UPES

UNIVERSITY WITH A PURPOSE

Report for Minor II

Submitted by

500069734	R134218123	Pratham Pandey
500068183	R134218125	Pulkit Mittal
500067543	R134218152	Shashwat Chitransh
500068730	R134218206	Karmanya Dadhich

Under the mentorship of

Ms. Shahina Anwarul

(Assistant Professor)

Market Sales Analysis by Identifying Frequent Patterns

INDEX

Abstract

A data mining approach used in markets are targeting more customers by using Frequent Pattern Analysis using FP Growth Algorithm. By the help of this technique, the store manager will able to fulfill the requirements more as per need of the customers by showing them other sets of products that are closely related to a specific product known as combinations. This in turn will help in reducing the efforts of the customers by overcoming the problem of increased lookouts for other products.

We will be using this technique of Data Mining to overcome the challenges of most of the business of marts and retailers as in todays' generation the major factor is "Population" which is increasing day by day globally. As more population results in more production of consumer goods as well as leading to more retailers and businesses. With all these the major issue is high competition among business organizations.

Keywords: *Data Mining, Fp-Tree, Fp-Growth, Frequent patterns, Analysis.*

Introduction

As in today's generation many marts and wholesalers are competing with each other to provide more satisfaction to their customers. As if the needs of the customer are fulfilled by their retailers along with their less efforts will create a chance of more shopping of the different products by the customer. To understand how to analyze this pattern we are going to implement Frequent Pattern Analysis by the help of FP Growth Algorithm.

FP Growth algorithm discovers the frequent itemset without the candidate generation. It follows two steps such as: In step one it builds a compact data structure called the FP-Tree, in step two it directly extracts the frequent item sets from the FP-Tree. FP-Tree was proposed by Han. The advantage is that it constructs conditional pattern base from database which satisfies minimum support, due to compact structure and no candidate generation it requires less memory. The disadvantage is that it performs badly with long pattern data sets. It is based on a prefix tree representation of the given database of transactions (called an FP-tree), which can save considerable amounts of memory for storing the transactions. The basic idea of the FP-growth algorithm can be described as a recursive elimination scheme

FP-Tree was proposed by Han. FP-Tree represents all the relevant frequent information from a data set due to its compact structure. Each and every path of FP-Tree represents a frequent itemset and nodes in the path are arranged in a decreasing order of the frequency. The great advantage of FP-Tree is that all the overlapping item sets share the same prefix path. Because of this the information of the data set is highly compressed. It scans the database only twice and it does not need any candidate generation.

Literature Review

1) Yuang, et al. recalls how Walmart in a study found that at certain time of a week (mostly Friday) products such as beer and diapers were bought together while applying FP growth algorithm.[1]

2) B.S Kumar, et al. proposed that one of the profound implementation of FP growth algorithm is in Web usage mining. This helps to better understand the common searching patterns that are used while visiting a particular website. For example, while reading about a certain smart phone on a website, some readers searched for headphones of the same manufacturer.[2]

3) In a research conducted by Sidhu, Shivam et al. FP growth algorithm was found particularly beneficial in order to obtain association rules between related data over old ideas such as Apriori algorithm which consist several drawbacks such as repeated scans of the whole database.[3]

4) Wang, Bowei, et al. explains how FP growth has also expanded its roots to medical science.It was used to reveal some association rules from health information. For example, more than 30 % people suffering from hypertension are having high systolic pressure and liver problems.[4]

Problem Statement

Nowadays people buy daily goods from super market nearby. There are many supermarkets that provide goods to their customer. The problem many retailers face is the placement of the items. They are unaware of the purchasing habits of the customer so they don't know which items should be laced together in their store. With the help of this application shop managers can determine the strong relationships between the items which ultimately helps them to put products that co-occur together close to one another and also to know which item to stock more.

Objectives

The primary objective of Market Sales Analysis is to improve the effectiveness of marketing and sales tactics using the customer data that is accumulated with the enterprise during the sales transaction

The objective of our project is to identify the next product that might interest a customer.

This way marketing and sales team can develop more effective pricing, product placement, and cross-sell and up-sell strategies.

Frequent Pattern Analysis can help to optimize and ease inventory control. Predicting product sales in specific locations can improve shipping times and warehouse operations.

This translates into increased revenues, lower costs and higher profit.

