Department of
Computer Science and Engineering

Title: Implementation of FP-Growth algorithm for frequent itemset mining

Data Mining Lab

CSE 436

Green University of Bangladesh

1 Objective(s)

- To gather knowledge of the basics of frequent itemset mining.
- To implement the FP-Growth algorithm on a real-world dataset.

2 FP-growth Algorithm

The FP-growth algorithm is an enhanced version of the Apriori algorithm used for Association Rule Mining in databases. The Apriori algorithm suffers from two major drawbacks: low speed and high computational cost. To address these limitations, the FP-growth algorithm offers a significantly faster and more efficient alternative.

2.1 FP-growth algorithm overview

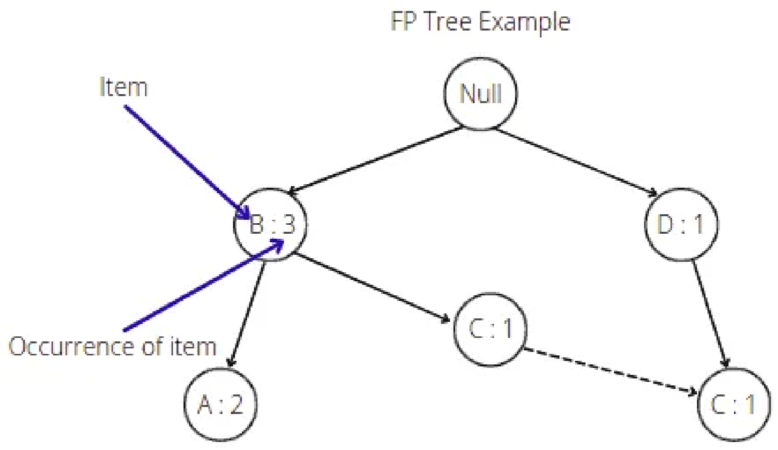
FP-growth algorithm is a tree-based algorithm for frequent itemset mining or frequent-pattern mining used for market basket analysis. The algorithm represents the data in a tree structure known as FP-tree, responsible for maintaining the association information between the frequent items.

The algorithm compresses frequent items into an FP-tree from the database while retaining association rules. Then it splits the database data into a set of conditional databases (a special kind of projected database), each of which is associated with one frequent data item.

2.2 FP-tree

FP-tree is the core concept of the FP-growth algorithm. The FP-tree is a compressed representation of the database itemset, storing the DB itemset in memory and keeping track of the association between items.

The tree is constructed by taking each itemset and adding it as a subtree. The FP-tree's whole idea is that items that occur more frequently will be more likely to be shared.



The root node in the FP-tree is null. Each node of the subtree stores at least the item name and the support (or item occurrence) number. Additionally, the node may contain a link to the node with the same name from another subtree (represents another itemset from the database).

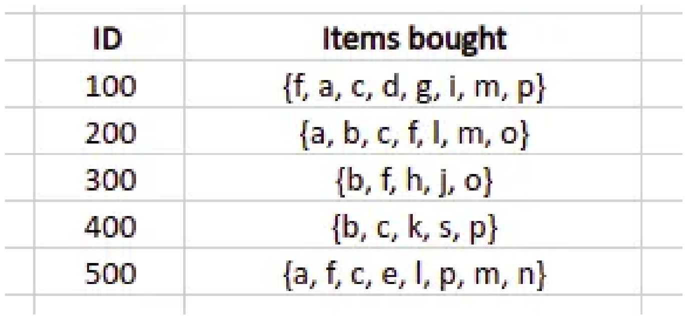
2.3 Building FP-tree

The FP-growth algorithm uses the following steps to build FP-tree from the database.

