Name : Ramzy Izza Wardhana
NIM : 21/472698/PA/20322
Class : IUP CS1

Assignment 3 – Binary Search Tree

Lab Algorithm and Data Structure

Complete Source Code *(Zip file consisting Main.java, BinaryTree.java, Node.java)* :
https://drive.google.com/file/d/1MYtY3Zd4TVw9VhLvunXzLE0TkDhbfbeK/view?usp=sharing

Given a row of input data with the data sequence as follows:

56 23 | 21 | 15 | 9 | 87 | 45 | 77 | 59 | 90 | 83 | 75 | 20 | 5 | 92 | 98 | 100

1. Implement a binary search tree based on the order of the data.

```
Enter desired size you want to insert: 17

Input data seperated by spaces: 56 23 21 15 9 87 45 77 59 90 83 75 20 5 92 98 100
Input data element you want to search: 30

Data Not Found
```

```
Enter desired size you want to insert: 17

Input data seperated by spaces: 56 23 21 15 9 87 45 77 59 90 83 75 20 5 92 98 100
Input data element you want to search: 21

Data Found!
```

2. Create a method to display the results of traversal using inorder, preorder and postorder.

```
Choose type of traversal you want:
1. Inorder
2. Preorder
3. Postorder

1
Inorder = Left Child - Parent - Right Child
5 9 15 20 21 23 45 56 59 75 77 83 87 90 92 98 100
```

```
Choose type of traversal you want:
1. Inorder
2. Preorder
3. Postorder

2

Preorder =  Parent - Left Child - Right Child
56 23 21 15 9 5 20 45 87 77 59 75 83 90 92 98 100
```

```
Choose type of traversal you want:
1. Inorder
2. Preorder
3. Postorder

3

Postorder: Left Child - Right Child - Parent
5 9 20 15 21 45 23 75 59 83 77 100 98 92 90 87 56
```

3. Create a method to calculate the sum of the values of all elements in a binary search tree

```
PROBLEM 3:
The total sum of data inside the binary tree is: 955
```

4. Create a method to determine the height of a binary search tree.

```
PROBLEM 4:
The height of the binary tree is: 5
```

5. Create a method showLevelOrder to print nodes based on their depth level (print all nodes on the first level, followed by print nodes on the second level, and so on). All nodes on each level are printed from left to right.

```
PROBLEM 5:
Display nodes value based on it's level:
56
23  87
21  45  77  90
15  59  83  92
9  20  75  98
5  100
```

6. Create a method to display the sibling value of a certain node. Give an example of a node that has sibling and does not.

```
PROBLEM 6:
Enter desired value to check it's sibling's value: 90
It's Sibling: 77


Input FIRST value to check if it has siblings: 23
Input SECOND value to check if it has siblings: 87
It has the same parents or a sibling
```

