CSP2348.3 - Data Structures

Assignment 3 - Paired programming project

Student ID: 10636908

Name: Poorav Sharma

Due Date: 29/05/2023

Table of Contents

Introduction

Binary tree sorting algorithm is used to sort data by clearly placing larger data values on the right side of the tree and smaller data values on the left side. It determines the size by comparing the new data values with the existing data values. The positions where data values are placed are called nodes. The first (top) node is referred to as the root node.

There are different types of nodes: leaf nodes and non-leaf nodes. Leaf nodes have no children branching off them, while non-leaf nodes have child nodes branching off them. Nodes can have only two branches, so if more data is added, it becomes a child node of the available nodes. The AVL tree is like a binary tree, with the main difference being that an AVL tree must be balanced. This means that the tree cannot have one side significantly longer than the other.

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.

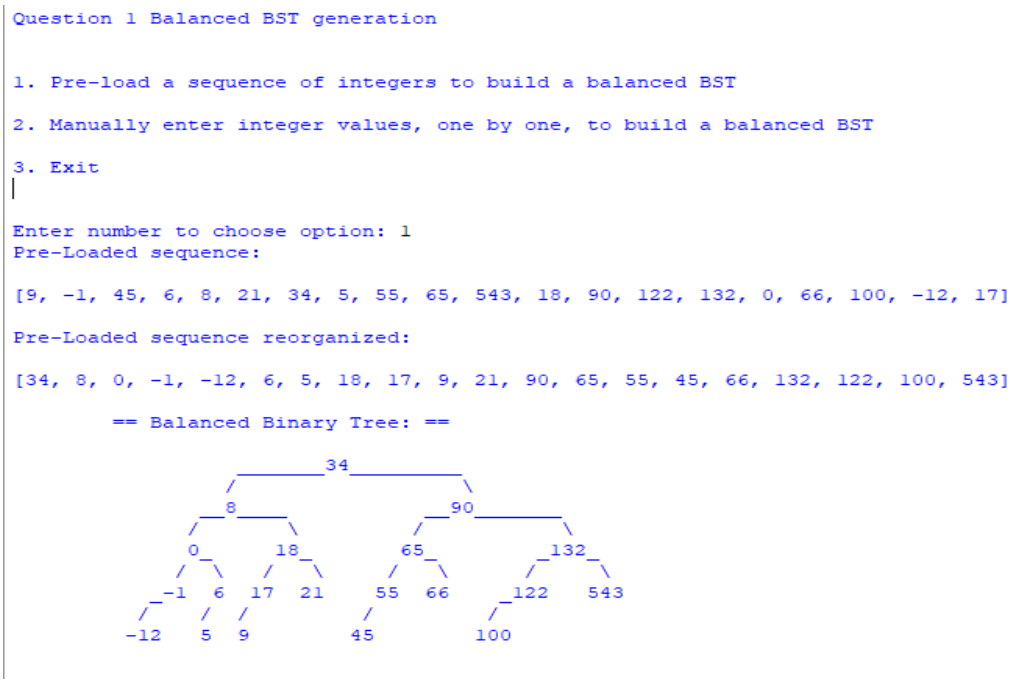


Image1.1 Sequence given by assignment is used.

