Coding + Theory Assignment-3

Name: Krishnan.R
Roll no: CS24BTECH11036

Coding - Algorithm Proof

5) Verification: Our goal is to output all keys $K \in T(x)$ such that $a \leq K \leq b$.
Let $K = x \rightarrow data$.
Since, we are implementing the algorithms recursively, we need a base case. The base case considered is when x = nullptr / external node where we return nothing.
Recursion: If key $> a$, it means some keys in the left subtree of root can have values between a, b. Hence, we recursively call the function for root → left.
If $a \leq$ key $\leq b$, then we print it.
If key $> b$, then it means some keys in the right subtree of root can have values between a, b. Hence, we recursively call the function for root → right.
Since the recursive calls are in inorder, the keys are printed in ascending order.
Hence, we print all keys as we traverse through the left, right subtree of every node in the tree until we hit an external / non-existent node.

Time complexity = $\Theta(m + \log n)$ proof:
We visit a node only if it lies on a path where we searching for nodes which match the condition $a \leq$ node → data $\leq b$ or it satisfies the condition itself and that is when we print it.
∴ We visit m nodes which satisfy the condition ~~and~~ + $K \log n$, $K \in N$ which are the no of nodes on any path from root to external for K paths we traverse. Since visiting each node is $O(1)$, we have time taken $\leq m + K \log n$ (Using Theorem 14.1 of CLRS which states inorder traversal visits each node only once)

Since we must output n keys, time = $\Omega(m)$
If $n=0$, we need to traverse atleast one path, time = $\Omega(\log n)$
$\therefore T(n) = \Omega(m+\log n)$ and $T(n) = O(m+\log n)$
$\therefore T(n) = \Theta(m+\log n)$

Theory

1) Conditions a, b are satisfied by RB-Trees.
If we look at condition c, it is always satisfied by RB-Trees even after recoloring as recoloring doesn't increase the number of red nodes and hence satisfies RB-Tree properties.
Condition d is already satisfied by RB Trees, even after recoloring as recoloring just increases black depth of all nodes by 1. Since, black depth were equal in the relaxed tree before, they remain the same even after recoloring.
$\therefore$ Tree is an RB-Tree.

2) ~~If t~~ Proof: Let the black height of an ext node x be bh(x), counting only no of internal black nodes.
Min path length must be bh(x) by definition if the path contains only black nodes.
We know that, a red node can't have a red child and the number of black nodes is fixed.
So, the only way to maximize the path is to alternate between red and black nodes giving no of red nodes = no of internal black nodes = bh(x). Therefore, the max path length = 2 bh(x).
$\therefore$ Longest simple path from node x in RB-Tree to a descendent leaf has a length at most twice that of the shortest simple path from node x to a descendent leaf.

3) Consider the tree: 30
/
20

Insert 25, we get 30 , we get an unbalanced AVL-Tree.
/ \
20 NULL
\
25

If we perform a single rotation (LL case), we get which is still unbalanced.

20
\
30
/
25

If we perform another type of single rotation (RR case), we get NULL) in fact RR doesn't do anything useful, gives us a non-existing tree.

30
/
20
\
25

∴ Single rotation doesn't solve the balance issue.
∴ We perform double rotation:
1) Left rotate 20 about 25 to get 30
/
25
\
20

2) Right rotate 30 about 25 (new subtree root) to get 25
/ \
20 30
, which is a balanced tree.
∴ Single cannot always restore balance.