L3 Report

Name : Priyanshu Garg
Branch: Computer Science and Engineering
Batch: O3
Enrollment No. : 18114058

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.

Convert the obtained BST into the corresponding red-black tree for the same input. Red-Black Tree is a self-balancing Binary Search Tree (BST) where every node follows following rules.
1) Every node has a color either red or black.
2) Root of tree is always black.
3) There are no two adjacent red nodes (A red node cannot have a red parent or red child).
4) Every path from a node (including root) to any of its descendant NULL node has the same number of black nodes.

Data Structures:

I used the concept of trees and make Binary Search Tree, AVL Tree, Red_Black Tree.I also use stack in the process of recusion.

Algorithm:

Firstly, I make a common node for all 3 trees and store the numbers in the trees according to the properties of the trees. I used recusion principle in almost every function as trees have this property of recusrion.
For printing path, first I call find_leaf function which find thee leaf and instantly call print_path function, in it I assign temp pointer to the root an apply loop until program reaches the leaf node by shifting the temp pointer accordingly, after reacing leaf node, program reasign root value to the temp and shift temp according to the leaf node and call print_path again, this will continue until root reaches leaf node, due to the recursion find_leaf is again called and same process repeats until all leaf nodes are covered.
For printing nodes with indentation, I used the value of node->level and do intendation accordingly.
For BST, program writes height in [], for AVL it is balance factor, for RBT it is height and color.

Screenshots:

```
priyanshu@Kratos:~/DS_LABS$ g++ problem1.c++
priyanshu@Kratos:~/DS_LABS$ ./a.out
Enter number of terms
6
Enter numbers:
10 20 30 40 50 25
1. To insert a node in the node and in the red-black tree
2. To create AVL tree from the inorder traversal of the node
3. To print the inorder traversal of the node/AVL/red-black tree
4. To display all the paths in the node/AVL tree/red-black tree
5. To print the node/AVL tree/red-black Tree in the terminal using level-wise indentation
6. Exit
Enter option:
2
1. To insert a node in the node and in the red-black tree
2. To create AVL tree from the inorder traversal of the node
3. To print the inorder traversal of the node/AVL/red-black tree
4. To display all the paths in the node/AVL tree/red-black tree
5. To print the node/AVL tree/red-black Tree in the terminal using level-wise indentation
6. Exit
Enter option:
3
Inorder Transversal: 10 20 25 30 40 50
```

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

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

    20[3]

        30[2]
            25[1]
            40[1]

                50[0]
AVL Tree:
30[0]
    20[0]
        10[0]
        25[0]
    40[-1]

        50[0]
Red-Black Tree:
20[4][1]
    10[3][1]
    40[3][0]
        30[2][1]
            25[1][0]

        50[1][1]
```

```
1. To insert a node in the node and in the red-black tree
2. To create AVL tree from the inorder traversal of the node
3. To print the inorder traversal of the node/AVL/red-black tree
4. To display all the paths in the node/AVL tree/red-black tree
5. To print the node/AVL tree/red-black Tree in the terminal using level-wise indentation
6. Exit
Enter option:
6
CPU time: 1.98 ms
Real time passed: 34895.89 ms
```

Problem 2:

For a given sequence of positive integers A1, A2, …, AN in decimal, find the triples (i, j, k), such that $1 \le i < j \le k \le N$ 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 structures:

I used array and linked list. In array program used 2D array to store the prefix xor with their stating and ending index.

Algorithms:

Firstly, Program makes a 2D array for dynamic programming and store the prefix xor after this we search a cell with zero entry and end index > start index and

count the difference them to count the number of triplets. Program search zero prefix xor as when prefix xor is zero the we can divide that subarray into 2 parts such that their xor is equal.

i.e. B^C == 0

=> B = C

Similar concept I used for linked list, but instead of 2D array I used head, bottom and righr pointer, where head poijnter stores the starting index of that subarray and right pointer stores the prefix xor starting from that index.

Screenshots:

```
priyanshu@Kratos:~/DS_LABS$ g++ problem2.c++
priyanshu@Kratos:~/DS_LABS$ ./a.out
Enters number of terms:
9
Enter numbers:
23 6 85 41 23 69 85 47 1
From dynamic programming:
Number of triplets: 5
Triplets are:
(4,5,9)
(4,6,9)
(4,7,9)
(4,8,9)
(4,9,9)

From Linked List:
Number of triplets: 5
Triplets are:
(4,5,9)
(4,6,9)
(4,7,9)
(4,8,9)
(4,9,9)
CPU time: 0.67 ms
Real time passed: 15285.52 ms
priyanshu@Kratos:~/DS_LABS$
```