

LAPORAN PRAKTIKUM MINGGU KE-13
“TREE”



Disusun oleh:
Daffa Aqila Rahmatullah
2041720098

D4 TEKNIK INFORMATIKA
TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG
2021

LAPORAN

A. KODE PROGRAM

Kode program dipaste di sini menggunakan font yang berbeda, misalkan courier new.

Kode program jangan discreen shoot agar lebih jelas.

1. PRAKTIKUM 13.2.1 (Percobaan 1)

- **Class Node:**

```
package Praktikum1;

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

    public Node() {
    }

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

- **Class BinaryTree:**

```
package Praktikum1;

public class BinaryTree {
    Node root;

    public BinaryTree() {
        root = null;
    }

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

```

    }

void add(int data){
    if(isEmpty()) //tree is empty
    {
        root = new Node(data);
    }
    else
    {
        Node current = root;
        while(true)
        {
            if(data<current.data)
            {
                if (current.left != null)
                {
                    current = current.left;
                }
                else
                {
                    current.left = new
Node(data);

                    break;
                }
            }
            else if(data>current.data)
            {
                if (current.right != null)
                {
                    current = current.right;
                }
                else
                {

```

```

                                current.right = new
Node(data);

                                break;
                                }
                                }
                                else
                                { //data is already exist
                                    break;
                                }
                                }
                                }
}
}

```

```

boolean find(int data){
    boolean hasil = false;
    Node 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;
}

```

```

void traversePreOrder(Node node){
    if(node != null)
    {
        System.out.print(" " + node.data);
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

void traversePostOrder(Node node){
    if(node != null)
    {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.print(" " + node.data);
    }
}

void traverseInOrder(Node node){
    if(node != null)
    {
        traverseInOrder(node.left);
        System.out.print(" " + node.data);
        traverseInOrder(node.right);
    }
}

```

```

Node getSuccessor(Node del){
    Node successor = del.right;
    Node 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()) //tree is empty
    {
        System.out.print("Tree is empty!");
        return;
    }
    // find node(current) that will be deleted
    Node parent = root;
    Node current = root;
    boolean isLeftChild = false;
    while(current != null)
    {
        if (current.data == data)
        {
            break;
        }
        else if(data<current.data)
        {
            parent = current;
            current = current.left;
            isLeftChild = true;
        }
        else if(data>current.data)
        {
            parent = current;

```

```

        current = current.right;
        isLeftChild = false;
    }
    if(current == null)
    {
        System.out.println("Couldn't find
data!");
        return;
    } else
    {
        // if there is no child simply
delete it
        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.left;

        }
        else
        {
            parent.right =
current.left;

```



```

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("");
    }
}

```

2. PRAKTIKUM 13.3.1 (Percobaan 2)

- **Class BinaryTreeArray:**

```

package Praktikum2;

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);
            }
        }
    }
}

```

- **Classs BinaryTreeArrayMain:**

```

package Praktikum2;

public class BinaryTreeArrayMain {
    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(0);
    }

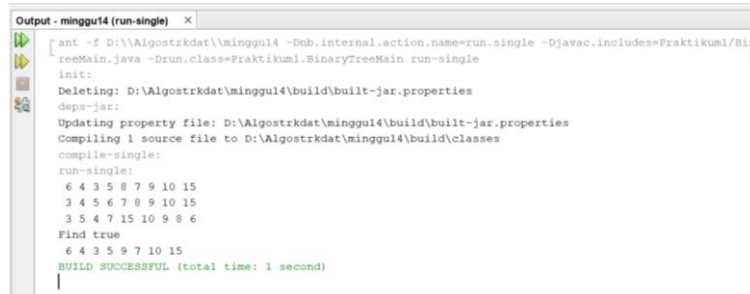
}

```

B. OUTPUT PROGRAM

Untuk hasil program silakan dilakukan screen shoot dengan tetap memperhatikan ukuran gambar agar dapat terlihat dengan jelas.

