CSP2348.3 - Data Structures

Assignment 3 - Paired programming project

Student ID: 10636908

Name: Poorav Sharma

Due Date: 29/05/2023

Table of Contents

[Introduction 1](#_Toc136133067)

[Q1: Balanced BST generation 1](#_Toc136133068)

[Q2: Application to binary tree traversal & BST 3](#_Toc136133069)

[a) Modify the in-order traversal algorithm to implement the inverse-in-order traversal algorithm. 3](#_Toc136133070)

[b) Modify the in-order traversal algorithm to form two code versions; one prints all leaf nodes, and the other prints all non-leaf nodes of the BST 3](#_Toc136133071)

[c) Modify the pre-order traversal algorithm to form a new method such that, for a given node N in a BST, it counts the total number of nodes of the sub-tree rooted at N, and prints all nodes, including N, of the sub-tree. 3](#_Toc136133072)

[d) Modify the search tree node algorithm to form a new method so that, for a given node N in a BST, it calculates the depth of the node N (in the BST). 3](#_Toc136133073)

[e) Modify the post-order traversal algorithm to form a new method so that, for a given node N in a BST, it calculates the depth of the sub-tree rooted at N. 3](#_Toc136133074)

[f) Design an algorithm to delete a node from a BST 3](#_Toc136133075)

[Q3: AVL tree: Deleting a Node 3](#_Toc136133076)

[Conclusion 3](#_Toc136133077)

[References 3](#_Toc136133078)

# Introduction

This report showcases the design, analysis, implementation, and performances of three programming tasks. The first task is to design and implement a simple algorithm to generate a balanced BST. The second task is to modify some existing tree-based algorithms/methods to implement some specific application scenario/s. The third is a dummy mini-project using AVL-tree ADT.

# Q1: Balanced BST generation

a) Pseudo Code design

n = pre-loaded sequence

O = empty new sequence

Sort n in ascending order

1. Find Middle by Dividing n by two.
2. Insert the element found in Middle index of n into O .
3. L = all the elements of n from the start to the Middle index.
4. R = all the elements of n from the Middle index to the end.

5. Repeat steps 1, 2, 3, 4 , 5, 6 using L instead of n

5.1 If nothing is inserted into O from L elements, Go to next step.

6. Repeat steps 1, 2, 3, 4 , 5, 6 using R instead of n

6.1 If nothing is inserted into O from R elements. Go to next step.

7. terminate

b) Analysis

The algorithm I made finds the middle element of the sequence and inserts it into the new sequence. This way when the sequence is added to an empty binary tree it will be the root node. The left or the right node of the root node is determined by finding the middle element from the left or right side from middle index respectively. This algorithm repeats these steps until all the elements in two sides have been added to the new sequence. The new sequence will be able to generate a balanced binary tree when its data items are inserted one by one into an empty binary tree.

c) Testing algorithm

A pre-loaded sequence that is given to the algorithm. The algorithm then reorganizes it so that it can generate a balanced binary tree. **Image1.1** shows the required sequence to test from the assignment. **Image 1.2** shows the other data set I created to test the algorithm.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence**Image1.1** Sequence given by assignment is used.

**Image1.2** Sequence created is used in the program.

# Q2: Application to binary tree traversal & BST

## a) Modify the in-order traversal algorithm to implement the inverse-in-order traversal algorithm.

N = tree node

2.IF n is not empty

3.Travers the right subtree of N

3.1 Repeat but change N to right subtree of N.

4.Display the element in N.

5. Travers the left subtree of N

5.Repeat but change N to left subtree of N.

END IF

6.IF traversed all nodes is true.

7.End looping.

END IF

## b) Modify the in-order traversal algorithm to form two code versions; one prints all leaf nodes, and the other prints all non-leaf nodes of the BST

### 1)To find all leaf nodes:

N = tree node

IF N is not empty

Traverse the left subtree of N in order

Repeat till all the u travers all the node on the left subtree of N

Traverse the right subtree of N in order

Repeat till all the u travers all the node on the right subtree of N

If N has no right subtree and no left subtree

Display the element at N.

END IF

END IF

IF traversed all nodes is true.

End looping.

END IF

### 2)To find all non-leaf nodes:

N = tree node

IF N is not empty

Traverse the left subtree of N in order

Repeat till all the u travers all the node on the left subtree of N.

Traverse the right subtree of N in order

Repeat till all the u travers all the node on the right subtree of N.

If N has right subtree or left subtree

Display the element at N.

END IF

END IF

IF traversed all nodes is true.

End looping.

END IF

## c) Modify the pre-order traversal algorithm to form a new method such that, for a given node N in a BST, it counts the total number of nodes of the sub-tree rooted at N, and prints all nodes, including N of the sub-tree.

N = node search input

Count = number of nodes = 0

search N in tree

IF N search is successful

IF N subtree not empty

Display element of N.

Count plus one

Traverse the left subtree of N in order

Repeat steps by change N to its left subtree.

Traverse the right subtree of N in order

Repeat steps by change N to its right subtree.

END IF

END IF

Else

N node not found in binary tree

Terminate loop.

END ELSE

## d) Modify the search tree node algorithm to form a new method so that, for a given node N in a BST, it calculates the depth of the node N (in the BST).

N = input integer from user will act as the node

R = root node of the tree

Dept = the dept of the N in the BST =0

WHILE R is not empty

IF N is less than the element at R

Let R be R’s left child.

Add one to Dept.

END IF

ELSE IF N is greater than the element at R.

Let R be R’s right child.

Add one to Dept.

END ELSE IF

ELSE

Display the Dept at which N was found

Terminate method.

END ELSE

End WHILE

Display N was not found in the tree.

Terminate method.

## e) Modify the post-order traversal algorithm to form a new method so that, for a given node N in a BST, it calculates the depth of the sub-tree rooted at N.

N = the integer input

1.search N in tree

2.IF N search is successful

2.1.Assign Depth as 0.

2.2.IF N is not empty

2.3Traverse the left subtree of N

leftDept = Depth

2.4Traverse the right subtree of N

rightDept = Depth

2.5Compare leftDept and rightDept.

Depth = The larger value between the two depths. +1 to count the depth of the node.

Repeat step 2.1 to 2.5 for all the node of the subtree.

END IF

Subtract Depth by 1 so that the counting begins from 0.

Display Depth of sub-tree rooted at N.

Terminate

END IF

ELSE

Display N not found in BST.

Terminate method.

## f) Design an algorithm to delete a node from a BST

N = the integer input

O = the root node of tree

1.search N in tree

2.IF N search is successful

3.IF N is less than the element of O

Repeat steps 1 to 3 with O being replaced with O left tree.

END IF

4.ELSE IF N is less than the element of O

Repeat from step 1 with O being replaced with O right tree.

END ELSE IF

5.ELSE

5.1IF O left subtree and O right tree are both empty

Delete O which is where the N is present

END IF

5.2ELSE IF O left subtree is empty

Assign O right subtree as O

END ELSE IF

5.3ELSE IF O right subtree is empty

Assign O left subtree as O

END ELSE IF

5.4ELSE

Let M be the minimum node of the O right subtree

Set O value as the M value

Repeat from step 1 with O right subtree assigned as O and M assigned as N

END ELSE

END IF

## g) Test Cases

# Q3: AVL tree: Deleting a Node

# Conclusion

# References