 3 fields which are the name of the item, count of the item and the node link where the item name registers to which item the node is representing. While the count registers the number of transactions which are represented by the portion of path reaching the node. The node link is linking to the next node of the FP-Tree. Each item present in the header table consists of 2 fields one is the name of the item and the other is head of the node link which is pointing to the 1st node in the FP-Tree which is having the name of the item.

Implementation of the FP Growth Algorithm

We have done the implementation of the FP-growth Algorithm in the following steps using java as the programming language:

- Step 1: Modifying of the dataset
- Step 2: Preprocessing of the Input Data.
- Step 3: FP Tree Construction.
- Step 4: Finally using the FP tree, the FP Growth function is written which is finally generating the frequent item sets.

Source Code

Google Drive Link:
https://drive.google.com/file/d/1sjdm6glrBiaORW8_mINNzzvbSpuoLbj6/view?usp=sharing

How to run the code:

Insert the dataset in dataset.dat file.
Insert the attribute name in name.dat file.
Clear temp.dat and final.dat file.
Run the main method to see the frequent item by entering the min_support as the input.

```
package com.ese589;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;


//Main Method
public class FPGrowthMain {

    static String data_set = "dataset.dat";
    static String name_data_set = "name.dat";

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        long currentTimeMillis_start = System.currentTimeMillis();
        Scanner min_support_scanner = new Scanner(System.in);
        System.out.println("Please enter the Min_Support");
        int min_support = min_support_scanner.nextInt();

        File da = new File(data_set);
        Scanner input_dataSet = null;
        try {
            input_dataSet = new Scanner(da);
        } catch (FileNotFoundException e1) {
            e1.printStackTrace();
        }
        FileWriter fw = null;
        BufferedWriter bw = null;
        PrintWriter out = null;
        while(input_dataSet.hasNextLine()) {
                try {
                    String temp = input_dataSet.nextLine();
                    if(temp.trim().equals("")) {
                        continue;
                    }
                  fw = new FileWriter("temp.dat", true);
                  bw = new BufferedWriter(fw);
                  out = new PrintWriter(bw);
                  bw.flush();

                  out.println(temp.replaceAll(",", " "));
                  out.close();
                } catch (IOException ioe) {
                    ioe.printStackTrace();
                }

        }


            File temp = new File("temp.dat");
            Scanner input_data_removed_spaces_scanner = null;
            try {
                input_data_removed_spaces_scanner = new
Scanner(temp);
            } catch (FileNotFoundException e1) {
                e1.printStackTrace();
            }
            File nameset = new File(name_data_set);
            while (input_data_removed_spaces_scanner.hasNextLine()) {
                Scanner nameset_scanner = null;
                try {
                    nameset_scanner = new Scanner(nameset);
                } catch (FileNotFoundException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
                while (nameset_scanner.hasNext() == true) {
                     fw = null;
                     bw = null;
                     out = null;
                    try {
                        fw = new FileWriter("final.dat", true);
                        bw = new BufferedWriter(fw);
                        out = new PrintWriter(bw);
                        StringBuilder a = new StringBuilder();
                        String[] a1 =
input_data_removed_spaces_scanner.nextLine().split(" ");
                        String fi = "";
                        for (String a2 : a1) {
                            if(a2.trim().equals("")) {
                                continue;
                            }
                            fi = nameset_scanner.next() + "_" +
a2 + " ";

                            a.append(fi);
                        }
                        out.println(a);
                        out.close();
                    } catch (IOException e) {

                    }

                }
            }
        long currentTimeMillis_mid = System.currentTimeMillis();
        FPGrowthServiceImpl fpGrowthServiceImpl = new
FPGrowthServiceImpl(new File("final.dat"), min_support);
        fpGrowthServiceImpl.print();
        long currentTimeMillis_end = System.currentTimeMillis();
        System.out.println("Time taken to run the algorithm in milliseconds
is: "+ (currentTimeMillis_end - currentTimeMillis_mid));




//      FPGrowthServiceImpl fpGrowthServiceImpl = new
FPGrowthServiceImpl(new File("dataset.dat"), min_support);
//      System.out.println("Frequent Item sets are: ");
//      fpGrowthServiceImpl.print();
    }

}

package com.ese589;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;

public class FPGrowthServiceImpl {

    // Initializing FPtree root to null
    FPTreeNode fpTreeNode = new FPTreeNode("null");
    // Frequent Patterns item set, Result
    HashMap<String, Integer> frequentPatterns = new HashMap<String,
Integer>();
    // To initialize FPTreeNode for each Item set
    ArrayList<FPTreeNode> eachItemSetTreeNode = new
ArrayList<FPTreeNode>();

    public FPGrowthServiceImpl(File file, int min_support) {
        fptree(min_support, file);
        fpgrowthFacade(fpTreeNode, min_support, eachItemSetTreeNode);
    }

    private void fptree(int min_support, File file) {
        // TODO Auto-generated method stub
        HashMap<String, Integer> itemsSetFrequency = new
HashMap<String, Integer>();
        Scanner input;
        try {
            input = new Scanner(file);

            List<String> sortedItemsSetFrequency = new
LinkedList<String>();
            ArrayList<String> removedItemSet = new ArrayList<String>();
            preProcessing(min_support, file, itemsSetFrequency, input,
sortedItemsSetFrequency, removedItemSet);
            building_fpTree(file, itemsSetFrequency, input,
sortedItemsSetFrequency, removedItemSet);
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

