National Institute of Technology, Calicut Department of Computer Science and Engineering CS2094D – Data Structures Lab (MCA)

Assignment-3

Policies for Submission and Evaluation:

You must submit your assignment in the moodle (Eduserver) course page, on or before the submission deadline. Also, ensure that your programs in the assignment must compile and execute without errors in Andromeda server. During evaluation your uploaded programs will be checked in Andromeda server only. Failure to execute programs in the assignment without compilation errors may lead to zero marks for that program. Detection of ANY malpractice regarding the lab course will also lead to awarding an F grade.

Naming Conventions for Submission:

Submit a single ZIP (.zip) file (do not submit in any other archived formats like .rar or .tar.gz).

The name of this file must be ASSG<NUMBER>_<ROLLNO>_<FIRSTNAME>.zip (For example: ASSG3_MxxyyyyCA_LAXMAN.zip). DO NOT add any other files (like temporary files, input files, etc.) except your source code, into the zip archive.

The source codes must be named as ASSG <NUMBER> _<ROLLNO> _<FIRSTNAME> _ <PROGRAM-NUMBER> .<extension> (For example: ASSG3_MxxyyyyCA_LAXMAN_1.c). If there is a part ‘a ‘ and a part ‘b’ for a particular question, then name the source files for each part separately as in ASSG3_MxxyyyyCA_LAXMAN_1b.c. Make sure that you follow the naming conventions.

Standard of Conduct:

Violations of academic integrity will be severely penalized.

Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work MUST BE an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or

evaluation and will be reported to the department council for record keeping and for permission to assign F grade in the course.

The department policy on academic integrity can be found at:

http://cse.nitc.ac.in/sites/default/files/Academic-Integrity.pdf

ASSIGNMENT QUESTIONS

General Instructions:

- The input should be read from the file **input.txt** and output should be written to the file **output.txt**.

Q1. Create a Red-Black tree (RBTree). The input should be read from the file ***input.txt*** and output should be printed to ***output.txt.*** Structure of each node contains an additional color field to indicate its color.

The following functions should be performed on the RBTree:

1. insert(tree, element) – adds the node specified by element (which contains the data) into the RBTree specified by tree.

2. delete (tree, element) – removes the node specified by element from the RBTree specified by tree. Replace the node with its inorder successor if exists otherwise choose inorder predecessor.

3. search(tree, key) – searches for the data specified by key in the RBTree specified by tree.

4. findMin(tree) – retrieves the smallest data in the RBTree specified by tree.

5. findMax(tree) – retrieves the largest data in the RBTree specified by tree.

6. predecessor(tree, element) – retrieves the inorder-predecessor of the node specified by element in the RBTree specified by tree.

7. successor(tree, element) – retrieves the inorder-successor of the node specified by element in the RBTree specified by tree.

8. inorder (tree) – To do a recursive inorder traversal of the RBTree. Print the nodes with their color.

9. preorder (tree) – To do a recursive preorder traversal of the RBTree. Print the nodes with their color.

10. postorder (tree) - To do a recursive preorder traversal of the RBTree. Print the nodes with their color.

Input Format:

The input consists of multiple lines, each containing a four letter string followed by zero or one integer. The meaning of each line is given below:

- String '**stop**' means stop the program.
- String "**insr**" means, create a node which contains the next integer(>= 0) from the input as the data part, and then, insert this node into the RBTree. In this case, the data is given on the same line as the string "insr", separated by a space.
- String "**delt**" means delete the data specified by the next integer(>= 0) in the input from the RBTree, if present. In this case, the data is given on the same line as the string "delt", separated by a space. (Here, the data to be deleted is guaranteed to be present in the RBTree).
- String "**srch**" means, search for the key specified by the next integer(>= 0) from the input, in the RBTree. In this case, the key is given on the same line as the string "srch", separated by a space. If the search is successful, output "FOUND". Otherwise, output "NOT FOUND".
- String "**minm**" means find and output the minimum value in the RBTree. Print "NIL" if the RBTree is empty.
- String "**maxm**" means find and output the maximum value in the RBTree. Print "NIL" if the RBTree is empty.
- String "**pred**" means find and output the inorder-predecessor of the data specified by the next integer(>= 0) from the input. In this case, the data is given on the same line as the string "**pred**", separated by a space. Output "NIL", if the data exists in the tree, but there is no inorder-predecessor for the data. Output "NOT FOUND", if the data is not present in the tree.
- String "**succ**" means find and output the inorder-successor of the data specified by the next integer(>= 0) from the input. In this case, the data is given on the same line as the string "succ", separated by a space. Output "NIL", if the data exists in the tree, but there is no inorder-successor for the data. Output "NOT FOUND", if the data is not present in the tree.
- String "**inor**" means output the in-order traversal of the RBTree.
- String **"prer"** means output the pre-order traversal of the RBTree.

- String **“post”** means output the post-order traversal of the RBTree.

Output Format:

The output (if any) of each command should be printed on a separate line.

Note: Strictly follow the output format . It should be NIL, NOT FOUND, FOUND. Color field can have one of the two values: **‘B’** for Black node , **‘R’** for Red node.

