**Finding Frequent Item sets**

In this Project there is implementation of 3 different algorithms to find frequent item-sets namely Park-Chen-Yu (PCY), Multi-Stage hashing and Toivonen. The transactions will be given as an input file.

**Problem 1: PCY Algorithm and Execution**

Implementing PCY algorithm using a single hash and print all frequent item sets. Each letter(alphabet) is assigned a value. The hash function calculates the sum of the values for pairs and so on and takes the mod with respect to the bucket size.

**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.

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 contains the frequent itemsets generated in each pass sorted lexicographically. It contains information about the memory usage during each pass over the data along with the hash buckets and their counts (Assume each count takes 4 bytes).

**Note:** In PCY the counts for pairs are stored in form of Triples (e.g. count of pair (“a”,”b”) is stored as (“a”, ”b”, <count>) and it uses 12 bytes.). Similarly, frequent itemsets of size 3 will be stored in form of quadruples taking 16 bytes and so on. 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. Also, while finding frequent pairs you should hash all the possible pairs but in finding frequent itemsets of **size >= 3**, as you know the frequent itemsets of **size-1** you can use those and hash only those itemsets whose all possible subsets of **size-1** are frequent.

Two sample solutions are shown below.

**Sample 1:**

Executing code: **python nayak\_nandan\_pcy.py input2.txt 3 20**

*memory for item counts: 48*

*memory for hash table counts for size 2 itemsets: 80*

*{0: 0, 1: 2, 2: 0, 3: 2, 4: 2, 5: 3, 6: 2, 7: 1, 8: 1, 9: 0, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0}*

*frequent item sets of size 1 : [['e']]*

*PASS : 2*

*memory for frequent itemsets of size 1 : 8*

*bitmap size : 20*

*frequent item sets of size 2 : []*

Here for the 1st pass we show the memory used for the item counts, hash table counts, hash\_buckets with their counts and frequent itemsets found respectively. As there are no frequent itemsets of size 2, we do not output anything for subsequent passes.

**Sample 2:**

Executing code: **python nayak\_nandan\_pcy.py input.txt 5 20**

*PASS : 1*

*memory for item counts: 96*

*memory for hash table counts for size 2 itemsets: 80*

*{0: 2, 1: 4, 2: 5, 3: 14, 4: 8, 5: 17, 6: 11, 7: 14, 8: 12, 9: 9, 10: 9, 11: 12, 12: 9, 13: 9, 14: 6, 15: 4, 16: 4, 17:3, 18: 2, 19: 2}*

*frequent item sets of size 1 : [['a'], ['b'], ['c'], ['d'], ['e'], ['f']]*

*PASS : 2*

*memory for frequent itemsets of size 1 : 48*

*bitmap size : 20*

*memory for candidates counts of size 2 : 180*

*memory for hash table counts for size 3 itemsets: 80*

*{0: 12, 1: 11, 2: 11, 3: 10, 4: 6, 5: 4, 6: 10, 7: 13, 8: 14, 9: 11, 10: 12, 11: 11, 12: 14, 13: 14, 14: 17, 15: 17, 16: 16, 17: 15, 18: 15, 19: 12}*

*frequent item sets of size 2 : [['a', 'b'], ['a', 'c'], ['a', 'e'], ['b', 'c'], ['b', 'f'], ['c', 'e'], ['d', 'e'], ['e', 'f']]*

*PASS : 3*

*memory for frequent itemsets of size 2 : 96*

*bitmap size : 20*

*memory for candidates counts of size 3 : 32*

*memory for hash table counts for size 4 itemsets: 80*

*{0: 18, 1: 20, 2: 22, 3: 22, 4: 21, 5: 19, 6: 17, 7: 15, 8: 15, 9: 12, 10: 13, 11: 12, 12: 12, 13: 11, 14: 11, 15: 13, 16: 14, 17: 17, 18: 17, 19: 17}*

*frequent item sets of size 3 : [['a', 'b', 'c']]*

*PASS : 4*

*memory for frequent itemsets of size 3 : 16*

*bitmap size : 20*

*frequent item sets of size 4 : []*

**Problem 2: Multi-Stage Algorithm**

Implementingthe Multi-Stage algorithm to generate frequent itemsets. There are **two** stages using two different hashing functions for finding frequent itemsets of each size. Usng hashing functions which are independent of each other. Both the hashes will have the same number of buckets. Also while finding frequent pairs, all the possible pairs are hashed but the frequent itemsets of **size >= 3**, as we know the frequent itemsets of **size-1**. The counts in the buckets can vary depending on the hashing function used.