1. Praktikum 1



```
Output - minggul4 (run-single) X
[ant -f D:\Algostrkdat\minggul4 -Dmb.internal.action.name=run.single -Djavac.includes=Praktikum1\BinaryTreeMain.java -Drun.class=Praktikum1.BinaryTreeMain run-single
init:
Deleting: D:\Algostrkdat\minggul4\build\build-jar.properties
deps-jar:
Updating property file: D:\Algostrkdat\minggul4\build\build-jar.properties
Compiling 1 source file to D:\Algostrkdat\minggul4\build\classes
compile-single:
run-single:
6 4 3 5 6 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
BUILD SUCCESSFUL (total time: 1 second)
```

2. Praktikum 2



```
Output - minggul4 (run-single) X
[ant -f D:\Algostrkdat\minggul4 -Dmb.internal.action.name=run.single -Djavac.includes=Praktikum2\BinaryTreeArrayMain.java -Drun.class=Praktikum2.BinaryTreeArrayMain run-single
init:
Deleting: D:\Algostrkdat\minggul4\build\build-jar.properties
deps-jar:
Updating property file: D:\Algostrkdat\minggul4\build\build-jar.properties
Compiling 1 source file to D:\Algostrkdat\minggul4\build\classes
compile-single:
run-single:
3 4 5 6 7 8 9 BUILD SUCCESSFUL (total time: 1 second)
```

C. PENJELASAN

Silakan diberikan penjelasan kode program yang Anda buat, potongan-potongan program yang menurut Anda penting.

13.2.2 Pertanyaan Percobaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?

Jawaban: Karena pada binary search tree semua left-child harus lebih kecil dari pada right child dan parentnya, sedangkan binary tree biasa tidak. Binary search tree adalah binary tree yang seluruh children dari tiap node terurut sehingga dalam pencarian data jauh lebih efektif dari awal sampai akhir.

2. Untuk apakah di class **Node**, kegunaan dari atribut **left** dan **right**?

Jawaban: Untuk menentukan left-child dan right-child. Karena pada tiap node hanya memiliki maksimal 2 child, yaitu left dan right.

3. a. Untuk apakah kegunaan dari atribut **root** di dalam class **BinaryTree**?

Jawaban: Untuk menentukan node dalam tree yang tidak memiliki predesesor.

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

Jawaban: nilai root Ketika objek tree pertama kali dibuat adalah bernilai null (kosong).

4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?

Jawaban: Proses yang akan terjadi adalah node baru tersebut langsung masuk dan menjadi root dalam sebuah tree.

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

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

Jawaban: Dari method tersebut, ketika data kurang dari current.data kemudian current.left tidak bernilai null maka akan dilakukan current = current.left dan apabila current.left bernilai null maka proses yang dilakukan current.left = new Node(data) atau penambahan node baru dan break (berhenti).

13.3.2 Pertanyaan Percobaan

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

Jawaban:

- Preorder : print data dari root -> leftchild -> rightchild
- Inorder : print data dari leftchild -> root -> rightchild
- Postorder : print data dari leftchild -> rightchild -> root

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

Jawaban: Kegunaan dari method populateData() adalah untuk mendeklarasikan data.

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

Jawaban: Kegunaan dari method `traverseInOrder()` adalah untuk mencetak / menampilkan data dengan InOrder.

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

Jawaban: Left child dari node tersebut akan berada di indeks $2i + 1$. Right child dari node tersebut akan berada di indeks $2i + 2$. Maka left child berada pada indeks ke 5 dan right child berada pada indeks ke 6.

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

Jawaban: Kegunaan dari statement `int idxLast = 6` adalah untuk menentukan posisi pada index terakhir.

D. KESIMPULAN

Praktikum 1

Dari percobaan pada kegiatan pratikum 1 bisa dijelaskan bahwa binary tree tersebut memiliki 3 atribut yaitu atribut left, atribut data, dan atribut right. Yang masing – masing digunakan untuk digunakan seperti diagram pohon yang dimana terdapat sebuah pucuk atau inti utama dari binaryTree yaitu root. Dan root sendiri bisa dikatakan sebagai parent dari child binary tree. Dan dari child bisa menjadi parent jika memiliki child lagi dibawah nya.

Ada beberapa method didalam percobaan binary tree diatas, yaitu :

- a. Add() : digunakan untuk menambah data Node dari binary tree.
- b. Find() : digunakan untuk menemukan data Node binary tree.
- c. traversePreOrder() : digunakan untuk Mengunjungi simpul akar (root), Melakukan traversal subpohon kiri (*left subtree*), dan Melakukan traversal sub pohon kanan (*right subtree*).
- d. traverseInOrder() : digunakan untuk Melakukan traversal subpohon kiri (*left subtree*), Mengunjungi simpul akar (root), Melakukan traversal subpohon kanan (*right subtree*)
- e. traversePostOrder() : digunakan untuk Melakukan traversal subpohon kiri (*left subtree*), Melakukan traversal subpohon kanan (*right subtree*), Mengunjungi simpul akar (*root*).
- f. Delete() : digunakan untuk menghapus data.
- g. getSuccessor() : digunakan untuk mendapat nilai paling kecil dari binary tree bagian kanan.

