CS301 A3

Gizem Topsakal

November 19, 2022

1

To calculate the black height of a Red-Black Tree, we need to count the black nodes from the root to the external nodes excluding the root. We can count it in any path since all paths from a node to its null descendants contain the same number of black nodes. We can calculate any node's black height this way.

In the question, we are asked to calculate black heights of nodes and insert them as attributes to each node. The maximum height of the tree is O(logn) so calculating the black height will take O(logn) time like the BST.

The insertion takes O(logn) time and it will cause different cases. First case will be just inserting to the right place which we require no rotations or coloring so it will not affect the time complexity.
Second case is when inserted node's uncle is red. This case will require only recoloring. We will resolve the issue by recoloring parent, grandparent and uncle of the newly inserted node. In this case the grandparent's black height will chance and calculating it will take constant time. This case did not affect the time complexity.
Third case is when the new node's uncle is black and new node is the left child. This case will require rotations and recoloring. But none of the node's black height will change so no need to update the black heights. Rotations will take O(1) time so this case will also have no affect on the time complexity.
The last case is the same with the third case but this time new node is the right child. Again nothing changed about the black heights and rotations will take constant time so time complexity is still O(logn)

As a result, when we examine each case one by one we can see that none of them affects the time complexity. It is O(logn) in each case.

2

The depth of a node can be found by counting the nodes starting from the root until the node itself. This means the nodes depth is dependent on the ancestors of the node. Hence any change in the ancestors will affect all the nodes after that node on the path.

As explained in the first question insertion will cause 4 cases. In the first case we can find node's depth while inserting the node so this case will take O(logn) time.
The second case requires just recoloring so it is same as the first case in term of complexity. The third and fourth cases will require both recoloring and rotating. So the children nodes' depth should be all updated. This means we should iterate through every node and update their depths. Hence in the worst case it will take O(n) time.

To sum up if the insertion requires a change in the RBT's structure, this will change the new inserted node's parent's depth and this will cause a change of the children nodes depth. So every time a chance occurs this will affect the total time complexity. Insertion takes O(logn) time and updating the depths will take O(n) time, in total O(logn)+O(n). This is equal to O(n).