**Semester Project**

**Data Structures and Algorithms**

Submitted to:

DR. INAYAT-ur-rehman

Submitted by:

MAHRUKH (034)

Mohammad Haris Zia (037)

Data Structures Used:

* Arrays
* Linked List
* AVL Tree
* Stacks Using Array
* Queue Using Linked List
* Priority Queue
* Graphs

Algorithms Used:

* Dijkstra Algorithm
* Prims Algorithm

Description:

**Jukebox (Music Player)**

* We used a Recursive implementation of AVL Tree in order to store our songs and its associated Credentials to reduce the search, insert and delete time complexity to O (log n).
* Initially, the program reads data from the file and inserts it in the AVL tree.
* While on runtime, the AVL keeps on getting updated but no changed occur on the file as authors tried to keep the time and space complexity to bare minimum
* Once the program terminates, only then the updated changes are rewritten to the file.

Functionalities:

1. Display Song List

* The AVL Tree is displaced in Alphabetical Order using In Order Traversal Logic.

1. Inserting a New Song

* The **AVL Tree** is Traversed from root to the Empty location for insertion.
* Which is found on the basis on alphabetical order the traversal node moves left if the alphabetical order of the node to be inserted is less than the current traversal node else it moves right.
* Finally, when the Node is inserted in the first Empty Location, **Back Tracking** of the Calls start from bottom to up.
* Back Tracking calls are used to update heights and Balance Tree with the condition being that It Balances from bottom up of the first unbalanced node found.

1. Deleting a Song

* The song is searched, If not found it generates an Error that Song not Found.
* The **AVL Tree** is Traversed from root to the location of node to be deleted.
* Which is found on the basis on alphabetical order the traversal node moves left if the alphabetical order of the node to be deleted is less than the current traversal node else it moves right.
* Finally, when the Node is found, it is replaced from the value at it right to the left most node. Then **Back Tracking** of the Calls start from bottom to up.
* Back Tracking calls are used to update heights and Balance Tree with the condition being that It Balances from bottom up of the first unbalanced node found.

1. Playing a Song

* Song is Searched from **AVL** and played.
* Else gives an Error that Song not Found
* The played song is pushed to the stack and priority queue afterwards

1. Creating a playlist

* To create a playlist, every song is enqueued in our **queue** and then it is streamed via dequeuing them one by one making sure no song is repeated.
* The played songs are pushed into the stack and priority queue respectively.

1. Recently Played Song

* To view recently played songs, songs are displayed from the **stack** which kept track of the played songs.

1. View Most Played Songs

* For this function , we used **priority queue** as it is the most efficient in such scenarios. Recall that every song was pushed into the priority queue after being played, turns out it had a reason. Every song has priority 1 when it gets played for the first time but it’s priority gets updated when its played multiple times. So to view most played songs, they are simply dequeued from the priority queue.

1. Finding Shortest Path to the Artist

* No other algorithm comes to one’s mind other than **dijsktra algorithm** when we talk about shortest path problem solving solutions. We used dijsktra algorithm to find the shortest path cost from our present continent to the artist.

1. Calculate World Tour Distance From Your Location

* For these feature, we used **prim’s algorithm** to generate a minimum spanning tree,as it is significantly faster, travelling across the globe evaluating the total distance in kms. We made use of **priority queue** here to reduce the time complexity to (O logN) as compared to O(N) if using normal array otherwise.