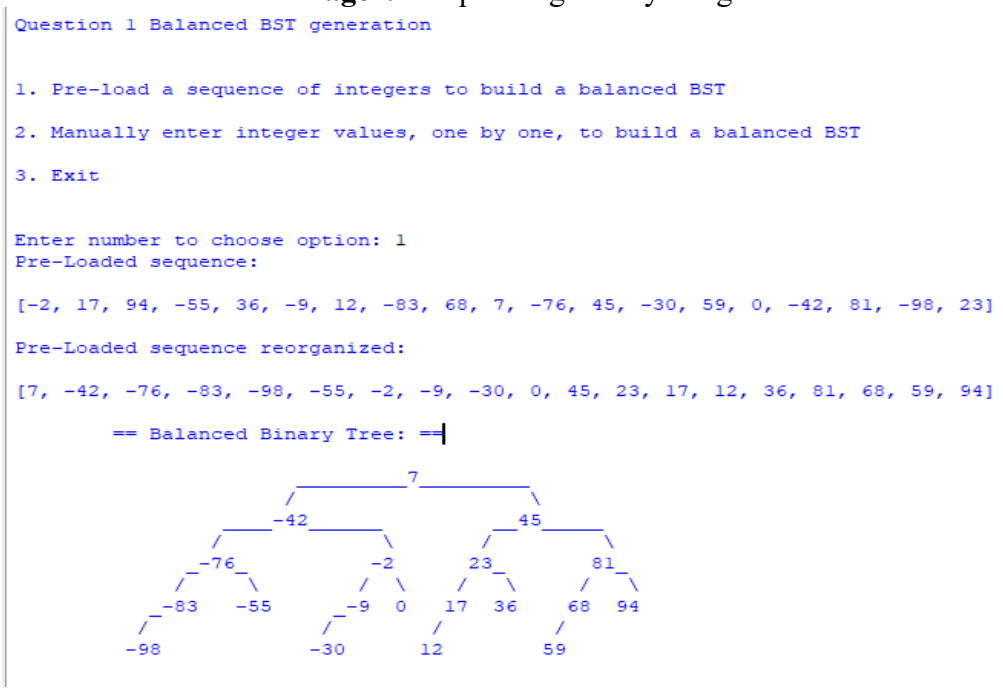


Image1.2 Sequence created is used in the program.

Q2: Application to binary tree traversal & BST

These algorithms will be used in a method function therefore it is easy to repeat the steps as instructed.

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

N = tree node

1.IF n is not empty

2.Travers the left subtree of N

2.1 Repeat from step 1 but change N to right subtree of N.

3.Display the element in N.

4. Travers the right subtree of N

4.1Repeat from step 1 but change N to right subtree of N.

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

1.IF N is not empty

Traverse the left subtree of N in order

Let N be the N left subtree

Repeat from step 1 till it traverses all the node on the left subtree of N

Traverse the right subtree of N in order

Let N be the N right subtree

Repeat from step 1 till it traverses 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

2)To find all non-leaf nodes:

N = tree node

1.IF N is not empty

Traverse the left subtree of N in order

Let N be the N left subtree

Repeat from step 1 till all it has travers all the node on the left subtree of N.

Traverse the right subtree of N in order

Let N be the N right subtree

Repeat from step 1 till it has 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

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 which starts off as zero

1. IF R is not empty

IF N is less than the element at R

Let R be R's left child.

Add one to Dept.

Start step 1

END IF

ELSE IF N is greater than the element at R.

Let R be R's right child.

Add one to Dept.

Start step 1

END ELSE IF

ELSE

Display the Dept at which N was found

Terminate

END ELSE

End IF

ELSE

Display N was not found in the tree.

Terminate.

END ELSE

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 subtree root begins from 0.

Display Depth of sub-tree rooted at N.

Terminate

END IF

ELSE

Display N not found in BST.

Terminate method.

END ELSE

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 empty

Return N

END IF

4.IF N is less than the element of O

Start from step 3 with the new O being replaced with O left tree.

END IF

5.ELSE IF N is more than the element of O

Start from step 3 with the new O being replaced with O right tree.

END ELSE IF

6.ELSE

6.1IF O left subtree and O right sub tree are both empty

Delete O

Start from step 3 with O being empty

END IF

6.2ELSE IF O left subtree is empty

Assign O right subtree as O

Start from step 3 again with the new O

END ELSE IF

6.3ELSE IF O right subtree is empty

Assign O left subtree as O

Start from step 3 again with the new O

END ELSE IF

6.4Successor = the left subtree of O right tree

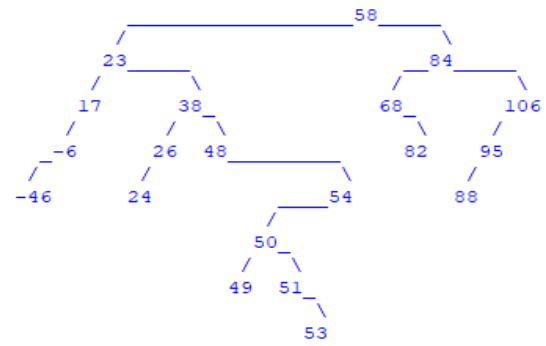
6.5Assing N as the element of successor

6.5For the updated O right tree subtree = start from step 3 with O right subtree as O and delete successor element

