# Data Mining Lab : Experiment 7

### Submitted By

NAME: Debatreya Das

Roll-No.: 12212070

CS-A4

Data Mining Lab 7

## Objective:

### To compute maximal frequent itemset.

## Part A

Compute item set of frequent items (1-itemsets) and their support counts from a given transactional dataset. Sort frequent itemsets and generate them in L order (descending order of support counts).

#### Importing Libraries

## IMPORTING LIBRARIES  
import pandas as pd  
from collections import defaultdict

#### Loading Transaction Data

# Load the CSV file containing transactions  
file\_path = 'fp\_growth\_transactions.csv' # Update with the correct path  
df\_transactions = pd.read\_csv(file\_path)  
df\_transactions

TID Items  
0 T1 bread,milk  
1 T2 bread,diaper,beer,eggs  
2 T3 milk,diaper,beer,cola  
3 T4 bread,milk,diaper,beer  
4 T5 bread,milk,cola

#### Function to compute support of 1-itemsets compute\_1\_itemsets

# Function to compute support counts of 1-itemsets  
def compute\_1\_itemsets(transactions):  
 item\_support\_count = defaultdict(int)  
 for transaction in transactions:  
 items = transaction.split(',')  
 for item in items:  
 item\_support\_count[item] += 1  
 return dict(item\_support\_count)

#### Calculating frequent 1-itemsets

# Extract transactions from the CSV data  
transaction\_list = df\_transactions['Items'].tolist()  
  
# Compute 1-itemsets with support counts  
item\_support\_counts = compute\_1\_itemsets(transaction\_list)  
  
# Sorting the 1-itemsets by support counts in descending order  
sorted\_item\_support\_counts = sorted(item\_support\_counts.items(), key=lambda x: x[1], reverse=True)  
  
# Display the sorted frequent 1-itemsets  
sorted\_item\_support\_counts

[('bread', 4),  
 ('milk', 4),  
 ('diaper', 3),  
 ('beer', 3),  
 ('cola', 2),  
 ('eggs', 1)]

## Part B

Sort items in the transactions of the dataset in L-order (descending order of support counts).

#### Function to sort items in each transaction by L-order

# Helper function to sort items in each transaction by L-order (support counts)  
def sort\_transactions\_by\_l\_order(transactions, support\_counts):  
 sorted\_transactions = []  
 for transaction in transactions:  
 items = transaction.split(',')  
 # Sort items in transaction based on the L-order (support counts)  
 sorted\_items = sorted(items, key=lambda item: support\_counts[item], reverse=True)  
 sorted\_transactions.append(sorted\_items)  
 return sorted\_transactions

#### Calculating sorted transaction by L order

# Sorting transactions by L-order  
sorted\_transactions = sort\_transactions\_by\_l\_order(transaction\_list, dict(sorted\_item\_support\_counts))  
  
# Display the sorted transactions  
sorted\_transactions

[['bread', 'milk'],  
 ['bread', 'diaper', 'beer', 'eggs'],  
 ['milk', 'diaper', 'beer', 'cola'],  
 ['bread', 'milk', 'diaper', 'beer'],  
 ['bread', 'milk', 'cola']]

## Part C

Construct FP tree using the Han’s book example. Display FP tree using appropriate notation/representation.

#### Class: FPTreeNode

# FP-Tree Node structure  
class FPTreeNode:  
 def \_\_init\_\_(self, item\_name, count, parent):  
 self.item\_name = item\_name  
 self.count = count  
 self.parent = parent  
 self.children = {}  
 self.link = None # Link to next node of the same item  
  
 def increment(self, count):  
 """Increment the count of the node."""  
 self.count += count  
  