- Scan itemsets from the database for the first time
- Find frequent items (single item patterns) and order them into a list L in frequency descending order. For example, $L = \{A : 5, C : 3, D : 2, B : 1\}$
- For each transaction order, its frequent items according to the order in L

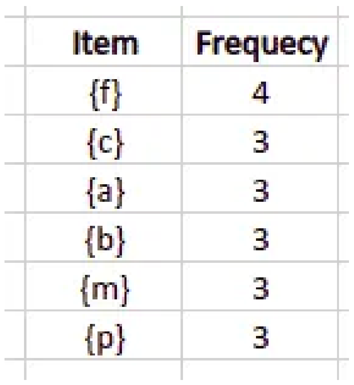
- Scan the database the second time and construct FP-tree by putting each frequency ordered transaction onto it.

To illustrate algorithm steps, let's take a sample database and create an FP-tree out of it:



ID	Items bought
100	{f, a, c, d, g, i, m, p}
200	{a, b, c, f, l, m, o}
300	{b, f, h, j, o}
400	{b, c, k, s, p}
500	{a, f, c, e, l, p, m, n}

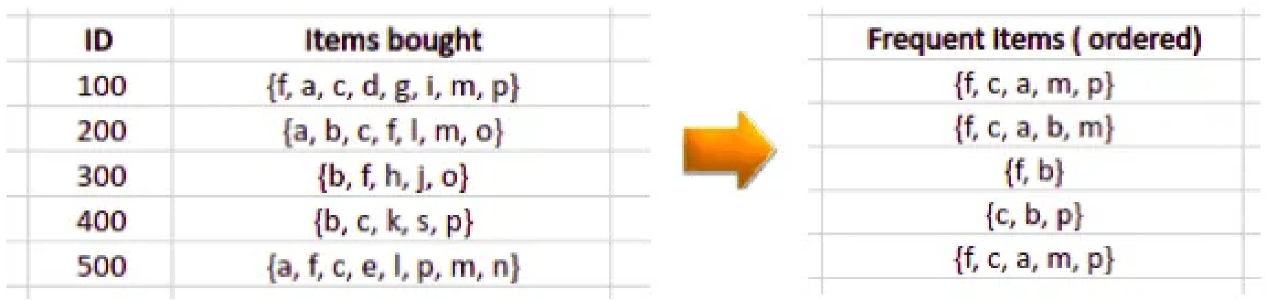
The first step is to scan the dataset, create a frequency table containing each item from the database, and arrange them in descending order. We also need to specify the minimum support (number of item occurrences). The algorithm will not put items with a support value less than the minimum support into the frequency table. For example, let's set up the minimum support value equal to 3. In that case, we will get the following frequency table:



Item	Frequecy
{f}	4
{c}	3
{a}	3
{b}	3
{m}	3
{p}	3

The next step is to scan the database the second time and arrange elements based on the frequency table:

- items with a higher frequency number will come first
- if two items have the same frequency number they will be arranged in alphabetical order