END IF

g) Screenshots

```
Enter number to choose option: 1
        == Binary Tree: shape ==
```




```
Pre-Order Traversal:

58, 23, 17, -6, -46, 38, 26, 24, 48, 54, 50, 49, 51, 53, 84, 68, 82, 106, 95, 88,

In-Order Traversal:

-46, -6, 17, 23, 24, 26, 38, 48, 49, 50, 51, 53, 54, 58, 68, 82, 84, 88, 95, 106,

Post-Order Traversal:

-46, -6, 17, 24, 26, 49, 53, 51, 50, 54, 48, 38, 23, 82, 68, 88, 95, 106, 84, 58,

Inverse-In-Order Traversal:

106, 95, 88, 84, 82, 68, 58, 54, 53, 51, 50, 49, 48, 38, 26, 24, 23, 17, -6, -46,
```

Image 2.1 The output of option one from second menu

```
Question 2: Application to binary tree traversal & BST


1. Pre-load a sequence of integers to build a BST

2. Manually enter integer values, one by one, to build a BST

3. Exit


Enter number to choose option: 1


Question 2: Application to binary tree traversal & BST

1. Display the tree shape of current BST, and then show the pre-order, in-order, post-order and inverse-in-o
rder traversal sequences of the BST

2. Show all leaf nodes of the BST, and all non-leaf nodes (separately)

3. Show a sub-tree and count its nodes

4. Show the depth of a given node in the BST

5. Show the depth of a subtree of the BST

6. Insert a new integer key into the BST

7. Delete an integer key from the BST

8. Exit

Enter number to choose option: 2

All Leaf nodes:

-46, 24, 49, 53, 82, 88,

All Non-leaf nodes:

-6, 17, 26, 51, 50, 54, 48, 38, 23, 68, 95, 106, 84, 58,
```

Image 2.2 The output of option two from second menu

```
All Leaf nodes:

-46, 24, 49, 53, 82, 88,

All Non-leaf nodes:

-6, 17, 26, 51, 50, 54, 48, 38, 23, 68, 95, 106, 84, 58,


Question 2: Application to binary tree traversal & BST

1. Display the tree shape of current BST, and then show the pre-order, in-order, post-order and inverse-in-o
rder traversal sequences of the BST

2. Show all leaf nodes of the BST, and all non-leaf nodes (separately)

3. Show a sub-tree and count its nodes

4. Show the depth of a given node in the BST

5. Show the depth of a subtree of the BST

6. Insert a new integer key into the BST

7. Delete an integer key from the BST

8. Exit


Enter number to choose option: 3
Enter a integer: 51
The nodes of subtree  51 :

51 53
Total number of node:  2
```

Image 2.3 The output of option three from second menu

```
Enter number to choose option: 4
Enter a integer: 51

The depth of node  51  in the BTS is:  6
```

Image 2.4 The output of option four from second menu

```
Enter number to choose option: 5
Enter a integer: 51

The depth of subtree rooted at node  51  is:  1
```

Image 2.5 The output of option five from second menu

```
Enter number to choose option: 6
Enter a integer: 52

Integer 52 has been inserted into the BST

Inverse-In-Order Traversal:

106, 95, 88, 84, 82, 68, 58, 54, 53, 52, 51, 50, 49, 48, 38, 26, 24, 23, 17, -6, -46,
```

Image 2.6 The output of option six from second menu

```
Enter number to choose option: 7
Enter a integer: 51

Integer 51 has been deleted into the BST


Inverse-In-Order Traversal:

106, 95, 88, 84, 82, 68, 58, 54, 53, 52, 50, 49, 48, 38, 26, 24, 23, 17, -6, -46,
```

Image 2.7 The output of option seven from second menu

h) Test Cases

*Option 1 from the first menu inserts all the integers from a pre-built sequence into the tree. It faster to use it while testing compared to manually inserting numbers into the tree.