    private void preProcessing(int min_support, File filename, HashMap<String,
Integer> itemsMaptoFrequencies,
            Scanner dataset_scanner_input, List<String>
sortedItemsbyFrequencies, ArrayList<String> itemstoRemove)
            throws FileNotFoundException {
```

```
// TODO Auto-generated method stub
while (dataset_scanner_input.hasNext()) {
	String temp = dataset_scanner_input.next().trim();
	if (temp.contains("?")) {
		continue;
	}
	if (itemsMaptoFrequencies.containsKey(temp)) {
		int count = itemsMaptoFrequencies.get(temp);
		itemsMaptoFrequencies.put(temp, count + 1);
	} else {
		itemsMaptoFrequencies.put(temp, 1);
	}
}
dataset_scanner_input.close();

sortedItemsbyFrequencies.add("null");
itemsMaptoFrequencies.put("null", 0);
for (String item : itemsMaptoFrequencies.keySet()) {
	int count = itemsMaptoFrequencies.get(item);
	// System.out.println( count );
	int i = 0;
	for (String listItem : sortedItemsbyFrequencies) {
		if (itemsMaptoFrequencies.get(listItem) < count) {
			sortedItemsbyFrequencies.add(i, item);
			break;
		}
		i++;
	}
}

for (String listItem : sortedItemsbyFrequencies) {
	if (itemsMaptoFrequencies.get(listItem) < min_support) {
		itemstoRemove.add(listItem);
	}
}
```

```
        for (String itemtoRemove : itemstoRemove) {
            sortedItemsbyFrequencies.remove(itemtoRemove);
        }
//
//      System.out.println(sortedItemsbyFrequencies);
//      System.out.println(itemstoRemove);

    }

    private void building_fpTree(File filename, HashMap<String, Integer>
itemsMaptoFrequencies,
            Scanner dataset_scanner_input, List<String>
sortedItemsbyFrequencies, ArrayList<String> itemstoRemove)
            throws FileNotFoundException {
        // TODO Auto-generated method stub
        for (String itemsforTable : sortedItemsbyFrequencies) {
            eachItemSetTreeNode.add(new FPTreeNode(itemsforTable));
        }
        dataset_scanner_input = new Scanner(filename);
        // the null node!

        fpTreeNode.item = null;
        fpTreeNode.root = true;
        // ordering frequent items transaction
        while (dataset_scanner_input.hasNextLine()) {
            String line = dataset_scanner_input.nextLine().trim();
            String[] itemSets = line.split(" ");
            ArrayList<String> transactionSortedItemsbyFrequency = new
ArrayList<String>();
            for (String item_value : itemSets) {
                String item = item_value.trim();
                if (item.contains("?")) {
                    continue;
                }
                if (itemstoRemove.contains(item)) {
                    continue;
                }
                int index = 0;
                for (String vectorString :
transactionSortedItemsbyFrequency) {

                    if (itemsMaptoFrequencies.get(vectorString) <
itemsMaptoFrequencies.get(item)
                            ||
((itemsMaptoFrequencies.get(vectorString) == itemsMaptoFrequencies.get(item))
                                    &&
(vectorString.compareToIgnoreCase(item) > 0 ? true : false))) {

transactionSortedItemsbyFrequency.add(index, item);
                        break;
                    }
                    index++;
                }
                if (!transactionSortedItemsbyFrequency.contains(item)) {
                    transactionSortedItemsbyFrequency.add(item);
                }
            }

            insertIteminFPTree(transactionSortedItemsbyFrequency,
fpTreeNode, eachItemSetTreeNode);
            transactionSortedItemsbyFrequency.clear();
        }

        for (FPTreeNode item : eachItemSetTreeNode) {
            int count = 0;
            FPTreeNode itemtemp = item;
            while (itemtemp.next != null) {
                itemtemp = itemtemp.next;
                count += itemtemp.frequencyOfItemInTree;
            }
            item.frequencyOfItemInTree = count;
        }
        Comparator c = new FPTreeComparitor();
        Collections.sort(eachItemSetTreeNode, c);
        Collections.reverse(eachItemSetTreeNode);
        dataset_scanner_input.close();
    }

    public class FPTreeComparitor implements Comparator<FPTreeNode> {

        @Override
        public int compare(FPTreeNode fptreeNode1, FPTreeNode
fptreeNode2) {
            if (fptreeNode1.frequencyOfItemInTree >
fptreeNode2.frequencyOfItemInTree) {
                return -1;
            } else if (fptreeNode1.frequencyOfItemInTree <
fptreeNode2.frequencyOfItemInTree)
                return 1;
            else
                return 0;
        }

    }

    private void insertIteminFPTree(ArrayList<String>
transactionSortedbyFrequencies, FPTreeNode fptreeNode,
            ArrayList<FPTreeNode> eachItemSetTreeNode) {
        // TODO Auto-generated method stub
        if (transactionSortedbyFrequencies.isEmpty()) {
            return;
        }
        String itemtoAddtotree = transactionSortedbyFrequencies.get(0);
        FPTreeNode fpTreenewNode = null;
        boolean newRootNode = false;
        for (FPTreeNode child : fptreeNode.children) {
            if (child.item.equals(itemtoAddtotree)) {
                fpTreenewNode = child;
                child.frequencyOfItemInTree++;
                newRootNode = true;
                break;
            }
        }
        if (!newRootNode) {
            fpTreenewNode = new FPTreeNode(itemtoAddtotree);
            fpTreenewNode.frequencyOfItemInTree = 1;
            fpTreenewNode.parent = fptreeNode;
            fptreeNode.children.add(fpTreenewNode);
            for (FPTreeNode fptreeNodeheaderPointer :
eachItemSetTreeNode) {
                if
(fptreeNodeheaderPointer.item.equals(itemtoAddtotree)) {
                    while (fptreeNodeheaderPointer.next != null) {
                        fptreeNodeheaderPointer =
fptreeNodeheaderPointer.next;
                    }
                    fptreeNodeheaderPointer.next = fpTreenewNode;
                }
            }
        }
        transactionSortedbyFrequencies.remove(0);
        insertIteminFPTree(transactionSortedbyFrequencies, fpTreenewNode,
eachItemSetTreeNode);
    }

    private void fpgrowthFacade(FPTreeNode fpTreeNode, int min_support,
            ArrayList<FPTreeNode> reverseSortedItemSetTreeNode) {
        // TODO Auto-generated method stub
        fpgrowthFacadeImpl(fpTreeNode, null, min_support,
reverseSortedItemSetTreeNode, frequentPatterns);
```

