National Institute of Technology, Calicut
Department of Computer Science and Engineering
Monsoon 2019-20
CS2094D – Data Structures Lab

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 Athena server. During evaluation your uploaded programs will be checked in Athena server only. Failure to execute programs in the assignment without compilation errors may lead to zero marks for that program.

Your submission will also be tested for plagiarism, by automated tools. In case your code fails to pass the test, you will be straightaway awarded zero marks for this assignment and considered by the examiner for awarding F grade in the course. 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: ASSG4_BxxyyyyCS_LAXMAN.zip). DO NOT add any other files (like temporary files, inputfiles, 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: ASSG4_BxxyyyyCS_LAXMAN_1.c). If there are multiple parts for a particular question, then name the source files for each part separately as in

ASSG4_BxxyyyyCS_LAXMAN_1b.c.

If you do not conform to the above naming conventions, your submission might not be recognized by some automated tools, and hence will lead to a score of 0 for the submission. So, 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 an F grade in the course. The department policy on academic integrity can be found at:

http://minerva.nitc.ac.in/cse/sites/default/files/attachments/news/Academic-Integrity_new.pdf .

Assignment Questions

1. Write a program to create a Binary Search Tree (BST). Your program should include the following functions.

main() - Repeatedly read one of the strings "insr", "srch", "minm", "maxm", "pred", "succ", "delt", "inor", "prer" and "post" from the terminal and call corresponding subfunctions until the string "stop" is read.

insert(*tree*, *element*) – adds the node specified by ***element*** (which contains the data) into the BST specified by ***tree***. If the element already exists in the tree, insert it to the left subtree.

search(*tree, element*) - Search of the data specified by ***element*** in the tree ***tree***. If found, return a pointer to the corresponding node; otherwise, return NULL. If there are multiple nodes that contain ***element***, return that node which has the smallest level.

level(*tree, node*) - Find the node ***node*** in the tree ***tree***. If found, return the level of node in the tree. Otherwise, return **-1**.

findMin(*tree*) – returns the smallest data in the BST specified by ***tree***.

findMax(*tree*) – returns the largest data in the BST specified by ***tree***.

predecessor(*tree, node)* – Find and return a pointer to the inorder-predecessor of the node specified by ***node*** in the BST specified by ***tree***. Return -1, if the element exists in the tree, but there is no inorder-predecessor for the node. Return NULL , if the data is not present in the tree.

successor(*tree, node*) – Find and return a pointer to the inorder-successor of the node specified by ***node*** in the BST specified by ***tree***. Return -1, if the element exists in the tree, but there is no inorder-successor for the data. Return NULL, if the data is not present in the tree.

delete(*tree, node*) – Find and remove the node specified by ***node*** from the BST specified by ***tree***. If not found, return NULL. Replace the deleted node with its inorder successor if exists otherwise choose inorder predecessor.

inorder(*tree*) – does a recursive inorder traversal of the BST.

preorder(*tree*) – does a recursive preorder traversal of the BST.

postorder(*tree*) – does a recursive postorder traversal of the BST.

Input format

Each line of the input contains one of the following:

1. The string "**insr**" followed by an integer ***i*** : Call function **insert(*tree*, *i*)**
2. The String "**srch**" followed by an integer ***k*** : Search of a node with key **k**. If found, print its level, otherwise print **-1** (assume **root** had level **0**).
3. The String "**minm**" : Print the output of **findMin(*tree*).** Print "NIL" if the BST is empty.
4. The String "**maxm**" :Print the output of **findMax(*tree*).** Print "NIL" if the BST is empty.
5. The String "**pred**" followed by an integer ***i***: print the output of **predecessor(*tree, i*)**
6. The String "**succ**" followed by an integer ***i*** : print the output of **successor(*tree, i*)**
7. The String "**delt**" followed by an integer ***i***: print the output of **delete(*tree, element*)**
8. The String "**inor**" : print the output of inorder(tree)
9. The String "**prer**" : print the output of preorder(tree)
10. The String "**post**" : print the output of postorder(tree)