Methodology

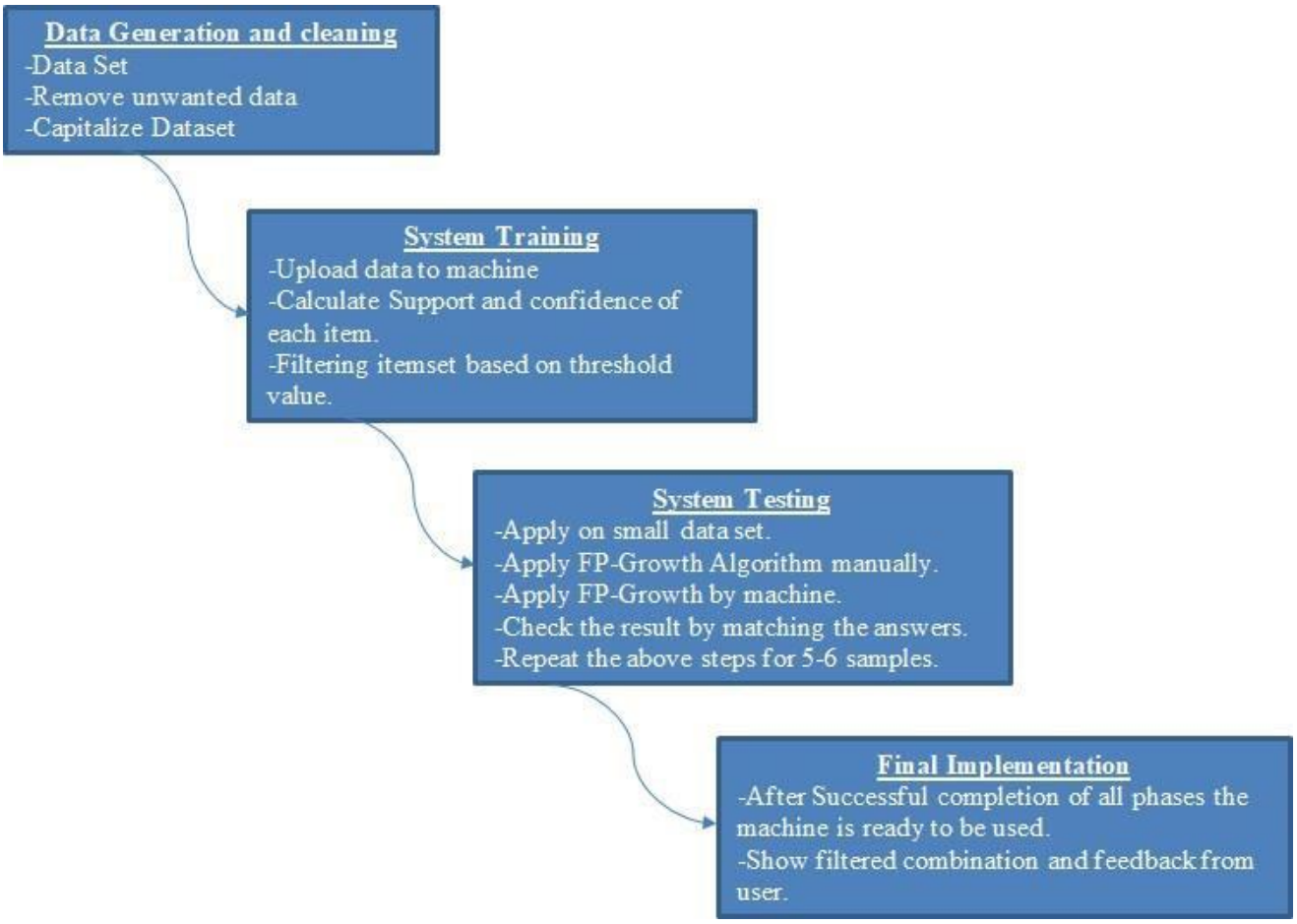
(A) Frequent Item set generation: -

1. Let I=i1, i2, i3… be a set of literals, called items.
2. Let D (D= {T1, T2, T3...}) be a set of transactions, where each transaction T, is a set of items such that T belongs to I.
3. A unique identifier TID (Transaction ID) is given to each transaction. A transaction T is said to contain X, a set of items in I.
4. For a given transaction in database D, the total number of transactions it contains is N.
5. Define the support count(X) of item set X such that (X is a proper subset of I) as the number of transactions T in D making X=T and the support "support(X)" of item set X as [count(X)/N].