Brief Screenshots of the code:

```java
//Creating node for tree - Problem 1
public void addNode(Node node){
    if(root == null)
        root = node;
    else
        insertNode(root, node);
}
```

```java
//Data insertion method - Problem 1
public void insertNode(Node parent, Node node){
    if(node.getValue() < parent.getValue()){
        if(parent.leftC == null)
            parent.leftC = node;
        else
            insertNode(parent.leftC, node);
    }
    else{
        if(parent.rightC == null)
            parent.rightC = node;
        else
            insertNode(parent.rightC, node);
    }
}
```

```java
//Binary search algorithm - Problem 1
public static String searchValue(Node root, int value){
    if(root == null)
        return "\nData Not Found\n";
    else{
        if(root.getValue() == value)
            return "\nData Found!\n";
        else if(root.getValue() > value)
            return searchValue(root.leftC, value);
        else
            return searchValue(root.rightC, value);
    }
}
```

```java
//In-order algorithm - Problem 2
public static void inorder(Node root) {
    if(root != null){
        inorder(root.leftC);
        System.out.print(root.value + " ");
        inorder(root.rightC);
    }
}
```

```java
//Pre-order algorithm - Problem 2
public static void preorder(Node root){
    if(root!= null){
        System.out.print(root.value + " ");
        preorder(root.leftC);
        preorder(root.rightC);
    }
}
```

```java
//Post-order algorithm - Problem 2
public static void postorder(Node root) {
    if(root != null){
        postorder(root.leftC);
        postorder(root.rightC);
        System.out.print(root.value + " ");
    }
}
```

```java
//Calculate sum of all value in BST algorithm - Problem 3
public static int sumValue(Node root){
    int sumLeft = 0,sumRight = 0,total = 0;
    if (root != null){
        if(root.leftC != null)
            sumLeft = sumValue(root.leftC);
        if(root.rightC != null)
            sumRight = sumValue(root.rightC);
        total = root.value + sumLeft + sumRight;
    }
    return total;
}
```

```java
//Calculate the highest depth of a binary tree algorithm - Problem 4
public static int getHeight(Node root){
    int rightHeight = 0, leftHeight = 0;
    int height;
    if(root!=null){
        if(root.leftC != null)
            leftHeight = getHeight(root.leftC);
        if(root!=null)
            rightHeight = getHeight(root.rightC);
    }
    if(leftHeight > rightHeight)
        height = leftHeight+1;
    else
        height = rightHeight+1;

    return height;
}
```

```java
//Display all the value of nodes based on depth levels algorithm - Problem 5
public static void showLevelOrder(Node root){
    //check the data is null or not
    if (root == null){
        System.out.println("There are no such data contain inside the binary Tree!");
        return;
    }
    //implement queue to push all the value to the queue
    Queue<Node> queue = new LinkedList<>();
    queue.add(root); //node value
    queue.add(null); //eliminator
    //loops until all queue is empty
    while (queue.isEmpty() == false) {
        root = queue.poll(); //remove first element and return
        if (root == null) { //check if the next node is occurs
            if (queue.isEmpty() == false) {
                queue.add(null);
                System.out.println();
            }
        }
        else {
            if (root.leftC != null)
                queue.add(root.leftC); //push left child
            if (root.rightC != null)
                queue.add(root.rightC); //push right child
            System.out.print(root.value + "  ");
        }
    }
}
```

```java
//Display sibling's value with given node algorithm - Problem 6
public Node displaySibling(Node root, int data) {
    if(root.value == data || root == null)
        return null; //tree is empty
    Node parentNode = null;

    while(root != null) {
        if(root.value > data) {
            parentNode = root;
            root = root.leftC;
        }
        else if(root.value < data) {
            parentNode = root;
            root = root.rightC;
        }
        else
        break;
    }

    if(parentNode.leftC.value == data && parentNode.leftC != null)
        return parentNode.rightC;
    if(parentNode.rightC.value == data && parentNode.rightC != null)
        return parentNode.leftC;
    return null;
}
```

```java
public Node insert(Node root, int data) {
    if(root == null)
     return createNewNode(data);
    if(root.value > data)
     root.leftC = insert(root.leftC, data);
    else if((root.value < data))
     root.rightC = insert(root.rightC, data);
    return root;
}
```

```java
public Node createNewNode(int i) {
    Node rootNew = new Node();
    rootNew.leftC = null;
    rootNew.rightC = null;
    rootNew.data = i;

    return rootNew;
}
```

```java
//Check if the given data has a sibling or not - Problem 6
public static boolean nodeSibling(Node root,int data1, int data2){
    if (root == null)
        return false;
    if (root.rightC != null && root.leftC != null){
        int rightSide = root.rightC.value;
        int leftSide = root.leftC.value;
    if (rightSide == data2 && leftSide == data1)
        return true;
    else if (rightSide == data1 && leftSide == data2)
        return true;
    }
    if (root.leftC != null) //check left side
        nodeSibling(root, data1, data2);
    if (root.rightC != null) //check right side
        nodeSibling(root, data1, data2);
    return true;
}
```

```java
package BinaryTree;

public class Node {
    public int value;
    public Node leftC, rightC;
    public int data;
    Node(int value){
        this.value = value;
    }
    public Node() {
    }
    public int getValue(){
        return value;
    }
}
```

Main.Java

Main.java 1 | BinaryTree.java | Node.java

src > BinaryTree > Main.java > Main > main(String[])

```java
            bt.addNode((new Node(j)));
        }

        //1. Implement binary search and searching algorithm
        System.out.println("PROBLEM 1:");
        System.out.print("Input data element you want to search: ");
        int find = sc.nextInt();
        System.out.println(BinaryTree.searchValue(bt.root, find));

        //2. User choose which traversal order
        System.out.println("PROBLEM 2: ");
        System.out.println("Choose type of traversal you want: ");
        System.out.println("1. Inorder \n2. Preorder \n3. Postorder\n");
        int choose = sc.nextInt();

        if(choose == 1){
            System.out.println("Inorder = Left Child - Parent - Right Child");
            BinaryTree.inorder(bt.root);
        }
        else if(choose == 2){
            System.out.println("\nPreorder =  Parent - Left Child - Right Child");
            BinaryTree.preorder((bt.root));
        }
        else if(choose == 3){
            System.out.println("\nPostorder: Left Child - Right Child - Parent");
            BinaryTree.postorder(bt.root);
        }
        else{
            System.out.println("\nWrong input");
        }

        //3. Sum up all the value inside the binary tree
        System.out.println("\n\nPROBLEM 3: ");
        System.out.print("The total sum of data inside the binary tree is: " + BinaryTree.sumValue(bt.root));

        //4. Display the maximum height of the binary tree
        System.out.println("\n\nPROBLEM 4: ");
        System.out.print("The height of the binary tree is: " + (BinaryTree.getHeight(bt.root)-1));

        //5. display nodes based on it's depth of level in binary tree
```