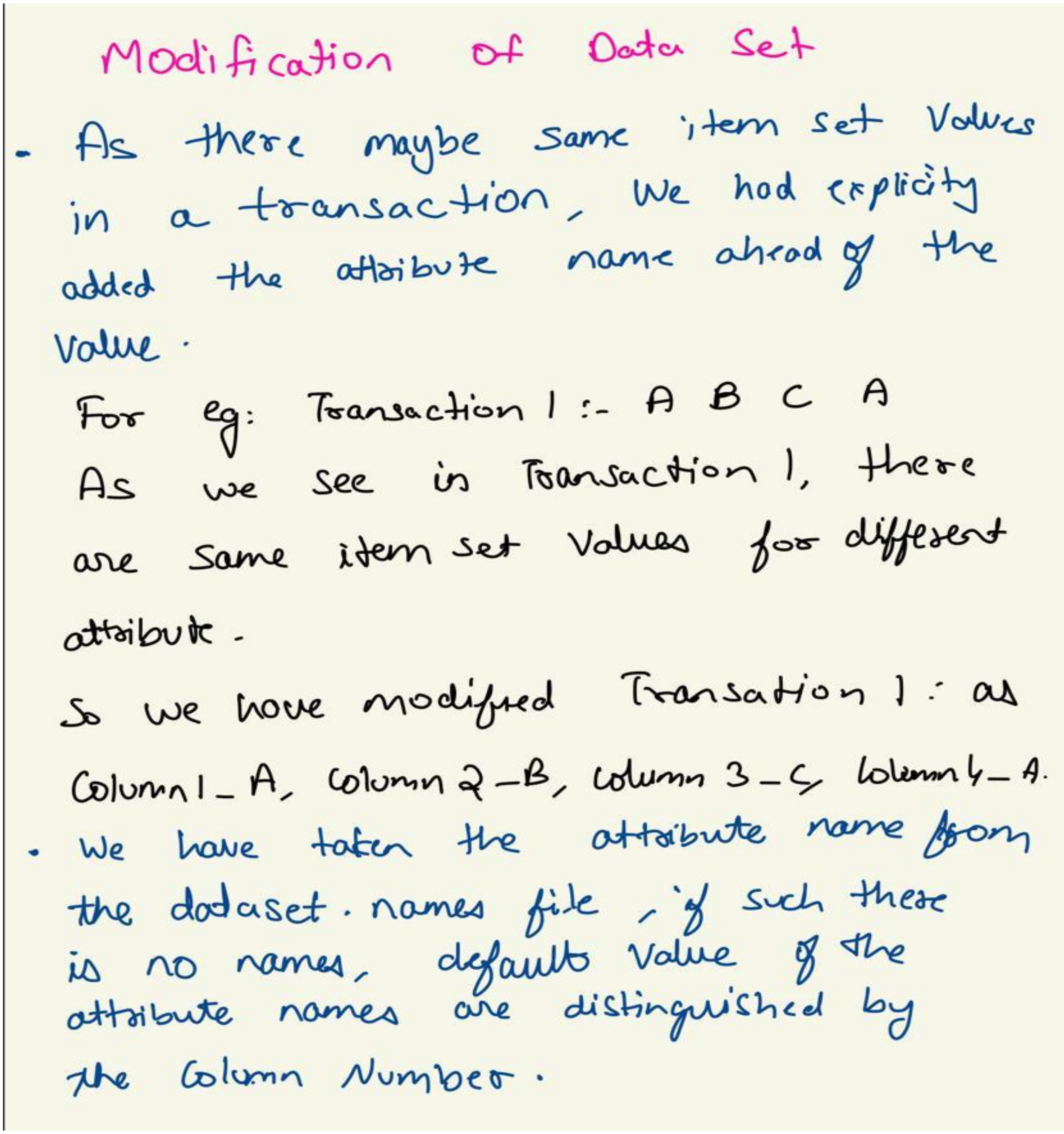
```
    }

    private void fpgrowthFacadeImpl(FPTreeNode fpTreeNode, String
present_path_base, int min_support,
            ArrayList<FPTreeNode> reverseSortedItemSetTreeNode,
HashMap<String, Integer> frequentPatterns) {
        // TODO Auto-generated method stub
        for (FPTreeNode iteminTree : reverseSortedItemSetTreeNode) {
            String path = (present_path_base != null ? present_path_base :
"") + (present_path_base != null ? " " : "")
                    + iteminTree.item;

            int frequencyofpresentPath = 0;
            HashMap<String, Integer> conditional_Pattern_Path = new
HashMap<String, Integer>();
            while (iteminTree.next != null) {
                iteminTree = iteminTree.next;
                frequencyofpresentPath +=
iteminTree.frequencyOfItemInTree;
                String conditional_Pattern = null;
                FPTreeNode conditional_Item = iteminTree.parent;

                while (!conditional_Item.isRoot()) {
                    conditional_Pattern = conditional_Item.item + " "
                            + (conditional_Pattern != null ?
conditional_Pattern : "");
                    conditional_Item = conditional_Item.parent;
                }
                if (conditional_Pattern != null) {
                    conditional_Pattern_Path.put(conditional_Pattern,
iteminTree.frequencyOfItemInTree);
                }
            }
            frequentPatterns.put(path, frequencyofpresentPath);
            HashMap<String, Integer> conditionalItemsMaptoFrequencies
= new HashMap<String, Integer>();
            for (String conditional_Pattern :
conditional_Pattern_Path.keySet()) {

                String[] conditional_Pattern_items =
conditional_Pattern.split(" ");

                for (String conditional_Pattern_item :
conditional_Pattern_items) {
                    if (conditional_Pattern_item.trim().equals("")) {
                        continue;
                    }
                    String item = conditional_Pattern_item.trim();
                    if
(conditionalItemsMaptoFrequencies.containsKey(item)) {
                        int count =
conditionalItemsMaptoFrequencies.get(item);
                        count +=
conditional_Pattern_Path.get(conditional_Pattern);

conditionalItemsMaptoFrequencies.put(item, count);
                    } else {

conditionalItemsMaptoFrequencies.put(item,
conditional_Pattern_Path.get(conditional_Pattern));
                    }
                }
            }

            ArrayList<FPTreeNode> conditional_headerTable = new
ArrayList<FPTreeNode>();
            for (String itemsforTable :
conditionalItemsMaptoFrequencies.keySet()) {
```

```
                int count =
conditionalItemsMaptoFrequencies.get(itemsforTable);
                if (count < min_support) {
                    continue;
                }
                FPTreeNode f = new FPTreeNode(itemsforTable);
                f.frequencyOfItemInTree = count;
                conditional_headerTable.add(f);
            }
            FPTreeNode conditional_fptree =
conditional_fptree_constructor(conditional_Pattern_Path,
                    conditionalItemsMaptoFrequencies, min_support,
conditional_headerTable);

            Collections.sort(conditional_headerTable, new
FPTreeComparitor());
            Collections.reverse(conditional_headerTable);
            if (!conditional_fptree.children.isEmpty()) {
                fpgrowthFacadeImpl(conditional_fptree, path,
min_support, conditional_headerTable, frequentPatterns);
            }
        }
    }

    private void insertIteminConditionalFPTree(ArrayList<String> pattern_list,
int count_of_pattern,
            FPTreeNode conditional_fptree, ArrayList<FPTreeNode>
conditional_headerTable) {
        // TODO Auto-generated method stub
        if (pattern_list.isEmpty()) {
            return;
        }
        String itemtoAddtotree = pattern_list.get(0);
        FPTreeNode fpTreenewNode = null;
        boolean ifisdone = false;
        for (FPTreeNode child : conditional_fptree.children) {
            if (child.item.equals(itemtoAddtotree)) {
                fpTreenewNode = child;
                child.frequencyOfItemInTree += count_of_pattern;
                ifisdone = true;
                break;
            }
        }
        if (!ifisdone) {
            for (FPTreeNode fpTreeNode_headerPointer :
conditional_headerTable) {

                if
(fpTreeNode_headerPointer.item.equals(itemtoAddtotree)) {
                    fpTreenewNode = new
FPTreeNode(itemtoAddtotree);
                    fpTreenewNode.frequencyOfItemInTree =
count_of_pattern;
                    fpTreenewNode.parent = conditional_fptree;
                    conditional_fptree.children.add(fpTreenewNode);
                    while (fpTreeNode_headerPointer.next != null) {
                        fpTreeNode_headerPointer =
fpTreeNode_headerPointer.next;
                    }
                    fpTreeNode_headerPointer.next = fpTreenewNode;
                }
            }
        }
        pattern_list.remove(0);
        insertIteminConditionalFPTree(pattern_list, count_of_pattern,
fpTreenewNode, conditional_headerTable);
    }

    private FPTreeNode conditional_fptree_constructor(HashMap<String,
Integer> conditionalPatternPath,
            HashMap<String, Integer> conditionalItemsMapedtoFrequency,
int min_support,
            ArrayList<FPTreeNode> conditional_headerTable) {
        // TODO Auto-generated method stub
        FPTreeNode conditional_fptreeNode = new FPTreeNode("null");
        conditional_fptreeNode.item = null;
        conditional_fptreeNode.root = true;
        for (String pattern : conditionalPatternPath.keySet()) {
            ArrayList<String> pattern_list = new ArrayList<String>();

            String[] pattern_items = pattern.split(" ");

            for (String pattern_item : pattern_items) {
                String item = pattern_item.trim();
                if (item.equals("")) {
                    continue;
                }
                if (conditionalItemsMapedtoFrequency.get(item) >=
min_support) {
                    pattern_list.add(item);
                }
            }
            insertIteminConditionalFPTree(pattern_list,
conditionalPatternPath.get(pattern), conditional_fptreeNode,
                    conditional_headerTable);
        }
        return conditional_fptreeNode;
    }

    public void print() {
        System.out.println("Total Number of Frequent Item sets are: " +
frequentPatterns.size());
        for (String frequent_PatternSets : frequentPatterns.keySet()) {
//          if (frequent_PatternSets.split(" ").length != 1) {
```

```
                              System.out.println("\t" + frequent_PatternSets + " " +
frequentPatterns.get(frequent_PatternSets));
//                            }
                    }
          }

}


package com.ese589;

import java.util.ArrayList;

public class FPTreeNode {

    public FPTreeNode(String item) {
        this.item = item;
        next = null;
        children = new ArrayList<FPTreeNode>();
        root = false;
    }

    boolean root;
    String item;
    ArrayList<FPTreeNode> children;
    FPTreeNode parent;
    FPTreeNode next;
    int frequencyOfItemInTree;

    boolean isRoot(){
        return root;
    }

}
```

