Indian Institute of Technology Roorkee
Department of Computer Science and Engineering

CSN-261: Data Structures Laboratory (Autumn 2019-2020)

Lab Assignment – 3

Name: Hemil Sanjaybhai Panchiwala

Enrollment No.: 18114031

Branch: Computer Science & Engineering

Sub Batch: O2

Problem Statement 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.

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.

Write a menu driven program as follows:

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 (print color for red-black tree)
6. Exit

Data Structures used :-

Array, Trees

Algorithm

- **In this code, I have inserted nodes one by one in Binary Search Tree and Red Black Tree.**

Insertion in BST:

- **Whenever an element is to be inserted, first locate its proper location.**
- **Start searching from the root node, then if the data is less than the insert value, search for the empty location in the left subtree and insert the data.**
- **Otherwise, search for the empty location in the right subtree and insert the data.**

Insertion in RB Tree:

- **Insert as you would into a BST, coloring the node red.**
- **If the parent of the node you just inserted was red, you must correct with zero or more recolorings followed by zero or one restructuring.**
- **Color the root node black.**

- **Then, I have converted the BST Tree inorder to AVL Tree**

Insertion in AVL Tree:

- **Insert the element in the AVL tree in the same way the insertion is performed in BST.**
- **After insertion, check the balance factor of each node of the resulting tree.**

CASE-1:

- **After the insertion, the balance factor of each node is either 0 or 1 or -1.**
- **In this case, the tree is considered to be balanced.**
- **Conclude the operation.**
- **Insert the next element if any.**

CASE-2:

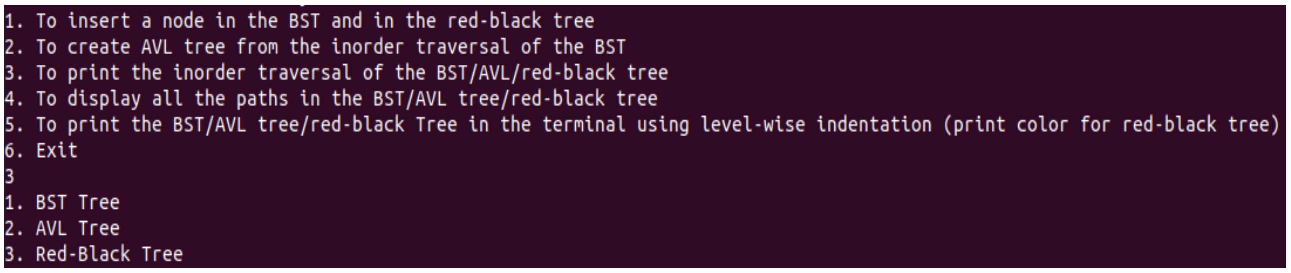
- **After the insertion, the balance factor of at least one node is not 0 or 1 or -1.**
- **In this case, the tree is considered to be imbalanced.**
- **Perform the suitable rotation to balance the tree.**
- **After the tree is balanced, insert the next element if any.**

Then, I have applied logics for printing the inorder traversal, all paths and the trees with proper indentation.

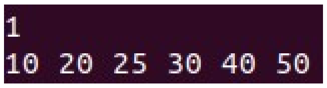
Snapshots

After insertion of 10,20,30,40,50 and 25 in BST

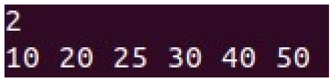
- ### Inorder of Trees



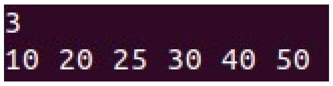
```
1. To insert a node in the BST and in the red-black tree
2. To create AVL tree from the inorder traversal of the BST
3. To print the inorder traversal of the BST/AVL/red-black tree
4. To display all the paths in the BST/AVL tree/red-black tree
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation (print color for red-black tree)
6. Exit
3
1. BST Tree
2. AVL Tree
3. Red-Black Tree
```



```
1
10 20 25 30 40 50
```

