Algorithm assignment 6

Nisharg Gosai

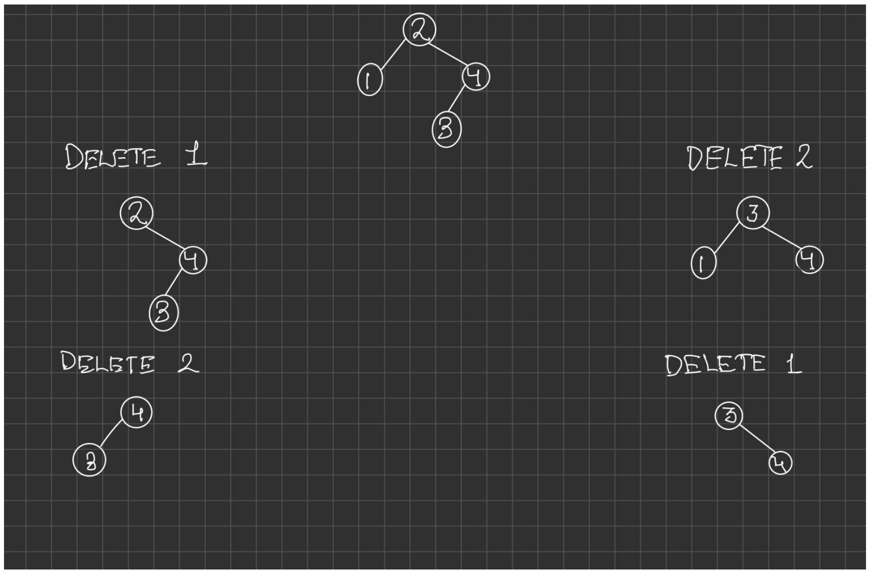
Problems

1. (20pts) Write in pseudocode a recursive version of the TREE-INSERT procedure for binary search trees.

```
RECURSIVE-TREE-INSERT(root, z):
1.  if root == NIL:
2.      return z
3.  if z.key < root.key:
4.      root.left = RECURSIVE-TREE-INSERT(root.left, z)
5.  else if z.key > root.key:
6.      root.right = RECURSIVE-TREE-INSERT(root.right, z)
7.  return root
```

2. (20pts) Is the operation of deletion “commutative” in the sense that deleting x and then y from a binary search tree leaves the same tree as deleting y and then x? Argue why it is or give a counterexample.

The operation of deletion is not commutative which means that deleting x and then y from a binary search tree does not leave the same tree as deleting y and then x. The example below shows that:



The BST at the end of deleting 1 and then 2 is different than the BST at the end of deleting 2 and then 1.

3. (10pts) Define a relaxed red-black tree as a binary search tree that satisfies red-black properties 1,3,4, and 5, but whose root may be either red or black. Consider a relaxed red-black tree T whose root is red. If the root of T is changed to black but no other changes occur, is the resulting tree a red-black tree? What happens to the black heights?

The resulting tree is a red-black tree. Consider the following properties of red-black trees:

Property 1 (Node Colors): The tree either has red or black nodes. Since no new colors are introduced in the transformation, this property remains valid.
Property 2 (Root Color): Originally, the relaxed red-black tree violates this property by allowing the root to be red. By changing the root to black, we now satisfy this property, aligning with the standard red-black tree requirements.
Property 3 (Leaf Nodes): Every leaf (NIL) is black. This property was already satisfied in the relaxed red-black tree and remains satisfied after the transformation.
Property 4 (Red Node Children): If a node is red, then both its children are black. No new red nodes are introduced, and changing the root color to black does not affect the color of other nodes or their children. Therefore, this property remains valid.
Property 5 (Black Height): From each node, all paths to the descendant leaves contain the same number of black nodes. In the original relaxed red-black tree, this property holds. Changing the root from red to black adds one black node to every path from the root to the leaves. However, since the root is a part of every path, this increase in black nodes is uniform across all paths. Therefore, the relative black heights between different paths remain the same, and this property still holds in the transformed tree.

Impact on Black Heights: The black height of a node is the number of black nodes on the path from the node to its descendant leaves. By changing the root color to black, the black height of the entire tree increases by one. However, this increase is uniform across all paths from the root, maintaining the required balance of black heights across different paths. Before the change, if the black height was h, after the change, it becomes h+1. This uniform increase preserves the property that all paths from any node to its descendant leaves have the same number of black nodes.

In summary, by changing the root of a relaxed red-black tree from red to black, the resulting tree satisfies all the properties of a standard red-black tree. The transformation increases the black height of the tree uniformly, ensuring the structure remains balanced and adheres to the red-black tree rules.