Sample Input	Sample Output
srch 10	NOT FOUND
minm	NIL
maxm	NIL
pred 10	NOT FOUND
succ 10	NOT FOUND
insr 10	
srch 10	FOUND
minm	10
maxm	10
pred 10	NIL
succ 10	NIL
insr 20	
insr 30	
insr 15	
inor	10B 15R 20B 30B
prer	20B 10B 15R 30B
post	15R 10B 30B 20B
stop	

Q2. Create an AVL Tree. The input should be read from the file ***input.txt*** and output should be printed to ***output.txt.*** The following functions should be performed on the AVLTree:

1. insert(tree, element) – adds the node specified by element (which contains the data) into the AVL tree specified by tree. **Note** assume all the values are distinct , no duplicate is present in the input.

2. delete (tree, element) – removes the node specified by element from the AVL tree specified by tree. Replace the node with its inorder successor if exists otherwise choose inorder predecessor.

3. search(tree, key) – searches for the data specified by key in the AVL tree specified by tree.

4. findMin(tree) – retrieves the smallest data in the AVL tree specified by tree.

5. findMax(tree) – retrieves the largest data in the AVL tree specified by tree.

6. predecessor(tree, element) – retrieves the inorder-predecessor of the node specified by element in the AVL tree specified by tree.

7. successor(tree, element) – retrieves the inorder-successor of the node specified by element in the RBTree specified by tree.

8. inorder (tree) – To do a recursive inorder traversal of the RBTree.

9. preorder (tree) – To do a recursive preorder traversal of the RBTree.

10. postorder (tree) - To do a recursive preorder traversal of the RBTree.

Input Format:

The input consists of multiple lines, each containing a four letter string followed by zero or one integer. The meaning of each line is given below:

- String '**stop**' means stop the program.
- String "**insr**" means, create a node which contains the next integer(>= 0) from the input as the data part, and then, insert this node into the AVL tree. In this case, the data is given on the same line as the string "insr", separated by a space.
- String "**delt**" means delete the data specified by the next integer(>= 0) in the input from the AVL tree, if present. In this case, the data is given on the same line as the string "delt", separated by a space. (Here, the data to be deleted is guaranteed to be present in the AVL tree).
- String "**srch**" means, search for the key specified by the next integer(>= 0) from the input, in the AVL tree. In this case, the key is given on the same line as the

string “srch”, separated by a space. If the search is successful, output “FOUND”. Otherwise, output “NOT FOUND”.

- String “**minm**” means find and output the minimum value in the AVL tree. Print “NIL” if the AVL tree is empty.
- String “**maxm**” means find and output the maximum value in the AVL tree. Print “NIL” if the AVL tree is empty.
- String “**pred**” means find and output the inorder-predecessor of the data specified by the next integer(>= 0) from the input. In this case, the data is given on the same line as the string “**pred**”, separated by a space. Output “NIL”, if the data exists in the tree, but there is no inorder-predecessor for the data. Output “NOT FOUND”, if the data is not present in the tree.
- String “**succ**” means find and output the inorder-successor of the data specified by the next integer(>= 0) from the input. In this case, the data is given on the same line as the string “succ”, separated by a space. Output “NIL”, if the data exists in the tree, but there is no inorder-successor for the data. Output “NOT FOUND”, if the data is not present in the tree.
- String “**inor**” means output the in-order traversal of the AVL tree.
- String **“prer”** means output the pre-order traversal of the AVL tree.
- String **“post”** means output the post-order traversal of the AVL tree.

Output Format:

The output (if any) of each command should be printed on a separate line.

Note: Strictly follow the output format . It should be NIL, NOT FOUND, FOUND

Sample Input	Sample Output
srch 10	NOT FOUND
minm	NIL
maxm	NIL
pred 10	NOT FOUND
succ 10	NOT FOUND
insr 10	
srch 10	FOUND
minm	10

maxm	10
pred 10	NIL
succ 10	NIL
insr 20	
insr 30	
insr 15	
inor	10 15 20 30
prer	20 10 15 30
post	15 10 30 20
delt 10	
stop	

Q3. Once Monk was watching a fight between an array and a tree, of being better. Tree got frustrated and converted that array into a Binary Search Tree by inserting the elements as nodes in BST, processing elements in the given order in the array. Now Monk wants to know the height of the created Binary Search Tree.

Help Monk for the same.

Note:

1) In Binary Search Tree, the left subtree contains only nodes with values less than or equal to the parent node; the right subtree contains only nodes with values greater than the parent node.

2) Binary Search Tree with one node, has height equal to 0.

Input Format :

The first line will consist of *1* integer *N*, denoting the number of elements in the array.

In the next line, there will be *N* space separated integers denoting the elements of array.

Output Format :

Print the height of the created Binary Search Tree.

Sample Input	Sample Output
4 2 1 3 4	2

Q4. Monk has an array *A* having *N* distinct integers and a Binary Search Tree which is initially empty. He inserts all the elements of the array from index *1* to *N* in the BST in the order given in the array. But wait! The tree so formed turns out to be cursed. Monk is having some weird experiences since he made that tree.