# FP-Tree structure  
class FPTree:  
 def \_\_init\_\_(self):  
 self.root = FPTreeNode(None, 1, None) # Root node with no item  
 self.header\_table = {}  
  
 def update\_header\_table(self, node, item):  
 """Update header table to point to nodes of the same item."""  
 if item in self.header\_table:  
 current\_node = self.header\_table[item]  
 while current\_node.link is not None:  
 current\_node = current\_node.link  
 current\_node.link = node  
 else:  
 self.header\_table[item] = node  
  
 def insert\_transaction(self, transaction):  
 """Insert a sorted transaction into the FP-Tree."""  
 current\_node = self.root  
 for item in transaction:  
 if item in current\_node.children:  
 current\_node.children[item].increment(1)  
 else:  
 new\_node = FPTreeNode(item, 1, current\_node)  
 current\_node.children[item] = new\_node  
 self.update\_header\_table(new\_node, item)  
 current\_node = current\_node.children[item]

#### Function to construct the FP-Tree from sorted transactions

# Construct the FP-Tree from sorted transactions  
def construct\_fp\_tree(transactions):  
 tree = FPTree()  
 for transaction in transactions:  
 tree.insert\_transaction(transaction)  
 return tree

#### Build the FP Growth Tree

# Build the FP-Tree from sorted transactions  
fp\_tree = construct\_fp\_tree(sorted\_transactions)

#### Printing the tree

def print\_fp\_tree(node, indent=0):  
 print(' ' \* indent + f'{node.item\_name}: {node.count}')  
 for child in node.children.values():  
 print\_fp\_tree(child, indent + 1)  
  
# Print the FP-Tree structure  
print\_fp\_tree(fp\_tree.root)

None: 1  
 bread: 4  
 milk: 3  
 diaper: 1  
 beer: 1  
 cola: 1  
 diaper: 1  
 beer: 1  
 eggs: 1  
 milk: 1  
 diaper: 1  
 beer: 1  
 cola: 1

## Part D

Using FP tree, construct pattern bases and conditional FP trees.

# Function to extract the conditional pattern base for an item  
def find\_prefix\_paths(base\_item, header\_table):  
 cond\_pattern\_base = []  
 node = header\_table[base\_item]  
 while node is not None:  
 prefix\_path = []  
 current\_node = node  
 while current\_node.parent.item\_name is not None:  
 prefix\_path.append(current\_node.parent.item\_name)  
 current\_node = current\_node.parent  
 if len(prefix\_path) > 0:  
 cond\_pattern\_base.append((prefix\_path, node.count))  
 node = node.link  
 return cond\_pattern\_base

#### Finding Pattern Bases for all Items

Items = ["bread", "milk", "diaper", "beer", "eggs", "cola"]  
  
PatternBases = {  
 "bread": find\_prefix\_paths("bread", fp\_tree.header\_table),  
 "milk": find\_prefix\_paths("milk", fp\_tree.header\_table),  
 "diaper": find\_prefix\_paths("diaper", fp\_tree.header\_table),  
 "beer": find\_prefix\_paths("beer", fp\_tree.header\_table),  
 "eggs": find\_prefix\_paths("eggs", fp\_tree.header\_table),  
 "cola": find\_prefix\_paths("cola", fp\_tree.header\_table)  
}  
  
for i in Items:  
 print(f"PatternBases for {i}: ", PatternBases[i])

PatternBases for bread: []  
PatternBases for milk: [(['bread'], 3)]  
PatternBases for diaper: [(['bread'], 1), (['milk'], 1), (['milk', 'bread'], 1)]  
PatternBases for beer: [(['diaper', 'bread'], 1), (['diaper', 'milk'], 1), (['diaper', 'milk', 'bread'], 1)]  
PatternBases for eggs: [(['beer', 'diaper', 'bread'], 1)]  
PatternBases for cola: [(['beer', 'diaper', 'milk'], 1), (['milk', 'bread'], 1)]

## Part E

Generate frequent patterns.

