JOBSHEET 14 - *BINARY TREE*

1. COMPETENCIES

- Students are able to understand the Binary tree model
- Students are able to create and declare the structure of a Binary tree algorithm.
- Students are able to apply and implement the Binary search tree

2. PRACTICUM

In this practicum, we will try to implement Binary search tree with basic operations, using arrays (practicum 2) and linked lists (practicum 1).

PRACTICUM 1

Node
data: int left: Node right: Node
Node(left: Node, data:int, right:Node)

BinaryTree
root: Node size : int
DoubleLinkedLists() add(data: int): void find(data: int) : boolean traversePreOrder (node : Node) : void traversePostOrder (node : Node) void traverseInOrder (node : Node): void getSuccessor (del: Node) add(item: int, index:int): void delete(data: int): void

1. Create `Node` , `BinaryTree` dan `BinaryTreeMain,` class
2. In the `Node` class, add **data**, **left, and right** attributes as well as default and constructors with parameters

```
3    public class Node {
4        int data;
5        Node left;
6        Node right;
7
8        public Node(){
9        }
10       public Node(int data){
11           this.left = null;
12           this.data = data;
13           this.right = null;
14       }
15   }
```

3. In the `BinaryTree` class, add **root** attribute.

```
3    public class BinaryTree {
4        Node root;
5
```

4. Add a default constructor and the **isEmpty()** method in the **BinaryTree** class

```
6        public BinaryTree(){
7            root = null;
8        }
9        boolean isEmpty(){
10           return root==null;
11       }
```

5. Create **add()** method in the **BinaryTree class**. In the following image, the process of adding nodes is **not done recursively**, so that it is easier to see the process of adding nodes in a tree. Actually, if it's done with recursive approach, the writing code will be more efficient.

```
12       void add(int data){
13           if(isEmpty()){//tree is empty
14               root = new Node(data);
15           }else{
16               Node current = root;
17               while(true){
18                   if(data<current.data){
19                       if(current.left!=null){
20                           current = current.left;
21                       }else{
22                           current.left = new Node(data);
23                           break;
24                       }
25                   }else if(data>current.data){
26                       if(current.right!=null){
27                           current = current.right;
28                       }else{
29                           current.right = new Node(data);
30                           break;
31                       }
32                   }else{//data is already exist
33                       break;
34                   }
35               }
36           }
37       }
```

6. Create **find() method**

```
38      boolean find(int data){
39          boolean hasil = false;
40          Node current = root;
41          while(current!=null){
42              if(current.data==data){
43                  hasil = true;
44                  break;
45              }else if(data<current.data){
46                  current = current.left;
47              }else{
48                  current = current.right;
49              }
50          }
51          return hasil;
52      }
```

7. Create the **traversePreOrder(), traverseInOrder() and traversePostOrder()** methods. All of hese traverse methods are used to visit and display nodes in the tree.

```
53      void traversePreOrder(Node node) {
54          if (node != null) {
55              System.out.print(" " + node.data);
56              traversePreOrder(node.left);
57              traversePreOrder(node.right);
58          }
59      }
60      void traversePostOrder(Node node) {
61          if (node != null) {
62              traversePostOrder(node.left);
63              traversePostOrder(node.right);
64              System.out.print(" " + node.data);
65          }
66      }
67      void traverseInOrder(Node node) {
68          if (node != null) {
69              traverseInOrder(node.left);
70              System.out.print(" " + node.data);
71              traverseInOrder(node.right);
72          }
73      }
```

8. Add the **getSuccessor()** method. This method will be used during the process of deleting a node that has 2 children.

```
74      Node getSuccessor(Node del){
75          Node successor = del.right;
76          Node successorParent = del;
77          while(successor.left!=null){
78              successorParent = successor;
79              successor = successor.left;
80          }
81          if(successor!=del.right){
82              successorParent.left = successor.right;
83              successor.right = del.right;
84          }
85          return successor;
86      }
```

9. Create the **delete() method.**

```
87      void delete(int data){
88
89      }
```

In the delete method, we need to add a further validation whether the tree is empty or not. If not find the position of the node to be deleted.

```
88          if(isEmpty()){
89              System.out.println("Tree is empty!");
90              return;
91          }
92          //find node (current) that will be deleted
93          Node parent = root;
94          Node current = root;
95          boolean isLeftChild = false;
96          while(current!=null){
97              if(current.data==data){
98                  break;
99              }else if(data<current.data){
100                 parent = current;
101                 current = current.left;
102                 isLeftChild = true;
103             }else if(data>current.data){
104                 parent = current;
105                 current = current.right;
106                 isLeftChild = false;
107             }
108         }
```

Then add the process of deleting the current node that has been found.

```
109             //deletion
110             if(current==null){
111                 System.out.println("Couldn't find data!");
                    return;
113             }else{
114                 //if there is no child, simply delete it
115                 if(current.left==null&&current.right==null){
116                     if(current==root){
117                         root  = null;
118                     }else{
119                         if(isLeftChild){
120                             parent.left = null;
121                         }else{
122                             parent.right = null;
123                         }
124                     }
125                 }else if(current.left==null){//if there is 1 child (right)
126                     if(current==root){
127                         root = current.right;
128                     }else{
129                         if(isLeftChild){
130                             parent.left = current.right;
131                         }else{
132                             parent.right = current.right;
133                         }
134                     }
135                 }else if(current.right==null){//if there is 1 child (left)
136                     if(current==root){
137                         root = current.left;
138                     }else{
139                         if(isLeftChild){
140                             parent.left = current.left;
141                         }else{
142                             parent.right = current.left;
143                         }
144                     }
145                 }else{//if there are 2 childs
146                     Node successor = getSuccessor(current);
147                     if(current==root){
148                         root = successor;
149                     }else{
150                         if(isLeftChild){
151                             parent.left = successor;
152                         }else{
153                             parent.right = successor;
154                         }
155                     }
156                     successor.left = current.left;
157                 }
158             }
```

