CS 130A
Project - Friend recommender

Assigned: November 8, 2018
Due On Demo Day: December 7 2018

PLEASE NOTE:

- Solutions have to be your own.
- You can form teams of **two** to collaboratively work on this project.
- Requests for a regrade must be submitted within seven days from the day when we return the assignment.

1 Introduction

Recommendation systems (or recommender systems) are highly popular and are used in a very wide range of applications including social media, online shopping, movies, etc,. In general, a recommendation system includes a set of users and their preferences, and performs the recommendation based on those preferences. The recommendation can be either to suggest one user to the other based on their preferences or to suggest one user's preference to the other.

In this project, you will implement a recommendation system which includes friends (i.e. users) and genres of movies (i.e. preferences) that a user likes. Your recommendation system will recommend a user to another user who likes the same movie genres. Each user will have two genre preferences. And there will be a friendship graph. Any two users who are not already friends and have at least one genre preference in common are eligible to be recommended as friends.

2 Project details

Your program will maintain a friendship graph to keep track of users and their friends. Each user will have four attributes: perm number, name, and two movie genres they like[1].

[1]**NOTE:** The User class does NOT store any friends information.

```
class User
{
    public:
        ...
    private:
        int perm_number; // unique id
        string name; // name of the person
        string genre1; // favorite movie genre
        string genre2; // second favorite movie genre
        ...
    ...
};
```

In the beginning of the program, you will be given a *text file* with user information i.e., a list of users with their perm number, name, genres they prefer, and *friendship information* which will be the perm numbers of their friends. Each entry in the file will have the following format. An example file is given in Figure 1:

$$PermNo; Name; Genre1; Genre2; PermNoOf1stFriend; PermNoOf2ndFriend...$$

```
1;Mert;Sci-Fi;Drama;2;4
2;Sydney;Comedy;Crime;1;3;4
3;Sujaya;Drama;Advanture;2
4;Amr;Sci-Fi;Crime;1;2
```

Figure 1: Sample file format of user information

Note that you cannot presume the numbers of friends for each user. From this file you are expected to create two data structures that will be explained in Section 3.

2.1 Genre Information

Each user will have two genres that they like. For simplicity, this assignment will contain only a small set of genres. You can find all possible genres in Table 1.

Action	Adventure	Comedy	Crime
Drama	Fantasy	Horror	Sci-Fi

Table 1: Available Genres

3 Data Structures

This section explains the two essential data structures that you will be using in this project. You may use additional data structures (array, linkedlist, hashtable, etc.,) if you find good use cases for them.

3.1 Graph

You will construct a friendship *graph* consisting of nodes and undirected and unweighted edges. Each node will represent *a user*, where each edge will express the friendship relation between two users (i.e. If there is an edge between two nodes [users], it means the two users are friends). You can see in Figure 2 a simple example of a friendship graph.

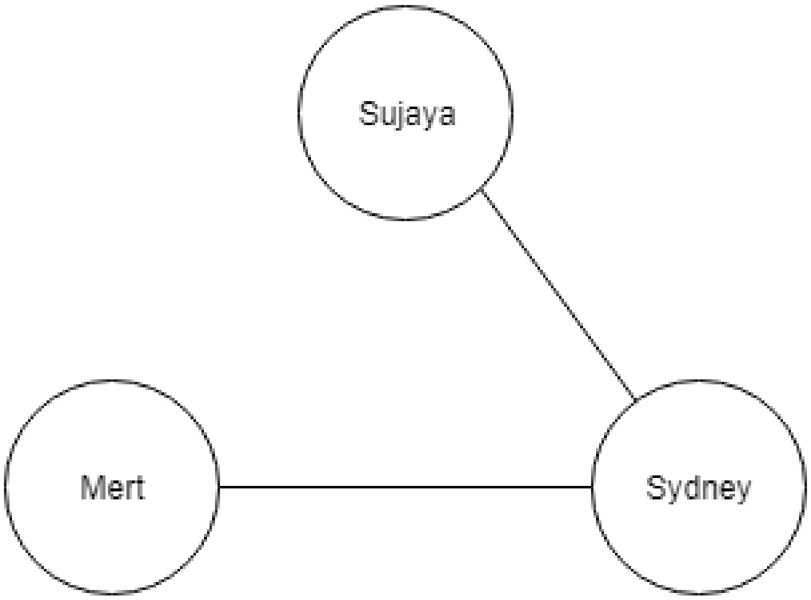

Figure 2: In this graph, we have three nodes [users]: Mert, Sujaya and Sydney. Edges represent Mert and Sydney are friends, Sujaya and Sydney are friends, but Sujaya and Mert are **not** friends.