Test case	Steps	Expected Result
Printing binary tree and the pre-order, in-order, post order and inverse in-order sequence of the tree.	1.Select option 1 from first menu 2. Select option 1	Prints Binary Tree diagram. Prints pre-order, in-order, inverse in-order, and post order sequence.
Manually inserting integer	1.Select option 2 from menu 1 2. Input integers (will not let you input same integer) 3. Input x to finish	Inserts all the integer input into the tree.
Printing leaf and non-leaf nodes	1.Select option 1 from first menu 2. Select option 2 from second menu	Prints leaf nodes. Prints non-leaf nodes
Show a sub-tree and count its nodes	1.Select option 1 from first menu 2. Select option 3 from second menu 3. Insert an integer (N) that the tree contains	Print the number of nodes in the subtree N. Print all the nodes in the subtree N.
Show the depth of a given node in the BST	1.Select option 1 from first menu 2. Select option 4 from second menu 3. Insert an integer (N) that the tree contains	Prints the depth at which the node N is in the tree.
Show the depth of a subtree of the BST	1.Select option 1 from first menu 2. Select option 5 from second menu 3. Insert an integer (N) that the tree contains	Prints the depth of the subtree rooted N.
Insert a new integer key into the BST	1.Select option 1 from first menu 2. Select option 6 from second menu	Inserted N into tree. Print N has been inserted into tree.

	3. Insert an integer (N) that the tree does not contains	
Delete an integer key from the BST	1.Select option 1 from first menu 2. Select option 7 from second menu 3. Insert an integer (N) that the tree contains	Delete N from tree. Rearrange tree to accordingly. Print N has been deleted from tree.
Insert number already inserted into tree	1.Select option 1 from first menu 2. Select option 6 from second menu 3. Insert an integer (N) that the tree contains	Print N already exists in tree.
Inserting number that does not exist in tree. *Perform test case for options (O) 3, 4,5 and 7 from second menu	1.Select option 1 from first menu 2. Select option (O) from second menu 3. Insert an integer (N) that the tree does not contain	Print N does not exist in tree.
Inserting invalid input	Insert a non-integer in anywhere the program askes for inputs	Print invalid input.

Q3: AVL tree: Deleting a Node

a) Algorithm

This algorithm almost the same as the deleting method of the BST from question 2. The only difference is that before the algorithm finished it used the AVL function rebalance() to check the binary tree and rebalance it if it was unbalanced after a node was deleted.

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 empty

Return N and end terminate

END IF

4.IF N is less than the element of O

Start from step 3 with the new O being replaced with O left tree.

END IF

5.ELSE IF N is more than the element of O

Start from step 3 with the new O being replaced with O right tree.

END ELSE IF

6.ELSE

6.1IF O left subtree and O right sub tree are both empty

Delete O which is where the N is present

Start from step 3

END IF

6.2ELSE IF O left subtree is empty

Assign O right subtree as O

Start from step 3 again with the new O

END ELSE IF

6.3ELSE IF O right subtree is empty

Assign O left subtree as O

Start from step 3 again with the new O

END ELSE IF

6.4Successor = the left subtree of O right tree

6.5Assing N as the element of successor

For the updated O right tree subtree = start from step 3 with O right subtree as O and delete successor element

7. Use Rebalance AVL tree method to balance the tree.

END IF

b) Screenshots

```
Enter number to choose option: 1
                === AVL Tree ===

                _______46_____________
               /                      \
           __20___                _____82_____
          /       \              /            \
     ____0_       35_         __58___       __92_
    /      \     /   \       /       \     /     \
 -55_      14   23   44_    49_      79   84_    103
     \                  \  /   \    /        \
     -21                45 47  50  72        89

 == AVL tree (printed left-side down, with [hights, balance_factors] & an 'L' for each leaf node) ==

            > 103 [0:0] L
        > 92 [2:1]
                > 89 [0:0] L
            < 84 [1:-1]
    > 82 [3:0]
            > 79 [1:1]
                < 72 [0:0] L
        < 58 [2:0]
                > 50 [0:0] L
            < 49 [1:0]
                < 47 [0:0] L
 46 [4:0]
                > 45 [0:0] L
            > 44 [1:-1]
        > 35 [2:-1]
            < 23 [0:0] L
    < 20 [3:0]
            > 14 [0:0] L
        < 0 [2:1]
                > -21 [0:0] L
            < -55 [1:-1]
```

