Assignment 2: Frequent Itemsets

In this assignment you will be implementing 3 different algorithms to find frequent itemsets namely PCY, Multi-Hash, Toivonen. The transactions will be given as an input file. The goal of the assignment is to make you understand the algorithms better by coding them.

Write your own code!

For this assignment to be an effective learning experience, you must write your own code! I emphasize this point because you will be able to find Python implementations of most or perhaps even all of the required functions on the web. Please do not look for or at any such code! **Do not share code with other students in the class!!**

Here's why:

- The most obvious reason is that it will be a huge temptation to cheat: if you include code written by anyone else in your solution to the assignment, you will be cheating. As mentioned in the syllabus, this is a very serious offense, and may lead to you failing the class.
- However, even if you do not directly include any code you look at in your solution, it surely will influence your coding. Put another way, it will short-circuit the process of you figuring out how to solve the problem, and will thus decrease how much you learn.

So, just don't look on the web for any code relevant to these problems. Don't do it.

Submission Details

For each problem, you will turn in a python programs. To allow you to test your programs, sample data will be provided in the data folder, and the corresponding solutions will be provided for each problem in the solutions folder.

You need to turn in the following python files.

1. Program that implements PCY algorithm. : <firstname>_<lastname>_pcy.py
2. Program that implements Multi-Hash : <firstname>_<lastname>_multihash.py
3. Program that implements Toivonen: <firstname>_<lastname>_toivonen.py

Problem 1: PCY Algorithm

Implement PCY algorithm using a single hash and print all frequent itemsets. You can use a hashing function of your choice.

Input Parameters:

1. Input.txt: This is the input file containing all transactions. Each line corresponds to a transaction. Each transaction has items that are comma separated. Use input.txt to test this algorithm.
2. Support: Integer that defines the minimum count to qualify as a frequent itemset.
3. Bucket size: This is the size of the hash table.

Output:

The output needs to contain the frequent itemsets of all sizes sorted lexicographically. It should also contain the hash buckets with their count of candidates. If the result just contains itemsets of size1 just print them and return. If it contains itemsets of size >= 2 print the bucket counts of the hash as well. For example consider the output below.

['a', 'b', 'd']

{0:0, 1:2, 3:5}
[['a', 'b']]

Here ['a', 'b', 'd'] represents itemsets of size 1 and {0:0, 1:2, 3:5} represents the hash counts before calculating frequent itemsets of size 2. [['a', 'b']] represents itemsets of size 2. Print all bucket counts only for ith frequent itemset where i >= 2. The counts in the buckets can vary depending on the hashing function used. So do not try to match this with the output files provided.

File Name: Please name your python script as <firstname>_<lastname>_pcy.py.

Executing code: python pooja_anand_pcy.py input.txt 4 20.

Where support = 4 and buckets = 20

(Please confirm the output with output_pcy.txt)

Problem 2: Multi-Hash Algorithm

Implement the multi-hash algorithm to generate frequent itemsets. You need to use 2 independent hashing functions for this. Make sure that all candidates are hashed to both the hashing functions to generate 2 different bit vectors. Both the hashes will have the same number of buckets.

Input parameters are same as above. For output follow the same format, but since we have two different hashing functions, print both the hash bucket counts. The counts in the buckets can vary depending on the hashing function used. So do not try to match this with the output files provided

For example:

['a', 'b', 'c']

{0:0, 1:2, 3:5}
{0:1, 1:4, 3:2}
[['a', 'b']]

File Name: Please name your python script as <firstname>_<lastname>_multihash.py.

Executing code: python pooja_anand_multihash.py input.txt 4 5

Where support = 4 and bucket size = 5

(Please confirm the output with output_multihash.txt)

Problem 3: Toivonen Algorithm

Implement the Toivonen algorithm to generate frequent itemsets. For this algorithm you need to use a sample size of less than 60% of your entire dataset. Use an appropriate sampling method to get the random sample set. Also perform a simple Apriori algorithm with the random sample set. Check for negative borders and run the algorithm again with a different sample set if required till there are no negative borders that have frequency > support.

Input Parameters:

1. Input.txt: This is the input file containing all transactions. Each line corresponds to a transaction. Each transaction has items that are comma separated. Use input1.txt to test this algorithm.
2. Support: Integer that defines the minimum count to qualify as a frequent itemset.

Output:

Line 1 <number of iterations performed>

Line 2 <fraction of transactions used>

Line 3 onwards <frequent itemsets lexicographically sorted>

File Name: Please name your python script as <firstname>_<lastname>_toivonen.py.

Executing code: **python pooja_anand_toivonen.py input1.txt 20**

Where support = 20

(Please confirm the output with output_toivonen.txt)

General Instructions:

1. Do not zip your files
2. Make sure your code compiles before submitting
3. Make sure to follow the output format and the naming format.
4. Make sure not to write the output to any files. Use standard output to print them.
5. We will be using Moss for plagiarism detection.