Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

EXPERIMENT NO. 7
PCY

NAME: Rishabh Patil
SAP: 60009200056
BATCH: D12

AIM: Implement program for PCY.

THEORY:

The PCY (Park, Chen, and Yu) algorithm is a classic method used in data mining for finding frequent item sets in large datasets. It's an improvement over the Apriori algorithm and is particularly efficient when dealing with large databases.

Here's a breakdown of how the PCY algorithm works:

1. Hashing: The first step involves hashing. PCY uses a hash function to generate hash buckets. Items from the dataset are hashed into these buckets. This step is crucial for speeding up the counting process of item occurrences.

2. First Pass: The algorithm goes through the dataset once to count the occurrences of individual items. It uses a bitmap or an array of counters associated with hash buckets to count the number of times an item appears in the dataset. This step helps identify potentially frequent items.

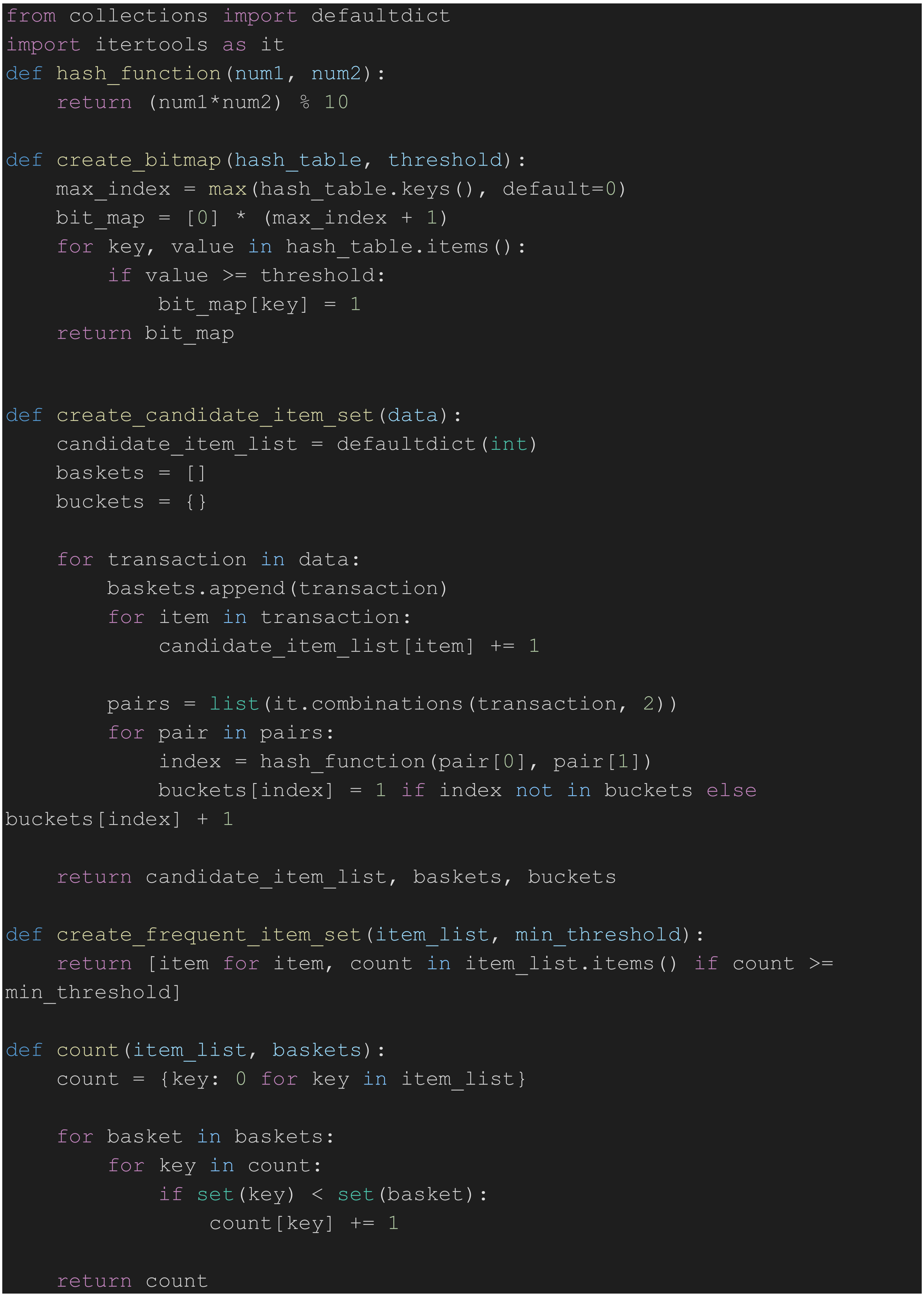
3. Second Pass: In this pass, PCY counts pairs of items that hash into the same bucket that was generated in the first pass. It uses a hash table to count the occurrence of pairs of items that hash to the same bucket. This is more efficient than the Apriori algorithm because it reduces the number of candidate pairs that need to be checked for frequency.

