Assignment 4

PART I:

APRIORI

Assignment Overview:

In this assignment you will implement the Apriori algorithm to find all frequent itemsets from a given list of transactions. Frequent itemsets are only those itemsets, which have support count greater than the minimum support provided to the program. The program takes in 2 parameters

- Minimum support
- Input file

Input file format:

A list of transactions where each line represents a single transaction. The items of the transactions are comma (",") separated in one transaction.

Example (input.txt)

```
beer, diaper, baby powder, bread, umbrella
diaper, baby powder
beer, diaper, milk
diaper, beer, detergent
beer, milk, coca-cola
```

Output format:

The output should contain the support for all k frequent itemsets. When k=1 frequent itemsets = frequent items. Do not redirect the output to a file, print it on standard output.

Example : (consider the above input and support as 2)

```
Frequent item sets of size 1
Beer-4
Diaper-4
Baby powder-2
Milk-2
Frequent item sets of size 2
Diaper, baby powder-2
Beer, milk-2
Beer diaper-3
```

Implementation Hints

1. Read input file and store the transaction in 2D arrays (array of array where each array consists of a transaction and every transaction is an array of items)

2. Finding Frequent items

Pass 1: Read the input file of baskets and count in main memory the occurrences of each unique item.

- Requires only memory proportional to #items.
- Find the support for these items using the given set of transactions
- Items that appear at least s times are the frequent items.

3. Finding Candidate itemsets of size k

- For Pass 2: Read input file of baskets again and count in main memory only those pairs both of which were found in Pass 1 to be frequent.
- Requires memory proportional to square of frequent items only (for counts), plus a list of the frequent items from the first pass (so you know what must be counted).

- Use Lexicographic ordering for storing itemsets so that it is easier to find candidate itemsets.

- For Pass k: For creating candidate itemsets of size k, use the frequent item sets of size k-1. Sort all the frequent item sets in k-1 lexicographically. To create candidate item sets only compare ith itemset with only item sets j such that j > i. Create a candidate item set of size k using i and j only if first k-2 elements of i and j are same.

4. Determine which candidate itemsets are frequent

- After finding the candidate itemsets find their support comparing them with every transaction.

5. Use this link for creating test data and testing your output
   http://codeding.com/?article=13

Running code

Name your files as apriori.py and should be run as follows

apriori.py inputFile supportThreshold

Or for the example provided:

apriori.py AprioriInput.txt 3

Where input.txt is the input file and 3 is the support

Sample output file is in AprioriOutput.txt

PART II:

PAGE RANK with Random Teleporting

Assignment Overview:

The objective of this assignment is to implement page rank algorithm to find the rank of all nodes in a directed graph given using power iteration. The program takes in 3 parameters: the name of the input file (e.g., input.txt), number of iterations of PageRank algorithm, and beta.

Input file format:
The input file consists of N lines. Consider that ith line represents the links for the ith node. Each line consists of 1s and 0s. 1 in the i^{th} line at j^{th} position represents that the i^{th} node links to the j^{th} node whereas 0 represents no connection. Each line consists of N numbers representing links to N-1 other nodes. Look at the example below for a clear understanding.

Example (input.txt)
1 0 1 0
0 0 0 1
1 0 0 1
0 0 0 1

In this example the nodes are the row numbers. Namely (0^{th}, 1^{st}, 2^{nd}, 3^{rd})
0^{th} node links to itself and 2^{nd} node.
1^{st} node links only to 3^{rd} node.
2^{nd} node links to 0^{th} node and 3^{rd} node.
3^{rd} node links only to itself.

Remember the number of lines equals number of nodes.

Output format:
The output should contain the page ranks of all N nodes, in the same order as given in the input file. (0^{th}, 1^{st}, 2^{nd},3^{rd}). This output should be a 1XN matrix.

Implementation Hints

1. **Read the input file and store the values in a 2D matrix (input matrix)**
2. **The parameter β will be an input to your program. This usually ranges from 0.8 to 0.9.**
3. **Create the matrix M_initial from the input_matrix such that M_initial[i][j] = probability of going from page i to page j for the ith node. Then create the new matrix M by computing the transpose of M_initial.**
4. **Then generate the matrix A from matrix M:**
   $$A_{ij} = \beta M_{ij} + (1-\beta)/N$$
   Where N is the number of nodes.
5. **Initialize R(old) to [1/N, 1/N …]**

6. **Let the number of iterations be k. For every iteration find R(new) as R(new) = Aij * R(old)**
7. **After k iterations R(new) gives the page rank.**

Running code

Name your file as pageRank.py should be run as follows:

```
pageRank.py inputFileName numIterations beta
```

Or for the provided example:

```
pageRank.py input.txt 6 0.85
```

Where input.txt is the input file, 6 is the number of iterations, and 0.85 is the value for β.