CS301 Assignment 3

Ahmet Bilal Yildiz

November 2022

1 Standard RBT

To use in the comparisons in question1 and question2, insertion algorithm and fix-coloring algorithm in the insertion algorithm in standard RBT are explained here.

Insertion algorithm in standard RBT:

$RBTinsert(T, x)$:
    $color[x] = red$ $\Theta(1)$
    $BSTinsert(T, x)$ $\Theta(h)$
    $FixColoring(T, x)$ $\Theta(h)$

Time complexity for the algorithm:
$\Theta(1) + \Theta(h) + \Theta(h) = \Theta(h)$ where h = $logn$ (height of the tree).
Therefore; asymptotic time-complexity of the insertion operation in a RBT is $\Theta(logn)$.

FixColoring algorithm in the insertion algorithm in standard RBT:

FixColoring algorithm fixes the colors of the RBT after the insertion operation to ensure the all RBT properties. To do this, this algorithm ensures property 4 which is a red node has two black children and so the algorithm deals with the red-red (parent-child) node cases. And all other properties are already satisfied since we add the new node as a red node.

FixColoring algorithm deal with 3 cases:

Case1: If the uncle of the child in the red-red pair(parent-child) is also red, and the child is the right child of the parent in the pair, the algorithm pushes the black color of the grandparent down on to the parent and uncle (a visualization can be seen in the figure2). And this step is repeated when the same conditions occurs in the upper levels again until the reach to the node. If node is red at the end of the step, node will assign to the black to ensure RBT property.

Worst-case Running time:
Worst-case occurs when the pushing operation repeated until reaching the root.
In each iteration changing the color is $\Theta(1)$.
Total number of iterations = the height of the RBT which is $logn$.
Therefore, the since $\Theta(1)$ is repeated $logn$ times the time-complexity $= \Theta(logn)$.

Case2: If the uncle of the child in the red-red pair(parent-child) is black, and the child is the right child of the parent, and parent is the left child of the grandparent, the algorithm left rotates around the parent in the pair and the tree becomes the case3.Then the algorithm right rotates around the parent (a visualization can be seen in the figure4). And this is a symmetric case.

Running-time:
Since the rotation operation means that doing 3 assignment operations with pointers in the nodes, the running time of a single rotation is $\Theta(1)$. Therefore, running time of 2 rotation is again in $\Theta(1)$ time.

Case3: If the uncle of the child in the red-red pair(parent-child) is black, and the child is the left child of the parent, and parent is the left child of the grandparent, the algorithm right rotates around the parent in the pair (a visualization can be seen in the figure5).

Running-time:
Since the rotation operation means that doing 3 assignment operations with pointers in the nodes, the running time of the rotation is $\Theta(1)$.

2 Question1

To compute the black-height of a given node in a red-black tree (RBT) in constant time, we use a black height augmented version of the RBT. And then, we investigate the changes in the time-complexity of the insertion operation.

In the below figure the augmented RBT can be seen with a black-height additional info in the nodes.

