Laporan Praktikum

Struktur Data Non Linear DP

Modul 4

Dosen Pengampu

JB. Budi Darmawan S.T., M.Sc.

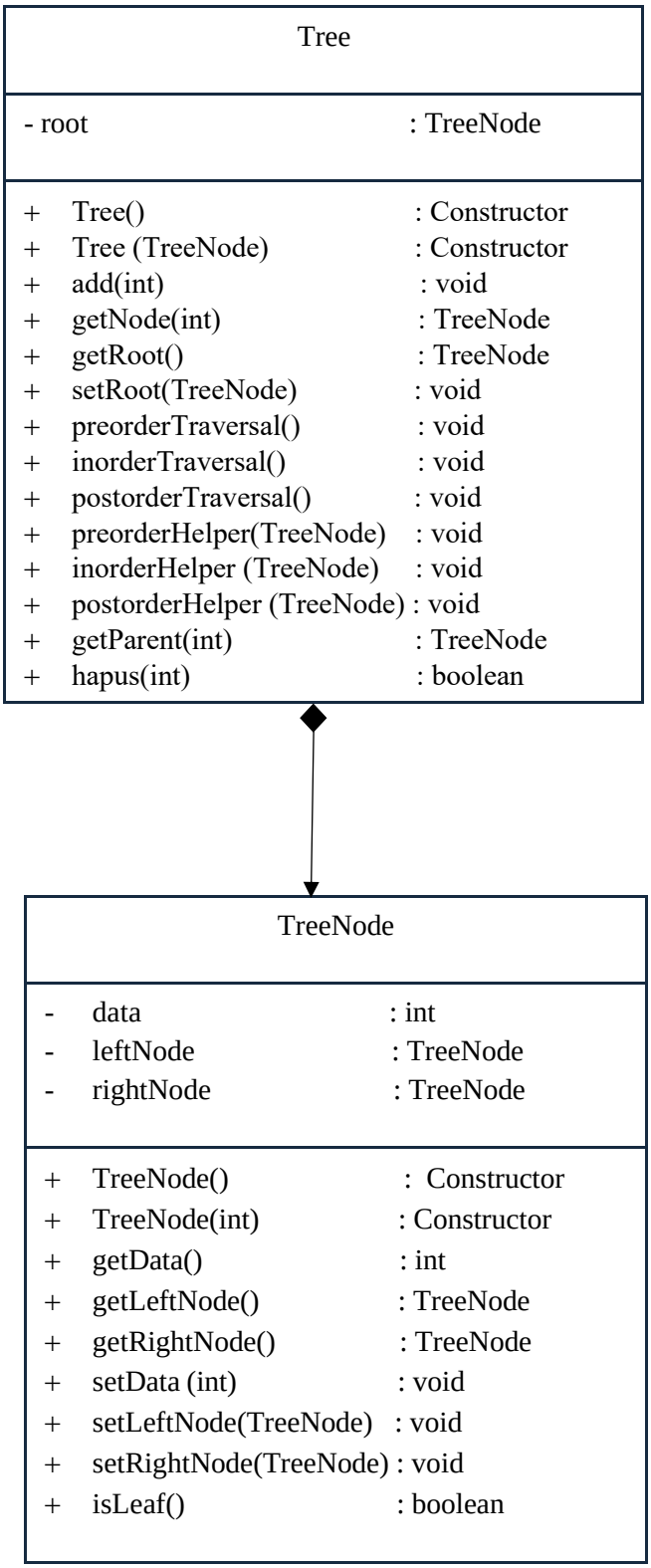
Oleh :

