Nama : Deanissa Sherly Sabilla

Kelas / Absen : SIB 1B / 06

Mata Kuliah : Pratikum Algoritma Struktur Data

Pertemuan : 14

JOBSHEET
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      }
```

5. 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.

```
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. 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      }
```

7. 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      }
```

8. 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      }
```

9. 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!");
113                 return;
114             }else{
115                 //if there is no child, simply delete it
116                 if(current.left==null&&current.right==null){
117                     if(current==root){
118                         root  = null;
119                     }else{
120                         if(isLeftChild){
121                             parent.left = null;
122                         }else{
123                             parent.right = null;
124                         }
125                     }
126                 }else if(current.left==null){//if there is 1 child (right)
127                     if(current==root){
128                         root = current.right;
129                     }else{
130                         if(isLeftChild){
131                             parent.left = current.right;
132                         }else{
133                             parent.right = current.right;
134                         }
135                     }
136                 }else if(current.right==null){//if there is 1 child (left)
137                     if(current==root){
138                         root = current.left;
139                     }else{
140                         if(isLeftChild){
141                             parent.left = current.left;
142                         }else{
143                             parent.right = current.left;
144                         }
145                     }
146                 }else{//if there is 2 childs
147                     Node successor = getSuccessor(current);
148                     if(current==root){
149                         root = successor;
150                     }else{
151                         if(isLeftChild){
152                             parent.left = successor;
153                         }else{
154                             parent.right = successor;
155                         }
156                         successor.left = current.left;
157                     }
158                 }
```

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

```java
public class BinaryTreeMain {
    public static void main(String[] args) {
        BinaryTree bt = new BinaryTree();

        bt.add(6);
        bt.add(4);
        bt.add(8);
        bt.add(3);
        bt.add(5);
        bt.add(7);
        bt.add(9);
        bt.add(10);
        bt.add(15);

        bt.traversePreOrder(bt.root);
        System.out.println("");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        bt.traversePostOrder(bt.root);
        System.out.println("");
        System.out.println("Find "+bt.find(5));
        bt.delete(8);
        bt.traversePreOrder(bt.root);
        System.out.println("");
    }
}
```

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

➢ **Input**

Class Node

Node06.java > Node06

```java
/**
 * Node06
 */
public class Node06 {
    int data;
    Node06 left;
    Node06 right;

    public Node06() {

    }
    public Node06(int data) {
        this.left = null;
        this.data = data;
        this.right = null;
    }
}
```

Class Binary Tree

```java
public class BinaryTree06 {
    Node06 root;
public BinaryTree06(){
    root = null;
}
boolean isEmpty() {
    return root == null;
}
void add (int data) {
    if(isEmpty()){
        root = new Node06(data);
    } else {
        Node06 current = root;
        while (true) {
            if (data<current.data){
                if(current.left!=null){
                    current = current.left;
                } else {
                    current.left = new Node06(data);
                    break;
                }
            }else if (data>current.data){
                if(current.right != null) {
                    current = current.right;
                } else {
                    current.right = new Node06(data);
                    break;
                }
            } else {
                break;
            }
        }
    }
}
boolean find(int data) {
    boolean hasil = false;
    Node06 current = root;
    while (current != null) {
        if (current.data== data) {
            hasil = true;
            break;
        } else if (data<current.data) {
            current = current.left;
        } else {
            current = current.right;
        }
    }
    return hasil;
}
```

```java
boolean find(int data) {
    boolean hasil = false;
    Node06 current = root;
    while (current != null) {
        if (current.data== data) {
            hasil = true;
            break;
        } else if (data<current.data) {
            current = current.left;
        } else {
            current = current.right;
        }
    }
    return hasil;
}
Node06 getSuccessor (Node06 del) {
    Node06 successor = del.right;
    Node06 successorParent = del;
    while(successor.left != null) {
        successorParent = successor;
        successor = successor.left;
    }
    if (successor !=del.right){
        successorParent.left = successor.right;
        successor.right = del.right;
```

```
    return successor;
}
void delete (int data) {
    if(isEmpty()){
        System.out.println(x:"Tree is empty");
        return;
    }
    Node06 parent = root;
    Node06 current = root;
    boolean isLeftChild = false;
    while(current != null){
        if (current.data == data) {
            break;
        } else if (data < current.data) {
            parent = current;
        }
    }
    if (current==null){
        System.out.println(x:"Cloud find data!");
        return;
    } else {
        if (current.left==null&&current.right==null) {
            if (current == root) {
                root = null;
            } else {
                if (isLeftChild){
                    parent.left = null;
                } else {
                    parent.right = null;
                }
            }
        } else if (current.left == null) {
            if (current == root) {
                root = current.right;
            } else {
                if (isLeftChild) {
                    parent.left =  current.right;
                } else {
                    parent.right = current.right;
                }
            }
        } else if (current.right == null){
            if (current == root) {
                root = current.left;
            } else {
                if (isLeftChild){
                    parent.left = current.right;
                } else {
                    parent.right = current.left;
                }
            }
        } else {
            Node06 successor = getSuccessor(current);
            if (current == root) {
                root = successor;
            } else {
                if (isLeftChild) {
                    parent.left = successor;
                } else {
                    parent.right = successor;
                }
                successor.left = current.left;
            }
        }
    }
}
public void traversePreOrder(Node06 node) {
    if (node != null) {
        System.out.print(" "+node.data);
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
public void traverseInOrder(Node06 node) {
    if (node != null) {
        traverseInOrder(node.left);
        System.out.print(" "+node.data);
        traverseInOrder(node.right);
    }
}
public void traversePostOrder(Node06 node) {
    if (node != null) {
        traverseInOrder(node.left);
        traverseInOrder(node.right);
        System.out.print(" "+node.data);
    }
}
}
```