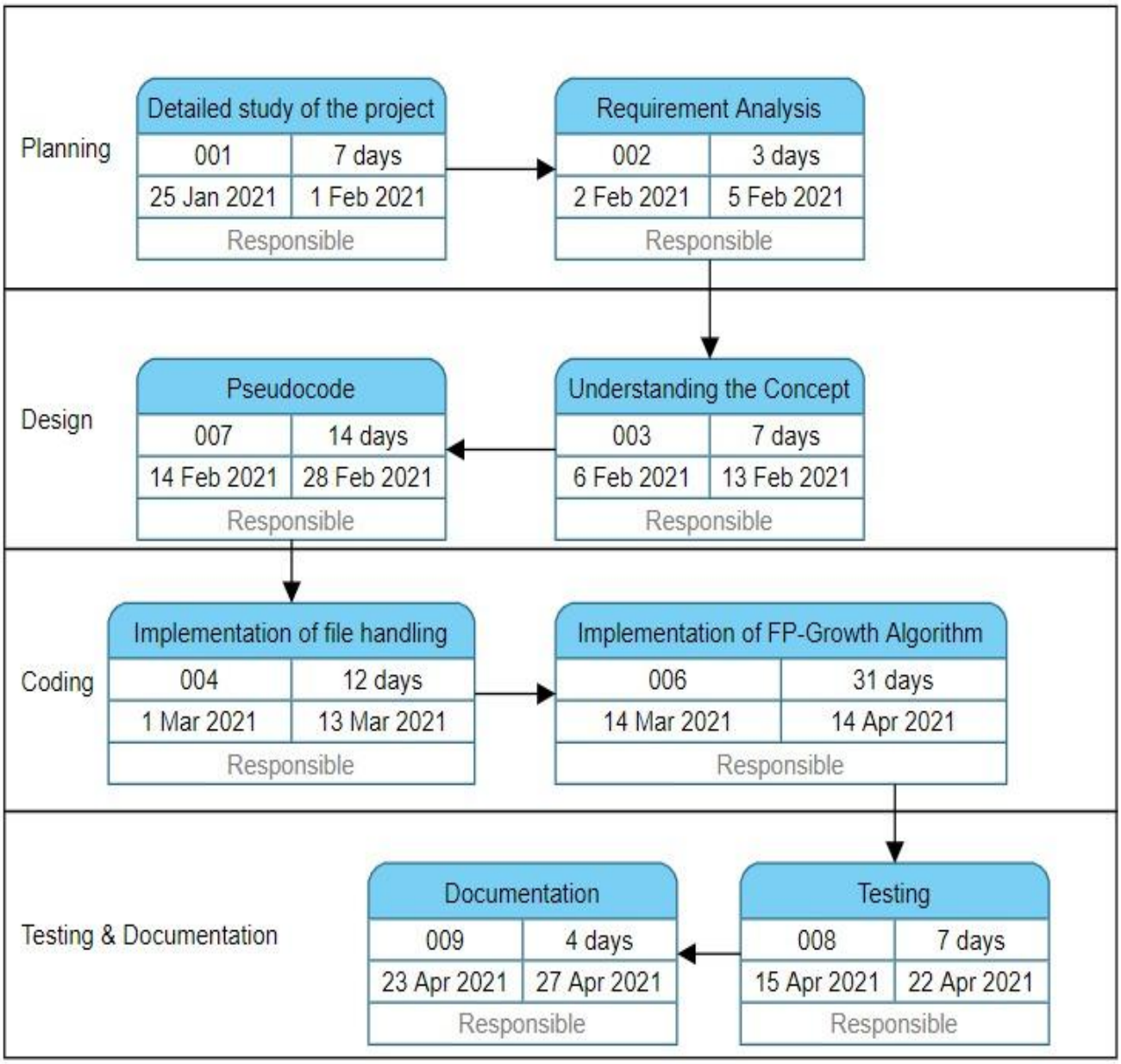
(B) Applying FP Growth Algorithm: -

1. FP-Growth algorithm compresses the database into a frequent pattern tree (FP-tree) and still maintains the information of associations between item sets.
2. Scanning the database for the first time, we can obtain a set of frequent items and their support count. The collection of frequent items is ordered by decreasing sequence of support count.
3. Then the algorithm creates the root node of the tree, with the tag “null.” Then it scans the database for the second time. Each item in a transaction is ordered by the sequence and later it creates a branch for each transaction.
4. For convenience of tree traversal, the algorithm creates an item header table. Each item through a node link points to itself in FP-tree.
5. The algorithm starts by the frequent patterns length of 1 (initial suffix pattern) and builds its conditional pattern base (a “sub database” consisting of the prefix path set which appears with the suffix pattern). Then, algorithm builds a FP-tree for the conditional pattern base and recursively digs the tree. The achievement of pattern growth gets through the link between frequent patterns generating by conditional FP-tree and suffix pattern.

