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

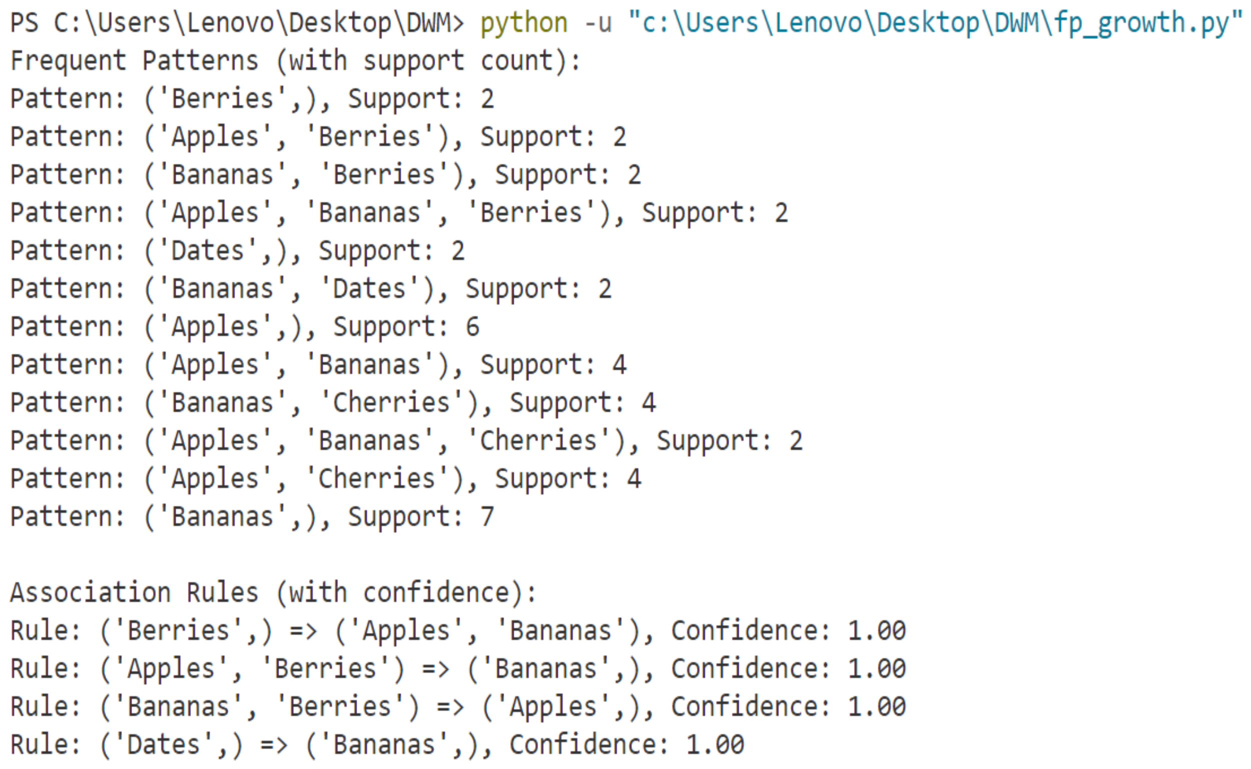
Given a transactional database:

Implement the FP-Growth algorithm to find all frequent itemsets with a specified minimum support threshold.

Code:-

```
import pyfpgrowth

transactions = [

    ['Apples', 'Bananas', 'Berries'],

    ['Bananas', 'Dates'],

    ['Bananas', 'Cherries'],

    ['Apples', 'Bananas', 'Dates'],

    ['Apples', 'Cherries'],

    ['Bananas', 'Cherries'],

    ['Apples', 'Cherries'],

    ['Apples', 'Bananas', 'Cherries', 'Berries'],

    ['Apples', 'Bananas', 'Cherries']

]

patterns = pyfpgrowth.find_frequent_patterns(transactions, 2)

print("Frequent Patterns (with support count):")

for pattern, support in patterns.items():

    print(f"Pattern: {pattern}, Support: {support}")


rules = pyfpgrowth.generate_association_rules(patterns, 0.7)


print("\nAssociation Rules (with confidence):")

for rule, (outcome, confidence) in rules.items():

    print(f"Rule: {rule} => {outcome}, Confidence: {confidence:.2f}")
```

Output:



```
PS C:\Users\Lenovo\Desktop\DWM> python -u "c:\Users\Lenovo\Desktop\DWM\fp_growth.py"
Frequent Patterns (with support count):
Pattern: ('Berries',), Support: 2
Pattern: ('Apples', 'Berries'), Support: 2
Pattern: ('Bananas', 'Berries'), Support: 2
Pattern: ('Apples', 'Bananas', 'Berries'), Support: 2
Pattern: ('Dates',), Support: 2
Pattern: ('Bananas', 'Dates'), Support: 2
Pattern: ('Apples',), Support: 6
Pattern: ('Apples', 'Bananas'), Support: 4
Pattern: ('Bananas', 'Cherries'), Support: 4
Pattern: ('Apples', 'Bananas', 'Cherries'), Support: 2
Pattern: ('Apples', 'Cherries'), Support: 4
Pattern: ('Bananas',), Support: 7

Association Rules (with confidence):
Rule: ('Berries',) => ('Apples', 'Bananas'), Confidence: 1.00
Rule: ('Apples', 'Berries') => ('Bananas',), Confidence: 1.00
Rule: ('Bananas', 'Berries') => ('Apples',), Confidence: 1.00
Rule: ('Dates',) => ('Bananas',), Confidence: 1.00
```

Conclusion: In this practical implementation of the FP-Growth algorithm, we learned how it effectively finds frequent itemsets in transactional databases. By using an FP-Tree structure, the algorithm simplifies the data, making it easier to analyze without needing to check every possible combination of items.

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

FP-Growth is an efficient algorithm for finding frequently occurring item sets in transactional data. It uses a special structure called an **FP-Tree** to manage and mine itemsets of different lengths.

FP-Tree: A compressed tree structure that stores transactions, linking frequent items together. Each node represents an item and its support count (how many transactions include that item).

Building the FP-Tree: First, the algorithm scans the transactions to count item frequencies. It creates the FP-Tree by inserting transactions, sorting items by their frequency, and linking nodes when possible.

Mining Item Sets: For each frequent item, it builds a **conditional pattern base**, which collects all paths leading to that item in the FP-Tree. A new conditional FP-Tree is created from this base, containing only the transactions relevant to the current item. This process is repeated recursively for each item, allowing the algorithm to discover frequent item sets of varying lengths (1-itemsets, 2-itemsets, etc.).

Benefits: FP-Growth avoids generating all possible item combinations, making it faster than other methods. It can find item sets of different lengths by recursively mining the FP-Tree.