Dept. of Computer Science and Engineering,

CS69011: Computing Lab 1

Assignment 2

Maximum Marks: 30

Consider a social network like Facebook, which has users $U = \{u_1, \dots, u_m\}$, and shared content $C = \{c_1, \dots, c_n\}$. The shared content may include images, videos, third party URLs etc., which signify user activity and interest on the social networks. Additionally, the social network has two more data sources: friendship links (undirected edges) between users $E \subseteq U \times U$, and sharing links (directed edges) $S \subseteq U \times V$, where each edge $(u_i, c_j) \in S$ denotes that user u_i has shared content c_j. Together, they form the user-content graph for the social network. For the purpose of this assignment, you can assume that this graph is a connected graph.

Input File format:

The user-content graph is input as follows:

```
n (number of users)
<user id 1>: <list of users connected to user 1>
…
<user id n>: <list of users connected to user n>
m (number of contents)
<user id 1>: <list of contents posted by user 1>
…
<user id n>: <list of contents posted by user n>
```

Task 1: **Marks: 10**

For the first task, consider the problem of content recommendation, a content c_j is shown to user u_i based on the distance (length of the shortest path) between u_i and c_j. Note that, there are no paths through the content nodes since all share edges are directed. Consider u_1 and u_2 are not friends (say connected through the social network with distance 5), but both have shared content c_2, and u_2 has also shared content c_2. Then, the distance between u_1 and c_2 is **not** 3 due to path (u_1 -> c_1 -> u_2 -> c_2).

Your task is to write a program which will list the first k recommended contents, $c_{i_1}, \dots, c_{i_k}$, in increasing order of their distances from user u_i. The content which is already shared by u_i should not be displayed. Here the user u_i and k should be taken as input, and name of the file containing user content graph should be taken as command line input.

Hint: Use a modification of Dijkstra's single source shortest path algorithm.

Input: filename for the user content graph in the above format, the user u_i for whom the content will be recommended and k the maximum number of recommended contents. These should be taken as command line input.

Output: The output for this task is a list of recommended content $c_{i_1}, \dots, c_{i_k}$ along with the corresponding distances $d_{i_1}, \dots, d_{i_k}$ from user u_i.

Algorithm description:
Dijkstra's algorithm maintains, at each stage:

1. Distances to all nodes in the graph, some of them infinity.
2. A set Q of all vertices which can be explored next.

In every step, the minimum distance node in set Q is finalized (it's distance is frozen as the shortest distance and the node is removed from Q), and its neighbors are processed or *relaxed* (added to the set Q and their distance are updated). You have to modify the above algorithm so that you keep a counter which counts the number of content nodes which are finalized. Once this number reaches k, you can stop the algorithm and output the k content nodes that you have discovered along with their distances.

Data structures: Both the distance and the Q set can be maintained as arrays of size the number of vertices. Min-distance node can be found using a linear scan or a min-heap. The min-heap implementation is more efficient and will get more marks.

You can call this function `int topKMinDist(Graph *G, int start, int * nodes, int * distances)`, where G is an input graph pointer (can be taken as an adjacency list), and nodes and distances are arrays for return top k node ids and distances. The return value from the function can be the actual number of content nodes (which could be less than k).

Task 2: **Marks: 10**

In the second task, consider the problem of friend recommendation, using the number of common contents in the k-neighborhood. Specifically, given a user u_i, you will rank users u_j using the count of contents c_l, such that distances between u_i and c_l, $d(u_i, c_l)$, and $d(u_j, c_l)$ are both less than k. Let $Q(u_i, u_j)$ be the count of k-neighbors which are content nodes in the user-content graph. Write a program which given a user content graph, a user u_i and k, prints the list of top p users u_j in decreasing order of $Q(u_i, u_j)$.

Input: the user content graph as above, a user u_i, neighborhood size k, list length l. The filename for the user content graph can be taken as first command line input, followed by other arguments in the order mentioned above.

Output: The list of top l users along with their corresponding distances.

Algorithm description:
In this problem, you have to find all content node whose distance is less than k from the starting user node u_i. Let that list be L. Next, find all user nodes u_j which are at distance at most k from each of these content nodes. Since, there are not paths starting with content nodes, you have consider all paths from all users nodes which are directly connected to the content nodes. The algorithm for finding distance are same as described in task 1.
You need two functions:

```
int distanceLContentNodes(Graph *G, int start, int * nodes, int * distances)
```

and

```
int connectedUserNodes(Graph *G, int start, int * nodes, int * distances)
```

Task 3: **Marks: 10**

Now, consider that there are privacy settings for each share link (u_i, c_j) where each link is marked as:

- Friends: only friends can see the post.
- Friends of friends: friends of friends can also see the post.

- Public: everyone can see the post.

The privacy setting induces additional and implicit "view" edges, which connect each shared content c_j to friends of u_i (1-hop neighbors), friends of friends of u_i (both 1-hop and 2-hop neighbors), and public (all users of the system).

The third task is to recommend content on the basis of distance in the "view" graph, which consists of the original edge as well as view edges introduced due to privacy settings. However, you are **not allowed** to generate the full view graph and then use the algorithm developed in task 1. Your algorithm should only "expand" the relevant share edges. For example, if the top list ends with content at distance 3 in the view graph, there is no need to expand shares of content beyond distance 5 in the user-content graph, since distance can reduce by at most 2 in the view graph, unless the share is a public one.

Note that, here all content can be recommended to all users. Only post information is private.

Algorithm hint: You should treat public shares separately, since they will be at distance 1 from all users. While updating the distances in the shortest path algorithm, you should take care of privacy setting of the edge.

Input File format:

The modified input format of privacy aware user-content graph is as follows:

```
n (number of users)
<user id 1>: <list of users connected to user 1>
…
<user id n>: <list of users connected to user n>
m (number of contents)
<user id 1>: <posted content 1>,<privacy flag 1>; <posted content 2>,<privacy
flag 2>; …
…
…
…
<user id n>: <posted content 1>,<privacy flag 1>; <posted content 2>,<privacy
flag 2>; …
```

Input: filename for the privacy-aware user-content graph in the above format, the user u_i for whom the content will be recommended and k the maximum number of recommended content. These should be taken as command line input.

Output: The output for this task is a list of recommended content $c_{i_1}, \dots, c_{i_k}$ along with the corresponding distances $d_{i_1}, \dots, d_{i_k}$ from user u_i.

Submission:

Submit 3 c source files for: task 1, task 2, and task 3, respectively. Write your name and roll number in a comment at the top of the file along with compilation instructions. The program should read the graph as input from a file and the name of the file should be first argument. Other inputs should be read from command line arguments as mentioned in the task. You can assume that maximum number of users and contents to be 1000 each.

Marks will be given for more understandable (logical order of functions, naming of variables, etc.), maintainable (proper comments) and concise (no code duplication) programs.