

DEPARTMENT OF COMPUTER SCIENCE

DSA PROJECT

ODI Record of Batsmen

And International Stadiums

Group Members:

* Moiza Imran (SP23-BCS-044)
* Mahnoor Bilal (SP23-BCS-038)
* M.Bilal Saleem(SP23-BCS-050)

Submitted To: Prof Inayat ur Rahman

**Introduction:**

The dataset includes the One Day International Cricket Format record of batsmen. This includes name, matches, runs, average, fifties and centuries scored by each player. We also used another dataset that has the record of Distances between International Stadiums.

**Functionalities:**

The cpp Program includes number of functionalities using different Data Structures.

**Functions on ODI record dataset:**

* Get All Player’s Data.
* Get Data w.r.t Matches played.
* Get Data w.r.t Average of each player.
* Get Data w.r.t Runs made.
* Get data of a particular Player.
* Add a New Player in data.
* List from lowest to hightest Runs Made.
* List from Most to Least Matches Played.
* Delete a Player by his name.
* Calculate Average runs for all Data Set players.
* Sorting Players by average.
* Searching about a particular player using average.
* Finding frequency of Centuries with constant time complexity.

**Functions on Stadium Distances dataset:**

* Calculating shortest distance from one Stadium to other(connected or not).
* Getting to shortest connection among all stadiums.
* Printing the graph and results of the algorithms.

Data Structures:

The Program consist of

* Array
* Doubly Link List
* Stack
* Queue
* Binary Search Tree(AVL Tree)
* Hashing
* Graphs
* Minimum Spanning Tree

Use of Data Structures;

* **Link List** is used to store complete information of a single player in single place.The data fro CSV file is stroed in Nodes and than a link list is made.
* **Stacks** are used to implement different functionalities where we are requires to deal with First in Last out operation like Sorting.
* **Queues** are used to implement different functionalities where we are requires to deal with First in First out operation like intermediate storage.
* **BST(precisely AVL Trees)** are used to sort data w.r.t average and to reduce the search to log(n).
* **Hashing** is used to find the frequency of a particular century number with O(1) time Complexity.
* **Graphs** are used to link data eith each other. We implement dijkhstra Algorithm using them.
* **Arrays** are used to implement graphs.
* **Minimum Spanning Trees** are used to create link between all nodes with minimum distance between them. We implement Prims Algorithm using it.

Libraries Used:

* #include <iostream>
* #include <fstream>
* #include <sstream>
* #include <string>
* #include <stack>
* #include <queue>
* #include <climits>

These libraries provide functionalities for input/output operations, file handling, string manipulation, and data structures such as stacks, and queues.

Node Structure for Linked List:

struct Node

{

    string playerName;

    int matches;

    int runs;

    int centuries;

    double average;

    Node \*next ;

    Node \*pre;

    Node(string n , int r , int m , int c , double a)

    {

        playerName = n;

        matches = m;

        runs = r;

        centuries = c;

        average = a;

        next = NULL;

        pre = NULL;

    }

};

struct Node \*head = NULL;

struct Node \*tail = NULL;

Defines a `Node` structure for the doubly linked list to store player data, and initializes `head` and `tail` pointers.

Node Structure for BST

struct BSTNode

{

    string playerName;

    int matches;

    int runs;

    int centuries;

    double average;

    BSTNode \*left ;

    BSTNode \*right;

    BSTNode(string n , int r , int m , int c , double a)

    {

        playerName = n;

        matches = m;

        runs = r;

        centuries = c;

        average = a;

        left = NULL;

        right = NULL;

    }

};

struct BSTNode \*BSTroot = NULL;

Defines a `BSTNode` structure for the binary search tree to store player data based on their batting averages. Initializes the root of the BST.

Function to add a new node to the BST. It places nodes in the left or right subtree based on the average.

Graph vertices(Stadiums):

string vertices[13] = {"Dehli","Channai","Mumbai","Chittagong","Kabul","Rawalpindi","Lahore","Eden Park","Galle","Lords","Sydney","Melbourne","Village Green"};

Graph Connections:

0 2106 1403 0 1008 0 0 0 0 0 0 0 0

2106 0 1387 0 0 0 0 0 0 0 0 0 0

1403 1387 0 2600 1757 0 0 0 0 0 0 0 0

0 0 2600 0 0 0 0 480 0 5090 0 0 0

1008 0 1757 0 0 650 814 9030 0 0 0 0 0

0 0 0 0 650 0 330 0 0 0 0 0 0

0 0 0 0 814 330 0 0 2850 0 0 0 0

0 0 0 480 9030 0 0 0 10850 18300 0 0 1300

0 0 0 0 0 0 2850 10850 0 0 0 8500 0

0 0 0 5090 0 0 0 18300 0 0 10998 0 0

0 0 0 0 0 0 0 0 0 10998 0 713 2226

0 0 0 0 0 0 0 0 8500 0 713 0 2581

0 0 0 0 0 0 0 1300 0 0 2226 2581 0

Main Function and Menu

int main()

{

    ifstream csvFile;

    csvFile.open("ODIdata.csv");

    string line , word;

    vector<string> vec;

    while(!csvFile.eof())

    {

        getline(csvFile , line);

        stringstream s(line);

        while(getline(s,word,','))

        {

            vec.push\_back(word);

        }

        int matches;

        stringstream ms(vec[3]);

        ms>>matches;

        int runs;

        stringstream rs(vec[6]);

        rs>>runs;

        double avg;

        stringstream as(vec[8]);

        as>>avg;

        int centuries;

        stringstream cs(vec[11]);

        cs>>centuries;

        // add data to Node and create link list

        addNode(vec[1] , runs , matches, centuries , avg );

        // add data to BSTNode and create BST

        BSTroot = addNodeInBST(BSTroot , vec[1] , runs , matches, centuries , avg );

        //clearvector

        vec.clear();

    }

User menu

Enter :

* 1 to print All Players Data
* 2 to print Data w.r.t Matches played
* 3 to print Data w.r.t Average
* 4 to print Data w.r.t Runs
* 5 to get data of particular Player
* 6 to Add New Player
* 7 to print from lowest to highest Runs Made
* 8 to Print from Most to Least Matches Played
* 9 to delete Player by name
* 10 to get Average runs for Data Set players
* 11 to sort by average using BST
* 12 search player by average using BST
* 13 to check Shortest Distance from Particula Stadium
* 14 to make Shortest connection between Stadiums
* 15 to print Graph
* Any other integer to exit

The `main` function loads player data from a CSV file, populates the linked list and BST, and provides a menu for user interaction and also reads stadium data from a CSV file, constructs the graph, and then prints the graph.

Functions:

1. The program currently contains placeholders for functions such as `printByMatches`, `printByAverage`, `printByRuns`, `printByName`, `addNewPlayer`, `reversePrint`, `addInQueue`, `sortingByMatches`, `printSortedlyByMatches`, `deleteByName`, `averageRunsInDataSet`, `inorder`, and `searchByAverage`. These functions need to be implemented for full functionality.

2. The CSV file must be formatted correctly, with columns: Name, Matches, Runs, Centuries, and Average. The CSV file `StadiumsDistance.csv` should have the following format: origin,destination,distance

Conclusion:

This program provides a robust framework for managing and analyzing ODI batsmen data using a doubly linked list and a binary search tree. With the complete implementation of all functions, it will offer comprehensive data management and query capabilities it also provides functionalities to manage and analyze cricket stadium distances using graph algorithms. It constructs a graph from a CSV file, and uses Dijkstra's and Prim's algorithms to find the shortest paths and the Minimum Spanning Tree, respectively.