Diagrammatic Representation



Pert Chart


Planning
Detailed study of the project
001
7 days
25 Jan 2021
1 Feb 2021
Responsible
Requirement Analysis
002
3 days
2 Feb 2021
5 Feb 2021
Responsible
Design
Pseudocode
007
14 days
14 Feb 2021
28 Feb 2021
Responsible
Understanding the Concept
003
7 days
6 Feb 2021
13 Feb 2021
Responsible
Coding
Implementation of file handling
004
12 days
1 Mar 2021
13 Mar 2021
Responsible
Implementation of FP-Growth Algorithm
006
31 days
14 Mar 2021
14 Apr 2021
Responsible
Testing & Documentation
Documentation
009
4 days
23 Apr 2021
27 Apr 2021
Responsible
Testing
008
7 days
15 Apr 2021
22 Apr 2021
Responsible

Program

Code is divided into three files:

1. **fileReader.java**
2. **FpTree.java**
3. **FpNode.java**

fileReader.java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

public class fileReader {

	public static String readFile(String file_name, String encoding) {
		File anyFile = new File(file_name);
		try {
			FileInputStream inStream = new FileInputStream(anyFile);
			BufferedReader reader = new BufferedReader(new
InputStreamReader(
					inStream, encoding));
			String line = new String();
			String text = new String();
			while ((line = reader.readLine()) != null) {
				text += line;
			}
```

```
                reader.close();
                return text;
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return null;
        }
//-------------------------------------------------------------------------------------------
        // Read and input file based on file name and separator
        public static List<String[]> scanChart(String file_name,
                String separator, String encoding) {
            List<String[]> matrix = new ArrayList<String[]>();
            File anyFile = new File(file_name);
            try {
                FileInputStream inStream = new FileInputStream(anyFile);
                BufferedReader reader = new BufferedReader(new
InputStreamReader(
                        inStream, encoding));
                String line = new String();
                while ((line = reader.readLine()) != null) {
                    matrix.add(line.split(separator));
                }
                reader.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            return matrix;
        }

        public static void main(String[] args) {
      List<String[]> matrix = scanChart("input.txt", " ", "UTF-8");
      System.out.println(matrix.size());
      readFile("input.txt", "UTF-8");}}
```

FpTree.java

```
import java.io.*;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;

public class FpTree
{
	private static final float sup_rate = 0.001f;
	private static long min_sup;

	public static void main(String[] args) throws FileNotFoundException
	{
		System.out.print("Enter Minimum Support : ");
		Scanner sc=new Scanner(System.in);
		min_sup = sc.nextLong();
		List<String[]> table = fileReader.scanChart("input.txt", " ", "utf-8");
		PrintStream o = new PrintStream(new File("output.txt"));
	System.setOut(o);
```

```
            long totalMilliSeconds1 = System.currentTimeMillis();
            System.out.println("Minimum Support Count: " + min_sup);
            Map<String, Integer> frequentLink = new LinkedHashMap<String,
Integer>();// First level frequent item
            Map<String, FpNode> topHead = getTop(table, frequentLink);
            FpNode root = findFpTree(table, topHead, frequentLink);
            Map<Set<FpNode>, Long> frequent = growthFunc(root, topHead, null);
            long totalMilliSeconds2 = System.currentTimeMillis();
            long total = (totalMilliSeconds2 - totalMilliSeconds1);
            System.out.println("Time Cost: " + total + " milliseconds");
            System.out.println("Frequent Patterns Generated: ");
      System.out.println("-----------------------------------------------");
            for (Map.Entry<Set<FpNode>, Long> Freq : frequent.entrySet())
            {
                  for (FpNode node : Freq.getKey())
                        System.out.print(node.ID + " ");
                  System.out.println("\t" + Freq.getValue());
            }
            System.out.println("-----------------------------------------------");


      }
//-------------------------------------------------------------------------------------


   //Based on FP growth, recursively finding frequent items
      private static Map<Set<FpNode>, Long> growthFunc(FpNode root, Map<String,
FpNode> topHead, String idMark)
      {
            Map<Set<FpNode>, Long> qualifyFreq = new HashMap<Set<FpNode>,
Long>();
            Set<String> keys = topHead.keySet();
            String[] keysArray = keys.toArray(new String[0]);
            String headID = keysArray[keysArray.length - 1];
            if (discretePath(topHead, headID)) // When there is only one path, all
combinations on the
```