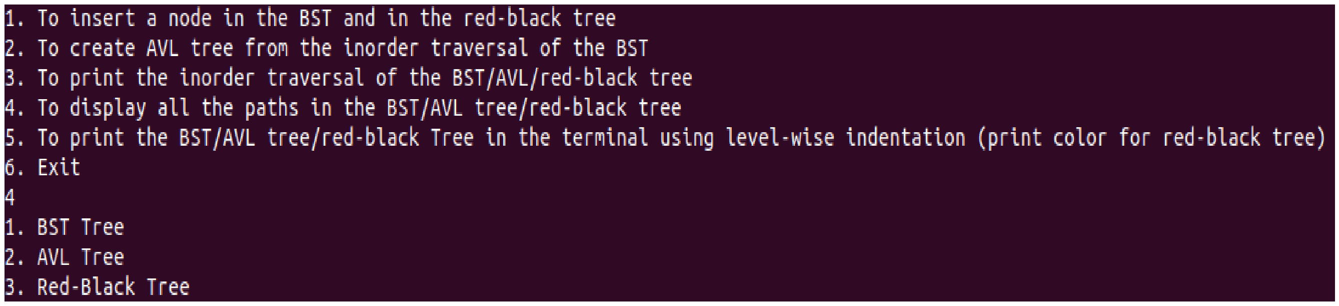


```
2
10 20 25 30 40 50
```

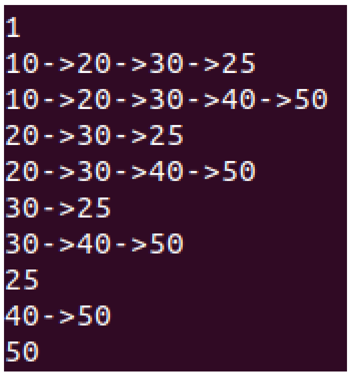


```
3
10 20 25 30 40 50
```

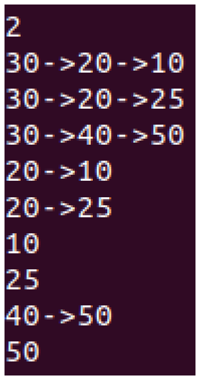
- ### All paths of Trees



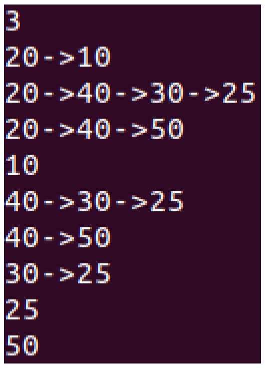
```
1. To insert a node in the BST and in the red-black tree
2. To create AVL tree from the inorder traversal of the BST
3. To print the inorder traversal of the BST/AVL/red-black tree
4. To display all the paths in the BST/AVL tree/red-black tree
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation (print color for red-black tree)
6. Exit
4
1. BST Tree
2. AVL Tree
3. Red-Black Tree
```



```
1
10->20->30->25
10->20->30->40->50
20->30->25
20->30->40->50
30->25
30->40->50
25
40->50
50
```

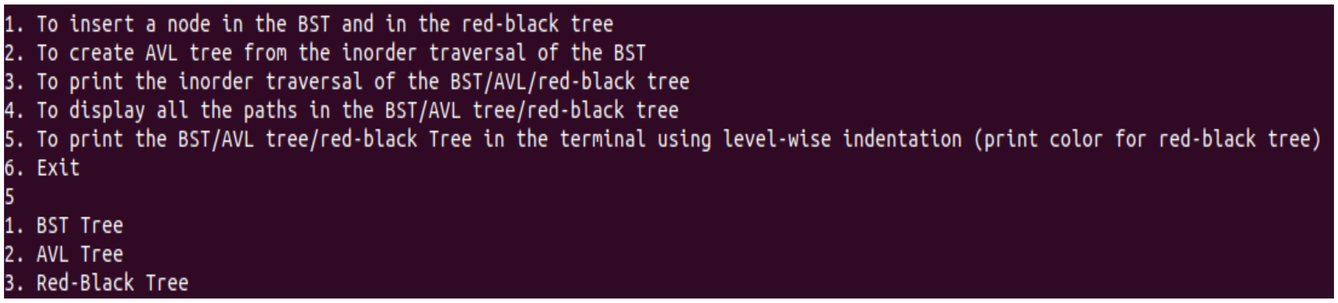


```
2
30->20->10
30->20->25
30->40->50
20->10
20->25
10
25
40->50
50
```

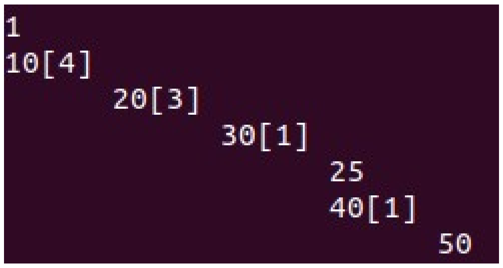


```
3
20->10
20->40->30->25
20->40->50
10
40->30->25
40->50
30->25
25
50
```