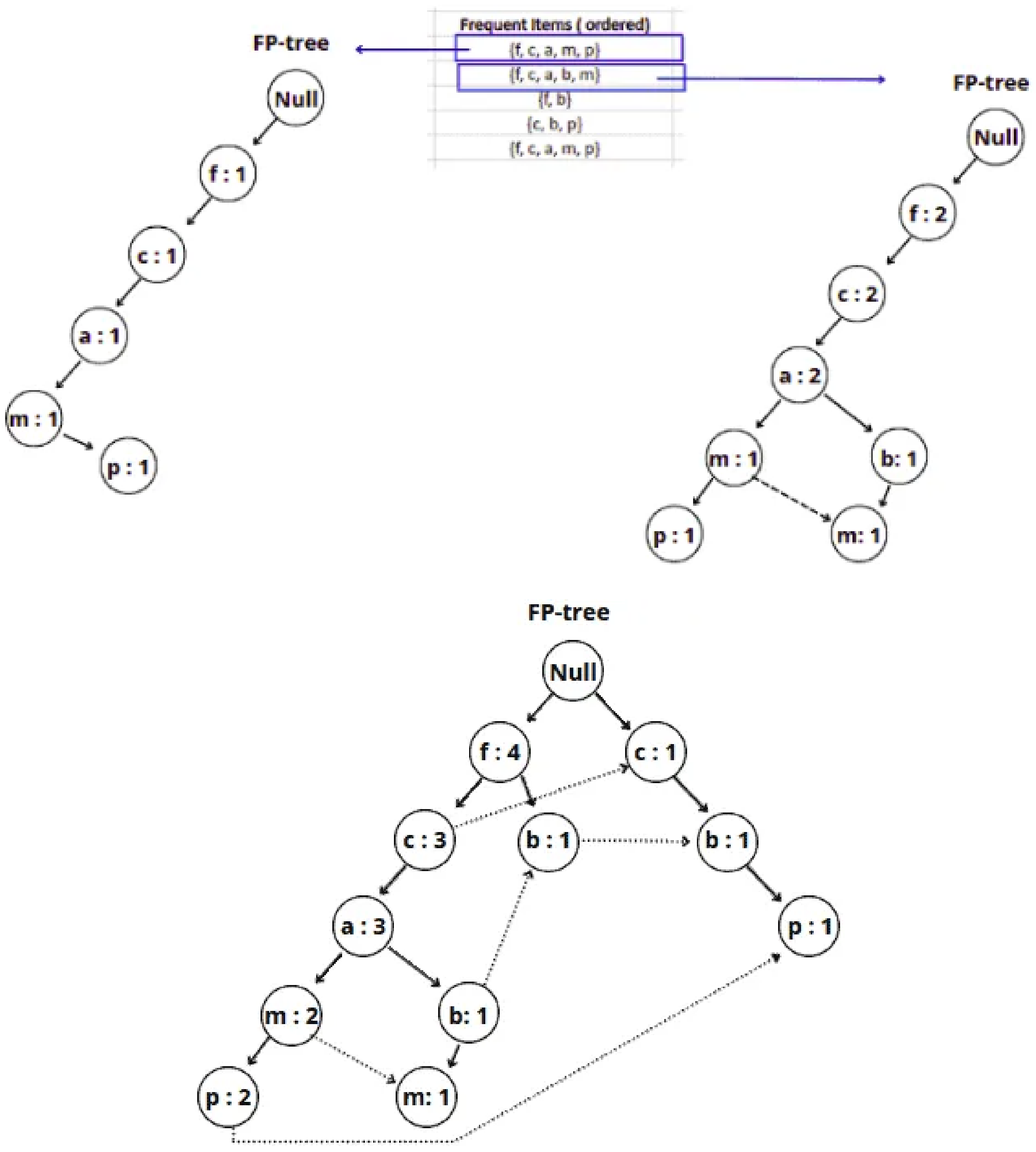
ID	Items bought
100	{f, a, c, d, g, i, m, p}
200	{a, b, c, f, l, m, o}
300	{b, f, h, j, o}
400	{b, c, k, s, p}
500	{a, f, c, e, l, p, m, n}

Frequent Items (ordered)
{f, c, a, m, p}
{f, c, a, b, m}
{f, b}
{c, b, p}
{f, c, a, m, p}

We will start with a null node, and then based on the frequent items table, we will add nodes to the tree.

For example, here's how the FP-growth will construct the FP-tree for the first frequent item from the list. Similarly, the next step will add the following item from the list to the FP-tree as shown below:

Similarly, we add other items from the frequent items table to the FP-tree.