Nama : Maria Gresia Plena Br Purba

NIM : 235314094

PROGRAM STUDI INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS SANATA DHARMA
YOGYAKARTA
2024

1. Diagram UML



2. Source Code

Tree()

```java
package modul4;

public class Tree {

    private TreeNode root;

    public Tree() {
        root = null;
    }

    public Tree(TreeNode root) {
        this.root = root;
    }

    public TreeNode getRoot() {
        return root;
    }

    public void setRoot(TreeNode root) {
        this.root = root;
    }

    public boolean isEmpty() {
        if (root == null) {
            return true;
        } else {
            return false;
        }
    }

    public void add(int x) {
        TreeNode bantu = root;

        if (isEmpty()) {
            root = new TreeNode(data: x);
            return;
        }

        while (true) {
            if (x < bantu.getData()) {
                if (bantu.leftNode == null) {
                    bantu.leftNode = new TreeNode(data: x);
                    return;
                } else {
                    bantu = bantu.leftNode;
                }
            } else if (bantu.rightNode == null) {
                bantu.rightNode = new TreeNode(data: x);
                return;
            } else {
                bantu = bantu.rightNode;
            }
        }
    }

    public TreeNode getNode(int key) {
```

```
57          TreeNode bantu = root;
58
59          while (bantu != null) {
60              if (key == bantu.getData()) {
61                  return bantu;
62              } else if (key < bantu.getData()) {
63                  bantu = bantu.getLeftNode();
64              } else {
65                  bantu = bantu.getRightNode();
66              }
67          }
68          return bantu;
69      }
70
71      public void preorderTraversal() {
72          preorderHelper(localRoot: root);
73      }
74
75      public void inorderTraversal() {
76          inorderHelper(localRoot: root);
77      }
78
79      public void postorderTraversal() {
80          postorderHelper(localRoot: root);
81      }
82
83      public void preorderHelper(TreeNode localRoot) {
84          if (localRoot != null) {
85              System.out.print(localRoot.getData() + " ");
86              preorderHelper(localRoot: localRoot.leftNode);
87              preorderHelper(localRoot: localRoot.rightNode);
88          }
89
90      }
91
92      public void inorderHelper(TreeNode localRoot) {
93          if (localRoot != null) {
94              inorderHelper(localRoot: localRoot.leftNode);
95              System.out.print(localRoot.getData() + " ");
96              inorderHelper(localRoot: localRoot.rightNode);
97          }
98      }
99
100     public void postorderHelper(TreeNode localRoot) {
101         if (localRoot != null) {
102             postorderHelper(localRoot: localRoot.leftNode);
103             postorderHelper(localRoot: localRoot.rightNode);
104             System.out.print(localRoot.getData() + " ");
105
106         }
107     }
108
109     public TreeNode getParent(int key) {
110         TreeNode bantu = root;
111         TreeNode parent = null;
112         while (bantu != null) {
113             if (bantu.data == key) {
114                 return parent;
115             } else if (bantu.data > key) {
116                 parent = bantu;
117                 bantu = bantu.leftNode;
118             } else {
119                 parent = bantu;
120                 bantu = bantu.rightNode;
121             }
122         }
123         return bantu;
124     }
125
126     public boolean hapus(int x) {
127         TreeNode bantu = root;
128
129         bantu = getNode(key:x);
130         if (bantu == null) {
131             return false;
132         } else {
133             if (bantu.data == root.data) {
134                 if (bantu.isLeaf()) {
135                     root = null;
136                 } else if (bantu.rightNode == null) {
137                     root = bantu.leftNode;
138                 } else if (bantu.leftNode == null) {
139                     root = bantu.rightNode;
140                 }
141                 return true;
```

```
            } else {
                TreeNode parent = getParent(key:x);

                if (bantu == null) {
                    return false;
                }
                if (x < parent.getData()) {
                    if (parent.isLeaf()) {
                        parent.leftNode = null;
                    } else if (bantu.rightNode == null) {
                        parent.leftNode = bantu.leftNode;
                    } else if (bantu.leftNode == null) {
                        parent.leftNode = bantu.rightNode;
                    }
                } else {
                    if (bantu.isLeaf()) {
                        parent.rightNode = null;
                    } else if (bantu.rightNode == null) {
                        parent.rightNode = bantu.leftNode;
                    } else if (bantu.leftNode == null) {
                        parent.rightNode = bantu.rightNode;
                    }
                }
            }
        }
        return true;
    }
}
```

TreeNode()

```
    public boolean isLeaf() {
        if (leftNode == null && rightNode == null) {
            return true;
        } else {
            return false;
        }
    }
}
```

TreeMain()

```
package modul4;

public class TreeMain {

    public static void main(String[] args) {
        Tree data = new Tree();

        data.add(x: 42);
        data.add(x: 21);
        data.add(x: 38);
        data.add(x: 27);
        data.add(x: 71);
        data.add(x: 82);
        data.add(x: 55);
        data.add(x: 63);
        data.add(x: 6);
        data.add(x: 2);
        data.add(x: 40);
        data.add(x: 12);

        System.out.println(x: "data yang ditambahkan = 42, 21, 38, 27, 71, 82, 55, 63, 6, 2, 40 dan 12");

        System.out.print(s: "Preorder = ");
        data.preorderTraversal();