```
{
    // path can be obtained to get the adjusted frequent set
       if (idMark == null)
                    {
                        return qualifyFreq;
                    }
                    FpNode leaf = topHead.get(headID);
                    List<FpNode> paths = new ArrayList<FpNode>();// Save path node
from itself to the top
                    paths.add(leaf);
                    FpNode node = leaf;
                    while (node.parent.ID != null)
                    {
                        paths.add(node.parent);
                        node = node.parent;
                    }
                    qualifyFreq = combiPattern(paths, idMark);
                    FpNode tempNode = new FpNode(idMark, -1L);
                    qualifyFreq = addLeafToFrequent(tempNode, qualifyFreq);

            }
            else
            {
                    for (int i = keysArray.length - 1; i >= 0; i--) //Recursively seeking
frequent sets of conditional trees
                    {
                        String key = keysArray[i];
                        List<FpNode> leafs = new ArrayList<FpNode>();
                        FpNode link = topHead.get(key);
                        while (link != null)
                        {
                            leafs.add(link);
                            link = link.next;
                        }
```

```
Map<List<String>, Long> paths = new HashMap<List<String>, Long>();
Long leafCount = 0L;
                    FpNode noParentNode = null;
                    for (FpNode leaf : leafs)
                    {
                            List<String> path = new ArrayList<String>();
                            FpNode node = leaf;
                            while (node.parent.ID != null)
                            {
                                    path.add(node.parent.ID);
                                    node = node.parent;
                            }
                            leafCount += leaf.count;
                            if (path.size() > 0)
                            {
                                    paths.put(path, leaf.count);
                            }
                            else      // No parent node
                            {
                                    noParentNode = leaf;
                            }
                    }
                    if (noParentNode != null)
                    {
                            Set<FpNode> oneItem = new HashSet<FpNode>();
                            oneItem.add(noParentNode);
                            if (idMark != null)
                            {
                                    oneItem.add(new FpNode(idMark, -2));
                            }
                            qualifyFreq.put(oneItem, leafCount);
                    }
                    Holder holder = getConditionFpTree(paths);
```

```
if (holder.topHead.size() != 0)
{
	Map<Set<FpNode>, Long> preFres = growthFunc(holder.root,
			holder.topHead, key);
if (idMark != null) {
		FpNode tempNode = new FpNode(idMark, leafCount);
		preFres = addLeafToFrequent(tempNode, preFres);
			}
			qualifyFreq.putAll(preFres);
			}
		}
	}
		return qualifyFreq;

}
//-----------------------------------------------------------------------------------------
	// Add leaf nodes to frequent set
	private static Map<Set<FpNode>, Long> addLeafToFrequent(FpNode leaf,
Map<Set<FpNode>, Long> qualifyFreq)
	{
		if (qualifyFreq.size() == 0)
		{
			Set<FpNode> set = new HashSet<FpNode>();
			set.add(leaf);
			qualifyFreq.put(set, leaf.count);
		}
		else
		{
			Set<Set<FpNode>>keys=new
HashSet<Set<FpNode>>(qualifyFreq.keySet());
			for (Set<FpNode> set : keys)
			{
				Long count = qualifyFreq.get(set);
				qualifyFreq.remove(set);
```

```
                        set.add(leaf);
                        qualifyFreq.put(set, count);
                    }
                }
                return qualifyFreq;
        }
//-------------------------------------------------------------------------------------------
        //Determine if an fp-tree is a single path
        private static boolean discretePath(Map<String, FpNode> topHead, String
tableLink)
        {
                if (topHead.size() == 1 && topHead.get(tableLink).next == null)
                {
                        return true;
                }
                return false;
        }
//-------------------------------------------------------------------------------------------
        //Generate condition tree
        private static Holder getConditionFpTree(Map<List<String>, Long> paths)
        {
                List<String[]> table = new ArrayList<String[]>();
                for (Map.Entry<List<String>, Long> entry : paths.entrySet())
                {
                        for (long i = 0; i < entry.getValue(); i++)
                        {
                                table.add(entry.getKey().toArray(new String[0]));
                        }
                }
                Map<String, Integer> frequentLink = new LinkedHashMap<String,
Integer>(); // First level frequent set
                Map<String, FpNode> cHeader = getTop(table, frequentLink);
                FpNode cRoot = findFpTree(table, cHeader, frequentLink);
                return new Holder(cRoot, cHeader);
```