Step 1: Modifying of the Dataset

In this step we have modified the dataset which is given to us. As some of the datasets are not containing all the data.



Modification of Data Set

- As there maybe same item set Values in a transaction, We had explicity added the attribute name ahead of the Value.

For eg: Transaction 1 :- A B C A

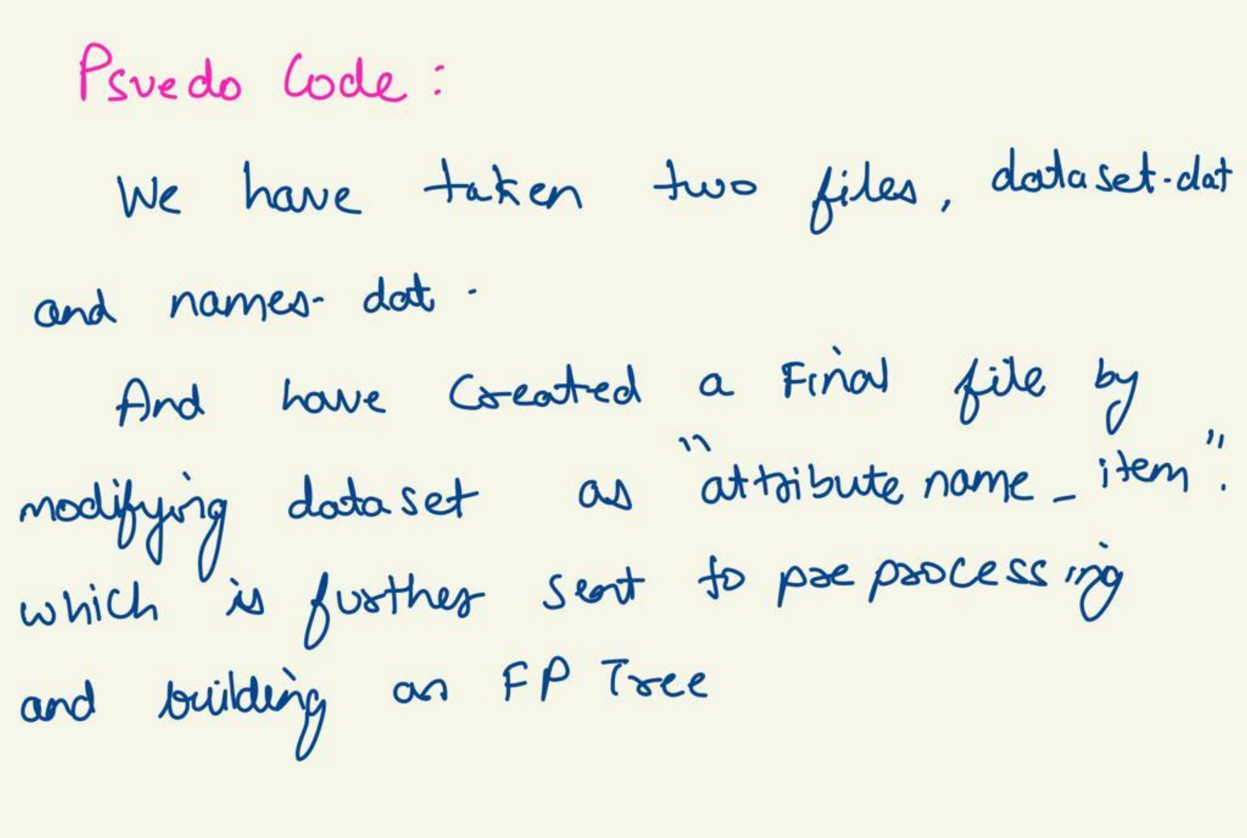
As we see in Transaction 1, there are same item set Values for different attribute.

So we have modified Transation 1 : as

Column1_A, column 2_B, column 3_C, column4_A.

- We have taken the attribute name from the dataset.names file, if such there is no names, default Value of the attribute names are distinguished by the Column Number.

The pseudo code of this step:



Psvedo Code :

We have taken two files, dataset.dat and names.dat.

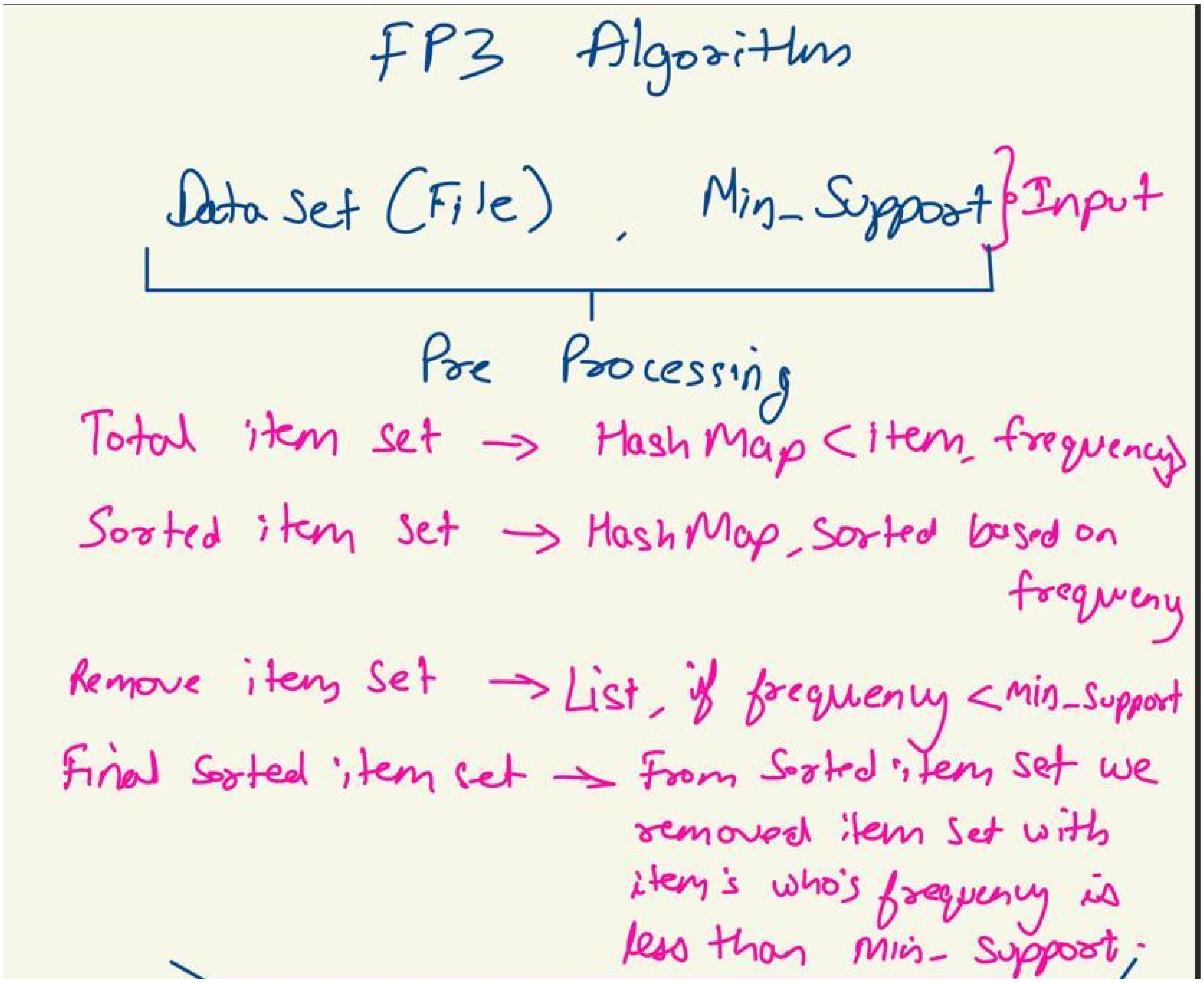
And have Created a Final file by modifying dataset as "attribute name _ item". which is further sent to preprocessing and building an FP Tree

Step 2: Preprocessing of the Input Data

Preprocessing of the input data is been done in the following steps:

- We have mapped each candidate of the input into a hashmap which contains the frequency of each candidate.
- Then we created another hashmap which contains all the candidates data which is sorted on the basis of their frequencies.
- After this step, we created a list which contains candidates to remove. In which we have added all the candidates which have frequencies less than a minimum threshold frequency.