Input parameters are same as above. For output please follow the format shown below:

**Sample 1:**

Executing code: **python nayak\_nandan\_multistage.py input2.txt 2 20**

**Output**

*memory for item counts: 48*

*memory for hash table 1 counts for size 2 itemsets: 80*

*{0: 0, 1: 2, 2: 0, 3: 2, 4: 2, 5: 3, 6: 2, 7: 1, 8: 1, 9: 0, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0}*

*Frequent itemsets of size 1 : [['a'], ['d'], ['e']]*

*memory for frequent itemsets of size 1 : 24*

*bitmap 1 size: 20*

*memory for hash table 2 counts for size 2 itemsets: 80*

*{0: 0, 1: 2, 2: 0, 3: 2, 4: 2, 5: 3, 6: 2, 7: 1, 8: 1, 9: 0, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0}*

*bitmap 2 size: 20*

*memory for candidates counts of size 2 : 36*

*memory for hash table 1 counts for size 3 itemsets: 80*

*{0: 0, 1: 0, 2: 0, 3: 0, 4: 1, 5: 0, 6: 2, 7: 1, 8: 2, 9: 2, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0}*

*Frequent itemsets of size 2 : [['a', 'e'], ['d', 'e']]*

*memory for frequent itemsets of size 2 : 24*

*bitmap 1 size: 20*

*memory for hash table 2 counts for size 3 itemsets: 80*

*{0: 0, 1: 0, 2: 0, 3: 0, 4: 1, 5: 0, 6: 2, 7: 1, 8: 2, 9: 2, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0}*

*bitmap 2 size: 20*

*memory for candidates counts of size 3 : 0*

*memory for hash table 1 counts for size 4 itemsets: 80*

*{0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 1, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0}*

**Sample 2:**

Executing code: **python nayak\_nandan\_multistage.py input.txt 5 20**

**Output**

*memory for item counts: 96*

*memory for hash table 1 counts for size 2 itemsets: 80*

*{0: 2, 1: 4, 2: 5, 3: 14, 4: 8, 5: 17, 6: 11, 7: 14, 8: 12, 9: 9, 10: 9, 11: 12, 12: 9, 13: 9, 14: 6, 15: 4, 16: 4, 17:3, 18: 2, 19: 2}*

*Frequent itemsets of size 1 : [['a'], ['b'], ['c'], ['d'], ['e'], ['f']]*

*memory for frequent itemsets of size 1 : 48*

*bitmap 1 size: 20*

*memory for hash table 2 counts for size 2 itemsets: 80*

*{0: 11, 1: 16, 2: 14, 3: 23, 4: 14, 5: 21, 6: 15, 7: 17, 8: 14, 9: 11, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0,17: 0, 18: 0, 19: 0}*

*bitmap 2 size: 20*

*memory for candidates counts of size 2 : 168*

*memory for hash table 1 counts for size 3 itemsets: 80*

*{0: 12, 1: 11, 2: 11, 3: 10, 4: 6, 5: 4, 6: 10, 7: 13, 8: 14, 9: 11, 10: 12, 11: 11, 12: 14, 13: 14, 14: 17, 15: 17, 16: 16, 17: 15, 18: 15, 19: 12}*

*Frequent itemsets of size 2 : [['a', 'b'], ['a', 'c'], ['a', 'e'], ['b', 'c'], ['b', 'f'], ['c', 'e'], ['d', 'e'], ['e', 'f']]*

*memory for frequent itemsets of size 2 : 96*

*bitmap 1 size: 20*

*memory for hash table 2 counts for size 3 itemsets: 80*

*{0: 24, 1: 22, 2: 25, 3: 24, 4: 23, 5: 21, 6: 26, 7: 28, 8: 29, 9: 23, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0,17: 0, 18: 0, 19: 0}*

*bitmap 2 size: 20*

*memory for candidates counts of size 3 : 16*

*memory for hash table 1 counts for size 4 itemsets: 80*

*{0: 18, 1: 20, 2: 22, 3: 22, 4: 21, 5: 19, 6: 17, 7: 15, 8: 15, 9: 12, 10: 13, 11: 12, 12: 12, 13: 11, 14: 11, 15: 13, 16: 14, 17: 17, 18: 17, 19: 17}*

*Frequent itemsets of size 3 : [['a', 'b', 'c']]*

*memory for frequent itemsets of size 3 : 16*

*bitmap 1 size: 20*

*memory for hash table 2 counts for size 4 itemsets: 80*

*{0: 31, 1: 32, 2: 34, 3: 33, 4: 32, 5: 32, 6: 31, 7: 32, 8: 32, 9: 29, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0,17: 0, 18: 0, 19: 0}*

*bitmap 2 size: 20*

*memory for candidates counts of size 4 : 0*

*memory for hash table 1 counts for size 5 itemsets: 80*

*{0: 11, 1: 13, 2: 12, 3: 15, 4: 15, 5: 20, 6: 20, 7: 21, 8: 20, 9: 21, 10: 20, 11: 21, 12: 19, 13: 20, 14: 15, 15: 15, 1*

*6: 10, 17: 12, 18: 10, 19: 11}*

**Problem 3: Toivonen Algorithm**

Implementing the Toivonen algorithm to generate frequent itemsets. For this algorithm we have used a sample size of less than 60% of your entire dataset. Also a simple Apriori algorithm is implemented with the random sample set. After Checking for negative borders, the algorithm is executed again with a different sample set ifrequired till there are no negative borders that have frequency > support.

**Note:** The number of iterations might differ in your output depending on the random sample which is generated.

**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 toivonen\_test.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>*

Executing code: **python nayak\_nandan\_toivonen.py toivonen\_test.txt 20**