Name : Azaria Cindy Sahasika
Number Id : 2341760169 / 06
Class : 1G – Business Information System
Lesson : Algorithm and Data Structure
Material : Material 11 – Tree
Github Link : https://github.com/azariacindy/algorithm-ds

JOBSHEET
Tree

12.1 Tujuan Praktikum

Setelah melakukan praktikum ini, mahasiswa mampu:

1. memahami model *Tree* khususnya *Binary Tree*
2. membuat dan mendeklarasikan struktur algoritma *Binary Tree*.
3. menerapkan dan mengimplementasikan algoritma *Binary Tree* dalam kasus *Binary Search Tree*

12.2 Kegiatan Praktikum 1

Implementasi Binary Search Tree menggunakan Linked List (45 Menit)

12.2.1 Percobaan 1

Pada percobaan ini akan diimplementasikan Binary Search Tree dengan operasi dasar, dengan menggunakan array (praktikum 2) dan linked list (praktikum 1). Sebelumnya, akan dibuat class Node, dan Class BinaryTree

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. Buatlah class **Node**, **BinaryTree** dan **BinaryTreeMain**
2. Di dalam class **Node**, tambahkan atribut **data**, **left** dan **right**, serta konstruktor default dan berparameter.

```
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. Di dalam class **BinaryTree**, tambahkan atribut **root**.

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

4. Tambahkan konstruktor default dan method **isEmpty()** di dalam class **BinaryTree**

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

4.4 Tambahkan method **add()** di dalam class **BinaryTree**. Di bawah ini proses penambahan node **tidak dilakukan secara rekursif**, agar lebih mudah dilihat alur proses penambahan node dalam tree. Sebenarnya, jika dilakukan dengan proses rekursif, penulisan kode akan lebih efisien.sus no

```
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      }
```

5. Tambahkan method find()

```
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      }
```

6. Tambahkan method **traversePreOrder()**, **traverseInOrder()** dan **traversePostOrder()**. Method traverse digunakan untuk mengunjungi dan menampilkan node-node dalam tree, baik dalam mode pre-order, in-order maupun post-order.

```
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      }
```

7. Tambahkan method **getSuccessor()**. Method ini akan digunakan ketika proses penghapusan node yang memiliki 2 child.

```
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      }
```

8. Tambahkan method **delete()**.

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

Di dalam method delete tambahkan pengecekan apakah tree kosong, dan jika tidak cari posisi node yang akan di hapus.

```
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             }
```

Kemudian tambahkan proses penghapusan terhadap node current yang telah ditemukan.

```
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 is 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                         successor.left = current.left;
156                     }
157                 }
158             }
```

9. Buka class BinaryTreeMain dan tambahkan method main().

```
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  }
```

10. Compile dan jalankan class BinaryTreeMain untuk mendapatkan simulasi jalannya program tree yang telah dibuat.
11. Amati hasil running tersebut.

```
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15
```

12.2.2 Pertanyaan Percobaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?
   ➔ Karena binary search tree node ini memiliki properti dimana nilai semua node di subTree left lebih kecil daripada nilai node parent, dan nilai semua node di subtree right lebih besar daripada nilai node parent. Pencarian data dilakukan secara lebih efisien dengan membandingkan nilai yang dicari dengan nilai node saat ini dan memilih subtree yang sesuai.
2. Untuk apakah di class **Node**, kegunaan dari atribut **left** dan **right**?
   ➔ Left : menunjuk ke node child yang lebih kecil dalam binary search tree saat ini.
   ➔ Right : menunjuk ke node child yang lebih besar dalam binary search tree saat ini.
3. a. Untuk apakah kegunaan dari atribut **root** di dalam class **BinaryTree**?
   - untuk menyimpan referensi ke node akar dari binary tree
   b. Ketika objek tree pertama kali dibuat, apakah nilai dari **root**?
   - unutk menunjukkan bahwa tree tersebut kosong dan belum memiliki node apa pun.
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
   ➔ Node baru akan ditambahkan menjadi root dari tree tersebut. Kemudian dilakukan pengalokasian node baru dan menetapkan atribut root ke node baru.

5. Perhatikan method **add()**, di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?

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

