CS-202

ASSIGNMENT - 04

S. SAMARTH REDDY

B19109

GROUP - 25

(1) Given n integers in sorted order, we can use the following algorithm to construct a RB Tree in O(n) time:

Algorithm:

- Make the middle integer of the sorted array the root of the Red black tree. Middlemost number = $\lceil n/2 \rceil$ th number.
- Then, make the middle integer of left part of array (number to the left of $\lceil n/2 \rceil^{th}$) as the left child of the root node, and the middlemost integer of right part of array as the right child of root node.
- Continue this process for the left side and right side arrays recursively till all the integers have been placed.
- Colouring all nodes black. except th. Ones at the bottom -pm (colour them Red), would result in a red black tree satisfying all the conditions. All null nodes with black.

The algorithm takes time T(n) such that

$$\{T(n) = 2T(n/2) + c\} \longrightarrow T(n) = O(n)$$

This can be easily verified by substitution.

Another way to look at it is, each recursion takes $O(1)$ time and there are n recursion (for each integer), $\therefore \boxed{T(n) = O(n)}$

---

(2) - Given a red black tree with m elements, we can find an array with non-decreasing order of elements of rb tree using inorder traversal. This can be done in $O(m)$ time.

$\therefore$ Given two RB trees with m and n number of elements, we can find the sorted arrays (non-decreasing) of these two trees in $O(m)$ and $O(n)$ time respectively.

- Now create a new array using these two arrays using merge sort. This can be done in $O(n+m)$ time.
- we have an array of $(n+m)$ non-decreasing (sorted) elements. Using the algorithm used in question 1, the RB Tree can be created in $O(m+n)$ time.

$\therefore$ Total Time taken = $O(m+n) + O(m+n) + O(n) + O(m)$

Part 1: $m = \sqrt{n}$

Total ~~Tim~~ Time = $O(n+\sqrt{n}) + O(n+\sqrt{n}) + O(n) + O(\sqrt{n})$

$= \underline{O(n)}$ [Since $n > \sqrt{n}$]

Similarly, Part 2: $m = n/2$

Total Time = $O(3n/2) + O(3n/2) + O(n) + O(n/2) = \underline{O(n)}$

3 - We know that the inorder traversal of BST generates a sorted (non-decreasing) array of the elements present in the BST.

- So, using Inorder traversal, we get two arrays representing the elements of the trees.

- We compare both the arrays, if they are same, then both BSTs have same set of elements.

- Given a BST, with n elements, the inorder traversal takes $O(n)$ time, And, comparing the two sorted arrays also takes $O(n)$ time.

$\therefore$ Time taken to compare two BSTs $= O(n) + O(n)$
$= O(n)$.

```
Inorder Traverse (root):

        Inorder Traverse (root → left)
        Print value of node
        Inorder Traverse (root → right)
```

(4) Mr. Lazyrus make a tweak in red-Black trees, Such that a node is not completely deleted to make deletion operation easier. Instead hes adds a boolean variable which denotes whether the node is active or not (i.e deleted or not)

Effectiveness:

- As mentioned above, the node is not completely deleted, so it still has the property of colour.. Therefore, it maintains the colour property of the RB tree after deletion and no extra rearrangements are required after deletion.
- Insertion of a new node can be done the same way as before, considering the colours of the nodes.
- As all elements are distinct, we cannot insert a node with the same value as that of a deleted (inactive) node. So we search the same way as before and search among active nodes to search a node.

Inaffectiveness:

- There is wastage of space as the nodes we delete are not really deleted and only the boolean is changed to inactive. It still uses the same space after deletion, and we cannot reuse this space.

⇒ If there are too many inactive (deleted) nodes, then insertion and Searching would be a lot more time consuming than normal RB tree.

5

For Insertion:-

- Rotations :- o(1) [at most 2 rotations]

Rotation happens when the uncle is of black colour. There are 4 types of rotations, {LL, LR, RR, RL}. At most 2 rotations happen.

- Recolouring:- O(log(n))

Recolowring happens when the Parent and uncle are of Red colour. we recolor Parent, uncle and grandparent and then recur on grand-Parent. This can repeat $h/2$ times, where h is height of tree, hence recoulering :- o(logn)

For Deletion:

- Rotations :- O(1) [at most 3 rotations]

At most 3 rotations happen in every case, and we do not have to recur on any Parent like in the case of recolouring.

- Re-coloring - O(logn)

Recoloring happens when Sibling and both it's children are black. This can repeat h times, where h is height of the tree, and hence there are o(logn) recolorings.

6 To find the Successor of a node in a Red Black Tree, we follow the following Procedure:-

- If the right Subtree of node is NULL, then successor lies in the right Subtree. So, go the the right Subtree and return the node with minimum key value in the right Subtree.
- If the Right Subtree is NULL, then Successor is one of the ancestors. So, we travel up the Parent Pointer until we see a node which is left child of it's Parent. The Parent of such a node is successor.