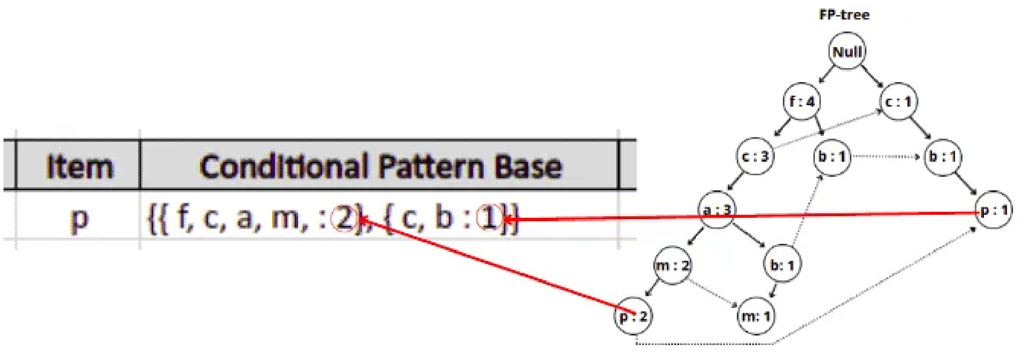


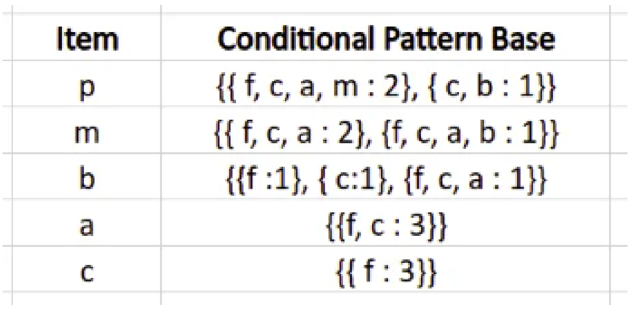
2.4 Building association rules

Once the FP-growth algorithm constructed the FP-tree, it can build different associations rules based on the minimum support value. It will take the item with the minor support count and trace that item through the FP-tree to achieve that goal. In our example, the item p has the lowest support count, and the FP-growth algorithm will produce the following paths: $\{\{f, c, a, m, p : 2\}, \{c, b : 1\}\}$

Note: The item p is located in two different subtrees of the FP-tree, so the algorithm traced both paths and added the minimum support value for every path. Similarly, the FP-growth will build the conditional pattern base table for all of the items from the FP-tree. Similarly, the FP-growth will build the conditional pattern base table for all of the items from the FP-tree. The final conditional pattern base table for all elements in our example, shown in figure 1. The next step is to get all items from the Conditional Pattern Base column that satisfy the minimum support requirement.

Let's calculate elements' occurrences for the p item: $\{f, c, a, m : 2\}, \{c, b : 1\}- > \{f : 2, c : 3, a : 2, m : 2, b : 1\}$



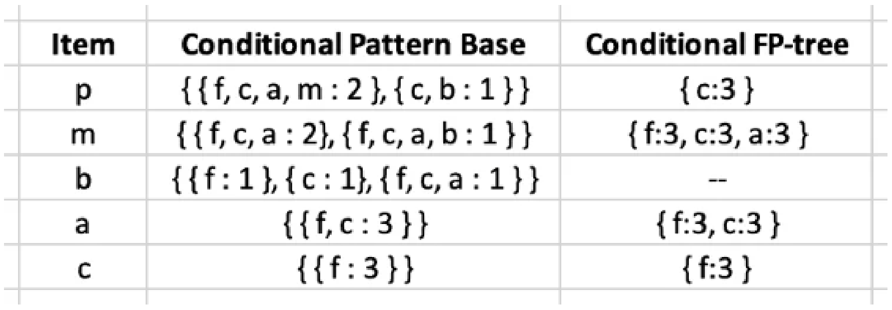


Item	Conditional Pattern Base
p	{{ f, c, a, m : 2}, { c, b : 1}}
m	{{ f, c, a : 2}, {f, c, a, b : 1}}
b	{{f :1}, { c:1}, {f, c, a : 1}}
a	{{f, c : 3}}
c	{{ f : 3}}

Figure 1: Conditional pattern base table for all elements

Only item c appears three times and satisfies the minimum support requirement. That means the algorithm will remove all other items except c.