```

```
26              TreeNode tes = data.getParent(key:12);
27              System.out.println("\nParent dari 12 adalah = " + tes.getData());
28              System.out.println(x: "");
29
30              data.hapus(x: 12);
31              System.out.println(x: "Hapus 12");
32              System.out.print(s: "Preorder = ");
33              data.preorderTraversal();
34
35              data.hapus(x: 27);
36              System.out.println(x: "\nHapus 27");
37              System.out.print(s: "Preorder = ");
38              data.preorderTraversal();
39
40              data.hapus(x: 6);
41              System.out.println(x: "\nHapus 6");
42              System.out.print(s: "Preorder = ");
43              data.preorderTraversal();
44
45              data.hapus(x: 55);
46              System.out.println(x: "\nHapus 55");
47              System.out.print(s: "Preorder = ");
48              data.preorderTraversal();
49          }
50      }
```

3. Output

```
run:
data yang ditambahkan = 42, 21, 38, 27, 71, 82, 55, 63, 6, 2, 40 dan 12
Preorder = 42 21 6 2 12 38 27 40 71 55 63 82
Parent dari 12 adalah = 6

Hapus 12
Preorder = 42 21 6 2 38 27 40 71 55 63 82
Hapus 27
Preorder = 42 21 6 2 38 40 71 55 63 82
Hapus 6
Preorder = 42 21 2 38 40 71 55 63 82
Hapus 55
Preorder = 42 21 2 38 40 71 63 82 BUILD SUCCESSFUL (total time: 0 seconds)
```

4. Analisa

Tree

Code	Penjelasan

public TreeNode getParent(int key)
{
TreeNode bantu = root;
TreeNode parent = null;
while (bantu != null) {
if (bantu.data == key) {
return parent;
} else if (bantu.data > key) {
parent = bantu;
bantu = bantu.leftNode;
} else {
parent = bantu;
bantu = bantu.rightNode;
}
}
return bantu;
}
	Method ini berfungsi untuk mencari dan mengembalikan parent dari suatu node dengan parameter "key" yang bertipe integer. Menginisialisasi bantu dengan root untuk mencari root dari node dan parent dengan null karena awalnya tidak memiliki parent. Keduanya bertipe class TreeNode. Terdapat perulangan while yang akan terus berjalan selama nilai bantu tidak null. Jika bantu.data sama dengan key atau node yang dicari ditemukan, maka akan mengembalikan parent. Selain itu, jika bantu.data lebih dari key atau node berada di kiri, maka parent sama dengan bantu dan bantu sama dengan bantu.leftNode. Jika tidak, maka parent sama dengan bantu dan bantu sama dengan bantu.rightNode. Saat nilai dari bantu sama dengan null atau perulangan berakhir, maka method akan mengembalikan nilai dari bantu.

public boolean hapus(int x) {
TreeNode bantu = root;
 	
bantu = getNode(x);
if (bantu == null) {
return false;
} else {
if (bantu.data == root.data) {
if (bantu.isLeaf()) {
root = null;
} else if (bantu.rightNode
== null) {
root = bantu.leftNode;
} else if (bantu.leftNode
== null) {
	Method ini berfungsi untuk menghapus node yang tidak memiliki anak atau memiliki 1 anak. Menginisialisasi bantu dengan root yang bertipe TreeNode. Kemudian, memanggil getNode(x). Jika bantu bernilai null, maka akan mengembalikan "false".
	
	Jika bantu.data sama dengan root.data atau node yang ditemukan adalah root. Maka akan memeriksa lagi, yaitu :
	Jika bantu.isLeaf() atau tidak memiliki anak, maka root bernilai null.
	Jika bantu.rightNode sama dengan null atau tidak memiliki anak di kanan, maka nilai

root = bantu.rightNode;
}
return true;
} else {
TreeNode parent = getParent(x);
 	
if (bantu == null) {
return false;
}
if (x < parent.getData()) {
if (parent.isLeaf()) {
parent.leftNode = null;
} else if (bantu.rightNode == null) {
parent.leftNode = bantu.leftNode;
} else if (bantu.leftNode == null) {
parent.leftNode = bantu.rightNode;
}
} else {
if (bantu.isLeaf()) {
parent.rightNode = null;
} else if (bantu.rightNode == null) {
parent.rightNode = bantu.leftNode;
} else if (bantu.leftNode == null) {
parent.rightNode = bantu.rightNode;
}
}
}
}
return true;
	dari root menyimpan nilai dari bantu.leftNode.
	Jika bantu.leftNode bernilai null atau tidak memiliki anak di kiri, maka root menyimpan nilai dari bantu.rightNode. Diakhiri dengan mengembalikan true saat sudah selesai penghapusan.
	
