Laporan Praktikum

Struktur Data Non Linear DP

Modul 5

Dosen Pengampu

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

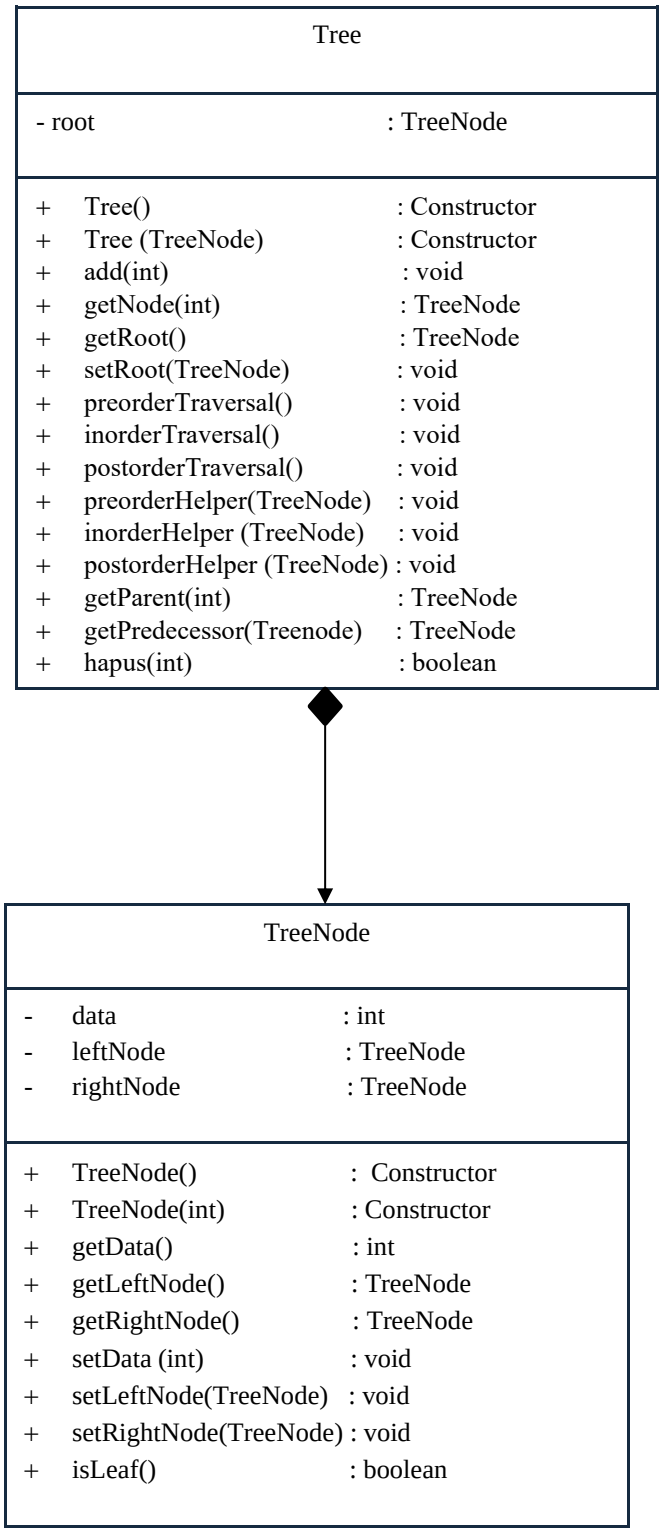
Oleh :

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 modul5;

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;
            }
        }
    }

```

```
56      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
```

```java
public TreeNode getParent(TreeNode key) {
    TreeNode bantu = root;
    TreeNode parent = null;
    while (bantu != null && bantu != key) {
        parent = bantu;

        if (key.data == bantu.data) {
        } else if (key.data > bantu.data) {
            bantu = bantu.rightNode;
        } else {

            bantu = bantu.leftNode;
        }
    }
    return parent;
}

public TreeNode getPredecessor(TreeNode node) {
    TreeNode bantu = node.leftNode;

    while (bantu.rightNode != null) {
        bantu = bantu.rightNode;
    }
    return bantu;
}
```