10. Open the **BinaryTreeMain** class and add the **main()** method.

```
3   public class BinaryTreeMain {
4       public static void main(String[] args) {
5           BinaryTree bt = new BinaryTree();
6
7           bt.add(6);
8           bt.add(4);
9           bt.add(8);
10          bt.add(3);
11          bt.add(5);
12          bt.add(7);
13          bt.add(9);
14          bt.add(10);
15          bt.add(15);
16
17          bt.traversePreOrder(bt.root);
18          System.out.println("");
19          bt.traverseInOrder(bt.root);
20          System.out.println("");
21          bt.traversePostOrder(bt.root);
22          System.out.println("");
23          System.out.println("Find "+bt.find(5));
24          bt.delete(8);
25          bt.traversePreOrder(bt.root);
26          System.out.println("");
27      }
28  }
```

11. Compile and run the **BinaryTreeMain** class to get more understanding of how the program tree was created.

12. Observe the results.

PRACTICUM 2

1. In this experiment, the data tree is stored in an array and entered directly from the **main()** method, and then the traversal process is simulated in the order.
2. Create `BinaryTreeArray` and `BinaryTreeArrayMain` class
3. Create **data and idxLast** attributes in the **BinaryTreeArray** class. Also create **populateData () and traverseInOrder ()** methods.

```
3   public class BinaryTreeArray {
4       int[] data;
5       int idxLast;
6
7       public BinaryTreeArray(){
8           data = new int[10];
9       }
10      void populateData(int data[], int idxLast){
11          this.data = data;
12          this.idxLast = idxLast;
13      }
14      void traverseInOrder(int idxStart){
15          if(idxStart<=idxLast){
16              traverseInOrder(2*idxStart+1);
17              System.out.print(data[idxStart]+" ");
18              traverseInOrder(2*idxStart+2);
19          }
20      }
21  }
```

4. Then in the **BinaryTreeArrayMain** class create the **main()** method as shown below.

```
3   public class BinaryTreeArrayMain {
4       public static void main(String[] args) {
5           BinaryTreeArray bta = new BinaryTreeArray();
6           int[] data = {6,4,8,3,5,7,9,0,0,0};
7           int idxLast = 6;
8           bta.populateData(data, idxLast);
9           bta.traverseInOrder(0);
10      }
11  }
```

5. Run the BinaryTreeArrayMain class and observe the results!

4. QUESTIONS

1. Why the data searching process is more efficient in the Binary search tree than in ordinary binary tree?
2. Why do we need the **Node** class? what are the **left** and **right** attributes?
3. a. What are the uses of the **root** attribute in the **BinaryTree** class?
   b. When the tree object was first created, what is the value of **root**?
4. When the tree is still empty, and a new node is added, what process will happen?
5. Pay attention to the **add()** method, in which there are program lines as below. Explain in detail what the program line is for?

```
if(data<current.data){
   if(current.left!=null){
      current = current.left;
   }else{
      current.left = new Node(data);
      break;
   }
}
```

6. What is the difference between pre-order, in-order and post-order traverse modes?

7. Look at the **delete()** method. Before the node removal process, it is preceded by the process of finding the node to be deleted. Besides intended to find the node to be deleted (current), the search process will also look for the parent of the node to be deleted (parent). In your opinion, why is it also necessary to know the parent of the node to be deleted?
8. For what is a variable named isLeftChild created in the **delete()** method?
9. What is the **getSuccessor()** method for?
10. In a theoretical review, it is stated that when a node that has 2 children is deleted, the node is replaced by the successor node, where the successor node can be obtained in 2 ways, namely 1) looking for the largest value of the subtree to the left, or 2) looking for the smallest value of subtree on the right. Which 1 of 2 methods is implemented in the **getSuccessor()** method in the above program?
11. What are the uses of the data and idxLast attributes in the **BinaryTreeArray** class?
12. What are the uses of the **populateData**() and **traverseInOrder**() methods?
13. If a binary tree node is stored in index array 2, then in what index are the left-child and rigth child positions respectively?

5. ASSIGNMENTS

1. Create a method inside the **BinaryTree** class that will add nodes with recursive approach.
2. Create a method in the **BinaryTree** class to display the smallest and largest values in the tree.
3. Create a method in the **BinaryTree** class to display the data in the leaf.
4. Create a method in the **BinaryTree** class to display the number of leaves in the tree.
5. Modify the **BinaryTreeMain** class, so that it has a menu option:
   a. *add*
   b. *delete*
   c. *find*
   d. *traverse inOrder*
   e. *traverse preOrder*
   f. *traverse postOrder*
   g. keluar
6. Modify the **BinaryTreeArray** class, and add:
   a. Add add method (int data) to enter data into the tree
   b. **traversePreOrder()** and **traversePostOrder()** methods