	Jika yang ingin dihapus bukan root, maka akan menginisialisasi parent dengan nilai dari getParent(x) yang bertipe TreeNode.
	
	Jika bantu bernilai null maka akan mengembalikan “false”.
	
	Jika x kurang dari parent.getData(), maka akan memeriksa lagi, yaitu :
	Jika parent.isLeaf(), maka parent.leftNode bernilai null.
	Jika bantu.rightNode bernilai null, maka parent.leftNode bernilai bantu.leftNode.
	Jika bantu.leftNode bernilai null, maka parent.leftNode bernilai bantu.rightNode.
	
	Jika node berada di kanan parent, maka akan memeriksa lagi, yaitu :
	Jika bantu.isLeaf(), maka parent.rightNode bernilai null.
	Jika bantu.rightNode bernilai null, maka parent.rightNode bernilai bantu.leftNode.
	Jika bantu.leftNode bernilai null, maka parent.rightNode bernilai bantu.rightNode.

}	

TreeNode

Code	Penjelasan
private int data;	Mendeklarasikan atribut data dengan tipe data integer yang bersifat private.
private TreeNode leftNode;	Mendeklarasikan atribut leftNode dengan tipe data TreeNode yang bersifat private.
private TreeNode rightNode;	Mendeklarasikan atribut rightNode dengan tipe data TreeNode yang bersifat private.

public TreeNode() {
this(0);
}
	Berfungsi untuk memanggil constructor yang lain. Lalu, menginisialisasikan data dengan nilai 0.

public TreeNode(int data) {
this.data = data;
leftNode = null;
rightNode = null;
}
	Menginisialisasi data dengan nilai yang ada. Menginisialisasi leftNode, serta rightNode dengan null.

public int getData() {
return data;
}
	Method ini berfungsi untuk mengembalikan nilai dari data.

public TreeNode getLeftNode() {
return leftNode;
}
	Method in berfungsi untuk mengembalikan nilai dari leftNode.

public TreeNode getRightNode() {
return rightNode;
}
	Method in berfungsi untuk mengembalikan nilai dari rightNode.

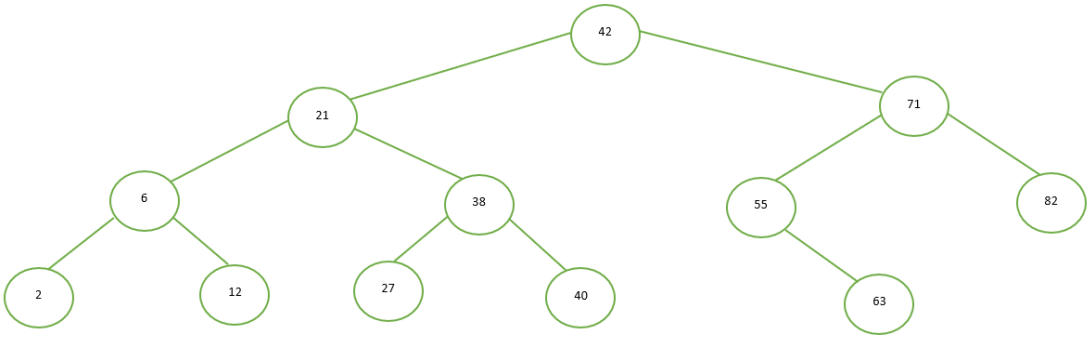
public void setData(int data) {
this.data = data;
}
	Method ini berfungsi untuk mengubah nilai data saat ini dengan data baru yang ingin diinput.

public void setLeftNode(TreeNode lefftNode) {
this.leftNode = leftNode;
}
	Method ini berfungsi untuk mengubah nilai leftNode saat ini dengan leftNode baru yang ingin diinput.

