AP21110011026 Ankush Kumar CSE - P

Implement FP Growth using the transactional database of 10,000 transactions or more

```
import pandas as pd
import numpy as np

# Generate a larger dataset with 10,000 transactions
np.random.seed(0)
transaction_data = {
    "TId": [f'T{i}' for i in range(1, 10001)],
    "Item_Ids": [np.random.choice(['I1', 'I2', 'I3', 'I4', 'I5'],
np.random.randint(1, 6), replace=False).tolist() for _ in
range(10000)]
}
df = pd.DataFrame(transaction_data)
df
```

```
         TId               Item_Ids
0         T1  [I5, I3, I2, I4, I1]
1         T2          [I2, I1, I3]
2         T3                [I4]
3         T4                [I1]
4         T5                [I3]
...      ...                 ...
9995   T9996          [I1, I2, I5]
9996   T9997                [I3]
9997   T9998              [I2, I5]
9998   T9999  [I2, I1, I4, I3, I5]
9999  T10000      [I1, I3, I4, I5]

[10000 rows x 2 columns]
```

```
msp = 2
flattened_list = [item for sublist in df['Item_Ids'].values.tolist()
for item in sublist]

itemset = pd.DataFrame(columns=['Itemset', 'Count'])

for item in set(flattened_list):
    itemset.loc[len(itemset)] = [item, flattened_list.count(item)]

itemset = itemset[itemset['Count'] >= msp]

itemset = itemset.sort_values(by='Count', ascending=False)

print(itemset)
```

```
  Itemset  Count
4      I5   6014
0      I2   5993
2      I1   5991
3      I3   5951
1      I4   5931
```

```
# Flatten the 'Item_Ids' column
flattened_list = [item for sublist in df['Item_Ids'].values.tolist()
for item in sublist]

# Create a DataFrame to store item counts
itemset = pd.DataFrame(columns=['Itemset', 'Count'])

# Calculate item counts
for item in set(flattened_list):
    itemset.loc[len(itemset)] = [item, flattened_list.count(item)]

# Filter itemset for items with counts >= msp
itemset = itemset[itemset['Count'] >= msp]

# Sort itemset by Count in descending order
itemset = itemset.sort_values(by='Count', ascending=False)

# Sort 'Item_Ids' within each row based on itemset order
df['Item_Ids'] = df['Item_Ids'].apply(lambda x: sorted(x, key=lambda
item: itemset[itemset['Itemset'] == item]['Count'].values[0],
reverse=True))

# Display the sorted DataFrame
df
```

```
         TId               Item_Ids
0         T1  [I5, I2, I1, I3, I4]
1         T2          [I2, I1, I3]
2         T3                  [I4]
3         T4                  [I1]
4         T5                  [I3]
...      ...                   ...
9995   T9996          [I5, I2, I1]
9996   T9997                  [I3]
9997   T9998              [I5, I2]
9998   T9999  [I5, I2, I1, I3, I4]
9999  T10000      [I5, I1, I3, I4]

[10000 rows x 2 columns]
```

```
import pyfpgrowth
from matplotlib import pyplot as plt

# Convert the dataset to the required format (list of transactions)
```

```
transactions = df['Item_Ids'].tolist()

min_support = 2

# Find frequent itemsets using FP-Growth
patterns = pyfpgrowth.find_frequent_patterns(transactions,
min_support)

frequent_itemsets_list = []

# Append frequent itemsets to the list
for itemset, support in patterns.items():
    if len(itemset) > 1:
        frequent_itemsets_list.append({"Itemset": itemset, "Support":
support})

frequent_itemsets_df = pd.DataFrame(frequent_itemsets_list)

frequent_itemsets_df
```

	Itemset	Support
0	(I2, I3, I4, I5)	2370
1	(I1, I2, I3, I4, I5)	1993
2	(I1, I3, I4, I5)	2400
3	(I1, I3, I4)	2990
4	(I1, I2, I3, I4)	2400
5	(I2, I3, I4)	2991
6	(I2, I4, I5)	2986
7	(I1, I2, I4, I5)	2420
8	(I1, I4, I5)	3014
9	(I2, I4)	3968
10	(I1, I2, I4)	2990
11	(I1, I4)	3981
12	(I1, I2, I3)	2986
13	(I1, I2, I3, I5)	2402
14	(I1, I3, I5)	2996
15	(I3, I5)	3982
16	(I2, I3, I5)	2986
17	(I2, I3)	3991
18	(I1, I2)	3989
19	(I1, I2, I5)	3044
20	(I1, I5)	4023
21	(I2, I5)	4013

```
import networkx as nx
import matplotlib.pyplot as plt

# Create an empty graph
G = nx.Graph()
```