4. Finding Frequent Item Sets: After counting the pairs, the algorithm applies a threshold to determine which pairs occur frequently enough to be considered as frequent item sets. These frequent item sets are then outputted as the final result.

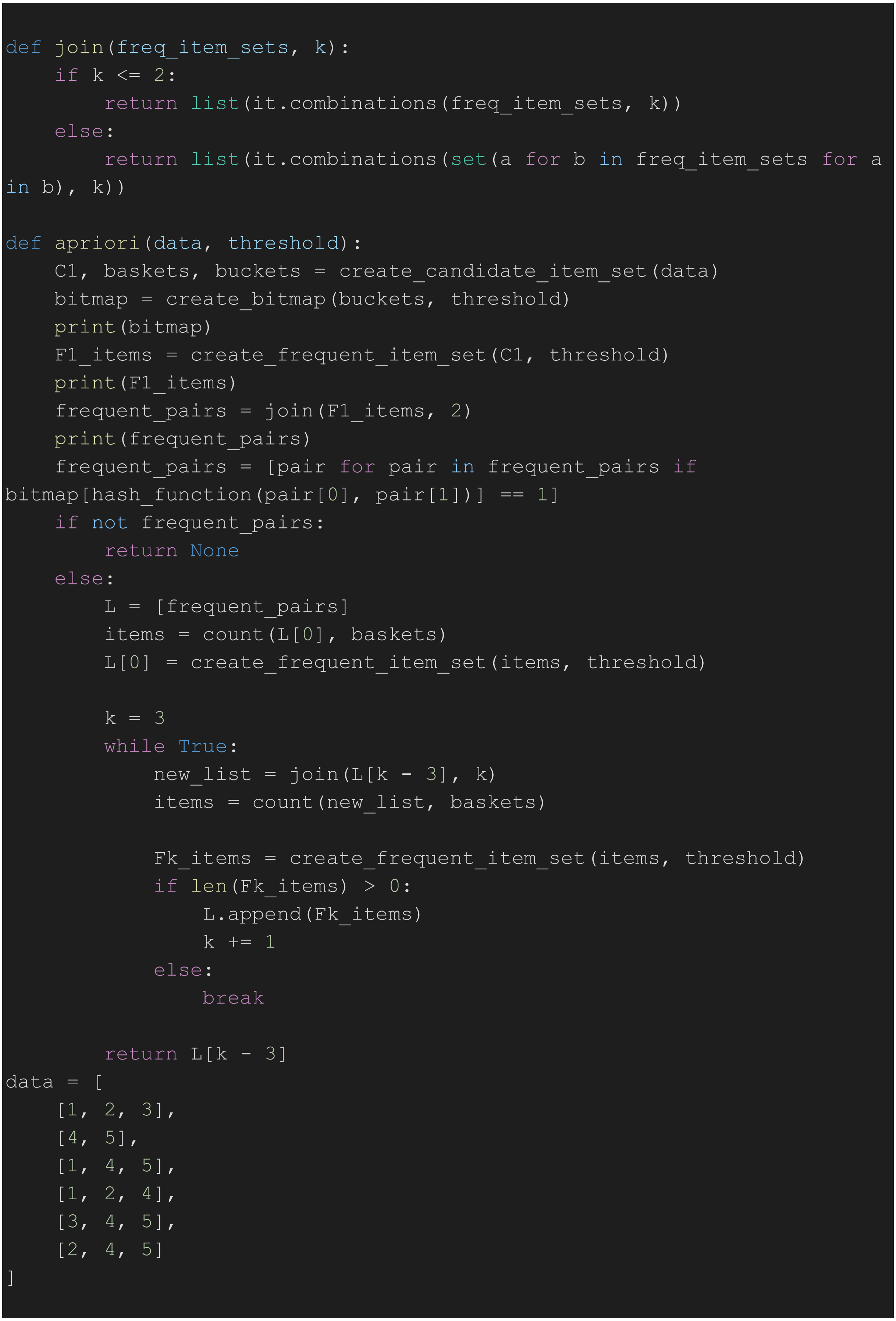
The key idea behind the PCY algorithm is its utilization of hashing to reduce the number of item pairs that need to be counted. By hashing items into buckets and focusing only on pairs that hash into the same bucket, it decreases the overall computational load compared to a straightforward counting of all possible pairs.