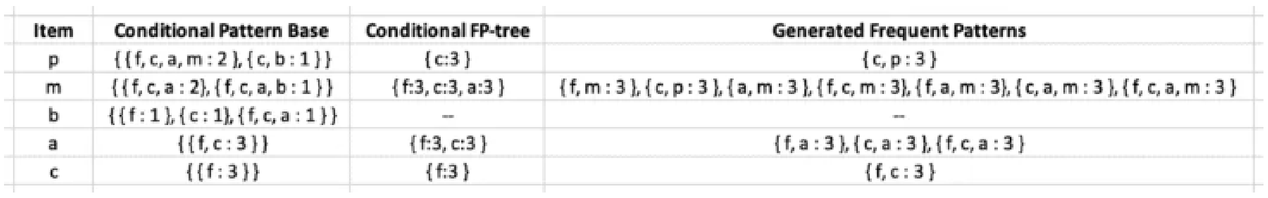
After removing items that do not meet the minimum support requirement, the algorithm will construct the following table:



Item	Conditional Pattern Base	Conditional FP-tree
p	{{f, c, a, m : 2 }, {c, b : 1 }}	{ c:3 }
m	{{f, c, a : 2}, {f, c, a, b : 1 }}	{ f:3, c:3, a:3 }
b	{{f : 1 }, {c : 1}, {f, c, a : 1 }}	--
a	{{f, c : 3 }}	{ f:3, c:3 }
c	{{f : 3 }}	{ f:3 }

The final step of creating association rules is to generate frequent patterns by pairing the items of the Conditional FP-tree column with the corresponding item from the Item column.

For example, for the first row, the algorithm needs to take c:3 from the Conditional FP-tree column, create its combination with the p element and add the support count value. Similarly, the FP-growth algorithm will generate frequent patterns (or association rules):



Item	Conditional Pattern Base	Conditional FP-tree	Generated Frequent Patterns
p	{{f, c, a, m : 2 }, {c, b : 1 }}	{ c:3 }	{ c, p : 3 }
m	{{f, c, a : 2}, {f, c, a, b : 1 }}	{ f:3, c:3, a:3 }	{ f, m : 3 }, { c, p : 3 }, { a, m : 3 }, { f, c, m : 3}, { f, a, m : 3}, { c, a, m : 3 }, { f, c, a, m : 3 }
b	{{f : 1 }, {c : 1}, {f, c, a : 1 }}	--	--
a	{{f, c : 3 }}	{ f:3, c:3 }	{ f, a : 3 }, { c, a : 3 }, { f, c, a : 3 }
c	{{f : 3 }}	{ f:3 }	{ f, c : 3 }