```
	}
//-------------------------------------------------------------------------------------------
	//Find all combinations on a single path and frequent items generate from ID
	private static Map<Set<FpNode>, Long> combiPattern(List<FpNode> paths,
String idMark)
	{
		Map<Set<FpNode>, Long> qualifyFreq = new HashMap<Set<FpNode>,
Long>();
		int size = paths.size();
		for (int mask = 1; mask < (1 << size); mask++) // Find out all the
combinations, count from 1 and ignore all empty set
		{
			Set<FpNode> set = new HashSet<FpNode>();

			for (int i = 0; i < paths.size(); i++) // Find out every possible
choice
			{
				if ((mask & (1 << i)) > 0)
				{
					set.add(paths.get(i));
				}
			}
			long minValue = Long.MAX_VALUE;
			for (FpNode node : set)
			{
				if (node.count < minValue)
				{
					minValue = node.count;
				}
			}
			qualifyFreq.put(set, minValue);
		}
		return qualifyFreq;
	}
```

```
//-----------------------------------------------------------------------------------------
   // Print out the FpTree
        private static void printTree(FpNode root)
        {
                System.out.println(root);
                FpNode node = root.pickChild(0);
                System.out.println(node);
                for (FpNode child : node.children)
                        System.out.println(child);
                System.out.println("*****");
                node = root.pickChild(1);
                System.out.println(node);
                for (FpNode child : node.children)
                        System.out.println(child);


        }
//-----------------------------------------------------------------------------------------
        // Build FpTree structure
        private static FpNode findFpTree(List<String[]> table, Map<String, FpNode>
topHead, Map<String, Integer> frequentLink)
        {
                FpNode root = new FpNode();
                int count = 0;
                for (String[] line : table)
                {
                        String[] orderLink = orderedLink(line, frequentLink);


                        FpNode parent = root;
                        for (String idMark : orderLink)
                        {
                                int index = parent.ifChild(idMark);
                                if (index != -1)    //Don't need build new node because
already contain this ID
```

```
                        {
                                parent = parent.pickChild(index);
                                parent.addCount();
                        }
        else
                {
                        FpNode node = new FpNode(idMark);
                        parent.addChild(node);
                        node.setParent(parent);
                        FpNode nextNode = topHead.get(idMark);
                        if (nextNode == null)    //If the node is empty, added to the node
                        {
                                topHead.put(idMark, node);
                        }
                        else    // added pointer to the node
                        {
                        while (nextNode.next != null)
                                {
                                        nextNode = nextNode.next;
                                }
                                nextNode.next = node;
                        }
        parent = node;   // All the child node will under this parent node
                                }
                        }
                }
                return root;
        }
//-------------------------------------------------------------------------------------------
        // Sort the ID in descending order based on the value of the frequentLink
        private static String[] orderedLink(String[] line, Map<String, Integer>
frequentLink)
```

```
{
        Map<String, Integer> countMap = new HashMap<String, Integer>();
        for (String idMark : line)
        {
            if (frequentLink.containsKey(idMark)) // Filter out non-primary
frequent items
            {
                countMap.put(idMark, frequentLink.get(idMark));
            }
        }
        List<Map.Entry<String, Integer>> mapList = new
ArrayList<Map.Entry<String, Integer>>(countMap.entrySet());
        Collections.sort(mapList, new Comparator<Map.Entry<String, Integer>>()
        {@Override public int compare(Entry<String, Integer> v1, Entry<String,
Integer> v2)
        {return v2.getValue() - v1.getValue();}});  //Descending order ID

        String[] orderLink = new String[countMap.size()];
        int i = 0;
        for (Map.Entry<String, Integer> entry : mapList)
        {
            orderLink[i] = entry.getKey();
            i++;
        }
        return orderLink;
    }
//-----------------------------------------------------------------------------------------
    // Generate table. The key is equal to ID value. Descend ordering based on
frequent value.
    private static Map<String, FpNode> getTop(List<String[]> table, Map<String,
Integer> frequentLink)
    {
        Map<String, Integer> countMap = new HashMap<String, Integer>();
        for (String[] line : table)
        {
```

```
                for (String idMark : line)
                {
                        if (countMap.containsKey(idMark))
                        {
                                countMap.put(idMark, countMap.get(idMark) + 1);
                        }
                        else
                        {
                                countMap.put(idMark, 1);
                        }
                }
        }
        for (Map.Entry<String, Integer> entry : countMap.entrySet())
        {
                if (entry.getValue() >= min_sup) // Filter out items that do not
meet the support value
                {
                frequentLink.put(entry.getKey(), entry.getValue());
                }
        }
        List<Map.Entry<String, Integer>> mapList = new
ArrayList<Map.Entry<String, Integer>>(frequentLink.entrySet());
        Collections.sort(mapList, new Comparator<Map.Entry<String, Integer>>()
        {@Override public int compare(Entry<String, Integer> v1, Entry<String,
Integer> v2)
        {return v2.getValue() - v1.getValue();}});  //Descending order ID

        frequentLink.clear();  // Clear the table for keeping key values in order
        Map<String, FpNode> topHead = new LinkedHashMap<String, FpNode>();
        for (Map.Entry<String, Integer> entry : mapList)
        {
                topHead.put(entry.getKey(), null);
                frequentLink.put(entry.getKey(), entry.getValue());
        }
```

