FRIEND RECOMMENDATION APPLICATION

PROJECT REPORT

TEAM 13

The Amigos App Source Code is hosted on GitHub.

[freyam/amigos]

Data Structures

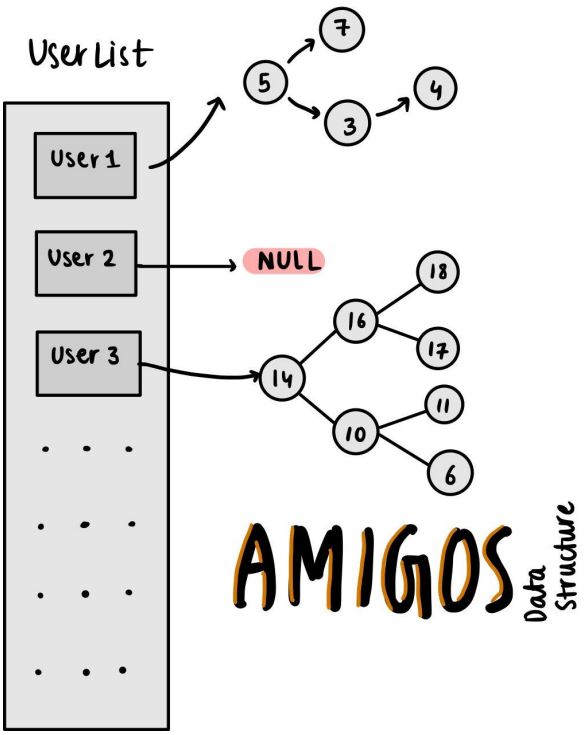
Adelson-Velsky and Landis (AVL) Tree

An AVL Tree is a self-balancing Binary Search Tree. Though it's similar to a binary search tree, there is one highlight of a difference: the height of the tree value should be <=1, and unlike the binary search tree, AVL has the elements on both sides of the tree to be balanced. The AVL tree structuring is implemented with the three primary data structure operations: insert, search, and delete.

Both Red-Black Trees and AVL Trees are the most commonly used balanced binary search trees, and they support insertion, deletion, and look-up in guaranteed O(log V) time. However, we finally decided to use AVL trees as they are more rigidly balanced and provide faster look-ups. Thus for a look-up intensive application such as Amigos, we used an AVL tree.

We have used an AVL Tree to represent the friend lists of a User.

```
- Adding a new friend to the friend list      O(log V)
- Searching for a friend in a friend list     O(log V)
- Removing a friend from a friend list        O(log V)
```



We have also implemented the entire Amigos App using a Red Black Tree.

Binary Min Heap

A Min-Heap is a complete binary tree in which the value in each internal node is smaller than or equal to the values in the children of that node. The exact property must be recursively true for all nodes in all the subtrees. The root of the Min Heap is always the smallest element in the Binary Tree.

We have used a Binary Min Heap to store the removed User IDs so a new user would first occupy the minimum position in the User List.

```
- Getting the Minimum User ID              O(1)
- Maintaining the Heap Property (Heapify)  O(log V)
```

Queue

A Queue is a memory-efficient linear data structure that follows FIFO (First In First Out) order in which the operations are performed. Elements are enqueued (added) at the rear and dequeued (removed) from the front.

We use a queue for implementing the creative friend recommendation system Amigos App has set up for the existing users. A queue is maintained for the Breadth-First Traversal Algorithm that we run on the Friend lists. The FIFO concept that underlies a Queue ensures that the friends that we discover first will be recommended first.

```
- Enqueuing a new element                          O(1)
- Deque an element                                 O(1)
- Getting the element at the front of the Queue    O(1)
- Getting the element at the rear of the Queue     O(1)
```

Algorithms

CheckStatus()

To check whether 2 users are friends of each other, we use **AVL Tree Search**.
Since we are storing all the friends of a user in an AVL Tree, finding whether a given friend exists in the user's friend list is very trivial. We search the AVL Tree in Pre Order Fashion and return TRUE if a friend is found and false otherwise.

```
Check whether 2 given UIDs are friends        O(logV)
Check whether 2 given Names are friends       O(VlogV)
```

RecommendFriendsNewUsers()

To recommend friends to a new user based on a compatibility score, we use **Sorting and Searching**.
We generate a simple integer array consisting of the Compatibility Scores of the given user with all the Users in the Database. We sort this array using Counting Sort in `O(V + K)` (K = 100) in non-increasing order. We display the first 10 users of the sorted array who have the highest compatibility with the given user.

```
Recommend 10 friends to a New User            O(V)
```

RecommendFriendsExistingUsers()

To recommend friends to a new user based on common friends, we use **Breadth-First Search**.
We generate 2 Queues to store Current Friends and Recommended Friends. Now, we traverse through the friend list and mark all the current friends to VISITED = 1. Finally, we run a Breadth-First Traversal on the Friendlist (an AVL Tree) and update the Queues accordingly. By the Level order Binary Tree Traversing approach, BFS recommends the 1st Common Friends > 2nd Common Friends > 3rd Common Friends … and so on.
Traverse through the first K Users in the queue and display the first K users of the queue.

```
Recommend K friends to an Existing User      O(V)
```