Praktikum 2

Dari percobaan kedua dapat kita ketahui bahwa binary tree tidak hanya dapat diterapkan menggunakan linked list. Binary tree juga dapat diimplementasikan menggunakan array. Namun apabila menggunakan array maka panjang dari arraynya harus ditentukan terlebih dahulu, sehingga lebih bersifat statis. Dalam percobaan 2 terdapat 2 method yang digunakan yaitu :

- a. populateData() : untuk melakukan inisialisasi pengisian data kedalam attribute.
- b. traverseInOrder : Untuk melakukan proses traverse menggunakan metode in order.

TUGAS

Tugas diberikan ketika diberikan intruksi untuk mengerjakan tugas, jangan lupa untuk menuliskan semua soalnya sebelum memberikan jawaban.

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

- **Class Node**

```
package Tugas1;
public class Node {
    int data;
    Node left;
    Node right;

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

- **Class BinaryTree**

```
package Tugas1;
public class BinaryTree {
    Node root;

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

    void add(int data) {
        root = insertRekusif(root, data);
    }

    //Fungsi Rekursif untuk memasukkan new data
    Node insertRekusif(Node root, int data) {

        if (root == null) {
            root = new Node(data);
            return root;
        }

        //Untuk menaruh leaf apakah di kiri atau
        kanan
        if (data < root.data)
```



```

        root.left = insertRekusif(root.left,
data);
    else if (data > root.data)
        root.right = insertRekusif(root.right,
data);

```

```

        return root;
    }
    boolean find(int data){
        boolean hasil = false;
        Node 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;
    }
    void traversePreOrder(Node node){
        if(node != null){
            System.out.print(" "+node.data);
            traversePreOrder(node.left);
            traversePreOrder(node.right);
        }
    }
    void traversePostOrder(Node node){
        if(node != null){
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            System.out.print(" "+node.data);
        }
    }
    void traverseInOrder(Node node){
        if(node != null){
            traverseInOrder(node.left);
            System.out.print(" "+node.data);
            traverseInOrder(node.right);
        }
    }
}

```

```

Node getSuccessor(Node del){
    Node successor = del.right;
    Node successorParent = del;
    while(successor.left != null){
        successorParent = successor;
    }
}

```

```

        successor = successor.left;
    }
    if(successor != del.right){
        successor.left = successor.right;
        successor.right = del.right;
    }
    return successor;
}
void delete(int data){
    if(isEmpty()){
        System.out.println("Data Tree kosong!");
    }
    //find node(current) itu akan segera dihapus
    Node parent = root;
    Node current = root;
    boolean isLeftChild = false;
    while(current != null){
        if(current.data == data){
            break;
        }else if(data < current.data){
            parent = current;
            current = current.left;
            isLeftChild = true;
        }else if(data > current.data){
            parent = current;
            current = current.right;
            isLeftChild = false;
        }
    }
    //deletion
    if(current == null){
        System.out.println("Couldn't find
data!");
        return;
    }else{
        //if there is no child, simply delete it
        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
there is 1 child(right)
            if(current == root){
                root = current.right;

```

```

        }else{
            if(isLeftChild){
                parent.left = current.right;
            }else{
                parent.right =
current.right;
            }
        }
        }else if(current.right == null){//if
there is 1 child (left)
            if(current == root){
                root = current.left;
            }else{
                if(isLeftChild){
                    parent.left = current.left;
                    parent.right = current.left;
                }
            }
        }else{//if there is 2 childs
            Node successor =
getSuccessor(current);
            if(current == root){
                root = root;
            }else{
                if(isLeftChild){
                    parent.left = successor;
                }else{
                    parent.right = successor;
                }
                successor.left = current.left;
            }
        }
    }
}

```

- **Class BinaryTreeMain**

```

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("");
    }
}

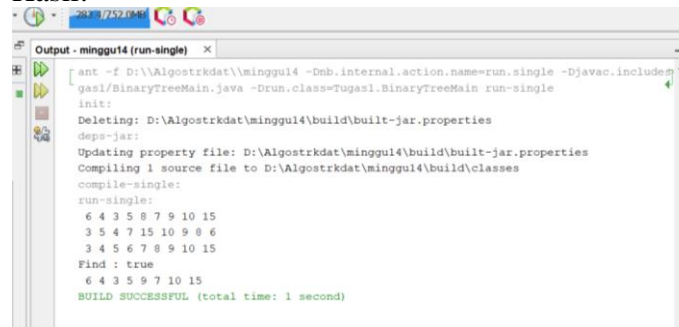
```

```

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

```

Hasil:



Penjelasan:

Pada penambahan node dengan cara rekursif dengan cara yaitu memastikan apakah root sama dengan null jika iya mereturnkan root itu sendiri jika data yang ditambah kurang dari root data maka akan ditaruh dikiri jika lebih dari root data maka akan di taruh di sebelah kanan dan selanjutnya mereturn root.

Kesimpulan:

Pada penambahan node dengan cara rekursif akan jauh lebih efisien dari pada yang ada di praktikum.

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

- Class Node