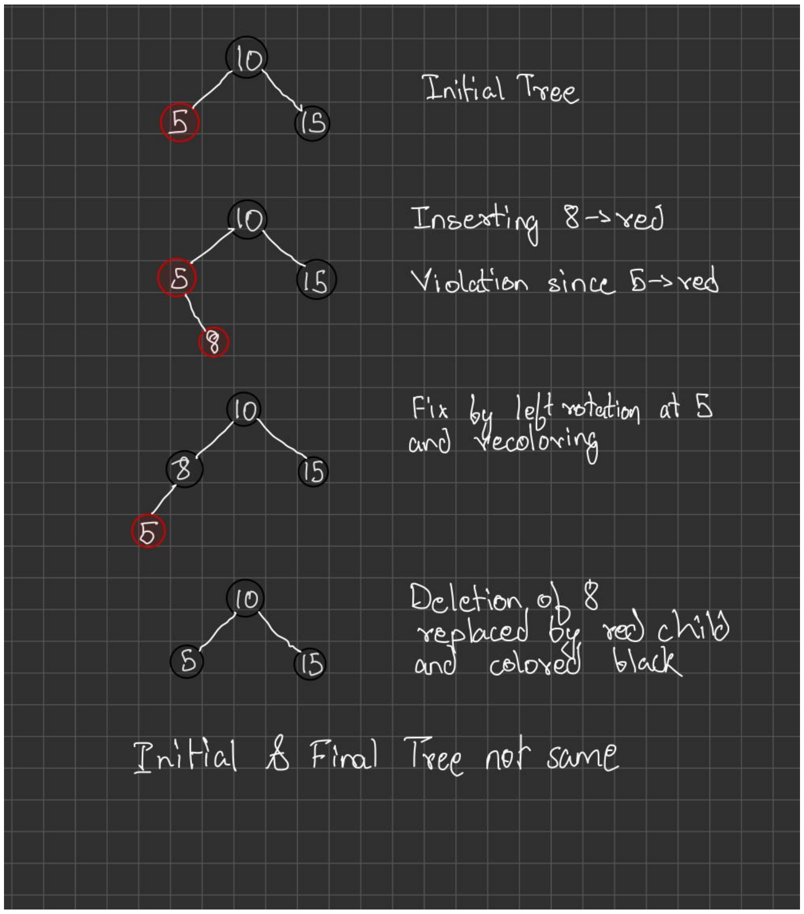
4. (15pts) Write the pseudocode for RIGHT-ROTATE.

```
RIGHT-ROTATE(Tree, PivotNode):
1.      NewRoot = PivotNode.left
2.      PivotNode.left = NewRoot.right
3.      if NewRoot.right != Tree.nil
4.              NewRoot.right.parent = PivotNode
5.      NewRoot.parent = PivotNode.parent
6.      if PivotNode.parent == Tree.nil
7.              Tree.root = NewRoot
8.      else if PivotNode == PivotNode.parent.left
9.              PivotNode.parent.left = NewRoot
10.     else
11.             PivotNode.parent.right = NewRoot
12.     NewRoot.right = PivotNode
13.     PivotNode.parent = NewRoot
```

5. (20pts) A node x is inserted into a red-black tree with RB-INSERT and then immediately deleted with RB-DELETE. Is the resulting red-black tree always the same as the initial red-black tree? Justify your answer.

No, the resulting red-black tree is not the same as the initial red-black tree.
The operations RB-INSERT and RB-DELETE can perform complex transformations on the tree to maintain its red-black properties. Because these operations may involve rotations and recoloring that are not necessarily symmetric (i.e., the insertion fixes are not always exactly reversed by the deletion fixes), the resulting tree after a deletion may not be structurally identical to the original tree, even if the set of values it contains is the same. Thus, the resulting red-black tree is not always the same as the initial red-black tree after a node is inserted and then deleted.

Take the following example,



6. (20pts) Show, using a counterexample, that the following “greedy” strategy does not always determine an optimal way to cut rods. Define the density of a rod of length i to be pi/i, that is, its value per inch. The greedy strategy for a rod of length n cuts off a first piece of length i, where 1 <= i<= n, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length n – i.

To show that the greedy strategy based on density $\frac{p_i}{i}$ does not always determine an optimal way to cut rods, consider the following counterexample:

Let's say we have a price table as follows:

Length(i)	1	2	3	4
Price(p_i)	1	5	8	10

The density of each length:

Density	1/1 = 1	5/2 = 2.5	8/3 = 2.67	10/4 = 2.5

Suppose we want to cut a rod of length 4 to maximize the price:

Using GREEDY,
First cut off a piece of length 3 because it has the highest density of 2.67
This leaves a piece of length 1
The price obtained would be 8+1 = 9

However, the optimal strategy would be to cut two pieces of length 2 which would give us a price of 2*5 = 10

The price achieved by the greedy strategy is 9, while the optimal strategy gives us a price of 10. Therefore, the greedy strategy does not always lead to an optimal solution for cutting rods based on the maximum density criterion.