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

Assignment-2

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 binary search tree (BST).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 BST:

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

2. search(tree, key) – searches for the data specified by key in the BST specified by tree.

3. findMin(tree) – retrieves the smallest data in the BST specified by tree.

4. findMax(tree) – retrieves the largest data in the BST specified by tree.

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

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

7. delete (tree, element) – removes the node specified by element from the BST specified by tree.

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

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

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

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 BST. In this case, the data is given on the same line as the string "insr", separated by a space.
- String "**srch**" means, search for the key specified by the next integer(>= 0) from the input, in the BST. 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 BST. Print "NIL" if the BST is empty.
- String "**maxm**" means find and output the maximum value in the BST. Print "NIL" if the BST 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 "**delt**" means delete the data specified by the next integer(>= 0) in the input from the BST, 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 BST).
- String "**inor**" means output the in-order traversal of the BST.
- String **"prer"** means output the pre-order traversal of the BST.
- String **"post"** means output the post-order traversal of the BST.

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 25	NOT FOUND
minm	NIL
maxm	NIL
pred 25	NOT FOUND
succ 25	NOT FOUND
insr 25	
srch 25	FOUND
minm	25
maxm	25
pred 25	NIL
succ 25	NIL
insr 13	
insr 50	
insr 45	
insr 55	
insr 18	
srch 10	NOT FOUND
srch 13	FOUND
srch 35	NOT FOUND

srch 55	FOUND
srch 80	NOT FOUND
inor	13 18 25 45 50 55
prer	25 13 18 50 45 55
post	18 13 45 55 50 25
minm	13
maxm	55
pred 13	NIL
succ 13	18
pred 18	13
succ 18	25
pred 25	18
succ 25	45
pred 45	25
stop	

Q2. 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 find something. He tells him two values, *X* and *Y*, present in the BST and ask him to find the maximum value that lie in the path between nodes having value *X* and node having value *Y*. (including *X* and *Y*).

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

Input Format:

- First line consists of a single integer denoting *N*.
- Second line consists of *N* space separated integers denoting the array *A*.
- Third line consists of two space separated integers denoting *X* and *Y*.

Output Format:

- Print the maximum value that lie in the path from node having value *X* and node having value *Y* in a new line.

Sample Input	**Sample Output**

5
4 7 8 6 3
3 7
	7

Q3. Given a complete binary tree with *N* nodes and each node have a distinct integer *ai* attached with it, find the minimum number of swaps you can make to convert the binary tree into binary search tree. In one swap, you can select any two nodes and swap their values.

You will be given the array representation of the binary tree. Root of the tree will be at A[0], Left child of root will be at A[1] and right child of root will be at A[2]. Left child of node at array position *k* will be at A[2*k + 1] and right child of node at array position *k* will be at A[2*k + 2].

Input Format :

First line contains an integer, *N* , denoting the number of nodes. Second line contains *N* space separated integers denoting the value attached to the ith node.

Output Format :

Print a single integer, denoting the minimum number of swaps needed to convert binary tree into a binary search tree.

Sample Input	**Sample Output**
3	1

1 2 3	

Q4. Huffman coding assigns variable length codewords to fixed length input characters based on their frequencies. More frequent characters are assigned shorter codewords and less frequent characters are assigned longer codewords. All edges along the path to a character contain a code digit. If they are on the left side of the tree, they will be a *0* (zero). If on the right, they'll be a *1* (one). Only the leaves will contain a letter and its frequency count. All other nodes will contain a null instead of a character, and the count of the frequency of all of it and its descendant characters.

For instance, consider the string *ABRACADABRA*. There are a total of 11 characters in the string. This number should match the count in the ultimately determined root of the tree. Our frequencies are A = 5, B = 2, R = 2, C = 1, and D = 1 . The two smallest frequencies are for C and D , both equal to 1 , so we'll create a tree with them. The root node will contain the sum of the counts of its descendants, in this case 1 + 1 = 2. The left node will be the first character encountered, C, and the right will contain D. Next we have 3 items with a character count of 2 : the tree we just created, the character B and the character R. The tree came first, so it will go on the left of our new root node. B will go on the right. Repeat until the tree is complete, then fill in the 1's and 0's for the edges.

Create 2 functions in your program:

- **create_tree(C, F):** Function will accept two arrays containing characters and their respective frequencies. Function will create the tree requires for huffman encoding and will return a pointer to the root node.
- **encode_huff(tree, data):** Using the tree, function will encode the string and prints the encoded string to output file.

Input Format :

- First line will contain two integers ‘N’ and T integer representing number of characters and test cases respectively.
- Next N lines will contain character and their frequencies separated by single space
- Next T lines will contain strings that will be encoded

Note: while assigning values to tree edges. Assign ‘0’ for left edge and ‘1’ for right edge

Output Format :

For each test case, print a single line of encoded string.

Sample Input	Sample Output
3 3 A 3 B 1 C 1 ABC CAB ABACA	10001 01100 1001011

Q5. Given a Binary Search Tree (BST), delete the kth largest node from it and print a level order traversal of the resulting BST.

Input Format :

- First line of input contains number of nodes 'N' followed by k (k^{th} largest node)
- 2nd line of input contains space separated keys for BST

Output Format :

- You have to print the level-order traversal of BST obtained after deletion of the kth largest element.

Sample Input	Sample Output
6 2 10 8 12 15 11 7	10 8 15 7 11 or 10 8 11 7 15

Q6. You will be given inorder and preorder traversals of a binary tree. Construct the binary tree using these traversals and print its postorder traversal.

Input Format :

- 1^{st} contains integer denoting the number of nodes in the tree.
- 2^{nd} line contains inorder traversal of binary tree.
- 3^{rd} line contains preorder traversal of binary tree.

Output Format :

- You have to print the postorder traversal of reconstructed binary tree.

Sample Input	**Sample Output**

8
4 8 2 5 1 6 3 7
1 2 4 8 5 3 6 7
	8 4 5 2 6 7 3 1

Q7. Given a binary search tree find a pair with given sum present in it.

Input Format :

- First line contains two integers ‘N’ and ‘t’ denoting number of nodes and test cases respectively.
- 2nd line contains space separated keys for BST.
- Next ‘t’ lines contain an integer denoting the sum for which you have to find the pair.

Output Format :

- For each test case, print pair in a single line
- If there is no such pair, print **NOT POSSIBLE**

Sample Input	**Sample Output**

11 3
15 10 25 8 12 20 30 6 9 18 22
14
21
15
	8 6
	NOT POSSIBLE
	6 9