#### Helper Functions

from collections import defaultdict  
  
# Function to filter items by support threshold  
def filter\_items\_by\_support(transactions, min\_support):  
 item\_count = defaultdict(int)  
 for transaction in transactions:  
 for item in transaction:  
 item\_count[item] += 1  
  
 # Filter out items that don't meet the minimum support  
 filtered\_items = {item for item, count in item\_count.items() if count >= min\_support}  
 return filtered\_items, item\_count

#### Function to generate the conditional fp-trees

# Function to create a conditional FP-Tree  
def construct\_conditional\_fp\_tree(base\_item, cond\_pattern\_base, min\_support):  
 tree = FPTree()  
 for prefix\_path, count in cond\_pattern\_base:  
 # Filter out items that don't meet min support in prefix path  
 filtered\_prefix\_path = [item for item in prefix\_path if item in tree.header\_table]  
 filtered\_prefix\_path.sort(key=lambda item: tree.header\_table[item].count, reverse=True)  
 for \_ in range(count):  
 tree.insert\_transaction(filtered\_prefix\_path)  
 return tree  
  
# Function to mine frequent patterns using conditional FP-trees  
def mine\_fp\_tree(tree, min\_support, prefix, frequent\_patterns):  
 # Sort items in the header table by frequency  
 sorted\_items = sorted(tree.header\_table.items(), key=lambda x: x[1].count)  
  
 for base\_item, node in sorted\_items:  
 new\_prefix = prefix.copy()  
   
 # Ensure no duplicate items in the new prefix  
 if base\_item not in new\_prefix:  
 new\_prefix.append(base\_item)  
  
 # Calculate total support for the current pattern  
 total\_support = 0  
 current\_node = node  
 while current\_node is not None:  
 total\_support += current\_node.count  
 current\_node = current\_node.link  
  
 # Add the frequent pattern (prefix + item) to the result  
 if total\_support >= min\_support:  
 frequent\_patterns.append((new\_prefix, total\_support))  
  
 # Find conditional pattern base  
 cond\_pattern\_base = find\_prefix\_paths(base\_item, tree.header\_table)  
  
 # Construct conditional FP-Tree for current item  
 cond\_tree = FPTree()  
 for path, count in cond\_pattern\_base:  
 cond\_tree.insert\_transaction(path \* count) # Insert with counts  
  
 # Recursively mine the conditional FP-tree  
 if cond\_tree.root.children:  
 mine\_fp\_tree(cond\_tree, min\_support, new\_prefix, frequent\_patterns)

#### Generate Frequent Patterns

# Main driver function  
def generate\_frequent\_patterns(transactions, min\_support):  
 # Step 1: Filter items by min\_support  
 filtered\_items, item\_count = filter\_items\_by\_support(transactions, min\_support)  
  
 # Step 2: Filtered transactions  
 filtered\_transactions = [[item for item in transaction if item in filtered\_items]  
 for transaction in transactions]  
  
 # Step 3: Construct the initial FP-Tree  
 tree = construct\_fp\_tree(filtered\_transactions)  
  
 # Step 4: Recursively mine the FP-Tree  
 frequent\_patterns = []  
 mine\_fp\_tree(tree, min\_support, [], frequent\_patterns)  
  
 return frequent\_patterns

# DRIVER CODE  
transactions = [items.split(',') for items in df\_transactions['Items']]  
min\_support = 2  
frequent\_patterns = generate\_frequent\_patterns(transactions, min\_support)  
  
  
# Display all frequent patterns  
for pattern, count in frequent\_patterns:  
 print(f"Pattern: {pattern}, Support: {count}")

Pattern: ['diaper'], Support: 3  
Pattern: ['diaper', 'bread'], Support: 2  
Pattern: ['diaper', 'milk'], Support: 2  
Pattern: ['beer'], Support: 3  
Pattern: ['beer', 'bread'], Support: 2  
Pattern: ['beer', 'bread', 'diaper'], Support: 2  
Pattern: ['beer', 'milk'], Support: 2  
Pattern: ['beer', 'milk', 'diaper'], Support: 2  
Pattern: ['beer', 'diaper'], Support: 3  
Pattern: ['cola'], Support: 2  
Pattern: ['cola', 'milk'], Support: 2  
Pattern: ['milk'], Support: 4  
Pattern: ['milk', 'bread'], Support: 3  
Pattern: ['bread'], Support: 4