This approach improves the efficiency of finding frequent item sets, making it suitable for handling large datasets efficiently. However, it's important to note that while PCY is effective, there might be trade-offs between memory usage, hash collisions, and the accuracy of the results based on the chosen hash functions and threshold values. Adjusting these parameters appropriately is crucial for optimal performance in different scenarios.

CODE WITH OUTPUT:



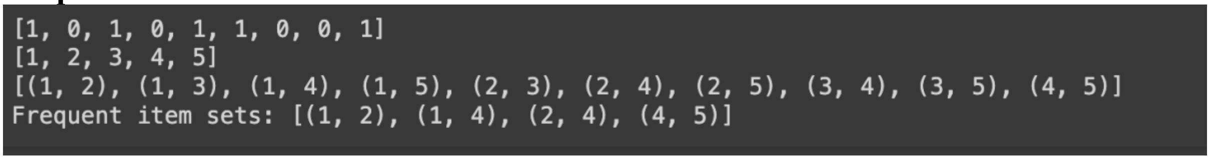
```
from collections import defaultdict
import itertools as it
def hash_function(num1, num2):
    return (num1*num2) % 10

def create_bitmap(hash_table, threshold):
    max_index = max(hash_table.keys(), default=0)
    bit_map = [0] * (max_index + 1)
    for key, value in hash_table.items():
        if value >= threshold:
            bit_map[key] = 1
    return bit_map


def create_candidate_item_set(data):
    candidate_item_list = defaultdict(int)
    baskets = []
    buckets = {}

    for transaction in data:
        baskets.append(transaction)
        for item in transaction:
            candidate_item_list[item] += 1

        pairs = list(it.combinations(transaction, 2))
        for pair in pairs:
            index = hash_function(pair[0], pair[1])
            buckets[index] = 1 if index not in buckets else
buckets[index] + 1

    return candidate_item_list, baskets, buckets

def create_frequent_item_set(item_list, min_threshold):
    return [item for item, count in item_list.items() if count >=
min_threshold]

def count(item_list, baskets):
    count = {key: 0 for key in item_list}

    for basket in baskets:
        for key in count:
            if set(key) < set(basket):
                count[key] += 1

    return count
```



```
def join(freq_item_sets, k):
    if k <= 2:
        return list(it.combinations(freq_item_sets, k))
    else:
        return list(it.combinations(set(a for b in freq_item_sets for a
in b), k))

def apriori(data, threshold):
    C1, baskets, buckets = create_candidate_item_set(data)
    bitmap = create_bitmap(buckets, threshold)
    print(bitmap)
    F1_items = create_frequent_item_set(C1, threshold)
    print(F1_items)
    frequent_pairs = join(F1_items, 2)
    print(frequent_pairs)
    frequent_pairs = [pair for pair in frequent_pairs if
bitmap[hash_function(pair[0], pair[1])] == 1]
    if not frequent_pairs:
        return None
    else:
        L = [frequent_pairs]
        items = count(L[0], baskets)
        L[0] = create_frequent_item_set(items, threshold)

        k = 3
        while True:
            new_list = join(L[k - 3], k)
            items = count(new_list, baskets)

            Fk_items = create_frequent_item_set(items, threshold)
            if len(Fk_items) > 0:
                L.append(Fk_items)
                k += 1
            else:
                break

        return L[k - 3]
data = [
    [1, 2, 3],
    [4, 5],
    [1, 4, 5],
    [1, 2, 4],
    [3, 4, 5],
    [2, 4, 5]
]
```

```
min_support_threshold = 2

result = apriori(data, min_support_threshold)
print("Frequent item sets:", result)
```

Output:



```
[1, 0, 1, 0, 1, 1, 0, 0, 1]
[1, 2, 3, 4, 5]
[(1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (3, 5), (4, 5)]
Frequent item sets: [(1, 2), (1, 4), (2, 4), (4, 5)]
```

CONCLUSION:
We have successfully implemented program for PCY.