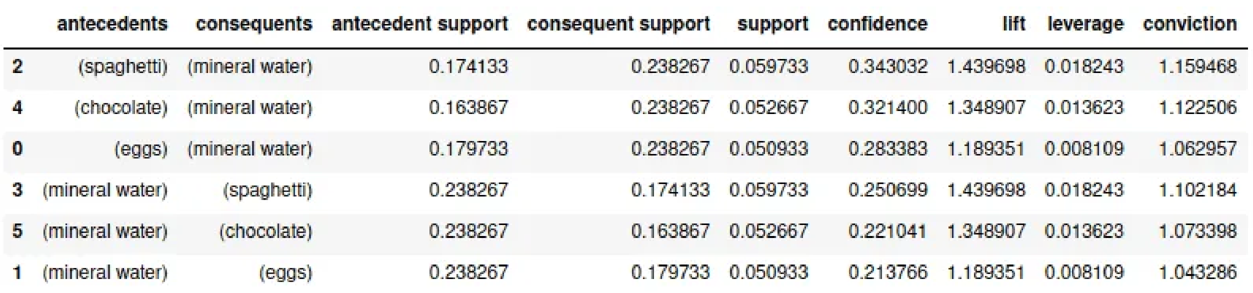
3 Implementation in Python

The following implementation demonstrates the FP-Growth algorithm on a real-world Market Basket Optimization dataset from Kaggle (Market Basket Optimisation Dataset). This dataset represents customer transactions in a supermarket.

```python
# importing module
import pandas as pd
# dataset
dataset = pd.read_csv("Market_Basket_Optimisation.csv")
# printing the shape of the dataset
dataset.shape
# printing the columns and few rows using head
dataset.head()
# importing module
import numpy as np
# Gather All Items of Each Transactions into Numpy Array
transaction = []
for i in range(0, dataset.shape[0]):
    for j in range(0, dataset.shape[1]):
        transaction.append(dataset.values[i,j])
# converting to numpy array
transaction = np.array(transaction)
print(transaction)

#  Transform Them a Pandas DataFrame
df = pd.DataFrame(transaction, columns=["items"])
# Put 1 to Each Item For Making Countable Table, to be able to perform Group By
df["incident_count"] = 1
#  Delete NaN Items from Dataset
indexNames = df[df['items'] == "nan" ].index
df.drop(indexNames , inplace=True)
# Making a New Appropriate Pandas DataFrame for Visualizations
df_table = df.groupby("items").sum().sort_values("incident_count", ascending=
    False).reset_index()
#  Initial Visualizations
df_table.head(5).style.background_gradient(cmap='Blues')

# importing required module
import plotly.express as px
# to have a same origin
df_table["all"] = "Top 50 items"
# creating tree map using plotly
fig = px.treemap(df_table.head(50), path=['all', "items"], values='
    incident_count',
                  color=df_table["incident_count"].head(50), hover_data=['items'
                      ],
                  color_continuous_scale='Blues',
                )
# ploting the treemap
fig.show()

# Transform Every Transaction to Seperate List & Gather Them into Numpy Array
transaction = []
for i in range(dataset.shape[0]):
    transaction.append([str(dataset.values[i,j]) for j in range(dataset.shape
        [1])])
# creating the numpy array of the transactions
transaction = np.array(transaction)
# importing the required module
from mlxtend.preprocessing import TransactionEncoder
# initializing the transactionEncoder
te = TransactionEncoder()
```

```python
te_ary = te.fit(transaction).transform(transaction)
dataset = pd.DataFrame(te_ary, columns=te.columns_)
# dataset after encoded
dataset.head()

# select top 30 items
first30 = df_table["items"].head(30).values
# Extract Top 30
dataset = dataset.loc[:,first30]
# shape of the dataset
dataset.shape

#Importing Libraries
from mlxtend.frequent_patterns import fpgrowth
#running the fpgrowth algorithm
res=fpgrowth(dataset,min_support=0.05, use_colnames=True)
# printing top 10
res.head(10)

# importing required module
from mlxtend.frequent_patterns import association_rules
# creating asssociation rules
res=association_rules(res, metric="lift", min_threshold=1)
# printing association rules
res

# Sort values based on confidence
res.sort_values("confidence",ascending=False)
```

4 Input/Output



	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2	(spaghetti)	(mineral water)	0.174133	0.238267	0.059733	0.343032	1.439698	0.018243	1.159468
4	(chocolate)	(mineral water)	0.163867	0.238267	0.052667	0.321400	1.348907	0.013623	1.122506
0	(eggs)	(mineral water)	0.179733	0.238267	0.050933	0.283383	1.189351	0.008109	1.062957
3	(mineral water)	(spaghetti)	0.238267	0.174133	0.059733	0.250699	1.439698	0.018243	1.102184
5	(mineral water)	(chocolate)	0.238267	0.163867	0.052667	0.221041	1.348907	0.013623	1.073398
1	(mineral water)	(eggs)	0.238267	0.179733	0.050933	0.213766	1.189351	0.008109	1.043286

5 Discussion & Conclusion

Based on the focused objective(s) to understand about simple frequent itemset mining using FP-Growth algorithm, the additional lab exercise will make the students more confident towards the fulfilment of the objectives(s).

6 Lab Task (Please implement yourself and show the output to the instructor)

Implement the FP-Growth algorithm using various other support thresholds in four other datasets and record. Data Link

7 Lab Exercise (Submit as a report)

Construct a lab report with comparative analysis of time complexity only using line-charts between Apriori algorithm and FP-growth algorithm using a same dataset and same minimum support threshold. use four other minimum support thresholds for a single dataset. Use four separate real-world given datasets. Your analysis and detailed discussions on the comparative report should be included.

8 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.