```
return topHead;
        }
}

//-------------------------------------------------------------------------------------
// Generate holder for conditional tree.
class Holder
{
        public final FpNode root;
        public final Map<String, FpNode> topHead;

        public Holder(FpNode root, Map<String, FpNode> topHead)
        {
                this.root = root;
                this.topHead = topHead;
        }
}
```

FpNode.java

```
import java.util.ArrayList;
import java.util.List;

public class FpNode
{
        String ID;
        List<FpNode> children;
        FpNode parent;
        FpNode next;
        long count;
//----
```

```
	public FpNode()  // Root Node
	{
		this.ID = null;
		this.count = -1;
		children = new ArrayList<FpNode>();
		next = null;
		parent = null;
	}
//-------------------------------------------------------
	public FpNode(String ID) //Non-root Node Structure
	{
		this.ID = ID;
		this.count = 1;
		children = new ArrayList<FpNode>();
		next = null;
		parent = null;
	}
//-------------------------------------------------------
	public FpNode(String ID, long count) // Generate non-root node
	{
		this.ID = ID;
		this.count = count;
		children = new ArrayList<FpNode>();
		next = null;
		parent = null;
	}
//-------------------------------------------------------
	public void addChild(FpNode child)   //Add a Child
	{
		children.add(child);
	}
```

```
public void addCount(int count)
	{
		this.count += count;
	}
//-------------------------------------------------------
	public void addCount()  //Count add 1
	{
		this.count += 1;
	}
//-------------------------------------------------------
	public void NextNode(FpNode next)   //Setup next node
	{
		this.next = next;
	}

	public void setParent(FpNode parent)
	{
		this.parent = parent;
	}

//-------------------------------------------------------
	public FpNode pickChild(int index) // Pick pointed Child
	{
		return children.get(index);
	}

//-------------------------------------------------------
	public int ifChild(String ID) // Search Child ID and see if exist, and return
result
	{
		for (int i = 0; i < children.size(); i++)
			if (children.get(i).ID.equals(ID))
```

```
                        return i;
                return -1;
        }

        public String outSearch()
        {
                return "id: " + ID + " Count: " + count + " Target amount "
                        + children.size();
        }
}
```

Input

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\pulki\Documents\Minor 2> javac FpTree.java
PS C:\Users\pulki\Documents\Minor 2> java FpTree
Enter Minimum Support : 8
PS C:\Users\pulki\Documents\Minor 2>
```