Attached below is the photo of pseudo code of the Preprocessing of the Input Data.



FP3 Algorithm

Data Set (File) , Min_Support } Input

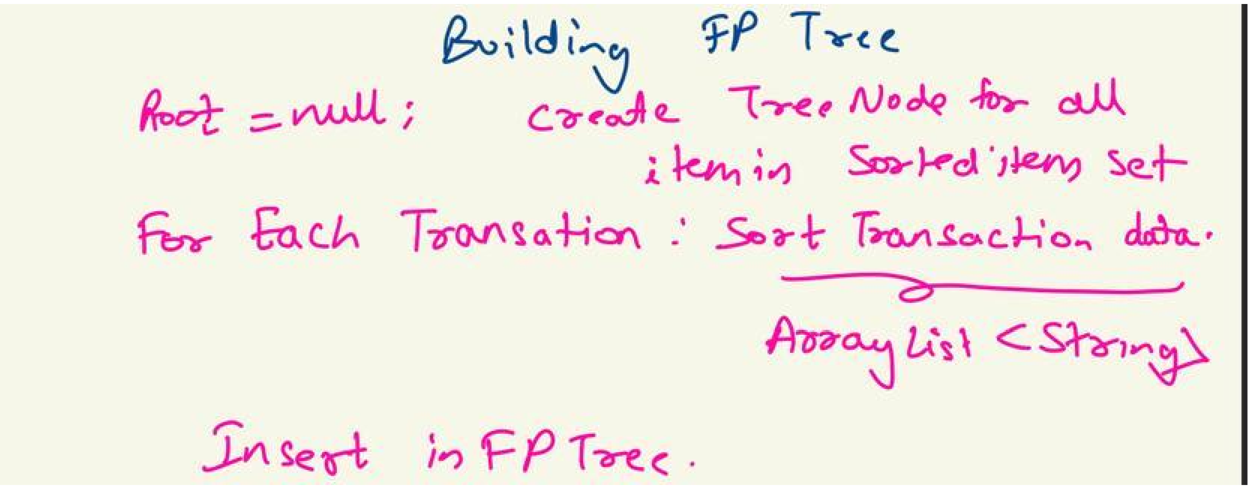
Pre Processing

```
Total item set → HashMap<item_ frequency>
Sorted item set → HashMap_Sorted based on frequency
Remove item Set → List, if frequency < min_Support
Final Sorted item set → From Sorted item set we removed item Set with item's who's frequency is less than min_Support;
```

Step 3: FP Tree construction

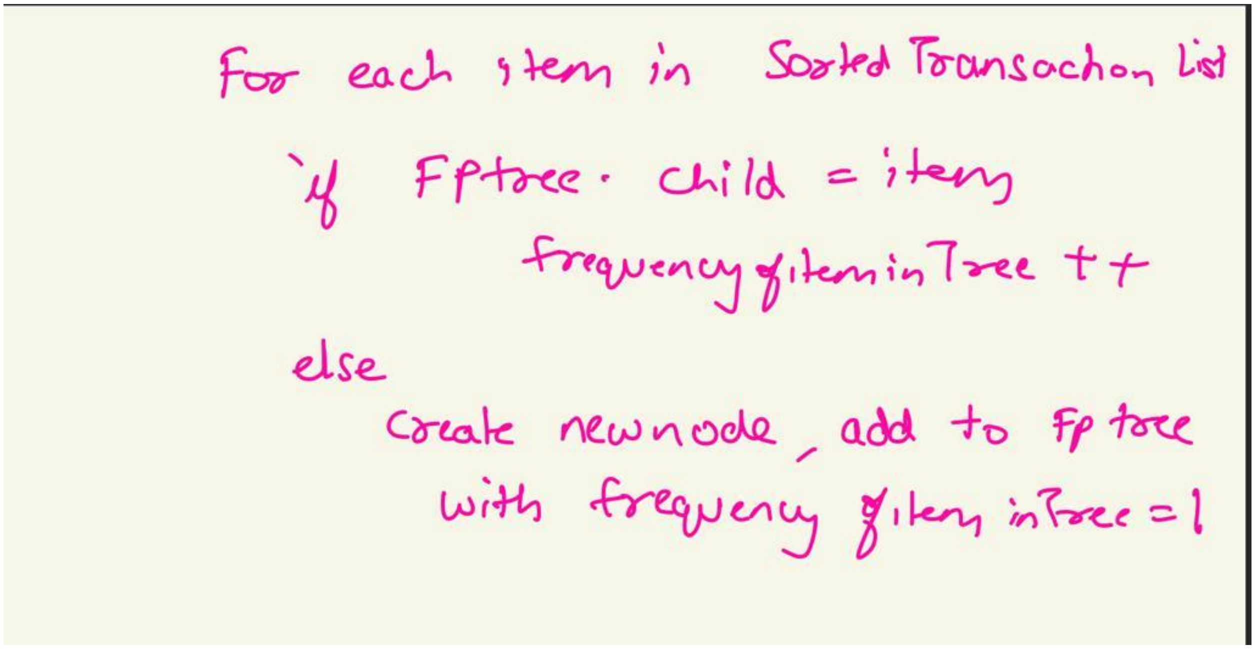
To construct the FP TREE we have initialized the tree root as null. And then started scanning each transaction using a scanner. And a while loop is run till the input is containing the next line. Then, we sorted each transaction based on their frequency and removed the infrequent single candidates. Then we took the sorted transaction list to insert into the tree. For each item in the sorted list get the first item as that is the most frequent item. We create a new node of the FP Tree and initialize it as null. If Fp tree children are equal to the item, the new node is equal to the fp tree children. And we increase the children's frequency count. Otherwise, the new node is created using the FP tree class. New node frequency is initialized as 1. Also, the new node is attached to its parent.

After this we remove the item. These whole steps are done using a recursive function. The pseudo code of the FP Tree construction screenshot is attached below.