Output format

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

Sample Input

```
srch 25
minm
maxm
pred 25
succ 25
insr 25
srch 25
minm
maxm
pred 25
succ 25
insr 13
insr 50
insr 45
insr 55
insr 18
srch 10
inor
prer
post
```

delt 55
delt 13
delt 25
minm
maxm
stop

Sample Output

NOT FOUND
NIL
NIL
NULL
NULL
FOUND
25
25
-1
-1
NOT FOUND
13 18 25 45 50 55
25 13 18 50 45 55
18 13 45 55 50 25
18
50

2. The Parenthesis Representation of a binary tree is recursively defined, as given below.

- The string **( )** represents an empty tree. That is, a left bracket followed by a **single space** followed by a right bracket.
- The string **(k *left-subtree right-subtree*)** represents a tree whose root node has key **k**, ***left-subtree*** is the left subtree of the root node in Parenthesis Representation and ***right-subtree*** is the right subtree of the root node in Parenthesis Representation. That is, a left bracket followed by a **single space** followed by the key value **k** followed by a **single space** followed by the parenthesis representation of **left-subtree** followed by a **single space** followed by the parenthesis representation of **right-subtree** followed by a **single space** and a right bracket.

Add a function **paren(tree)** to your program for Question 1. The function should take as input a pointer to the root node of a binary search tree **tree** and print the tree in its Parenthesis Representation.

Input/Output Format (in addition to input/output specification for Question 1)
If the input contains the string "**prep**", print the BST in its Parenthesis Representation.

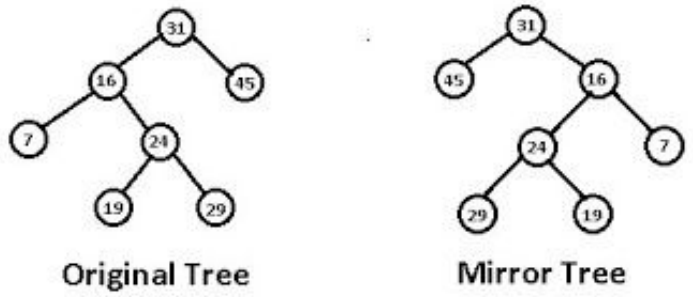
Sample Input

insr 43
insr 15
insr 8
insr 30
insr 20
insr 35
insr 60
insr 50
insr 82
insr 70
prep
stop

Sample Output

```
( 43 ( 15 ( 8 ( ) ( ) ) ( 30 ( 20 ( ) ( ) ) ( 35 ( ) ( ) ) ) ) (
60 ( 50 ( ) ( ) ) ( 82 ( 70 ( ) ( ) ) ( ) ) ) )
```

3. Write a program to construct the mirror of a Binary Search Tree (BST) from the given input recursively. Refer to the figure below for an example of mirror tree.



Add a function **mirror(*t*)** to your program of qn 2.

mirror(*tree*) - function should take as input a pointer to the root node of a tree **tree** and print its mirror tree in its Parenthesis Representation.

Input Format
Single line containing the parenthesis representation of a BST.

Output Format

Single line containing the parenthesis representation of the mirror BST.

Sample Input:

```
( 31 ( 16 ( 7 ( ) ( ) ) ( 24 ( 19 ( ) ( ) ) ( 29 ( ) ( ) ) ) ) ( 45 ( ) ( ) ) )
```

Sample Output

```
( 31 ( 45 ( ) ( ) ) ( 16 ( 24 ( 29 ( ) ( ) ) ( 19 ( ) ( ) ) ) ( 7 ( ) ( ) ) ) )
```

4. Given the preorder traversal with unique keys, write a program to construct the Binary Search Tree. Your program should include the following functions.

main() - reads the input as specified in the input format from the terminal and calls the appropriate subfunctions to construct the BST.

Input format

The first line contains a positive integer, representing the number of nodes in the tree.
The second line contains space separated integers, representing a valid preorder traversal of a binary search tree.

Output format

Single line representing the binary tree in the Parenthesis Representation

Sample Input

10
43 15 8 30 20 35 60 50 82 70

Sample Output

```
( 43 ( 15 ( 8 ( ) ( ) ) ( 30 ( 20 ( ) ( ) ) ( 35 ( ) ( ) ) ) ) (
60 ( 50 ( ) ( ) ) ( 82 ( 70 ( ) ( ) ) ( ) ) ) )
```

5. Write a recursive program to find the height and the longest path of a binary tree in linear time. Your program should include the following functions.

main() – reads the input as specified in the input format from the terminal and calls the appropriate subfunctions to find the height and the longest path.

height(*tree*) – returns the height of a binary tree specified by ***tree***, which is the number of edges between the tree's root and its farthest leaf. Assume that a binary tree with a single node has height zero.

longest_path(*tree*) – return the length of the longest path, which is the maximum number of edges between any two nodes in the tree specified by ***tree***.