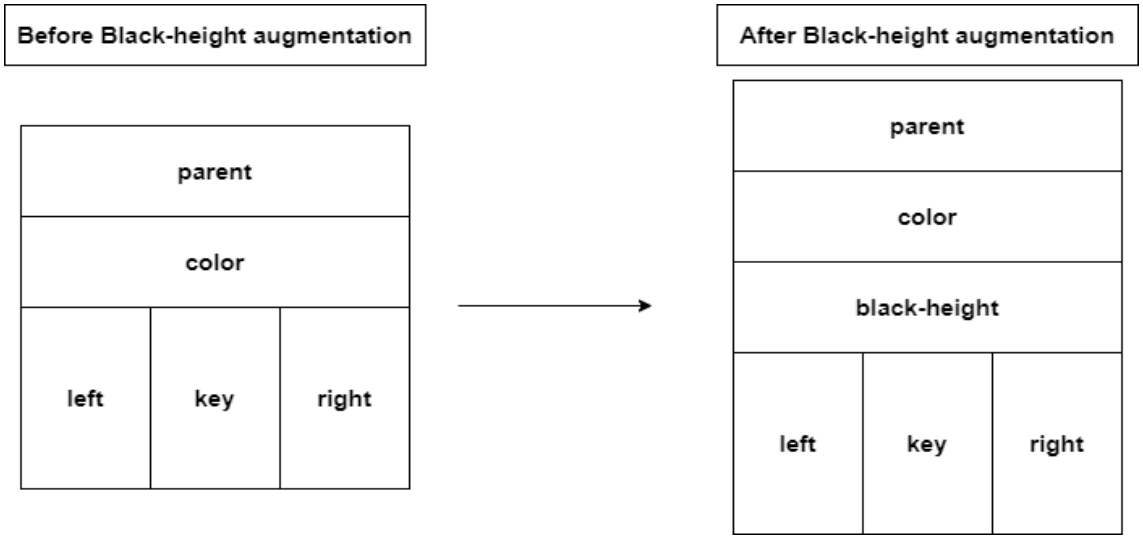

Figure 1: Black Height Augmentation to the RBT

Changes in the Insertion algorithm: When the augmented RBT is considered, there is no change in the first 2 steps of the standard insertion algorithm, since the new node first pushes as a red node, the black-heights of each node does not change. However, the changed part is the FixColoring part, since the black-heights of the nodes must be updated after the insertion operation in the new black-height augmented RBT.

Changes in the Fix-coloring algorithm: When the augmented RBT is considered, since the color of any node do not change in the rotations and RBT property 5 is preserved which is "black-heights in any path mus be equal to each other", the black-height of nodes does not changes. Therefore, the worst-case occurs when the case1

occurs and the color of nodes are changes repeatedly until the algorithm reaches to the root.

Worst-case running time in Black-height augmented RBT:

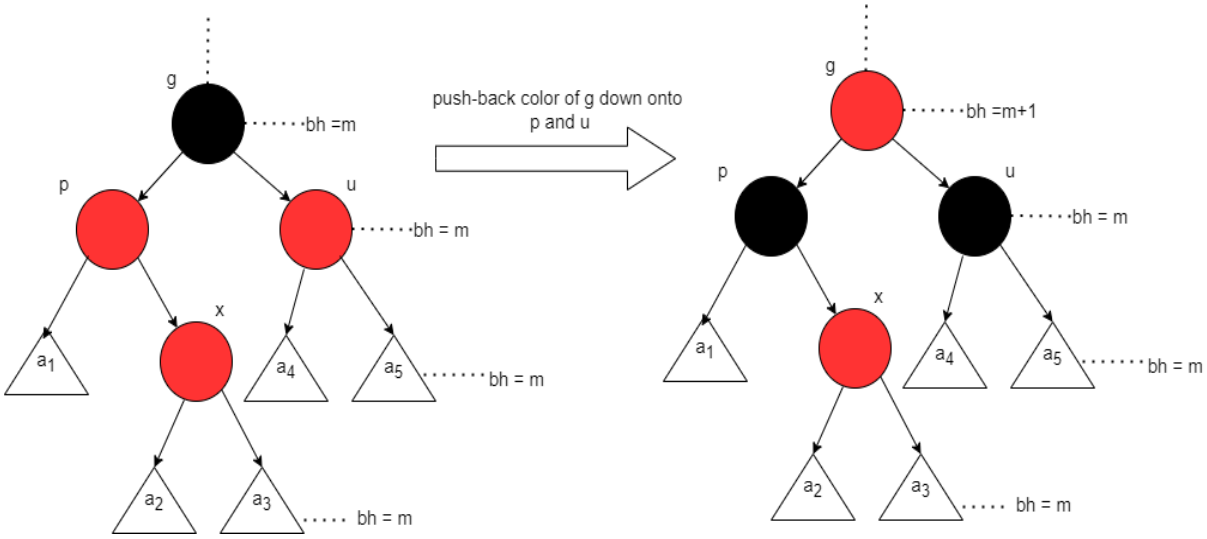

Figure 2: Fix-Coloring Case 1 in Black-Height Augmented RBT

Numerical Example: Normally, these figures represent a part of a whole RBT. However, to make it a numerical example, we can think the above process is the insertion of x(9) to the tree of g(10) which is the root, p(8), u(12), and a_1, a_2 a_3 a_4 a_5 null black leaves. After insertion, all the changes in colors can be seen from the figure and changes in black heights are same with below where m = 1, and black-heights of the leaf nodes = 0.