```java
public boolean hapus(int x) {
    TreeNode bantu = getNode(key: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) {
                root = bantu.rightNode;
            } else {
                TreeNode predecessor = getPredecessor(node: bantu);
                TreeNode parentPredecessor = getParent(key:predecessor);
                bantu.data = predecessor.data;
                if (parentPredecessor != bantu) {
                    if (predecessor.leftNode != null) {
                        parentPredecessor.rightNode = predecessor.leftNode;
                    } else {
                        parentPredecessor.rightNode = null;
                    }
                } else {
                    bantu.leftNode = predecessor.leftNode;
                }
            }
            return true;
```

```
164                 } else {
165                     TreeNode parent = getParent(key:bantu);
166                     if (x < parent.data) {
167                         if (bantu.isLeaf()) {
168                             parent.leftNode = null;
169                         } else if (bantu.rightNode == null) {
170                             parent.leftNode = bantu.leftNode;
171                         } else if (bantu.leftNode == null) {
172                             parent.leftNode = bantu.rightNode;
173                         } else {
174                             TreeNode predecessor = getPredecessor(node: bantu);
175                             TreeNode parentPredecessor = getParent(key:predecessor);
176                             bantu.data = predecessor.data;
177                             if (parentPredecessor != bantu) {
178                                 if (parentPredecessor != null) {
179                                     parentPredecessor.rightNode = predecessor.leftNode;
180                                 } else {
                                        parentPredecessor.rightNode = null;
182                                 }
183                             } else {
184                                 bantu.leftNode = predecessor.leftNode;
185                             }
186                         }
187                     } else {
188                         if (bantu.isLeaf()) {
189                             parent.rightNode = null;
190                         } else if (bantu.rightNode == null) {
191                             parent.rightNode = bantu.leftNode;
192                         } else if (bantu.leftNode == null) {
193                             parent.rightNode = bantu.rightNode;
```

```
194                         } else {
195                             TreeNode predecessor = getPredecessor(node: bantu);
196                             TreeNode parentPredecessor = getParent(key:predecessor);
197                             bantu.data = predecessor.data;
198                             if (parentPredecessor != bantu) {
199                                 if (predecessor.leftNode != null) {
200                                     parentPredecessor.rightNode = predecessor.leftNode;
201                                 } else {
202                                     parentPredecessor.rightNode = null;
203                                 }
204                             } else {
205                                 bantu.leftNode = predecessor.leftNode;
206                             }
207                         }
208                     }
209                     return true;
210                 }
211 
212             }
213         }
214     }
215 
```

TreeNode

```java
package modul5;

public class TreeNode {

    int data;
    TreeNode leftNode;
    TreeNode rightNode;

    public TreeNode() {
        this(data: 0);
    }

    public TreeNode(int data) {
        this.data = data;
        leftNode = null;
        rightNode = null;
    }

    public int getData() {
        return data;
    }

    public TreeNode getLeftNode() {
        return leftNode;
    }

    public TreeNode getRightNode() {
        return rightNode;
    }

    public void setData(int data) {
        this.data = data;
    }

    public void setLeftNode(TreeNode leftNode) {
        this.leftNode = leftNode;
    }

    public void setRightNode(TreeNode rightNode) {
        this.rightNode = rightNode;
    }

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

TreeMain

```
package modul5;

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");

        int x = 21;
        TreeNode bantu = data.getNode( key: x);
        TreeNode lala = data.getPredecessor( node: bantu);
        System.out.println("Predecessor " + x + " : " + lala.getData());

        int[] angka = {12, 55, 21, 42};
        for (int i = 0; i < angka.length; i++) {
            int angkaHapus = angka[i];
            data.hapus( x: angkaHapus);
            System.out.print( x: "\nPreorder = ");
            data.preorderTraversal();
        }
    }
}
```

3. Output

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

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

4. Analisa

Tree

Code	Penjelasan

public TreeNode
getPredecessor(TreeNode node) {
TreeNode bantu = node.leftNode;
 	
while (bantu.rightNode != null) {
bantu = bantu.rightNode;
}
return bantu;
}
	Method ini berfungsi untuk mendapatkan Predecessor. Menginisialisasi bantu dengan node yang berada di kiri. Terdapat perulangan while selama nilai dari bantu.rightNode tidak sama dengan null, maka akan mengeset nilai dari bantu dengan bantu.rightNode. Kemudian akan mengembalikan nilai dari bantu.