```
if(data < current.data) { // Checks whether the data to be added is smaller than the current node's data. If yes, then the new data should be placed in the left subtree.
    if(current.left != null) { // Checks whether the left child of the current node exists (not null). If it does, it means that it is necessary to continue the comparison further in the left subtree.
        current = current.left; // Updates the current node's reference to the left child, and the loop will continue from this child node in the next iteration.
    } else { // If the left child of the current node is null, then the right position for the new data is here.
        current.left = new Node06(data); // Creates a new node with the data to be added and sets it as the left child of the current node.
        break; // stop looping
    }
```

12.3 Kegiatan Praktikum 2

Implementasi binary tree dengan array (45 Menit)

13.3.1 Tahapan Percobaan

1. Di dalam percobaan implementasi binary tree dengan array ini, data tree disimpan dalam array dan langsung dimasukan dari method main(), dan selanjutnya akan disimulasikan proses traversal secara inOrder.
2. Buatlah class **BinaryTreeArray** dan **BinaryTreeArrayMain**
3. Buat atribut **data** dan **idxLast** di dalam class **BinaryTreeArray**. Buat juga method **populateData()** dan **traverseInOrder().**

```
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. Kemudian dalam class **BinaryTreeArrayMain** buat method main() seperti gambar berikut ini.

```
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. Jalankan class **BinaryTreeArrayMain** dan amati hasilnya! 

12.3.2 Pertanyaan Percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class **BinaryTreeArray**?
   ➔ Fungsi atribut data: menyimpan data array
   ➔ Fungsi idxLast: menyimpan batas index array
2. Apakah kegunaan dari method **populateData()**?
   ➔ Unutk menunjukkan data pada idxLast
3. Apakah kegunaan dari method **traverseInOrder()**?
   ➔ Untuk menelusuri tree untuk menggunakan metode order dan prinsip left visit right.
4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?
   ➔ Jika array dari 0, maka left child index ke-5 dan right child index ke-6.
5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
   ➔ Untuk menunjukkan idxLast / max index arraynya merupakan 6.

11.4 Tugas Praktikum

Waktu pengerjaan: 90 menit

1. Buat method di dalam class **BinaryTree** yang akan menambahkan node dengan cara rekursif.

```
172     // no1
173     void addRecursive(int data) {
174         root = addRecursive(root, data);
175     }
176
177     Node06 addRecursive(Node06 current, int data) {
178         if (current == null) {
179             return new Node06(data);
180         }
181
182         if (data < current.data) {
183             current.left = addRecursive(current.left, data);
184         } else if (data > current.data) {
185             current.right = addRecursive(current.right, data);
186         } else {
187             // Data already exists
188             return current;
189         }
190
191         return current;
192     }
```

2. Buat method di dalam class **BinaryTree** untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.

```
194     // no2
195     int findMinValue() {
196         if (root == null) {
197             throw new IllegalStateException(s:"Tree is empty");
198         }
199         return findMinValue(root);
200     }
201
202     private int findMinValue(Node06 node) {
203         return node.left == null ? node.data : findMinValue(node.left);
204     }
205
206     int findMaxValue() {
207         if (root == null) {
208             throw new IllegalStateException(s:"Tree is empty");
209         }
210         return findMaxValue(root);
211     }
212
213     private int findMaxValue(Node06 node) {
214         return node.right == null ? node.data : findMaxValue(node.right);
215     }
```

```
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15
Data Max:
15
Data Min:
3
```

```
28          System.out.println(x:"Data Max: ");
29          bt.max();
30          System.out.println(x:"Data Min: ");
31          bt.min();
```

3. Buat method di dalam class **BinaryTree** untuk menampilkan data yang ada di leaf.

```
218     // no3
219     void printLeafNodes() {
220         printLeafNodes(root);
221     }
222
223     private void printLeafNodes(Node06 node) {
224         if (node != null) {
225             if (node.left == null && node.right == null) {
226                 System.out.print(node.data + " ");
227             }
228             printLeafNodes(node.left);
229             printLeafNodes(node.right);
230         }
231     }
```

```
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15
Data Max:
15
Data Min:
3
Data at leaf:
3 5 7 15
```

```
33          System.out.println(x:"Data at leaf: ");
34          bt.printLeafNodes();
```

4. Buat method di dalam class **BinaryTree** untuk menampilkan berapa jumlah leaf yang ada di dalam tree.

