Data Structures and Algorithms
Mini-Project Report
Friends Recommendation System
Team: 55

Summary:

The project is a system which recommends friends based on certain parameters. Various functions are supported by the system, some of which are addition of a new user, making a user follow another user, un-register a user from the system and many more! The project also has a GUI which simplifies the process of using the system. The data structures and algorithms have been selected so as to decrease the overall complexity of the programs.

Data Structures used:

1. Structures :
   Various structures have been used for making hash maps, storing details of users etc.
2. Graphs :
   We have used graphs to form a directional network of users such that there is an edge from user1 to user2 if user1 is a friend of user2.
3. Binary Tree :
   We used self adjusting binary tree (AVL tree) to store data for which search and deletion operations were frequently used,like to store list of followers and following users of a given user.
4. Linked lists :
   Linked lists were used in other data structure like queues and Hash tables.
5. Queue :
   Queues have been implemented with the help of linked lists.
   We use queues in recommending friends to existing users via BFS, and since queues preserve the order of who comes first,it is easier to determine which user is given more priority while recommending.
6. Hash Tables :
   Hash tables provide us a faster way to access information related to a particular key. Since we do not know how many user might register,we take a hash table with separate chaining, so that we can store and access information in a fast and organized way. Hash tables were used in graphs, and parameter grouping.

DETAILS OF ALGORITHMS USED

1.Registering New User:

- When a user registers, we give them the minimum positive integer id which is not currently in use.
- If the user unregisters, we store that id in an AVL tree. While allocating id to a new user we check if this AVL tree is empty, if not we allocate minimum element from AVL tree as the user id, else we allocate a new id.
- We maintain a Graph for storing the user Nodes. Each Node in graph has two AVL trees associated with it, one for followers and others for following. This will help to minimize the time for search when we are finding if there exists a friendship between two users and helps to remove an unregister user faster.
- Now we add the new user to the graph Hash table.

➔ Find the minimum Id which is not in use using findmin() function in AVL tree (AVL tree containing Id's not in use)

Time Complexity: O(log n){n is number of id's currently not in use}

➔ Then Store the User details in Hash Table corresponding to the key.

Time Complexity: O(α) where is the load factor.

---

2. Adding friends to user:

➔ We need to find the User1 (O(α)) in graph and insert User2 into its following AVL tree (O(log n))(where n is number of nodes in AVL tree).

➔ Find the User2 in hash table (O (α)) and insert initial user into its followers AVL tree (O (log n))(where n is number of nodes in AVL tree).

3.Friends Recommendation For a New User:

➔ We make hash tables for each of the parameters (city/ organization) and maintain a AVL tree for each Node of a Hash Table and add users of same parameter(city/organization) to the AVL tree.

➔ It helps in finding people with common parameters. So, it will be used to recommend friends to new users as we are having the users of common parameters.

➔ Time Complexity: Let the number of people with same city be c and same organization be r, then time complexity is O(r+c+α) , where α is the load factor for the city, organization hash tables.

4.Un-registering User:

➔ Un-register from Hash Table and insert that Id in AVL Tree(O(log n)) (AVL tree containing Ids not in use).

➔ Traverse through the following AVL tree and go to each id and remove User from their followers AVL tree.

Time complexity for traversing is O(n)(where n is number of nodes)

➔ Traverse through the followers AVL tree and go to each id and remove User from their following AVL tree. Time complexity is same as above(Note that deleting in AVL tree is O(log n).

➔ Remove user from parameter hash tables.

Time complexity:O(α) where α is load factor.

5.Friends Recommendation for already registered users:

➔ We used BFS here since in recommending friends for existing user,the nodes at equal depth are given same priority and the ones with greater height come first.

➔ Time Complexity: O(K+F), where K is the number of recommendations required and F is the number of friends the user has.

6.Checking whether User1 is friend of User2:

➔ Check the following(friends) AVL Tree with Node pointing to User2.

➔ If User 1 is present in the AVL tree, then User 1 is friend of User2 else he is not.

Time Complexity: O(log n) where n is number of nodes in

following AVL tree of User2.

Part C: Division of Work

- Keyur Ganesh Chaudhari:

  BFS Implementation, GUI interface, Menu

- Thatipammula Harshavardhan:

  AVL Tree Implementation, Un-registering of User

- Rohith Reddy Lingala:

  Hash table for user details

- Yalaka Surya Teja Reddy:

  String Hash Table, Friend recommendation for new user, Adding new user.

- M Nandini Reddy:

  Network Graph.