public boolean hapus(int x) {
TreeNode 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) {
root = bantu.rightNode;
} else {
TreeNode predecessor = getPredecessor(bantu);
TreeNode parentPredecessor = getParent(predecessor);
bantu.data = predecessor.data;
if (parentPredecessor != bantu) {
if (predecessor.leftNode != null) {
 	
parentPredecessor.rightNode = predecessor.leftNode;
} else {
 	
parentPredecessor.rightNode = null;
}
} else {
bantu.leftNode = predecessor.leftNode;
}
	Method ini berfungsi untuk menghapus node yang memiliki 0 anak, 1 anak, ataupun 2 anak.
	
	Menginisialisasi bantu dengan nilai dari node. 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 bernilai null atau tidak memiliki anak di kanan, maka nilai dari root akan di set dengan nilai dari bantu.leftNode.
	Jika bantu.leftNode bernilai null atau tidak memiliki anak di kiri, maka nilai dari root akan di set dengan nilai dari bantu.rightNode.
	
	Mencari nilai dari predecessor dengan getPredecessor(bantu). Mencari nilai dari parent predecessor dengan getParent(predecessor) yang akan disimpan ke parentPredecessor. Mengeset nilai dari bantu sama dengan predecessor.
	Jika parentPredecessor tidak sama dengan bantu, maka akan memeriksa lagi, yaitu
	Jika predecessor.leftNode tidak bernilai null atau memiliki anak di kiri, maka parent dari predecessor di kanan akan diset dengan predecessor yang di kiri. Jika tidak, parent dari predecessor di kanan akan diset menjadi null.
	Jika nilai dari parent predecessor sama dengan bantu, maka akan mengeset nilai dari bantu di kiri sama dengan predecessor di kiri. Saat pemeriksaan selesai, akan mengembalikan nilai true.

}
return true;
 	
} else {
TreeNode parent = getParent(bantu);
if (x < parent.data) {
if (bantu.isLeaf()) {
parent.leftNode = null;
} else if (bantu.rightNode == null) {
parent.leftNode = bantu.leftNode;
} else if (bantu.leftNode == null) {
parent.leftNode = bantu.rightNode;
} else {
TreeNode predecessor = getPredecessor(bantu);
TreeNode parentPredecessor = getParent(predecessor);
bantu.data = predecessor.data;
if (parentPredecessor != bantu) {
if (parentPredecessor != null) {
 	
parentPredecessor.rightNode = predecessor.leftNode;
} else {
 	
parentPredecessor.rightNode = null;
}
} else {
bantu.leftNode = predecessor.leftNode;
}
}
} else {
if (bantu.isLeaf()) {
parent.rightNode = null;
} else if (bantu.rightNode == null) {
parent.rightNode = bantu.leftNode;
} else if (bantu.leftNode == null) {
	Jika yang ingin dihapus bukan root, maka akan menginisialisasi parent dengan nilai dari getParent(bantu) yang bertipe TreeNode.
	
	Jika nilai dari x kurang dari parent maka akan memeriksa lagi, yaitu
	Jika bantu.isLeaf() atau tidak memiliki anak, maka nilai dari parent di kiri akan diset menjadi null.
	Jika bantu.rightNode bernilai null atau tidak memiliki anak di kanan, maka nilai dari root akan di set dengan nilai dari bantu.leftNode.
	Jika bantu.leftNode bernilai null atau tidak memiliki anak di kiri, maka nilai dari root akan di set dengan nilai dari bantu.rightNode.
	
	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 akan diset dengan bantu.leftNode.
	Jika bantu.leftNode bernilai null, maka parent.leftNode akan diset dengan bantu.rightNode.
	Jika tidak, maka akan mencari nilai dari predecessor dengan getPredecessor(bantu) dan disimpan ke predecessor. Kemudian, mencari nilai dari parent predecessor dengan getParent(predecessor) dan disimpan ke parentPredecessor. Mengeset nilai dari bantu sama dengan predecessor.
	Jika parentPredecessor tidak sama dengan bantu, maka akan memeriksa lagi, yaitu
	Jika predecessor.leftNode tidak bernilai null atau memiliki anak di kiri, maka parent dari predecessor di kanan akan diset dengan predecessor yang di kiri. Jika tidak, parent dari predecessor di kanan akan diset menjadi null.
	Jika nilai dari parent predecessor sama dengan bantu, maka akan mengeset nilai dari bantu di kiri sama dengan predecessor di kiri.
	