Class Main

```java
public class BinaryTreeMain06 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTree06 bt = new BinaryTree06();
        bt.add(data:6);
        bt.add(data:4);
        bt.add(data:8);
        bt.add(data:3);
        bt.add(data:5);
        bt.add(data:7);
        bt.add(data:9);
        bt.add(data:10);
        bt.add(data:11);

        bt.traversePreOrder(bt.root);
        System.out.println(x:"");
        bt.traverseInOrder(bt.root);
        System.out.println(x:"");
        bt.traversePostOrder(bt.root);
        System.out.println(x:"");
        System.out.println("Find " +bt.find(data:5));
        bt.delete(data:8);
        bt.traversePreOrder(bt.root);
        System.out.println(x:"");
    }
}
```

➢ **Hasil Output**

```
PS C:\Users\TOSHIBA\Semester 2\Pertemuan14>  & 'C:\Program Files\Java\
C:\Users\TOSHIBA\AppData\Roaming\Code\User\workspaceStorage\d8ad5da517
inaryTreeMain06'
 6 4 3 5 8 7 9 10 11
 3 4 5 6 7 8 9 10 11
 3 4 5 7 8 9 10 11 6
Find true
```

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 memiliki aturan left-child harus lebih kecil daripada right-child dan parent-nya dan Binary Search Tree dapat mengatasi kelemahan dalam pencarian node tertentu dalam binary search.**
2. Untuk apakah di class **Node**, kegunaan dari atribut **left** dan **right**?
   - **Left : Menghubungkan ke sub-pohon kiri yaitu Atribut left menunjuk ke anak kiri dari node saat ini. Node yang ditunjuk oleh left berisi nilai yang lebih kecil daripada nilai di node saat ini.**
   - **Right : Menghubungkan ke sub-pohon kanan yaitu Atribut right menunjuk ke anak kanan dari node saat ini. Node yang ditunjuk oleh right berisi nilai yang lebih besar daripada nilai di node saat ini.**
3. a. Untuk apakah kegunaan dari atribut **root** di dalam class **BinaryTree**?
   - **Digunakan untuk menyimpan referensi ke node pertama atau node utama dari pohon (tree). Tanpa root, tidak akan ada cara untuk mengakses node lain dalam tree karena semua node terhubung melalui root.**

   b. Ketika objek tree pertama kali dibuat, apakah nilai dari **root**?
   - **Adalah null.**
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
   - **Pertama, method add akan mengecek apakah tree kosong dengan menggunakan metode isEmpty yang memeriksa apakah root bernilai null, selanjutnya jika tree kosong, metode akan membuat node baru dan mengatur root untuk menunjuk ke node baru ini.**
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){ : perbandingan data, menentukan apakah nilai data yang akan ditambahkan (data) lebih kecil daripada nilai data di node saat ini (current.data).**
   - **if(current.left!=null){ : memeriksa anak kiri, mengecek apakah node saat ini (current) memiliki anak kiri (left).**
   - **current = current.left; : pindah ke anak kiri, memindahkan referensi current ke node anak kiri.**
   - **current.left = new Node(data); : menambahkan node baru sebagai anak kiri, membuat node baru dengan nilai data yang diberikan dan menempatkannya sebagai anak kiri dari node saat ini (current).**

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!

➢ Input

Binary Tree Array

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

Class Main

```java
public class BinaryArrayMain06 {
    public static void main(String[] args) {
        BinaryTreeArray06 bta = new BinaryTreeArray06();
        int[] data = {6,4,8,3,5,7,9,0,0,0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        bta.traverseInOrder(idxStart:0);
    }
}
```

