INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

CSN-261 : Data Structures Laboratory

Assignment-3

Name : **Abhishek Patil**

Enr. No. : **18114003**

E-mail : **apatil@cs.iitr.ac.in**

Problem 1 :

Given the set of integers, write a C++ program to create a binary search tree (BST) and print all possible paths for it. You are not allowed to use subarray to print the paths. Convert the obtained BST into the corresponding AVL tree for the same input. AVL tree is a selfbalancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property

Algorithms Used :

Insertion in binary search tree:

Insert function is used to add a new element in a binary search tree at appropriate location. Insert function is to be designed in such a way that, it must node violate the property of binary search tree at each value.

1.Allocate the memory for tree.

2.Set the data part to the value and set the left and right pointer of tree, point to NULL.

3.If the item to be inserted, will be the first element of the tree, then the left and right of this node will point to NULL.

4.Else, check if the item is less than the root element of the tree, if this is true, then recursively perform this operation with the left of the root.

5.If this is false, then perform this operation recursively with the right sub-tree of the root.

Insertion in AVL tree:

Insertion in AVL tree is performed in the same way as it is performed in a binary search tree. The new node is added into AVL tree as the leaf node. However, it may lead to violation in the AVL tree property and therefore the tree may need balancing.

The tree can be balanced by applying rotations. Rotation is required only if, the balance factor of any node is disturbed upon inserting the new node, otherwise the rotation is not required.

Depending upon the type of insertion, the Rotations are categorized into four categories.

SN	Rotation	Description
1	LL Rotation	The new node is inserted to the left sub-tree of left sub-tree of critical node.
2	RR Rotation	The new node is inserted to the right sub-tree of the right sub-tree of the critical node.
3	LR Rotation	The new node is inserted to the right sub-tree of the left sub-tree of the critical node.
4	RL Rotation	The new node is inserted to the left sub-tree of the right sub-tree of the critical node.

Insertion in AVL tree:

In Red-Black tree, we use two tools to do balancing.

1) Recoloring

2) Rotation

We try recoloring first, if recoloring doesn't work, then we go for rotation. Following is detailed algorithm. The algorithms has mainly two cases depending upon the color of uncle. If uncle is red, we do recoloring. If uncle is black, we do rotations and/or recoloring.

Color of a NULL node is considered as BLACK.

Let x be the newly inserted node.

1) Perform standard BST insertion and make the color of newly inserted nodes as RED.

2) If x is root, change color of x as BLACK (Black height of complete tree increases by 1).

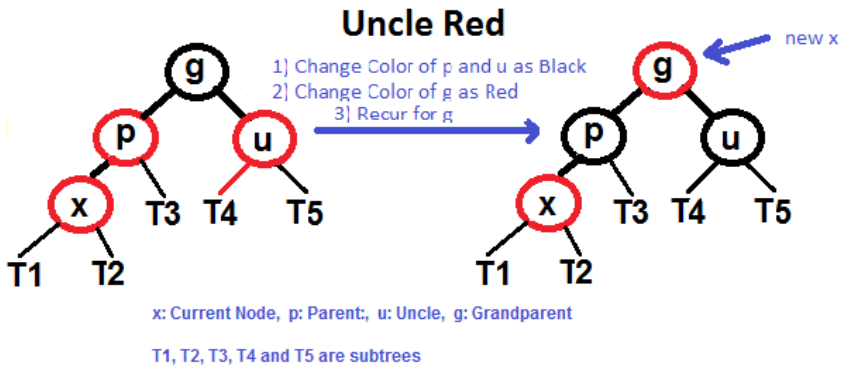
3) Do following if color of x's parent is not BLACK and x is not root.

....a) If x's uncle is RED (Grand parent must have been black from property 4)

.....(i) Change color of parent and uncle as BLACK.

.....(ii) color of grand parent as RED.

.....(iii) Change x = x's grandparent, repeat steps 2 and 3 for new x.