```
233     // no4
234     int countLeafNodes() {
235         return countLeafNodes(root);
236     }
237
238     private int countLeafNodes(Node06 node) {
239         if (node == null) {
240             return 0;
241         }
242         if (node.left == null && node.right == null) {
243             return 1;
244         }
245         return countLeafNodes(node.left) + countLeafNodes(node.right);
246     }
```

```
33          System.out.println(x:"Data at leaf: ");
34          bt.printLeafNodes();
35          System.out.println(x:"");
36          System.out.println("Jumlah data leaf: " + bt.countLeafNodes());
```

```
Data at leaf:
3 5 7 15
Jumlah data leaf: 4
```

5. Modifikasi class **BinaryTreeArray**, dan tambahkan :
   - method **add(int data)** untuk memasukan data ke dalam tree
   - method **traversePreOrder()** dan **traversePostOrder()**

jobsheet11_Tree > pract2Tree > BinaryTreeArray06.java > ...

```java
package pract2Tree;

public class BinaryTreeArray06 {
    int[] data;
    int idxLast;

    public BinaryTreeArray06() {
        data = new int[10];
    }

    void populateData(int data[], int idxLast) {
        this.data = data;
        this.idxLast = idxLast;
    }

    void add(int data) {
        if (idxLast == this.data.length - 1) {
            System.out.println(x:"The tree is full");
            return;
        }
        idxLast++;
        this.data[idxLast] = data;
    }

    void traverseInOrder(int idxStart) {
        if(idxStart <= idxLast) {
            traverseInOrder(2 * idxStart + 1);
            System.out.print(data[idxStart] + " ");
            traverseInOrder(2 * idxStart + 2);
        }
    }

    void traversePreOrder(int idxStart) {
        if (idxStart <= idxLast) {
            System.out.print(this.data[idxStart] + " ");
            traversePreOrder(2 * idxStart + 1);
            traversePreOrder(2 * idxStart + 2);
        }
    }

    void traversePostOrder(int idxStart) {
        if (idxStart <= idxLast) {
            traversePostOrder(2 * idxStart + 1);
            traversePostOrder(2 * idxStart + 2);
            System.out.print(this.data[idxStart] + " ");
        }
    }
}
```

jobsheet11_Tree > pract2Tree > BinaryTreeArrayMain06.java > ...

```java
package pract2Tree;

public class BinaryTreeArrayMain06 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray06 bta = new BinaryTreeArray06();
        int[] data = {6, 4, 8, 3, 7, 9, 0, 0, 0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        bta.traverseInOrder(idxStart:0);

        System.out.println(x:"In-Order Traversal:");
        bta.traverseInOrder(idxStart:0);
        System.out.println();

        System.out.println(x:"Pre-Order Traversal:");
        bta.traversePreOrder(idxStart:0);
        System.out.println();

        System.out.println(x:"Post-Order Traversal:");
        bta.traversePostOrder(idxStart:0);
        System.out.println();

        // Adding a new data to the tree
        bta.add(data:5);
        bta.add(data:10);

        System.out.println(x:"In-Order Traversal after adding new data:");
        bta.traverseInOrder(idxStart:0);
        System.out.println();

        System.out.println(x:"Pre-Order Traversal after adding new data:");
        bta.traversePreOrder(idxStart:0);
        System.out.println();

        System.out.println(x:"Post-Order Traversal after adding new data:");
        bta.traversePostOrder(idxStart:0);
        System.out.println();
    }
}
```

```
3 4 7 6 9 8 0
In-Order Traversal:
3 4 7 6 9 8 0
Pre-Order Traversal:
6 4 3 7 8 9 0
Post-Order Traversal:
3 7 4 9 0 8 6
In-Order Traversal after adding new data:
5 3 10 4 7 6 9 8 0
Pre-Order Traversal after adding new data:
6 4 3 5 10 7 8 9 0
Post-Order Traversal after adding new data:
5 10 3 7 4 9 0 8 6
```