Image 3.1 Output of option one from second menu

```
Question 3: AVL tree: deleting a node

1. Display the AVL tree, showing the height and balance factor for each node.

2. Print the pre-order, in-order, and post-order traversal sequences of the AVL tree

3. Print all leaf nodes of the AVL tree, and all non-leaf nodes (separately)

4. Insert a new integer key into the AVL tree

5. Delete an integer key from the AVL tree

6. Exit

Enter number to choose option: 2

Pre-Order Traversal:

[46, 20, 0, -55, -21, 14, 35, 23, 44, 45, 82, 58, 49, 47, 50, 79, 72, 92, 84, 89, 103]

In-Order Traversal:

[-55, -21, 0, 14, 20, 23, 35, 44, 45, 46, 47, 49, 50, 58, 72, 79, 82, 84, 89, 92, 103]

Post-Order Traversal:

[-21, -55, 14, 0, 23, 45, 44, 35, 20, 47, 50, 49, 72, 79, 58, 89, 84, 103, 92, 82, 46]
```

Image 3.2 Output of option two from second menu

```
Enter number to choose option: 3

All Leaf nodes:

-21, 14, 23, 45, 47, 50, 72, 89, 103,

All Non-leaf nodes:

-55, 0, 44, 35, 20, 49, 79, 58, 84, 92, 82, 46,
```

Image 3.3 Output of option three from second menu

```
Enter number to choose option: 4
Enter a integer: 41
Inserted key [41]
```

Image 3.4 Output of option four from second menu

```
Enter number to choose option: 5
Enter a integer: 51

Integer 51 has been deleted into the AVL
```

Image 3.5 Output of option five from second menu

c) Test Case

*Option 1 from the first menu inserts all the integers from a pre-built sequence into the tree. It faster to use it while testing compared to manually inserting numbers into the tree.

Test case	Steps	Expected Result
Display the AVL tree, showing the height and balance factor for each node.	1.Select option 1 from first menu 2. Select option 1	Prints AVL Tree diagram. Prints the height and balance factor for each node.
Manually inserting integer	1.Select option 2 from menu 1 2. Input integers (will not let you input same integer) 3. Input x to finish	Inserts all the integer input into the tree.
Pre-order, in-order, and post-order traversal sequences of the AVL tree.	1.Select option 1 from first menu 2. Select option 2 from second menu	Print the pre-order, in-order, and post-order traversal sequences of the AVL tree.
Printing leaf and non-leaf nodes	1.Select option 1 from first menu 2. Select option 3 from second menu	Prints leaf nodes. Prints non-leaf nodes
Insert a new integer key into the AVL	1.Select option 1 from first menu 2. Select option 4 from second menu 3. Insert an integer (N) that the tree does not contains	Inserted N into tree. Rearrange and rebalance tree accordingly. Print N has been inserted into tree.
Delete an integer key from the AVL	1.Select option 1 from first menu 2. Select option 5 from second menu 3. Insert an integer (N) that the tree contains	Delete N from tree. Rearrange and rebalance tree to accordingly. Print N has been deleted from tree.
Insert number already inserted into tree	1.Select option 1 from first menu 2. Select option 4 from second menu 3. Insert an integer (N) that the tree contains	Print N already exists in tree.

Inserting number that does not exist in tree.	1.Select option 1 from first menu 2. Select option 6 from second menu 3. Insert an integer (N) that the tree does not contain	Print N does not exist in tree.
Inserting invalid input	Insert a non-integer in anywhere the program askes for inputs	Print invalid input.

Conclusion

The binary tree in question 2 is not balanced as it depends on the order in which integers are inserted into the tree. In contrast, both the AVL tree in question 3 and the binary tree in question 1 are balanced, but they achieve balance in different ways. The binary tree in question 1 achieves balance by altering the sequence of numbers that is to be inserted into an empty tree, while the AVL tree rotates nodes within the tree to ensure balance is maintained.

References

Stack Overflow. (2015, December 3). Print binary tree level by level in Python [Online forum post]. Stack Overflow. Retrieved from https://stackoverflow.com/questions/34012886/print-binary-tree-level-by-level-in-python

GeeksforGeeks. (n.d.). Deletion in Binary Search Tree. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/deletion-in-binary-search-tree/

GeeksforGeeks. (n.d.). Count Non-Leaf Nodes in Binary Tree. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/count-non-leaf-nodes-binary-tree/