	Jika nilai dari x tidak kurang dari parent, maka akan memeriksa lagi, yaitu
	Jika bantu tidak memiliki anak , maka parent di kiri akan diset dengan null.
	Jika bantu di kanan bernilai null, maka parent di kanan akan diset dengan bantu di kiri.

parent.rightNode = bantu.rightNode;
} else {
TreeNode predecessor = getPredecessor(bantu);
TreeNode parentPredecessor = getParent(predecessor);
bantu.data = predecessor.data;
if (parentPredecessor != bantu) {
if (predecessor.leftNode != null) {
parentPredecessor.rightNode = predecessor.leftNode;
} else {
parentPredecessor.rightNode = null;
}
} else {
bantu.leftNode = predecessor.leftNode;
}
}
}
return true;
}
}
}
	Jika bantu di kiri bernilai null, maka parent di kanan akan diset dengan bantu di kanan.
	
	Jika tidak, maka akan mencari nilai dari predecessor dengan getPredecessor(bantu) dan disimpan ke predecessor. Kemudian, mencari nilai dari parent predecessor dengan getParent(predecessor) dan disimpan ke parentPredecessor. Mengeset nilai dari bantu dengan predecessor.
	Jika parentPredecessor tidak sama dengan bantu, maka akan memeriksa lagi, yaitu
	Jika predecessor di kiri tidak bernilai null, maka parent dari predecessor di kanan akan diset dengan predecessor yang di kiri. Jika tidak, parent predecessor di kanan akan diset dengan null. Jika nilai dari parent predecessor sama dengan bantu, maka nilai dari bantu akan diset dengan predecessor di kiri. Saat pemeriksaan sudah selesai, maka akan mengembalikan nilai true.

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.

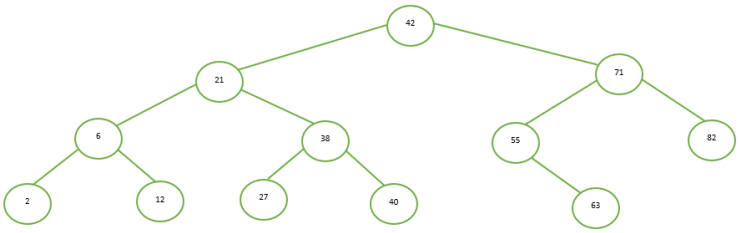
public void setData(int data) {
this.data = data;
}
	Method ini berfungsi untuk mengubah nilai data saat ini dengan data baru yang ingin diinput.

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

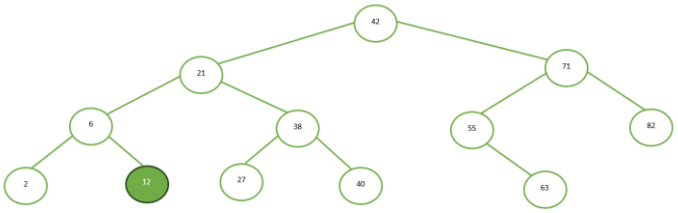
public void setRightNode(TreeNode rightNode) {
this.rightNode = rightNode;
}
	Method ini berfungsi untuk mengubah nilai rightNode saat ini dengan rightNode baru yang ingin diinput.

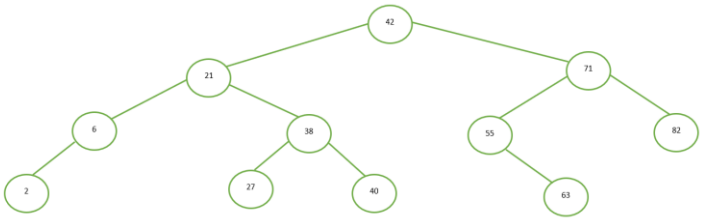
public boolean isLeaf() {
if (leftNode == null && rightNode == null) {
return true;
} else {
return false;
}
}
	Method ini berfungsi untuk mengetahui apakah suatu node memiliki anak atau tidak dengan syarat jika leftNode dan rightNode bernilai null.