Input format

Single line representing the binary tree in the parenthesis format.

Output format

Two integers separated by a space denoting height and longest path respectively.

Sample Input

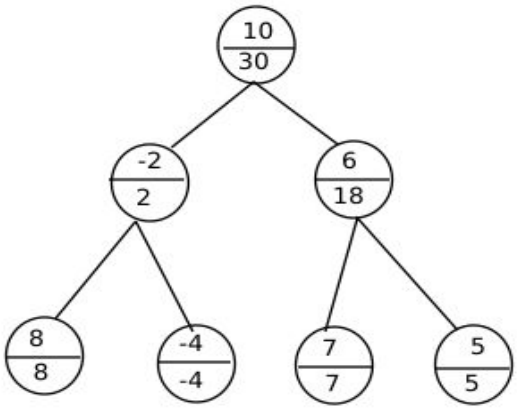
```
( 43 ( 15 ( 8 ( ) ( ) ) ( 30 ( 20 ( ) ( ) ) ( 35 ( ) ( ) ) ) ) (
60 ( 50 ( ) ( ) ) ( 82 ( 70 ( ) ( ) ) ( ) ) ) )
```

Sample Output

3 6

6. Given a Binary Tree with each of the nodes having positive and negative values. Each node in the tree has an extra field with the sum of its left subtree, right subtree and the node's value itself. Your program should include the following function:

main() – reads the input as specified in the input format from the terminal and calls the appropriate function to find the new tree as specified.
mod_tree(*tree*) – prints the tree in the specified format from the given binary tree specified by ***tree***.



Input format

Single line representing the binary tree in the parenthesis format.

Output format

Single line representing the sum calculated for each node of binary tree in the parenthesis format.
Sample Input
```
( 10 ( -2 ( 8 ( ) ( ) ) ( -4 ( ) ( ) ) ) ( 6 ( 7 ( ) ( ) ) ( 5 ( ) ( ) ) ) )
```
Sample Output
```
( 30 ( 2 ( 8 ( ) ( ) ) ( -4 ( ) ( ) ) ) ( 18 ( 7 ( ) ( ) ) ( 5 ( ) ( ) ) ) )
```

7. Given a binary tree, find the size (number of nodes in the tree) of the largest subtree which is also a Binary Search Tree (BST). Your program should run in Linear time and include the following functions.

main() - reads the input as specified in the input format from the terminal and calls the appropriate function to find the largest BST present in the given binary tree.
maxSubBST(*tree*) - returns the size of the largest BST in the binary tree specified by **tree**.

For Example
If the binary tree is as shown below Fig 1. Identify maximum size BST subtree as shown in Fig 2 and output the number of nodes in Fig 2 as 5.

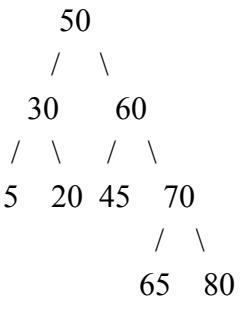


Fig 1
The following subtree is the maximum size BST subtree

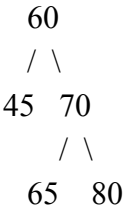


Fig 2

Input format

Single line representing the binary tree in the parenthesis format.

Output format

Single integer representing the size (number of nodes in the tree) of the largest BST in the given binary tree.

Sample Input

```
( 10 ( 40 ( 20 ( ) ( ) ) ( 60 ( ) ( ) ) ) ( 30 ( 90 ( ) ( ) ) ( ) ) )
```

Sample Output

3

8. Write a program to find the Huffman encoding for a given message using binary tree. Your program should include the following functions:

main() – reads the input as specified in the input format from the terminal and calls the appropriate functions to print the huffman code.

Find_huffman_Code(*m*) – It should build a Huffman tree for the given message *m* and assign codes by traversing the Huffman Tree. Path from the top or root of this tree to a particular node/character will determine the code group associated with that node/character.

print_Code_Length (*m*) – This function should print the total number of bits required to store the final binary code.

Input File Format:

The input consists of multiple lines, each line of the input contains a message of type string.

Output File Format:

The output consists of multiple lines each containing the total length of corresponding encoded message.

Sample Input:

malayalam
mississippi

Sample Output:

17
21