➢ Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\TOSHIBA\Semester 2\Pertemuan14>  & 'C:\Program
'-cp' 'C:\Users\TOSHIBA\AppData\Roaming\Code\User\workspace
e3bcb7\bin' 'BinaryArrayMain06'
3 4 5 6 7 8 9
PS C:\Users\TOSHIBA\Semester 2\Pertemuan14>
```

12.3.2 Pertanyaan Percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class **BinaryTreeArray**?
   - **data : Digunakan untuk menyimpan elemen-elemen dari binary tree dalam representasi berbasis array.**
   - **idxLast : Digunakan menunjukkan indeks dari elemen terakhir dalam array data yang merupakan bagian dari tree.**
2. Apakah kegunaan dari method **populateData()**?
   - **Digunakan untuk mengisi array data dengan elemen-elemen yang mewakili node-node dari binary tree.**
3. Apakah kegunaan dari method **traverseInOrder()**?
   - **Digunakan untuk melakukan traversal in-order pada binary tree yang diwakili oleh array data.**
4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?
   - **posisi anak kiri (left child) dan anak kanan (right child) dari suatu node dengan indeks tertentu dapat dihitung menggunakan rumus berikut:**
     Left child : 2 * i + 1.
     Right child : 2 * i + 2.
     Menghitung Posisi Anak Kiri dan Anak Kanan
     Jika suatu node disimpan dalam array pada indeks 2:
     Left child: 2 * 2 + 1 = 4 + 1 = 5
     Right child: 2 * 2 + 2 = 4 + 2 = 6
     Jadi, jika node disimpan pada indeks 2:
     Posisi left child ada di indeks 5.
     Posisi right child ada di indeks 6.
5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
   - **Pernyataan int idxLast = 6; digunakan untuk memastikan bahwa traversal in-order yang dilakukan oleh program hanya berjalan hingga ke node terakhir yang valid dalam tree yang diwakili oleh array.**

12.4 Tugas Praktikum

Waktu pengerjaan: 90 menit

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

```
148
149   Rekursif (Menambahkan Node)
150   public Node06 addRekursif(Node06 current, int data) {
151       if (current == null) {
152           return new Node06(data);
153       } if (data < current.data) {
154           current.left = addRekursif(current.left, data);
155       } else if (data > current.data) {
156           current.right = addRekursif(current.right, data);
157       }
158       return current;
```

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

```
160
161   // Menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree
162   public int findMin() {
163       if (isEmpty()) {
164           throw new IllegalStateException(s:"Tree is empty");
165       }
166       Node06 current = root;
167       while (current.left != null) {
168           current = current.left;
169       }
170       return current.data;
171   }
```

```
172   public int findMax() {
173       if (isEmpty()) {
174           throw new IllegalStateException(s:"Tree is empty");
175       }
176       Node06 current = root;
177       while (current.right != null) {
178           current = current.right;
179       }
180       return current.data;
181   }
```

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

```
182   //Menampilkan data yang ada di leaf
183   public void displayLeafNodes(Node06 node) {
184       if (node != null) {
185           if (node.left == null && node.right == null) {
186               System.out.print(" " + node.data);
187           }
188           displayLeafNodes(node.left);
189           displayLeafNodes(node.right);
190       }
191   }
```

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

```
192   //Menampilkan berapa jumlah leaf yang ada di dalam tree
193   public int countLeafNodes(Node06 node) {
194       if (node == null) {
195           return 0;
196       }
197       if (node.left == null && node.right == null) {
198           return 1;
199       } else {
200           return countLeafNodes(node.left) + countLeafNodes(node.right);
201       }
202   }
```

➢ **Tambahan pada bagian main :**

```
29            System.out.println("Min : " + bt.findMin());
30            System.out.println("Max : " + bt.findMax());
31
32            System.out.println(x:"Leaf :");
33            bt.displayLeafNodes(bt.root);
34            System.out.println();
35
36            System.out.println("Jumlah Leaf: " + bt.countLeafNodes(bt.root));
37        }
```

➢ **Hasil Output :**

```
PS C:\Users\TOSHIBA\Semester 2\Pertemuan14>  c:; cd 'c:\Users\TOSHIBA\Semes
va.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\TOSHIBA\A
88ef1c6c22d1c878\redhat.java\jdt_ws\Pertemuan14_afe3bcb7\bin' 'BinaryTreeMa
 6 4 3 5 8 7 9 10 11
 3 4 5 6 7 8 9 10 11
 3 4 5 7 8 9 10 11 6
Min : 3
Max : 11
Leaf :
 3 5 7 11
Jumlah Leaf: 4
```

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

```
    //add [int data]
    void add(int data) {
        if (idxLast + 1 >= this.data.length) {
            resizeArray();
        }
        this.data[++idxLast] = data;
    }
    void resizeArray() {
        int[] newData = new int[this.data.length * 2];
        System.arraycopy(this.data, srcPos:0, newData, destPos:0, this.data.length);
        this.data = newData;
    }
}
```

   - method **traversePreOrder()** dan **traversePostOrder()**

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

➢ **Modifikasi main :**

```
public class BinaryArrayMain06 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray06 bta = new BinaryTreeArray06();
        int[] data = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0};
        int idxLast = 6;
        bta.populateData(data, idxLast);

        System.out.print(s:"In-order : ");
        bta.traverseInOrder(idxStart:0);
        System.out.println();

        System.out.print(s:"Pre-order : ");
        bta.traversePreOrder(idxStart:0);
        System.out.println();

        System.out.print(s:"Post-order : ");
        bta.traversePostOrder(idxStart:0);
        System.out.println();
    }
}
```

➢ **Hasil Output :**

```
78\redhat.java\jdt_ws\Pertemuan14_afe3bcb7\bin' 'BinaryArrayMain0
In-order : 3 4 5 6 7 8 9
Pre-order : 6 4 3 5 8 7 9
Post-order : 3 5 4 7 9 8 6
PS C:\Users\TOSHIBA\Semester 2\Pertemuan14>
```