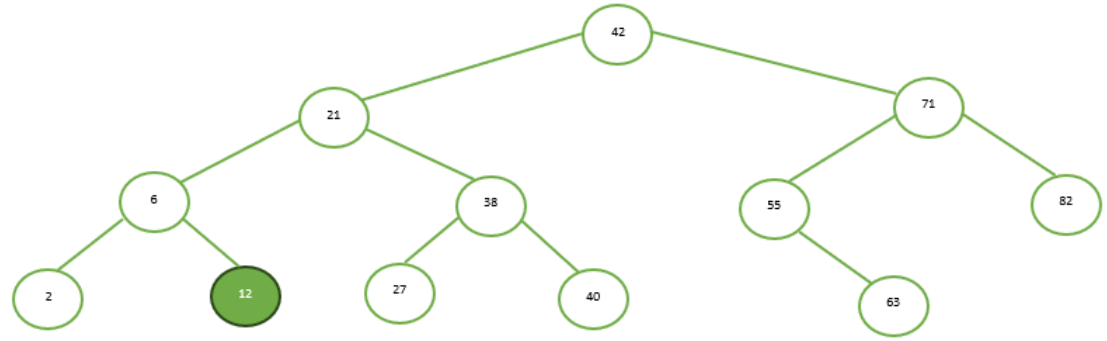
public void setRightNode(TreeNode rightNode) {
this.rightNode = rightNode;
}
	Method ini berfungsi untuk mengubah nilai rightNode saat ini dengan rightNode baru yang ingin diinput.

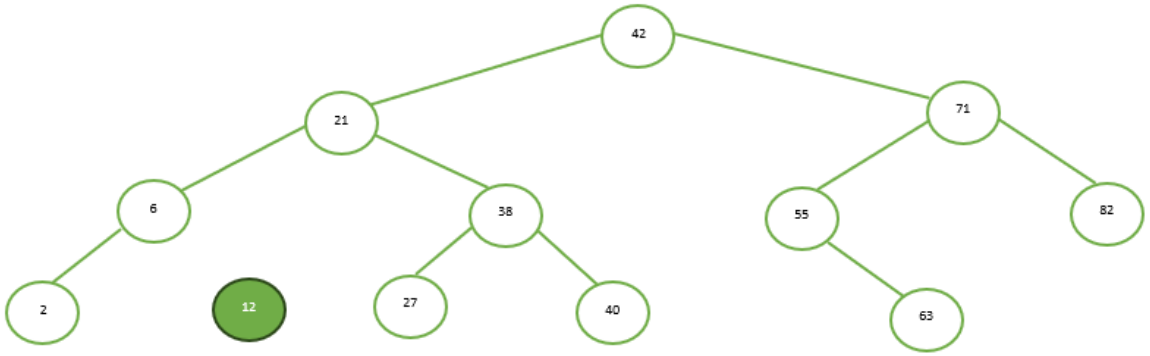
public boolean isLeaf() {
if (leftNode == null && rightNode == null) {
return true;
} else {
return false;
}
}
	Method ini berfungsi untuk mengetahui apakah suatu node memiliki anak atau tidak dengan syarat jika leftNode dan rightNode bernilai null.