You will be using an **adjacency list** to represent the friendship graph. The adjacency list will be a list of linked lists, where the head of each linked list stores the perm number of a unique user. The adjacency list for the above graph will look like:
[
$perm_{sujaya} \rightarrow perm_{sydney}$,
$perm_{sydney} \rightarrow perm_{sujaya} \rightarrow perm_{mert}$,
$perm_{mert} \rightarrow perm_{sydney}$
]

Note that the adjacency list does **not** contain any other information about a user. It only stores the friendship relation between the users.

3.2 B-Tree

A B-Tree data structure will be used to find a given user in the friendship graph. Your B-Tree will have ***M=4, L=2*** and each leaf node will consist of two user's information (as $L = 2$); each user's information is: a User class object with information on the name, perm number and two favorite genres, *and* an index of the adjacency list pertaining to that user. The B-Tree will be **sorted in ascending order of perm numbers** of the users. Figure 3 shows an example of the B-tree.

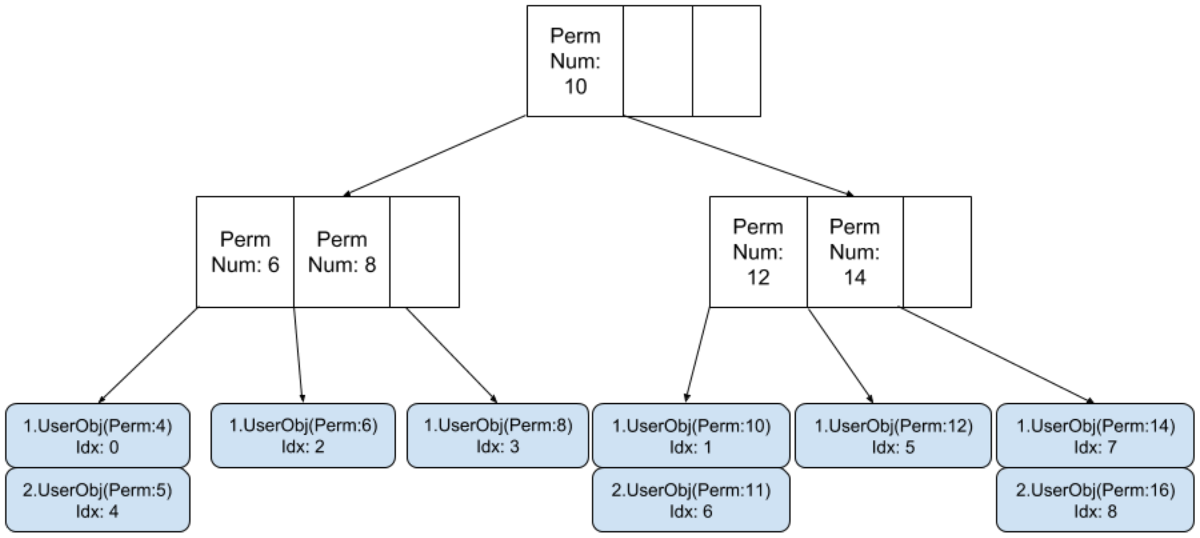

Figure 3: B-Tree example

Please note: B-tree is the *only* way to find a user. From the friendship graph (adjacency list) you can extract a list of perm numbers of a given user's friends and use B-tree to find the information about each friend.

4 Implementation details

As a part of this homework, you will implement 5 functions as explained below:

- *Input file:* This function takes the reference to a file consisting of all the users' information and will construct both data structures: the B-tree and the friendship graph (represented as adjacency list).
- *Add a user:* This function provides a way to add a new user into the system. The function will be called with parameters of *perm number, name, 2 favorite movie genres* and *a list of perm numbers* representing the user's friends. The user's information will be added into both the B-tree and the graph, and may require updating the friendship graph of existing users as well.
- *Find a user:* This function takes *perm number* as input and finds the respective user on the B-tree. If the user is present in the system, the function will print a success message; and a 'not- found' message otherwise.
- *Find a user's details:* This function takes *perm number* as input and finds the user in the B-tree. If the user is found, the function will further fetch the user's details using the object stored in B-tree and will print the user details (name, favorite genres). using the index stored in B-tree, it will also fetch and print the perm numbers of the user's friends. The function prints a 'not-found' message if the user is not in the system.
- *Recommend friends:* This function takes *perm number* as input and extracts the user's favorite genres and their friends (similar to the previous function). The function will then traverse the whole friendship graph (i.e., adjacency list) either using **BFS** or **DFS** and identifies all the users that have at least one favorite genre that is common to the current user. And then filters out the users that are already friends with the current user. This list will be the recommended friend list for the current user. The function should then clearly print the < *perm number, name, 2 genres* > for each recommended friend.

NOTE:

1. We do not want any front end UI for this project. Your project will be run on the terminal and the input/output for the demo will use `stdio`.
2. Your program should be interactive and give us the option of choosing one of the 5 functions to execute during your demo. Failing to do so will result in penalty.

5 Demo

We will have a short demo for each project. It will be on **December 7, 2018** in CSIL. Time details will be announced later. Please be ready with the working program at the time of your demo.