Above figure is a visualization of the fix-coloring case1 (the worst-case) in a part of the black-height augmented tree. All of the information about black-heights of the nodes are represented on the right side after the dashed line with bh, and m represents the black-height of the nodes which are the parents in the continued parts represented with triangles. As it can be seen, black-heights of the nodes p, u, x, and all the nodes in the triangular parts are not change after the fix-coloring operation. However, the only change is the g which is the root of the current node.

In short, all the changes in the standard implementation of the RBT continues with addition to the updating the black-height info of the top-most node iteratively until the algorithm reaches to the root. Updating a nodes black-height info runs in $\Theta(1)$ time because it is just an assignment operation which is:

if the child node is a red node:
black-height of the parent node = black-height of the child node
if the child node is a black node: black-height of the parent node = black-height of the child node +1

Worst-case running time:
In each iteration:
change the color $\Theta(1)$
update the black-height info of the top-most node $\Theta(1)$
Total number of iterations = the height of the RBT which is $logn$.
Since $\Theta(1)$ is repeated $logn$ times the time-complexity = $\Theta(logn)$.

Conclusion: Since all other parts in the insertion operation are same with the standard version and black-height augmentation does not change running time of the fix-coloring part.

Asymptotic time-complexity of insertion: $\Theta(1) + \Theta(h) + \Theta(h) = \Theta(n)$ where h denotes the height of the node which is$logn$, and time complexity = $\Theta(logn)$

Therefore, the black-height augmentation to the RBT does not change the asymptotic time complexity of the insertion operation.

3 Question2

To compute the depth of a given node in a red-black tree (RBT) in constant time, we use a depth augmented version of the RBT. And then, we investigate the changes in the time-complexity of the insertion operation.

In the below figure the augmented RBT can be seen with a black-height additional info in the nodes.