....b) If x's uncle is BLACK, then there can be four configurations for x, x's parent (p) and x's grandparent (g) (This is similar to AVL Tree)

.....i) Left Left Case (p is left child of g and x is left child of p)

.....ii) Left Right Case (p is left child of g and x is right child of p)

.....iii) Right Right Case (Mirror of case i)

.....iv) Right Left Case (Mirror of case ii)

Following are operations to be performed in four subcases when uncle is BLACK.

Data Structure Used :

1. Binary search tree

2. AVL Tree

3. Red Black Tree

Snapshots :

```
abhi@abhi-laptop:~/Desktop/DataStructureAssignment/Assignment3/Question1$ ./a.out
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
1
10
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
1
20
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
1
30
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
1
40
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
1
50
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
1
25
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
```

```
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
2
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
3
        1. inorder of BST
        2. inorder of AVL Tree
        3. inorder of red-black tree
1
10 20 25 30 40 50
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
3
        1. inorder of BST
        2. inorder of AVL Tree
        3. inorder of red-black tree
2
10 20 25 30 40 50
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
3
        1. inorder of BST
        2. inorder of AVL Tree
        3. inorder of red-black tree
3
10 20 25 30 40 50
```

```
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
4
        1. paths of BST
        2. paths of AVL Tree
        3. paths of red-black tree
2
30->20->10->NULL
20->10->NULL
10->NULL

30->20->25->NULL
20->25->NULL
25->NULL

30->40->50->NULL
40->50->NULL
50->NULL

1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
4
        1. paths of BST
        2. paths of AVL Tree
        3. paths of red-black tree
3
20->10->NULL
10->NULL

20->40->30->25->NULL
40->30->25->NULL
30->25->NULL
25->NULL

20->40->50->NULL
40->50->NULL
50->NULL
```

```
5
        1. level-wise indentation of BST
        2. level-wise indentation of AVL Tree
        3. level-wise indentation of red-black tree
1
10[4]
        20[3]
                30[1]
                        25
                        40[1]
                                50
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
5
        1. level-wise indentation of BST
        2. level-wise indentation of AVL Tree
        3. level-wise indentation of red-black tree
2
30[0]
        20[0]
                10
                25
        40[1]
                50
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
5
        1. level-wise indentation of BST
        2. level-wise indentation of AVL Tree
        3. level-wise indentation of red-black tree
3
20[2][b]
        10[b]
        40[1][r]
                30[1][b]
                        25[r]
                50[b]
1. To insert a node in the BST and in the red-black tree.
2. To create AVL tree from the inorder traversal of the BST.
3. To print the inorder traversal of the BST/AVL/red-black tree.
4. To display all the paths in the BST/AVL tree/red-black tree.
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation.
6. Exit.
6
cpu time used is 0.004728
abhi@abhi-laptop:~/Desktop/DataStructureAssignment/Assignment3/Question1$
```

Problem 2 :

For a given sequence of positive integers A1, A2, …, AN in decimal, find the triples (i, j, k), such that $1 \leq i < j \leq k \leq N$ and $A_i \oplus A_{i+1} \oplus \ldots \oplus A_{j-1} = A_j \oplus A_{j+1} \oplus \ldots \oplus A_k$, where ⊕ denotes bitwise XOR. This problem should be solved using dynamic programming approach and linked list data structures.

Data Structure Used :

1. Stack
2. Linked List

Algorithms Used :

if xor of two sub array are equal then, their xor is zero.

So, we have to find a sub array whose xor is zero. Then we can break that aub array at any place, then we get two subaray whose xor will be equal.

Snapshots :

```
abhi@abhi-laptop:~/Desktop/DataStructureAssignment/Assignment3/Question2$ ./a.out
3
2 5 7
(1,2,3)
(1,3,3)
Total number of triplets are : 2
cpu time used is 0.000381
abhi@abhi-laptop:~/Desktop/DataStructureAssignment/Assignment3/Question2$
```