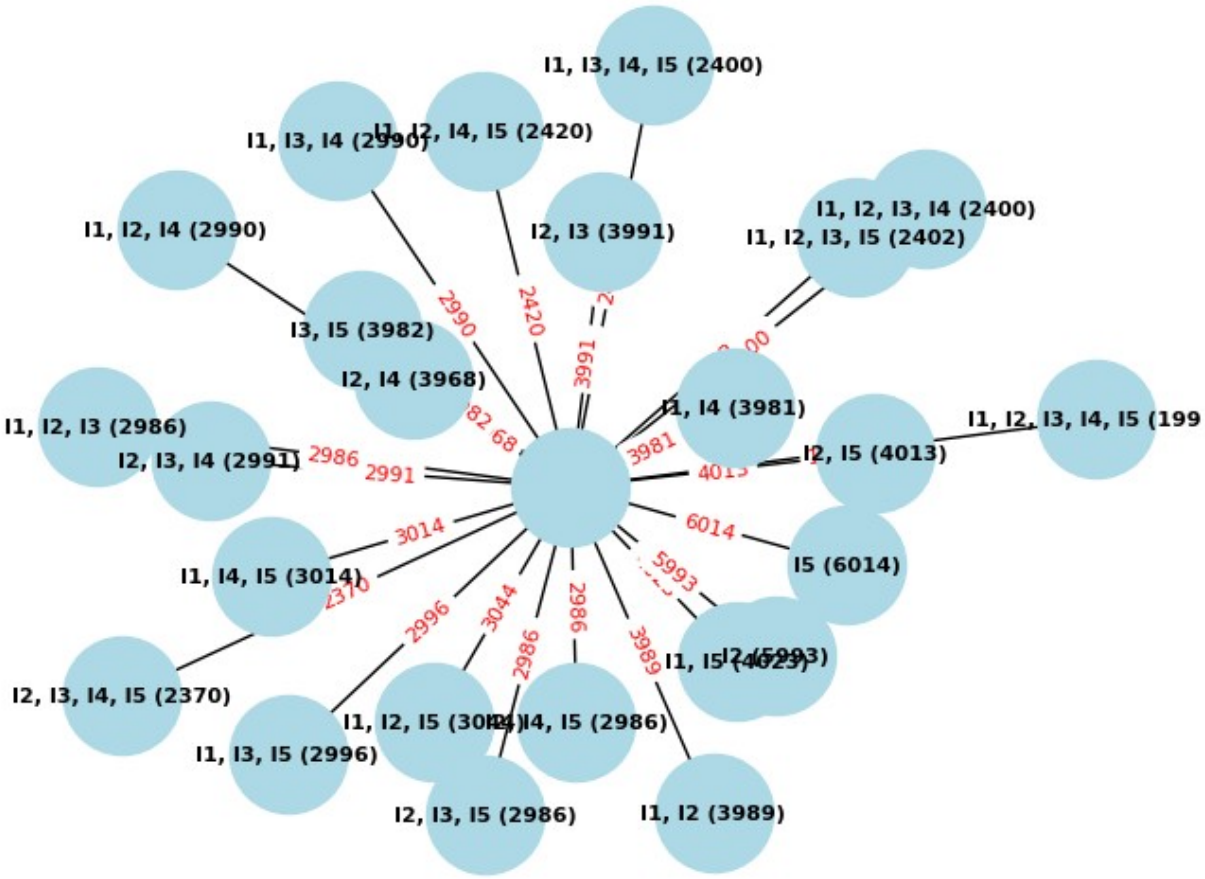
```
# Define a recursive function to build the FP-Growth tree
def build_fpgrowth_tree(graph, node, prefix, support):
    for item, count in node.items():
        if isinstance(item, tuple):
            item_str = ', '.join(item)
        else:
            item_str = item
        child_prefix = prefix + [item_str]
        child_support = support + [count]
        graph.add_node('-'.join(child_prefix), label=f'{item_str}
({count})')
        graph.add_edge('-'.join(prefix), '-'.join(child_prefix),
weight=count)
        if isinstance(node[item], dict):
            build_fpgrowth_tree(graph, node[item], child_prefix,
child_support)

# Call the function to build the tree
build_fpgrowth_tree(G, patterns, [], [])

# Draw the tree using a suitable layout
pos = nx.spring_layout(G)
labels = nx.get_node_attributes(G, 'label')
edge_labels = nx.get_edge_attributes(G, 'weight')

# Draw nodes and edges
nx.draw(G, pos, with_labels=True, labels=labels, node_size=2000,
node_color='lightblue', font_size=8, font_weight='bold')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
font_size=8, font_color='red')

# Show the tree
plt.show()
```


I1, I3, I4, I5 (2400)
I1, I3, I4 (2990)
I1, I2, I4, I5 (2420)
I1, I2, I4 (2990)
I2, I3 (3991)
I1, I2, I3, I4 (2400)
I1, I2, I3, I5 (2402)
I3, I5 (3982)
2990
2420
3991
I2, I4 (3968)
I1, I4 (3981)
I1, I2, I3 (2986)
I2, I3, I4 (2991)
2986
2991
3981
I2, I5 (4013)
I1, I2, I3, I4, I5 (199
3014
6014
I1, I4, I5 (3014)
I5 (6014)
5993
2996
3044
2986
2986
3989
I1, I5 (4023)
I2, I3, I4, I5 (2370)
I1, I2, I5 (3044)
I4, I5 (2986)
I1, I3, I5 (2996)
I2, I3, I5 (2986)
I1, I2 (3989)