AY: 2025-26

Class:	**TE**	**Semester:**	**V**
Course Code:	**CSC504**	**Course Name:**	**Data Warehousing and Mining**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**51**
Experiment No.:	**09**
Title of the Experiment:	**Implementation of association mining algorithms like FP Growth using languages like JAVA/ python.**
Date of Performance:	
Date of Submission:	

Evaluation

Performance Indicator	**Max. Marks**	**Marks Obtained**
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	**Exceed Expectations (EE)**	**Meet Expectations (ME)**	**Meet Expect Below Expectations (BE)**
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Ms. Neha Raut
Signature :
Date:

Aim :-To implement the FP-Growth algorithm using Python.

Objective: Understand the working principles of the FP-Growth algorithm and implement it in Python.

Theory

FP-Growth (Frequent Pattern Growth) is an algorithm for frequent item set mining and association rule learning over transactional databases. It efficiently discovers frequent patterns by constructing a compact data structure called the FP-Tree and mining it to extract frequent item sets.

Key Concepts:

1. FP-Tree: A data structure that represents the transaction database compressed by linking frequent items in a tree structure, along with their support counts.
2. Header Table: A compact structure that stores pointers to the first occurrences of items in the FP-Tree and their support counts.
3. Frequent Item Set Mining:
    - Conditional Pattern Base: For each frequent item, construct a conditional pattern base consisting of the prefix paths in the FP-Tree.
    - Conditional FP-Tree: Construct a conditional FP-Tree from the conditional pattern base and recursively mine frequent item sets.

Steps in FP-Growth Algorithm:

1. Build FP-Tree: Construct the FP-Tree by inserting transactions and counting support for each item.
2. Create Header Table: Build a header table with links to the first occurrences of items in the FP-Tree.
3. Mine FP-Tree:
    - Identify frequent single items by their support.
    - Construct conditional pattern bases and conditional FP-Trees recursively.
    - Combine frequent item sets from conditional FP-Trees to find all frequent item sets.

Example

Given a transactional database:

```
import pandas as pd

from mlxtend.frequent_patterns import fpgrowth, association_rules

# Sample transactional dataset

dataset = [

    ['milk', 'bread', 'eggs'],

    ['milk', 'bread'],

    ['milk', 'eggs'],

    ['bread', 'butter'],
```

```
    ['bread', 'eggs'],
    ['milk', 'bread', 'butter'],
]

# Convert dataset to one-hot encoded DataFrame
from mlxtend.preprocessing import TransactionEncoder
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Apply FP-Growth
frequent_itemsets = fpgrowth(df, min_support=0.3, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
print("Frequent Itemsets:")
print(frequent_itemsets)
print("\nAssociation Rules:")
print(rules[['antecedents','consequents','support','confidence','lift']]
```

Output:

```
Frequent Itemsets:
    support         itemsets
0  0.833333          (bread)
1  0.666667           (milk)
2  0.500000          (eggs)
3  0.333333         (butter)
4  0.500000    (bread, milk)
5  0.333333     (eggs, milk)
6  0.333333    (bread, eggs)
7  0.333333  (butter, bread)

Association Rules:
  antecedents consequents   support  confidence  lift
0      (eggs)      (milk)  0.333333    0.666667   1.0
1      (milk)      (eggs)  0.333333    0.500000   1.0
2    (butter)     (bread)  0.333333    1.000000   1.2
3     (bread)    (butter)  0.333333    0.400000   1.2
PS C:\Users\student\Desktop\New folder>
```

Conclusion

Explain how FP-Growth manages and mines item sets of varying lengths in transactional databases.

The FP-Growth algorithm efficiently mines frequent itemsets by compressing the database into an FP-Tree and recursively exploring conditional sub-trees. Unlike Apriori, it does not generate a large number of candidate sets, which makes it faster and more memory-efficient. FP-Growth is capable of handling itemsets of varying lengths by exploring all frequent items through conditional pattern bases and conditional FP-Trees, ensuring that both short and long frequent patterns are discovered effectively.