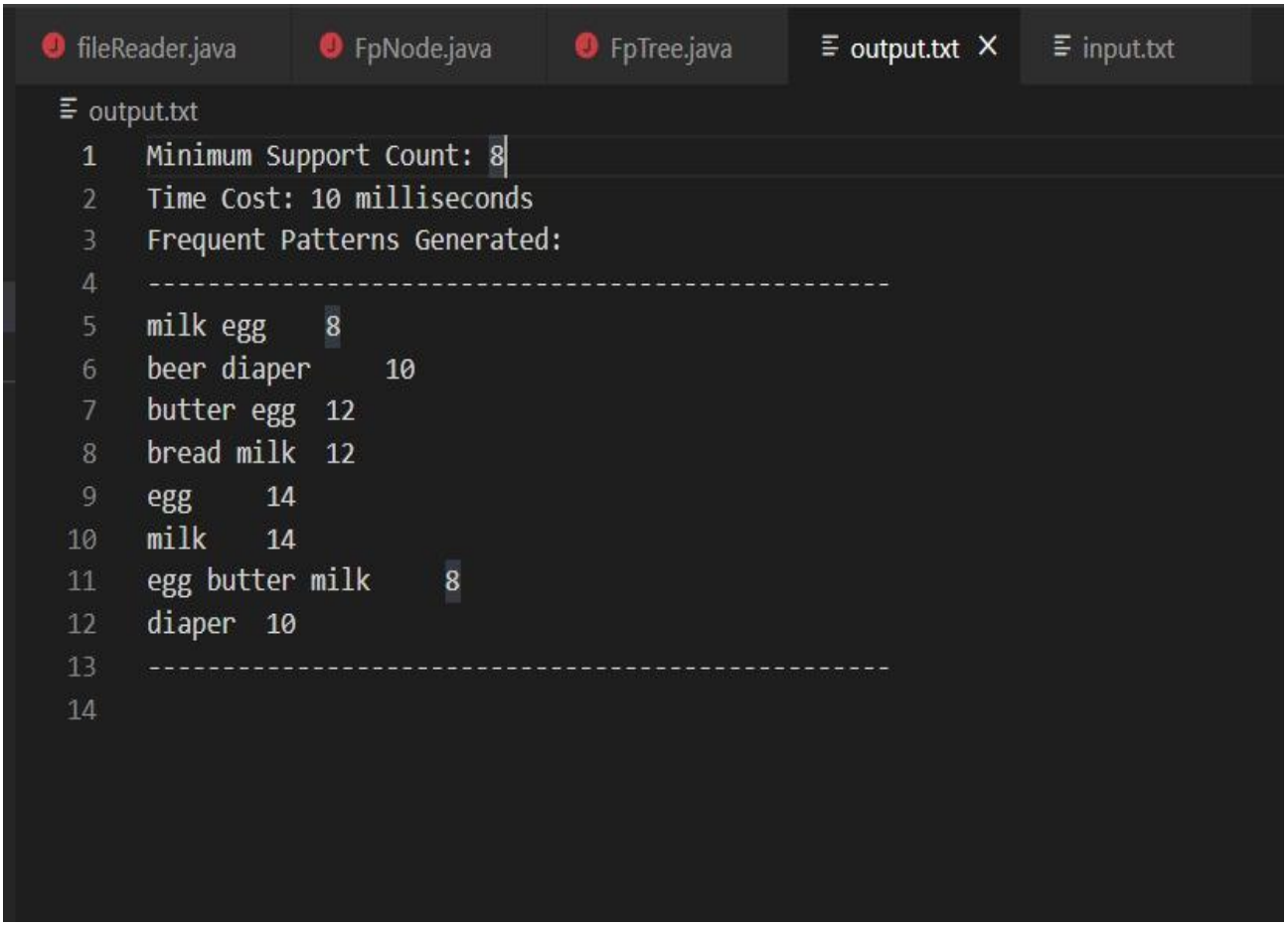
Data set

fileReader.java | FpNode.java | FpTree.java | output.txt | input.txt

input.txt

```
1   milk bread beer diaper
2   milk bread
3   butter bread milk egg
4   beer diaper
5   milk bread butter egg
6   butter egg
7   butter diaper egg beer
8   milk beer diaper bread
9   butter egg rice milk
10  beer diaper rice egg
11  bread butter milk egg
12  milk bread beer diaper
13  milk bread
14  butter bread milk egg
15  beer diaper
16  milk bread butter egg
17  butter egg
18  butter diaper egg beer
19  milk beer diaper bread
20  butter egg rice milk
21  beer diaper rice egg
22  bread butter milk egg
```

Output


fileReader.java
FpNode.java
FpTree.java
output.txt
input.txt
output.txt
1 Minimum Support Count: 8
2 Time Cost: 10 milliseconds
3 Frequent Patterns Generated:
4 ---
5 milk egg 8
6 beer diaper 10
7 butter egg 12
8 bread milk 12
9 egg 14
10 milk 14
11 egg butter milk 8
12 diaper 10
13 ---
14

References

[1] Yuan, M., Pavlidis, Y., Jain, M. and Caster, K., 2016, November. Walmart online grocery personalization: Behavioral insights and basket recommendations. In *International Conference on Conceptual Modeling* (pp. 49-64). Springer, Cham.

[2] Kumar, B.S. and Rukmani, K.V., 2010. Implementation of web usage mining using APRIORI and FP growth algorithms. *Int. J. of Advanced networking and Applications*, *1*(06), pp.400-404.

[3] Sidhu, Shivam, et al. "FP Growth algorithm implementation." *International Journal of Computer Applications* 93.8 (2014).

[4] Wang, Bowei, et al. "Comprehensive association rules mining of health examination data with an extended FP-growth method." *Mobile Networks and Applications* 22.2 (2017): 267-274.

[5] F. Bonchi and B. Goethals. FP-Bonsai: The Art of Growing and Pruning Small FP-trees. Proc. 8thPacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04, Sydney, Australia),155–160. Springer-Verlag, Heidelberg, Germany 2004.

[6] Data Mining - concept and techniques by Jiawei Han, Jian Pei, Micheline Kamber.

[7] Use the Link: https://link.springer.com/article/10.1007/s10462-018-9629-z.

[8] https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Frequent_Pattern_Mining/The_FP-Growth_Algorithm