```

package Tugas1;
public class Node {
    int data;
    Node left;
    Node right;

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

```

- Class BinaryTree

```

package Tugas1;
public class BinaryTree {
    Node root;

    BinaryTree() {
        root = null;
    }
}

```

```

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

void add(int data) {
    root = insertRekusif(root, data);
}

//Fungsi Rekursif untuk memasukkan new data
Node insertRekusif(Node root, int data) {

    if (root == null) {
        root = new Node(data);
        return root;
    }

    //Untuk menaruh leaf apakah di kiri atau
kanan
    if (data < root.data)
        root.left = insertRekusif(root.left,
data);
    else if (data > root.data)
        root.right = insertRekusif(root.right,
data);

    return root;
}

int getMin(Node node){
    if (node == null) {
        System.out.println("Tree Empty");
        return 0;
    }
    while (node.left != null) {
        node = node.left;
    }
    return node.data;
}

int getMax(Node node){
    if (node == null) {
        System.out.println("Tree Empty");
        return 0;
    }
    while (node.right != null) {
        node = node.right;
    }
    return node.data;
}

```

```

void printMinMax(Node node){
    System.out.println(" ");
    int Min = getMin(node);
    System.out.println("Nilai Min : " + Min);
    int Max = getMax(node);
    System.out.println("Nilai Max : " + Max);
}

```

```

boolean find(int data){
    boolean hasil = false;
    Node 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;
}

```

```

void traversePreOrder(Node node){
    if(node != null){
        System.out.print(" "+node.data);
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

```

```

void traversePostOrder(Node node){
    if(node != null){
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.print(" "+node.data);
    }
}

```

```

void traverseInOrder(Node node){
    if(node != null){
        traverseInOrder(node.left);
        System.out.print(" "+node.data);
        traverseInOrder(node.right);
    }
}

```

```

Node getSuccessor(Node del){
    Node successor = del.right;
    Node successorParent = del;
    while(successor.left != null){
        successorParent = successor;
    }
}

```

```

        successor = successor.left;
    }
    if(successor != del.right){
        successor.left = successor.right;
        successor.right = del.right;
    }
    return successor;
}
void delete(int data){
    if(isEmpty()){
        System.out.println("Data Tree kosong!");
    }
    //find node(current) itu akan segera dihapus
    Node parent = root;
    Node current = root;
    boolean isLeftChild = false;
    while(current != null){
        if(current.data == data){
            break;
        }else if(data < current.data){
            parent = current;
            current = current.left;
            isLeftChild = true;
        }else if(data > current.data){
            parent = current;
            current = current.right;
            isLeftChild = false;
        }
    }
    //deletion
    if(current == null){
        System.out.println("Couldn't find
data!");
        return;
    }else{
        //if there is no child, simply delete it
        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
there is 1 child(right)
            if(current == root){
                root = current.right;
            }
        }
    }
}

```

```

        }else{
            if(isLeftChild){
                parent.left = current.right;
            }else{
                parent.right =
current.right;
            }
        }
        }else if(current.right == null){//if
there is 1 child (left)
            if(current == root){
                root = current.left;
            }else{
                if(isLeftChild){
                    parent.left = current.left;
                    parent.right = current.left;
                }
            }
        }else{//if there is 2 childs
            Node successor =
getSuccessor(current);
            if(current == root){
                root = root;
            }else{
                if(isLeftChild){
                    parent.left = successor;
                }else{
                    parent.right = successor;
                }
                successor.left = current.left;
            }
        }
    }
}

```

- **Class BinaryTreeMain**

```

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("");
    }
}

```

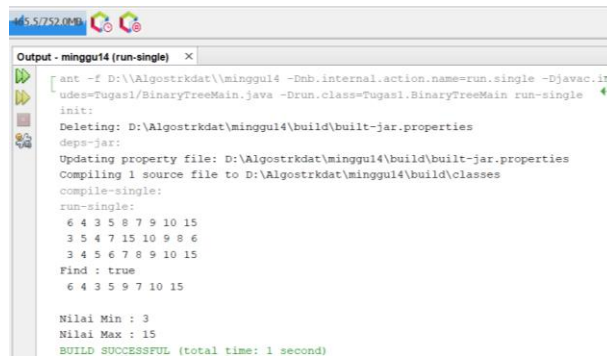