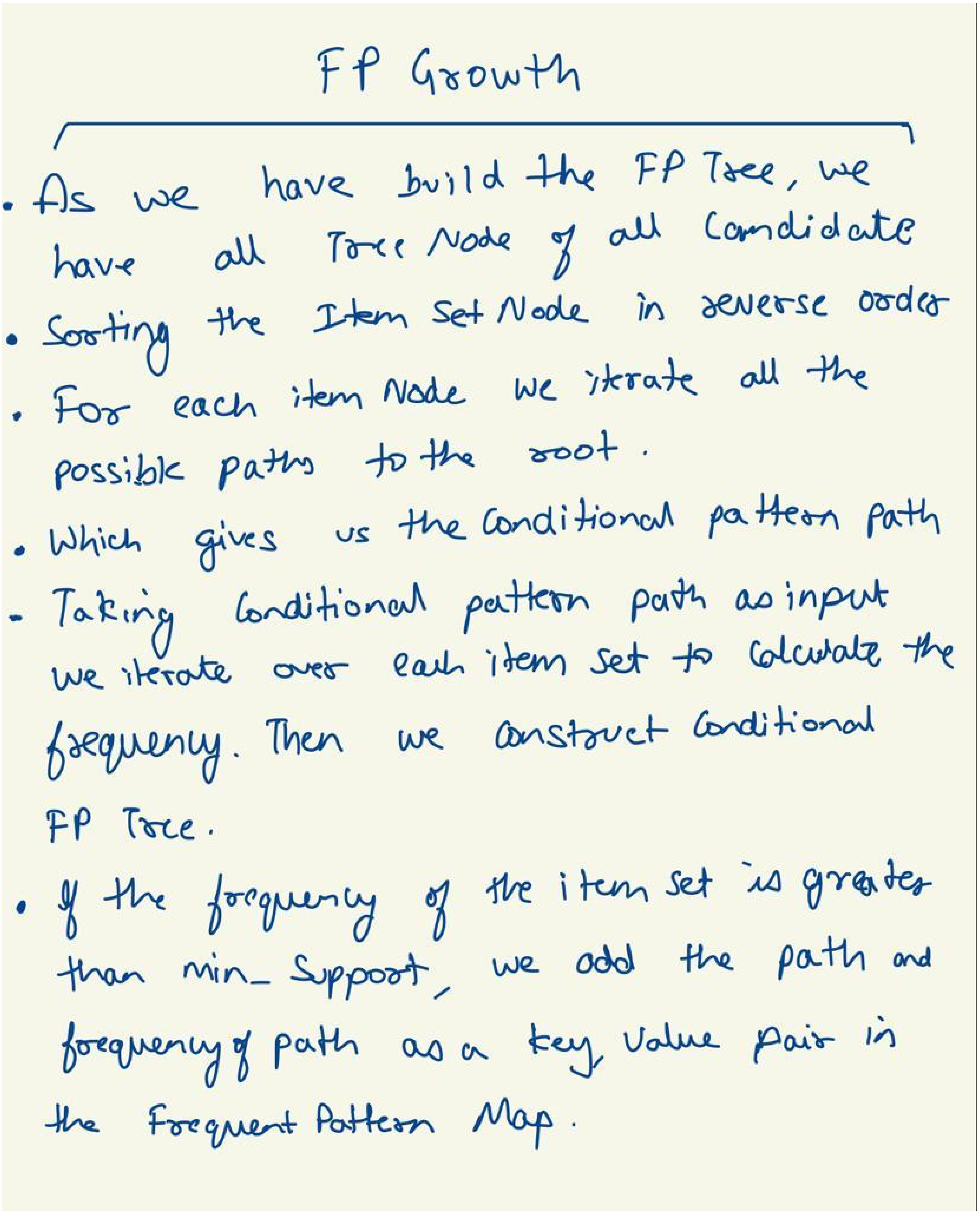
Building FP Tree

```
Root = null;      create Tree Node for all
                        item in Sorted item set
For Each Transation : Sort Transaction data.
                              → ArrayList <String>

   Insert in FP Tree.
```



```
For each item in Sorted Transaction List
    if FPtree. child = item
            frequency of item in Tree ++
    else
         Create newnode, add to Fp tree
           with frequency of item inTree = 1
```

Step 4: FP Growth



FP Growth

- As we have build the FP Tree, we have all Tree Node of all Candidate
- Sorting the Item Set Node in reverse order
- For each item Node we iterate all the possible paths to the root.
- Which gives us the Conditional pattern path
- Taking Conditional pattern path as input we iterate over each item set to calculate the frequeny. Then we construct Conditional FP Tree.
- If the frequency of the item set is greater than min_Support, we add the path and frequency of path as a key, Value pair in the Frequent Pattern Map.

Validation of the Code

In order to check the validation of our code we ran our code on 21 benchmarks and compared our results with the other existing FP Growth algorithms code. Also we did experimental analysis of our algorithm using these benchmarks to understand our code and to do validation of our code which is discussed in the next section of the report.

In the table below we have written about the overall findings on the benchmarks datasets.

S.No.	Dataset URL	Dataset Name	Findings of our Algorithm
1.	https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/	Adalone	Execution time: 1.1 s Minimum Support Provided: 11 Number of Frequent Itemsets Produced: 1544
2.	https://archive.ics.uci.edu/ml/datasets/Annealing	Anneal	Execution time: 0.7s Minimum Support Provided: 500 Number of Frequent Itemsets Produced: 45

3.	https://archive.ics.uci.edu/ml/datasets/Automobile	Automobile	Execution time: 0.8 s Minimum Support Provided: 50 Number of Frequent Itemsets Produced: 345
4.	https://archive.ics.uci.edu/ml/datasets/Adult	Adult	Execution time: 3s Minimum Support Provided: 5000 Number of Frequent Itemsets Produced: 1184
5.	https://archive.ics.uci.edu/ml/datasets/Balance+Scale	Balance Scale	Execution time: 0.44s Minimum Support Provided: 100 Number of Frequent Itemsets Produced: 22
6.	https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer/	Breast Cancer	Execution time: 0.43s Minimum Support Provided: 50 Number of Frequent Itemsets

			Produced: 237
7.	https://archive.ics.uci.edu/ml/datasets/Balloons	Balloons	Execution time: 0.12 s Minimum Support Provided: 3 Number of Frequent Itemsets Produced: 88
8.	https://archive.ics.uci.edu/ml/datasets/Pittsburgh+Bridges	Bridges Pittsburgh	Execution time: 0.39s Minimum Support Provided: 50 Number of Frequent Itemsets Produced: 17
9.	https://archive.ics.uci.edu/ml/datasets/Car+Evaluation	Car Evaluation	Execution Time: 1.38s Minimum Support Provided: 20 Number of Frequent Itemsets Produced: 24
10.	https://archive.ics.uci.edu/ml/datasets/Computer+Hardware	Computer Hardware	Execution Time: 1.2s Minimum Support

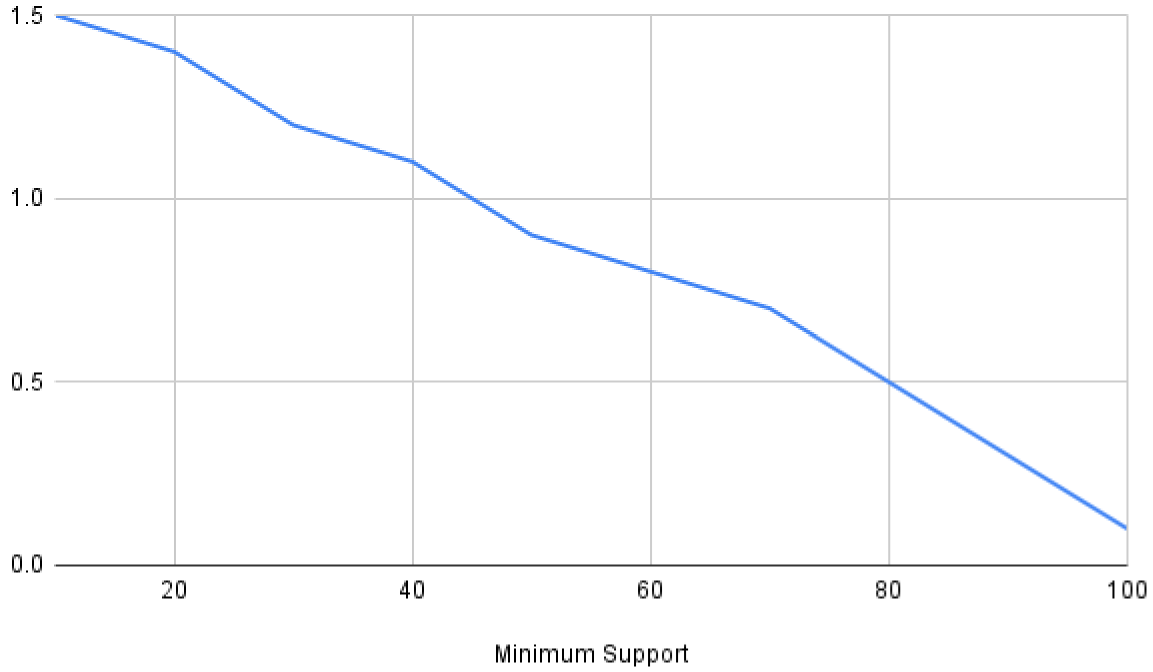
			Provided: 50 Number of Frequent Itemsets Produced: 148
11.	https://archive.ics.uci.edu/ml/datasets/Contraceptive+Method+Choice	Contraceptive Method Choice	Execution Time: 1.42 s Minimum Support Provided: 50 Number of Frequent Itemsets Produced: 1290
12.	https://archive.ics.uci.edu/ml/datasets/Cylinder+Bands	Cylinder Bands	Execution Time: 2.3s Minimum Support Provided: 1000 Number of Frequent Itemsets Produced: 4
13.	https://archive.ics.uci.edu/ml/datasets/Echocardiogram	Echocardiogram	Execution Time: 0.59 s Minimum Support Provided: 20 Number of Frequent Itemsets Produced: 69
14.	https://archive.ics.uci.edu/ml/datasets/Ecoli	Ecoli	Execution Time: 0.48 s Minimum Support