So, now to stop all that, Monk has two options, to destroy the BST or to pray to God and ask for a solution. Now since Monk has to use this BST in a Code Monk Challenge, he cannot destroy it. So he prays to God.

God answer his prayers and sends an angel named Micro. Now, Micro asks Monk to recreate the BST. He tells him to treat array elements as **preorder** traversal of the tree and create BST again.

Now since, Monk is very afraid of that tree he asks for your help.

Input Format :

The first line will consist of *1* integer *N*, denoting the number of elements in the array.

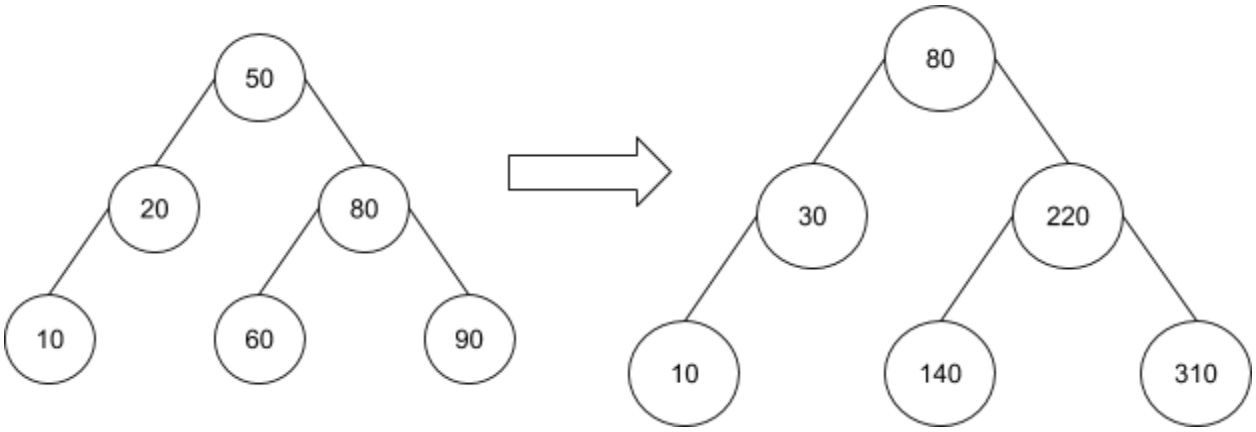
In the next line, there will be *N* space separated integers denoting the elements of array.

Output Format :

Print the postorder and inorder traversal of the BST in separate lines.

Sample Input	Sample Output
6 10 5 1 7 40 50	1 7 5 50 40 10 1 5 7 10 40 50

Q5. Monk has an array *A* having *N* distinct integers and a Binary Search Tree which is initially empty. He inserts all the elements of the array from index *1* to *N* in the BST in the order given in the array. Now Monk wants to convert this BST to another BST in which **every node contains a summation of all the lesser nodes including itself**. You can see this from the following figure.



Input Format :

The first line will consist of *1* integer *N*, denoting the number of nodes in BST.

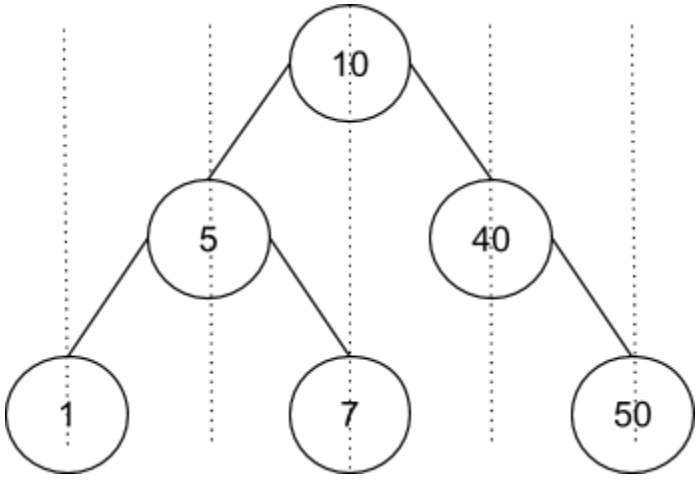
In the next line, there will be *N* space separated integers denoting nodes of BST.

Output Format :

Print preorder traversal and post order traversal of the new BST .

Sample Input	Sample Output
6 50 80 60 90 20 10	80 30 10 220 140 310 10 30 140 310 220 80

Q6. Given a binary search tree perform vertical traversal of it. In vertical traversal, we print nodes of a binary search tree in vertical order assuming that the left and right child of a node makes a 45 degree angle with the parent.



Input Format :

The first line will consist of *1* integer *N*, denoting the number of nodes in BST.

In the next line, there will be *N* space separated integers denoting nodes of BST.

Output Format :

Print vertical traversal of BST from left to right, top to bottom.

Sample Input	Sample Output
6 10 5 40 7 1 50	1 5 10 7 40 50