5. Ilustrasi

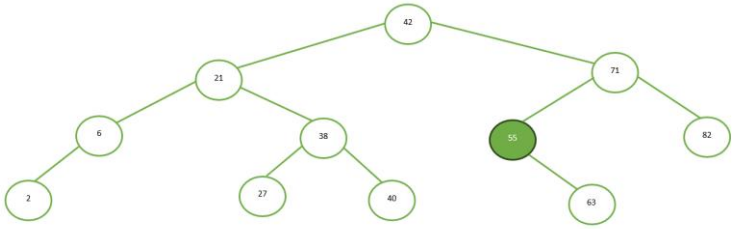


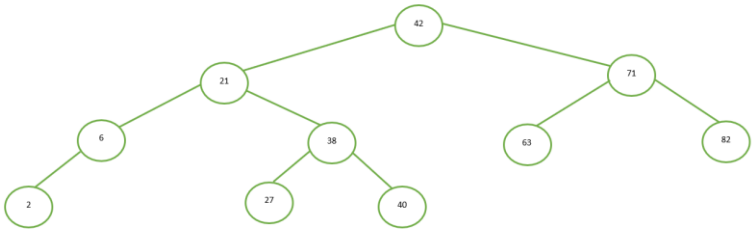
Hapus 12



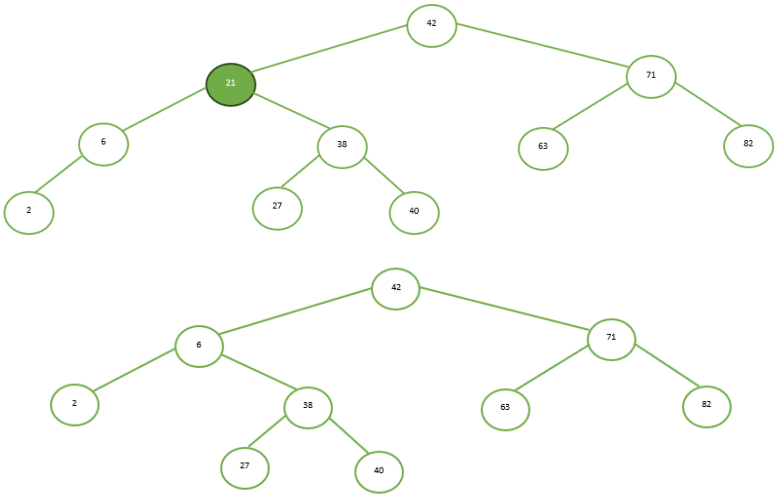


Hapus 55

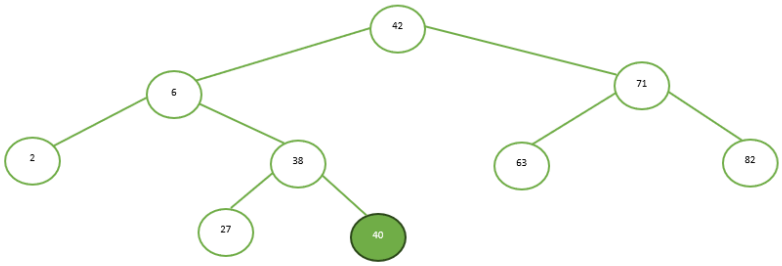


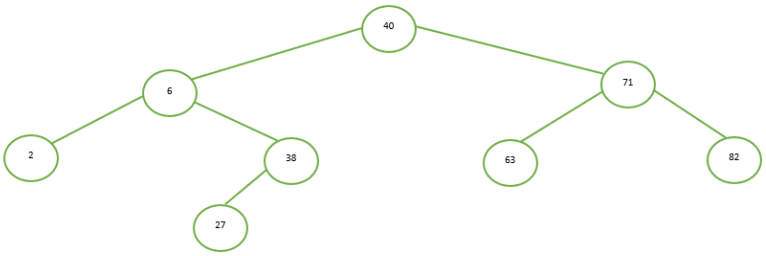


Hapus 21



Hapus 42





6. Referensi

-