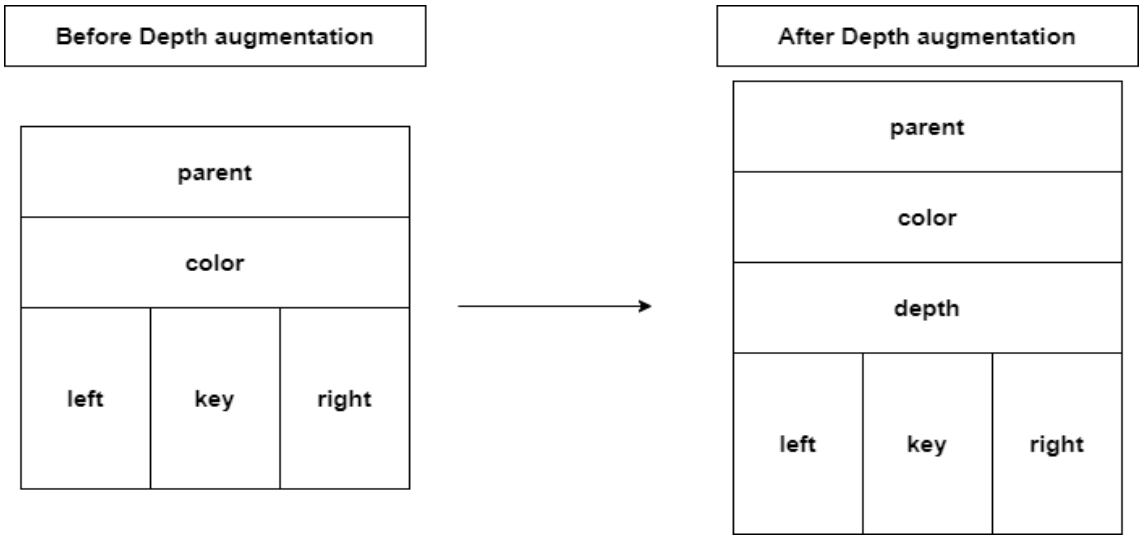

Figure 3: Black Height Augmentation to the RBT

Changes in the Insertion algorithm: When the augmented RBT is considered, there is no change in the first 2 steps of the insertion operation in the standard RBT algorithm, since the new node is pushed to the bottom, the depths of each node does not change. However, the changed part is the FixColoring part, since the depths of the nodes must be updated after the insertion operation in the new depth augmented RBT.

Changes in the Fix-coloring algorithm: When the augmented RBT is considered, since all changes are the colors of the nodes in case1, the depths of nodes do not change. However, in case2 or case3 the structure of the RBT changes so depths of the nodes change. Therefore, the worst-case occurs when the case2 occurs (since it comprises case3) and the RBT left-Rotate and right-Rotate.

Worst-case running time in Depth augmented RBT:

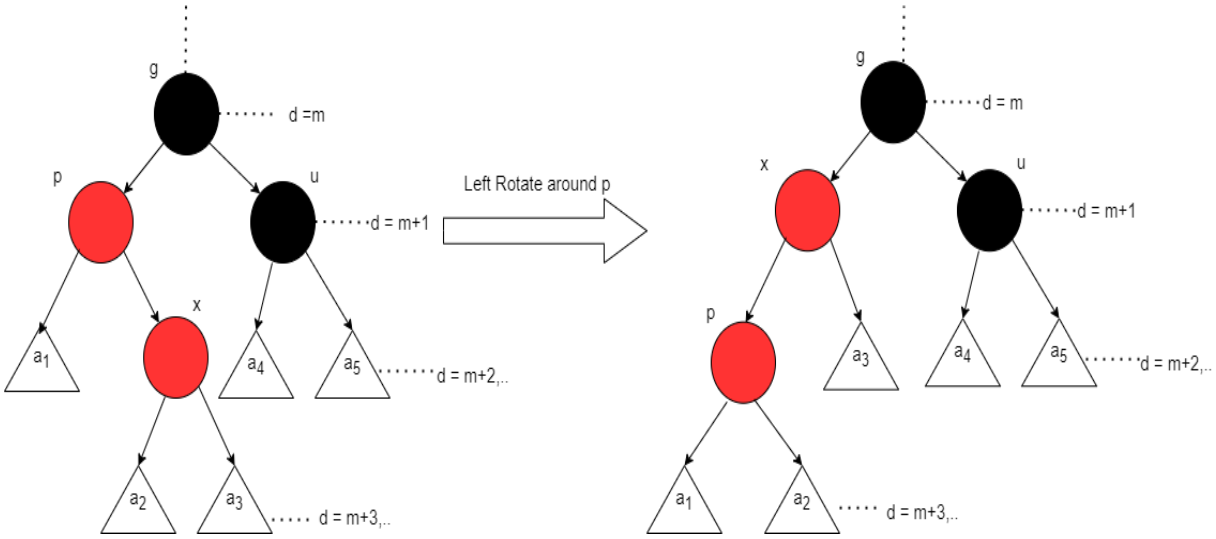

Figure 4: Fix-Coloring Case 2 Left Rotate in Depth Augmented RBT

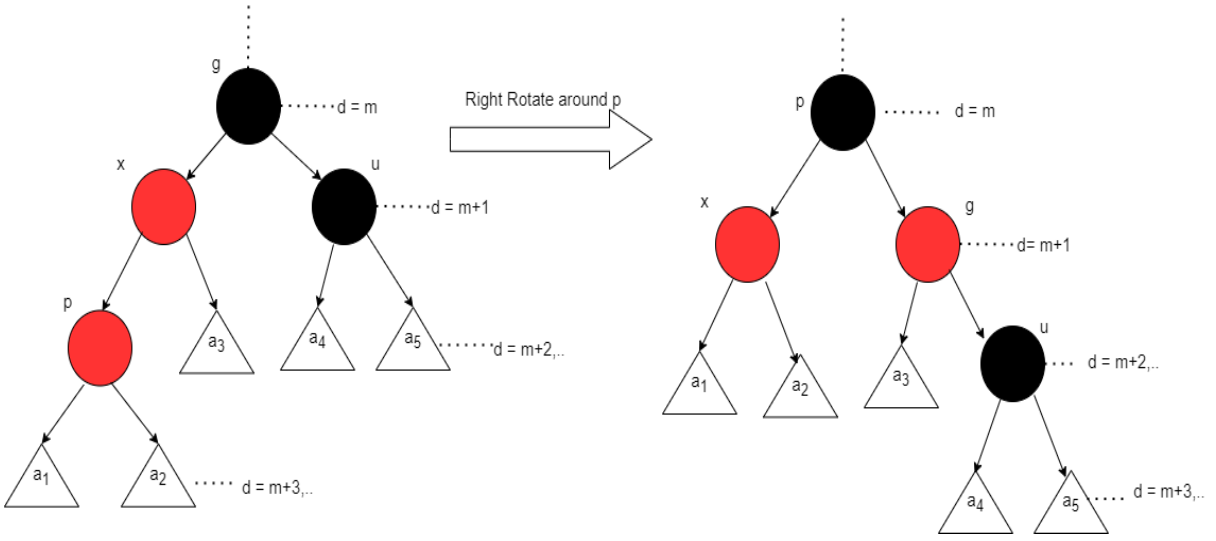

Figure 5: Fix-Coloring Case 2 Right Rotate in Depth Augmented RBT

Numerical Example: Normally, these figures represent a part of a whole RBT. However, to make it a numerical example, we can think the above process is the insertion of x(9) to the tree of g(10) which is the root, p(8), u(12), and a_1, a_2 a_3 a_4 a_5 null black leaves. After the insertion operation the RBT becomes, p(8) which is the root, x(9), g(10), u(12), and a_1, a_2 a_3 a_4 a_5 null black leaves.The depths and depth changes are same where m = 0.

Above two figures is a visualization of the fix-coloring case2 (the worst-case) in a part of the depth augmented RBT. All of the information about depths of the nodes are represented on the right side after the dashed line with d, and m represents the depth of the topmost node. As it can be seen, depths of the most of the nodes change in rotations: g($m- > m+1$), p($m+1- > m$), u($m+1- > m+2$), x($m+2- > m+1$), all the nodes in the a_2, a_3 triangle decreases by 1, all nodes in a_4, a_5 triangle increases by 1. Update from here to end .
In short, all the changes in the standard implementation of the RBT continues with addition to the updating the depth info of the almost all nodes(all nodes-nodes in $a_1 triangle$ which is approximated to the all nodes so n).
Updating depth info of a node runs in $\Theta(1)$ time because it is just an assignment operation which is:
Black-height of the child node = black-height of the parent node+1

Worst-case running time:

do the left and right rotations $\Theta(1)$
update the depth info of almost all nodes approximated to all nodes (n nodes) $\Theta(n)$
updating the depth of a single node $\Theta(1)$
Since $\Theta(1)$ is repeated n times the time-complexity for the updating depth info = $\Theta(n)$.
Because of the time-complexity of updating depth info = $\Theta(n)$, time-complexity of fix-coloring = $\Theta(n)$.

Conclusion: Since all other parts in the insertion operation are same with the standard version and depth augmentation changes the running time of the fix-coloring part to $\Theta(n)$,

Asymptotic time-complexity: $\Theta(1) + \Theta(h) + \Theta(n) = \Theta(n)$ where n denotes the total number of nodes in the RBT.

Therefore, the depth augmentation to the RBT change the asymptotic time complexity of the insertion operation from $\Theta(logn)$ to $\Theta(n)$.