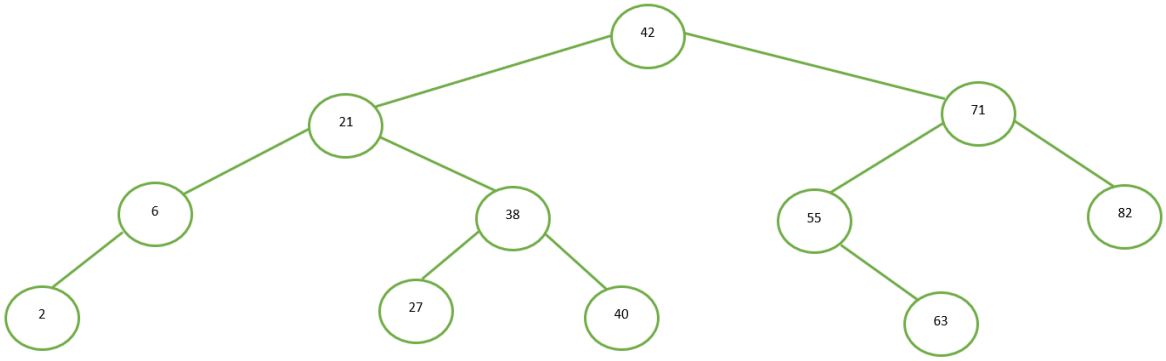
5. Ilustrasi



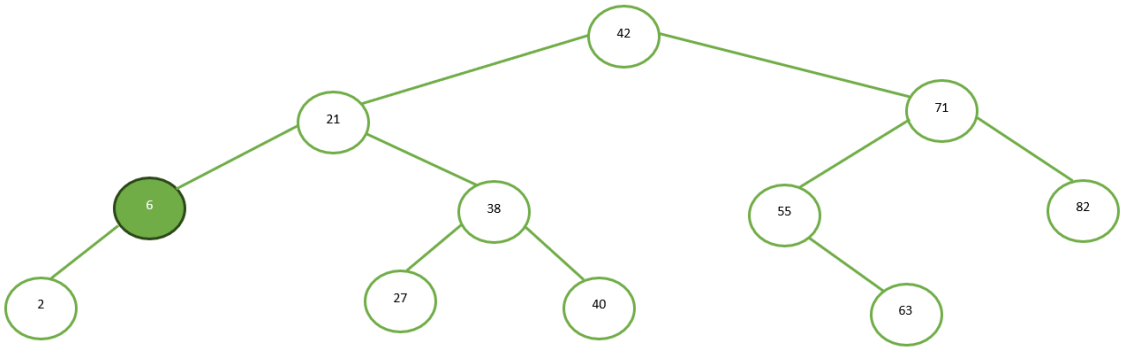
Hapus 12

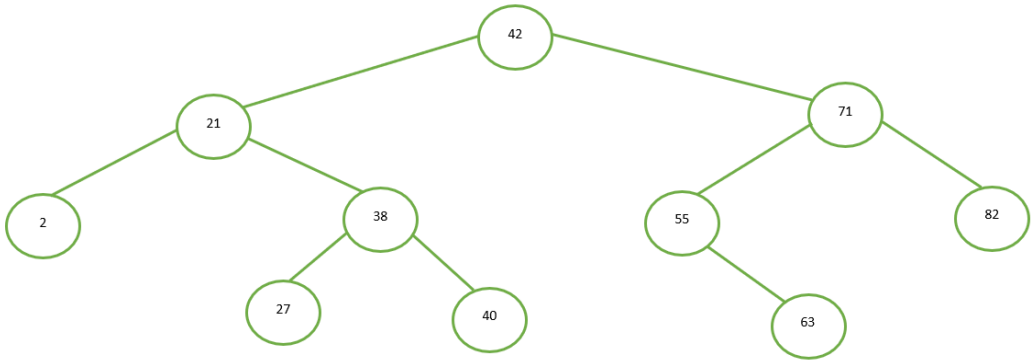

42
21
71
6
38
55
82
2
12
27
40
63


42
21
71