			Provided: 20 Number of Frequent Itemsets Produced: 21
15.	https://archive.ics.uci.edu/ml/datasets/Flags	Flag	Execution Time: 4.6 s Minimum Support Provided: 2000 Number of Frequent Itemsets Produced: 1561
16.	https://archive.ics.uci.edu/ml/datasets/Glass+Identification	Glass	Execution Time: 0.71 Minimum Support Provided: 15 Number of Frequent Itemsets Produced: 27
17.	https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival	Haberman Survival	Execution Time: 0.33 Minimum Support Provided: 10 Number of Frequent Itemsets Produced: 68
18.	https://archive.ics.uci.edu/ml/datasets/Hayes-Roth	Hayes Roth	Execution Time: 0.28 s Minimum Support

			Provided: 10 Number of Frequent Itemsets Produced: 125
19.	https://archive.ics.uci.edu/ml/datasets/Hepatitis	Hepatitis	Execution Time: 0.81 s Minimum Support Provided:100 Number of Frequent Itemsets Produced: 36
20.	https://archive.ics.uci.edu/ml/datasets/Iris	Iris	Execution Time: 0.26 s Minimum Support Provided: 10 Number of Frequent Itemsets Produced: 23
21.	https://archive.ics.uci.edu/ml/datasets/Lenses	Lenses	Execution Time: 0.13 s Minimum Support Provided: 10 Number of Frequent Itemsets Produced: 8

Experimental Analysis

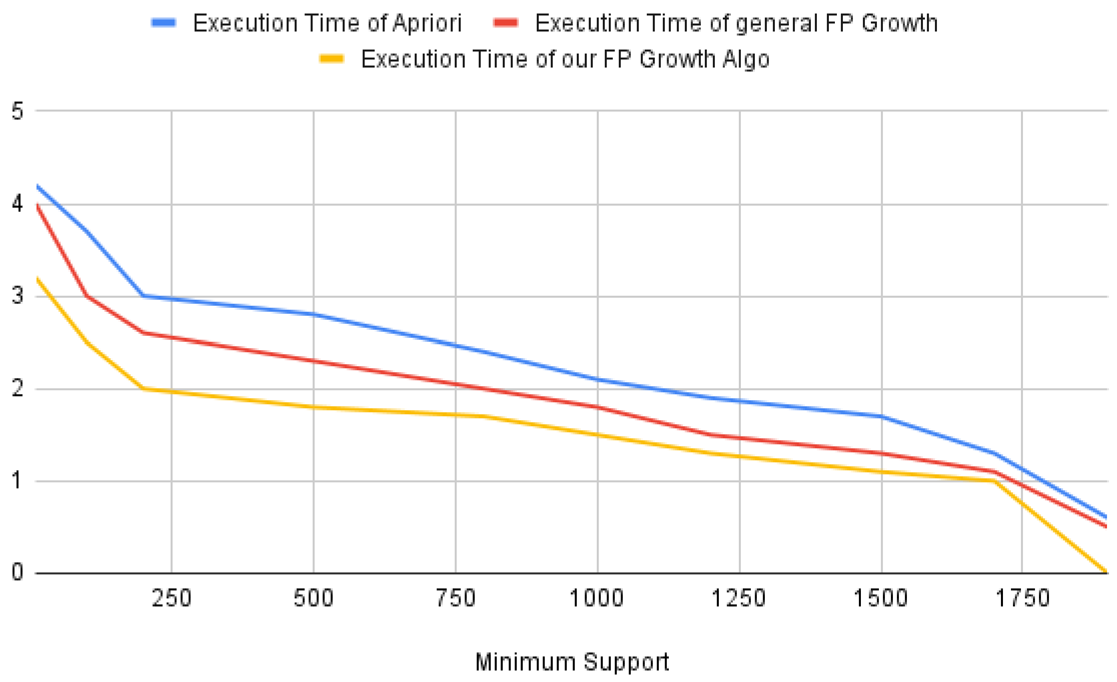
We took benchmark datasets from https://archive.ics.uci.edu/ml/datasets/Balloons and ran our algorithm. We calculated how much time our algorithm took with respect to the change of the minimum support. Attached below is the graphical representation of the execution time in seconds with respect to the minimum support.



Execution time with respect to the minimum support

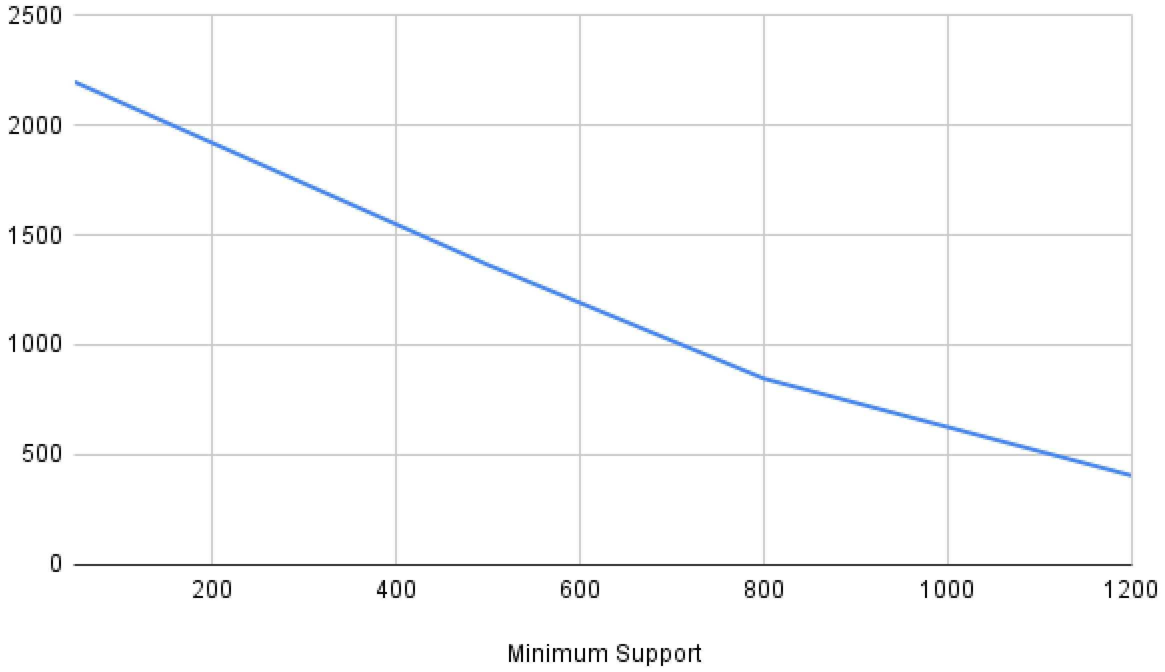
We took the following benchmark datasets:
https://archive.ics.uci.edu/ml/datasets/Adult
https://archive.ics.uci.edu/ml/datasets/Flags
Then we ran our algorithm along with existing code of apriori algorithm(https://github.com/asaini/Apriori) and FP Growth algorithm(https://github.com/chonyy/fpgrowth_py) and found that our algorithm was taking the least amount of time as compared with both the existing algorithms

code. Attached below is the graphical representation of the comparison between the algorithms based upon the execution timings.



