ALGORITMA STRUKTUR DATA

Praktikum - Tree

Lavina 2341760062

Praktikum 1

Node.java

```java
public class Node {
  int data;
  Node left;
  Node right;

  public Node() {
  }

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

BinaryTree.java

```java
public class BinaryTree {
  Node root;

  public BinaryTree() {
    root = null;
  }

  boolean isEmpty() {
    return root == null;
  }
```

```
12    void add(int data) {
13      if (isEmpty()) {
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 {
33            break;
34          }
35        }
36      }
37    }
38
39    boolean find(int data) {
40      boolean hasil = false;
41      Node current = root;
42      while (current != null) {
43        if (current.data == data) {
44          hasil = true;
45          break;
46        } else if (data < current.data) {
47          current = current.left;
48        } else {
49          current = current.right;
50        }
51      }
52      return hasil;
53    }
```

```
55    void traversePreOrder(Node node) {
56      if (node != null) {
57        System.out.print(" " + node.data);
58        traversePreOrder(node.left);
59        traversePreOrder(node.right);
60      }
61    }
62
63    void traversePostOrder(Node node) {
64      if (node != null) {
65        traversePostOrder(node.left);
66        traversePostOrder(node.right);
67        System.out.print(" " + node.data);
68      }
69    }
70
71    void traverseInOrder(Node node) {
72      if (node != null) {
73        traverseInOrder(node.left);
74        System.out.print(" " + node.data);
75        traverseInOrder(node.right);
76      }
77    }
78
79    Node getSuccessor(Node del) {
80      Node successor = del.right;
81      Node successorParent = del;
82
83      while (successor.left != null) {
84        successorParent = successor;
85        successor = successor.left;
86      }
87      if (successor != del.right) {
88        successorParent.left = successor.right;
89        successor.right = del.right;
90      }
91      return successor;
92    }
93
94    void delete(int data) {
95      if (isEmpty()) {
96        System.out.println("Tree is empty!");
```

```
97          return;
98        }
99        Node parent = root;
100       Node current = root;
101       boolean isLeftChild = false;
102       while (current != null) {
103
104         if (current.data == data) {
105           break;
106         } else if (data < current.data) {
107           parent = current;
108           current = current.left;
109           isLeftChild = true;
110         } else if (data > current.data) {
111           parent = current;
112           current = current.right;
113           isLeftChild = false;
114         }
115       }
116
117       if (current == null) {
118         System.out.println("Couldn't find data!");
119         return;
120       } else {
121         if (current.left == null && current.right == null) {
122           if (current == root) {
123             root = null;
124           } else {
125             if (isLeftChild) {
126               parent.left = null;
127             } else {
128               parent.right = null;
129             }
130           }
131         } else if (current.left == null) {
132           if (current == root) {
133             root = current.right;
134           } else {
135             if (isLeftChild) {
136               parent.left = current.right;
137             } else {
138               parent.right = current.right;
```

```
            }
          }
        } else if (current.right == null) {
          if (current == root) {
            root = current.left;
          } else {
            if (isLeftChild) {
              parent.left = current.left;
            } else {
              parent.right = current.left;
            }
          }
        } else {
          Node successor = getSuccessor(current);
          if (current == root) {
            root = successor;
          } else {
            if (isLeftChild) {
              parent.left = successor;
            } else {
              parent.right = successor;
            }
            successor.left = current.left;
          }
        }
      }
    }

}
```

BinaryTreeMain.java

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

        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:15);

        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:"");
    }
}
```

Output :

```
 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
```

Pertanyaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?
2. Untuk apakah di class **Node**, kegunaan dari atribut **left** dan **right**?

   Jawab : Gunanya sama dengan kegunaan pointer dalam double linked list, left menunjuk pada child kiri dan right menunjuk pada child kanan.
3. a. Untuk apakah kegunaan dari atribut **root** di dalam class **BinaryTree**?

   Jawab : Sebagai node yang berada paling atas dan tidak memiliki parent.

   b. Ketika objek tree pertama kali dibuat, apakah nilai dari **root**?