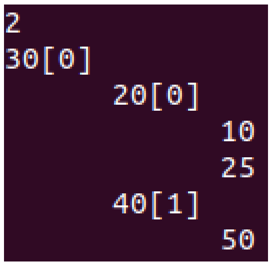
- **Print all trees using level-wise indentation**



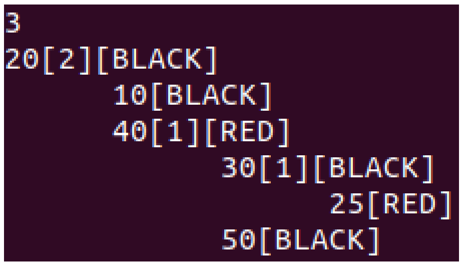
```
1. To insert a node in the BST and in the red-black tree
2. To create AVL tree from the inorder traversal of the BST
3. To print the inorder traversal of the BST/AVL/red-black tree
4. To display all the paths in the BST/AVL tree/red-black tree
5. To print the BST/AVL tree/red-black Tree in the terminal using level-wise indentation (print color for red-black tree)
6. Exit
5
1. BST Tree
2. AVL Tree
3. Red-Black Tree
```



```
1
10[4]
        20[3]
                30[1]
                        25
                        40[1]
                                50
```



```
2
30[0]
        20[0]
                10
                25
        40[1]
                50
```



```
3
20[2][BLACK]
        10[BLACK]
        40[1][RED]
                30[1][BLACK]
                        25[RED]
                50[BLACK]
```

Time taken: 0.004338 s

Problem Statement 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 $\oplus$ denotes bitwise XOR. This problem should be solved using dynamic programming approach and linked list data structures.

Input:

(a) Number of positive integers N.

(b) N space-separated integers A1, A2, …, AN.

Output:

Print the number (count) of triples and list all the triplets in lexicographic order (each triplet in a new line).

Data Structures used :-

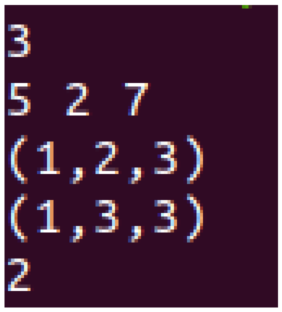
Linked List

Algorithm

- **Here first I created linked list with elements having value equal to xor of all its previous elements.**
- **Then in this linked list, first I have checked whether any element has value equal to 0 then we have to print all subarray’s index from 1 to that index in linked list where node value is 0.**
- **Then, I have checked that if two values in the linked list are equal (suppose at index p and q) then print all subarrays of index from p to q ((p,i,q) where p<i<=q).**
- **And I have given the total number of subarrays having such property as answer.**

Snapshots

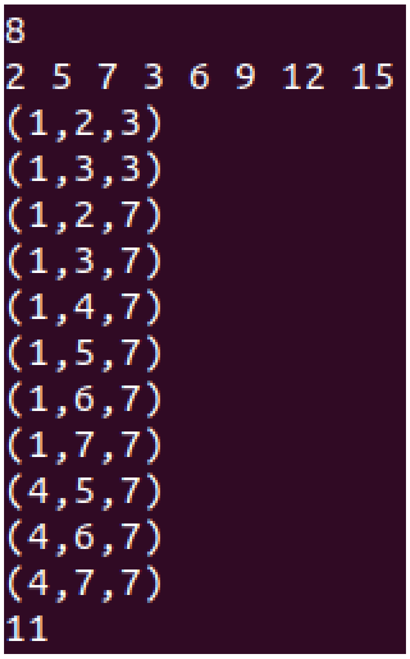
- **Test Case 1:**



```
3
5 2 7
(1,2,3)
(1,3,3)
2
```

Time taken: 0.000295 s

- **Test Case 2:**



```
8
2 5 7 3 6 9 12 15
(1,2,3)
(1,3,3)
(1,2,7)
(1,3,7)
(1,4,7)
(1,5,7)
(1,6,7)
(1,7,7)
(4,5,7)
(4,6,7)
(4,7,7)
11
```

Time taken: 0.000342 s