Comparison with existing algorithms on the basis of Execution time with respect to Minimum Support

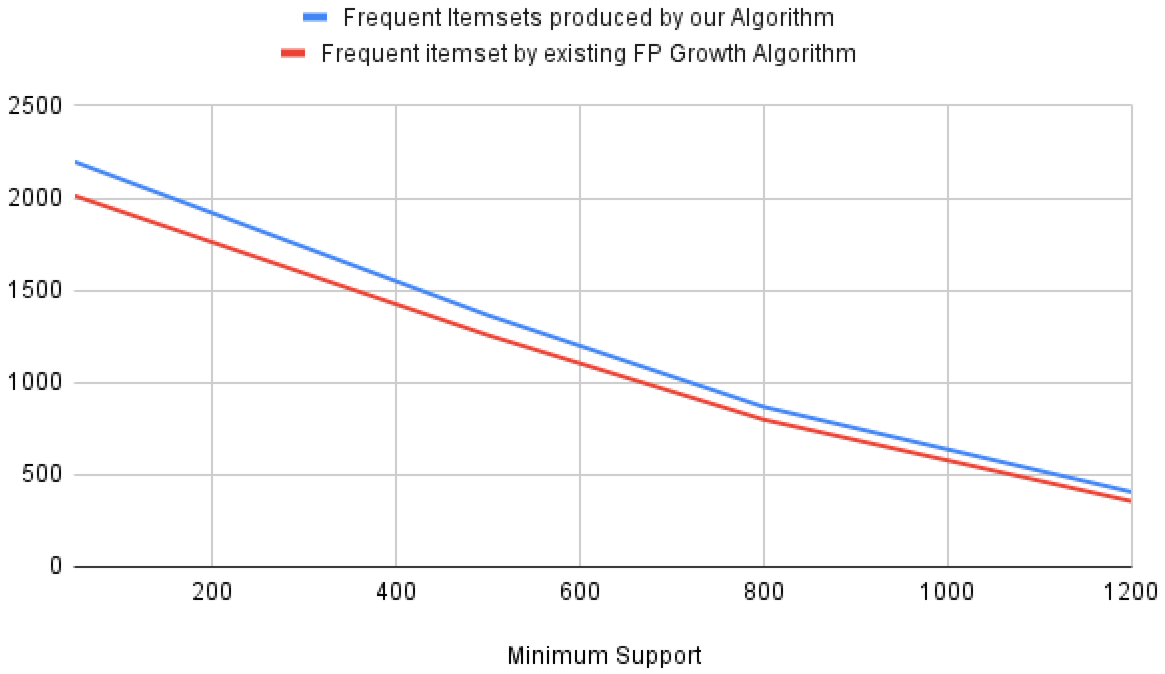
We took a bench mark dataset from https://archive.ics.uci.edu/ml/datasets/Car+Evaluation and ran our algorithm. We calculated how many frequent itemsets our algorithm was giving with respect to the change of the minimum support. Attached below is the graphical representation of the number of frequent itemsets with respect to the minimum support.



Number of Frequent itemsets with respect to the minimum support

Through the analysis we found that our results were getting influenced by the minimum support the higher the minimum support the lesser the number of frequent itemsets we were getting in our output.

We took a bench mark dataset from https://archive.ics.uci.edu/ml/datasets/Car+Evaluation https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer/ and ran our algorithm along with existing code of FP Growth algorithm(https://github.com/chonyy/fpgrowth_py) and found that our algorithm was producing more frequent items sets as compared to the existing algorithm. Attached below is the graphical representation of the comparison between the algorithms based upon the number of frequent itemsets.

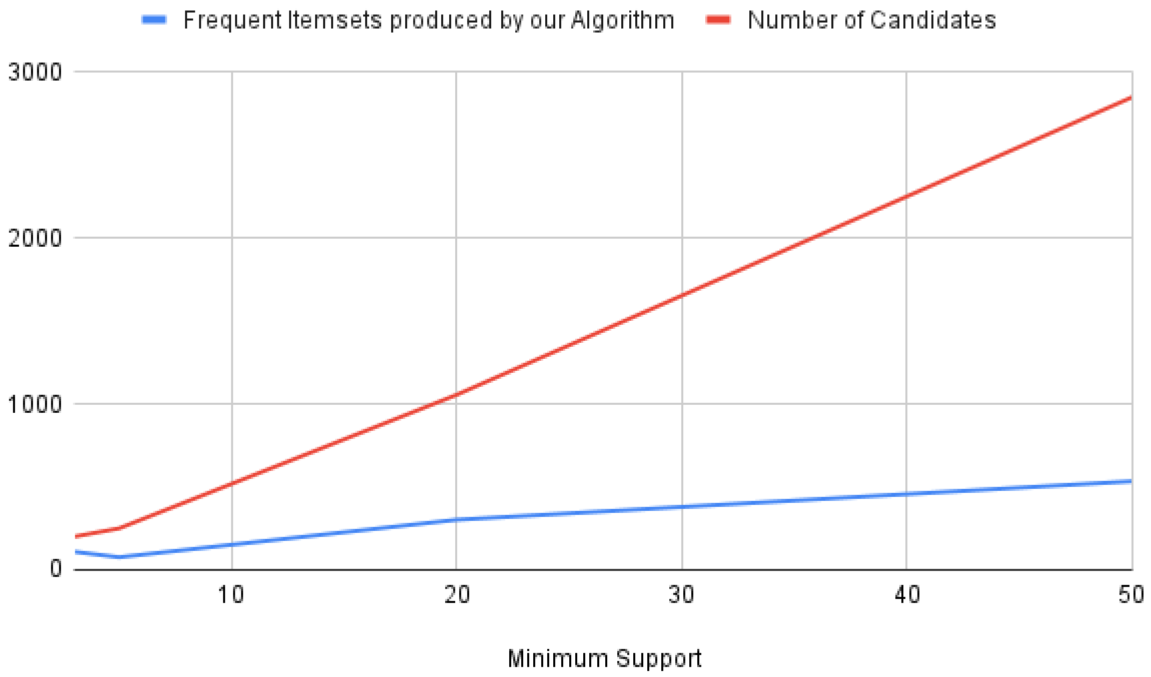


Comparison with existing algorithms on the basis of Number of Frequent Itemsets with respect to Minimum Support

We also worked upon the distribution on the basis of Number of Candidates and Number of Patterns/Frequent Itemsets with respect to the Minimum Support provided. We did this analysis on the following benchmark datasets:

https://archive.ics.uci.edu/ml/datasets/Balloons
https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer/
https://archive.ics.uci.edu/ml/datasets/Flags
https://archive.ics.uci.edu/ml/datasets/Glass+Identification
https://archive.ics.uci.edu/ml/datasets/Cylinder+Bands

Below is the graphical representation of our distribution analysis.



Distribution on the basis of Number of Candidates and Number of Frequent Itemsets with respect to the Minimum Support