6
38
55
82
2
12
27
40
63

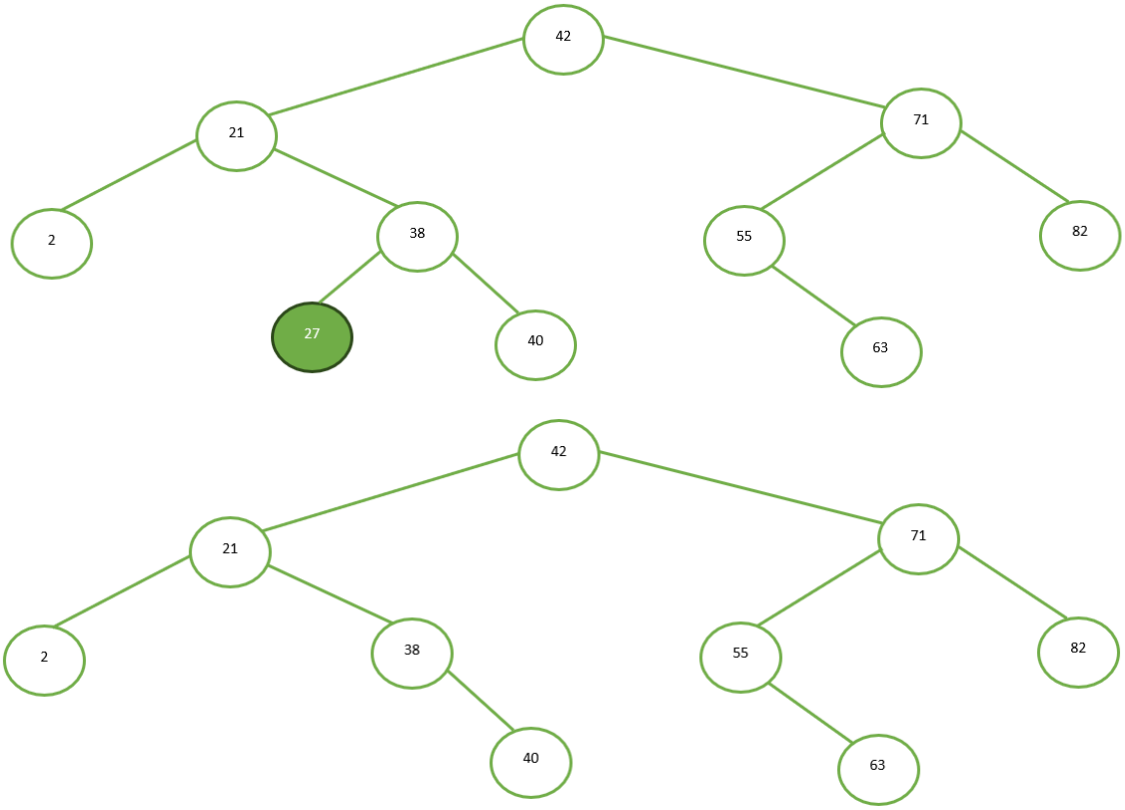

42
21
71
6
38
55
82
2
27
40
63

Hapus 6

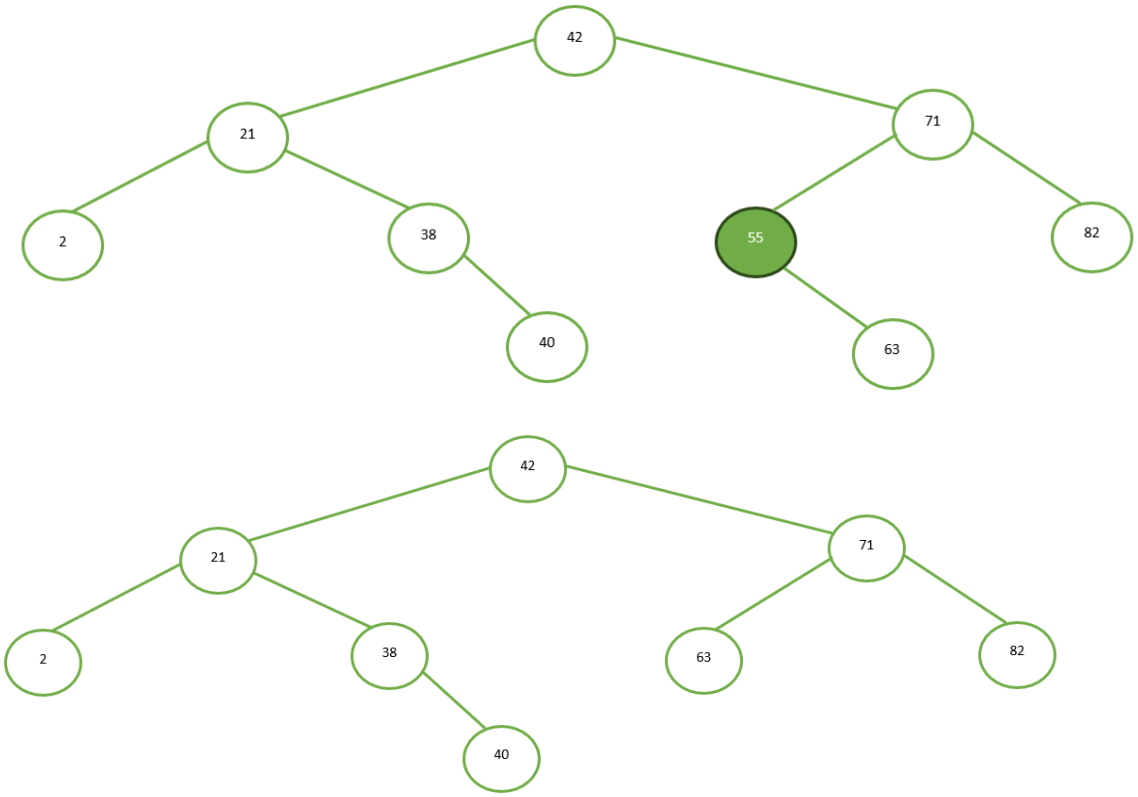




Hapus 27



Hapus 55



6. Referensi

-