   Jawab : Kondisi awal **root** berisi nilai **null**
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?

   Jawab : Ketika masih kosong maka node root akan diisi dengan data baru yang ingin ditambahkan.
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;
            }
 }
```

   Jawab : Pertama mengecek apakah data yang ingin ditambah lebih kecil dengan data saat ini, jika true maka dilakukan pengecekan apakah child kiri dari current kosong atau tidak, jika kosong maka current saat ini dipindah ke sebelah kiri tersebut sampai current left nya bernilai null, ketika sudah null diisi dengan data yang baru ingin ditambahkan.

Percobaan 2

BinaryTreeArray.java

```java
public class BinaryTreeArray {
  int[] data;
  int idxLast;

  public BinaryTreeArray() {
    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);
    }
  }
}
```

BinaryTreeArrayMain.java

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

Output :

```
Lavina@LAPTOP-VRURDV67 MING
$ /usr/bin/env C:\\Program
ser\\workspaceStorage\\65b1
3 4 5 6 7 8 9
```

Pertanyaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class **BinaryTreeArray**?

   Jawab : Atribut data berfungsi untuk menyimpan array dari node-node pada tree, sedangkan idxLast adalah variabel untuk menyimpal value index terakhir dalam tree.

2. Apakah kegunaan dari method **populateData()**?

Jawab : Method tersebut berguna sebagai konstruktor yang mengatur kondisi awal dari array data dan isi dari variable idxLast yang didapat melalui parameter.

3. Apakah kegunaan dari method **traverseInOrder()**?

   Jawab : Berguna untuk menelusuri tree secara dengan metode in order.

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?

   Jawab : Posisi left child = 2 * (2) + 1 = 5 dan posisi right child = 2 * (2) + 2 = 6

5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?

   Jawab : Untuk menetapkan index terakhir dari tree menjadi 6, dibuat 6 karena data yang valid untuk dimasukkan kedalam tree hanya ada 6.

Tugas Praktikum

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

```
167   void setRoot(int data) {
168     root = insert(root, data);
169   }
170
171   Node insert(Node current, int data) {
172     if (current == null) {
173       return new Node(data);
174     }
175     if (current.data > data) {
176       current.left = insert(current.left, data);
177     } else if (current.data < data) {
178       current.right = insert(current.right, data);
179     }
180     return current;
181   }
```

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

```
183   int setNodeMin() {
184     return findMin(root);
185   }
186
187   int setNodeMax() {
188     return findMax(root);
189   }
190
191   int findMin(Node node) {
192     return node.left == null ? node.data : findMin(node);
193   }
194
195   int findMax(Node node) {
196     return node.right == null ? node.data : findMax(node);
197   }
```

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

```
199   void displayLeaf(Node node) {
200     if (node == null) {
201       return;
202     }
203
204     if (node.left == null && node.right == null) {
205       System.out.println(node.data);
206     }
207     displayLeaf(node.left);
208     displayLeaf(node.right);
209   }
210 }
```

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

```
211   int amountLeaf(Node node) {
212     if (node == null) {
213       return 0;
214     }
215
216     if (node.left == null && node.right == null) {
217       return 1;
218     }
219     int leftLeaf = amountLeaf(node.left);
220     int rightLeaf = amountLeaf(node.right);
221
222     return leftLeaf + rightLeaf;
223   }
224 }
```

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

```
  void add(int data) {
    if (this.data[0] == 0) {
      this.data[0] = data;
      return;
    } else {
      int i = 0;
      while (true) {
        if (data < this.data[i]) {
          int indexLeft = 2 * i + 1;
          if (this.data[indexLeft] == 0) {
            this.data[indexLeft] = data;
          } else {
            i = indexLeft;
          }
        } else if (data > this.data[i]) {
          int indexRight = 2 * i + 2;
          if (this.data[indexRight] == 0) {
            this.data[indexRight] = data;
          } else {
            i = indexRight;
          }
        }
      }
    }
  }
}
```

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

  Method traversePostOrder()

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

  Method traversePreOrder()

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