Assignment 3 - Report

Muhammed Sefa Aydoğan

150150124

Part 2. Report

A) Complexity :

It is a balance tree. In seach, I traverse through left and right child.
Changing the colors of nodes during recoloring is O(1). However, we might then need to handle a double-red situation further up the path from the added node to the root. In the worst-case, we end up fixing a double-red situation along the entire path from the added node to the root. So, in the **worst-case**, the recoloring that is done during insert is O(log N) (time for one recoloring * max number of recolorings done = O(1) * O(log N)).
-Asymptotic upper bound for insertion for average case: O(logN). No matter how many recoloring will be made, it will always be O(1).

The seach-time results from the traversal from root to leaf, and therefore a balanced tree of n nodes, having the least possible tree height, results of the search time will be O(logN). It is a balanced tree, no mather how many nodes will be in the tree, height will always be O(logN) and searching in that will be O(logN)
-Asymptotic upper bound for search for worst case is : O(logN)

-Asymptotic upper bound for search for average case is : O(logN)

B) RBT vs BST:

Because RBT is a balance tree, It has the least possible tree height which is always O(logN). Because of this, in running time, it will run faster in any way. Because one of the dependencies of the running time is the tree height. In BST, worst case running time is O(N). Because it is not a balanced tree so it does not have a fixed tree height.

C) Augmenting Data Structures:

My strategy will be like in finding the leader for point, assist and rebound. I would create a struct. It will have 5 array for 5 positions. Whenever a new player added to the tree, I would check his position and add his information to the appropriate array. My struct could be like this:

```
struct positions{
        string PG[ ];
        string SG[ ];
        string SF[ ];
        string PF[ ];
        string C[ ];

};

Positions position_informations;
```

So, after implementing a struct like this, whenever a new player inserted to the tree, I will update this position_information struct variable in the insert function.

If user wants to know i_th Point Guard, i_th Shooting Guard, i_th Small Forward, i_th Power Forward or i_th Center, a function named “get_ith_positions” will be called and it can be like this:

```
String get_ith_positions(int i,string position):

        If(position==”PG”)

                return position_informations.PG[i]

        If(position==” SG”)

                return position_informations.SG[i]

        If(position==” SF”)

                return position_informations.SF[i]

        If(position==” PF”)

                return position_informations.PF[i]

        If(position==” C”)

                return position_informations.C[i]
```