```

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

```

Hasil:



```

Output - minggui4 (run-single) x
ant -f D:\Algostrkdat\minggui4\build\build-jar.properties
  udes=Tugas1/BinaryTreeMain.java -Drun.class=Tugas1.BinaryTreeMain run-single
Init:
Deleting: D:\Algostrkdat\minggui4\build\build-jar.properties
depa-jar:
Updating property file: D:\Algostrkdat\minggui4\build\build-jar.properties
Compiling 1 source file to D:\Algostrkdat\minggui4\build\classes
compile-single:
run-single:
6 4 3 5 8 7 9 10 15
3 5 4 7 15 10 9 8 6
3 4 5 6 7 8 9 10 15
Find : true
6 4 3 5 9 7 10 15

Nilai Min : 3
Nilai Max : 15
BUILD SUCCESSFUL (total time: 1 second)

```

Penjelasan:

Pada mencari nilai terkecil yaitu dengan cara melakukan pengecekan apakah node itu sama dengan null jika iya langsung keluar tree is empty jika tidak maka akan melakukan perulangan while node sebelah kiri tidak sama dengan null maka data yang di inputkan masuk sebelah kiri mereturnkan data.

Pada mencari nilai terbesar yaitu dengan cara melakukan pengecekan apakah node itu sama dengan null jika iya langsung keluar tree is empty jika tidak maka akan melakukan perulangan while node sebelah kanan tidak sama dengan null maka data yang di inputkan masuk sebelah kanan mereturnkan data dan selanjutnya akan ditampilkan di method tampil printMinMax.

Kesimpulan:

Binary Search Tree adalah salah satu bentuk khusus dari binary tree dengan sifat bahwa semua left-child harus lebih kecil daripada right child dan parent-nya.

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

- Class Node


```

package Tugas1;
public class Node {
    int data;
    Node left;
    Node right;

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

```

- **Class BinaryTree**

```
public class BinaryTree {
    Node root;

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

    void add(int data) {
        root = insertRekusif(root, data);
    }

    //Fungsi Rekursif untuk memasukkan new data
    Node insertRekusif(Node root, int data) {

        if (root == null) {
            root = new Node(data);
            return root;
        }

        //Untuk menaruh leaf apakah di kiri atau
        kanan
        if (data < root.data)
            root.left = insertRekusif(root.left,
data);
        else if (data > root.data)
            root.right = insertRekusif(root.right,
data);

        return root;
    }

    int getMin(Node node){
        if (node == null) {
            System.out.println("Tree Empty");
            return 0;
        }
        while (node.left != null) {
            node = node.left;
        }
        return node.data;
    }

    int getMax(Node node){
        if (node == null) {
            System.out.println("Tree Empty");
            return 0;
        }
    }
}
```

```

        while (node.right != null) {
            node = node.right;
        }
        return node.data;
    }

    void printMinMax(Node node){
        System.out.println(" ");
        int Min = getMin(node);
        System.out.println("Nilai Min : " + Min);
        int Max = getMax(node);
        System.out.println("Nilai Max : " + Max);
    }

    boolean find(int data){
        boolean hasil = false;
        Node 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;
    }

    void traversePreOrder(Node node){
        if(node != null){
            System.out.print(" "+node.data);
            traversePreOrder(node.left);
            traversePreOrder(node.right);
        }
    }

    void traversePostOrder(Node node){
        if(node != null){
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            System.out.print(" "+node.data);
        }
    }

    void traverseInOrder(Node node){
        if(node != null){
            traverseInOrder(node.left);
            System.out.print(" "+node.data);
            traverseInOrder(node.right);
        }
    }
}

```

```

Node getSuccessor(Node del){
    Node successor = del.right;
    Node successorParent = del;
    while(successor.left != null){
        successorParent = successor;
        successor = successor.left;
    }
    if(successor != del.right){
        successor.left = successor.right;
        successor.right = del.right;
    }
    return successor;
}

void delete(int data){
    if(isEmpty()){
        System.out.println("Data Tree kosong!");
    }
    //find node(current) itu akan segera dihapus
    Node parent = root;
    Node current = root;
    boolean isLeftChild = false;
    while(current != null){
        if(current.data == data){
            break;
        }else if(data < current.data){
            parent = current;
            current = current.left;
            isLeftChild = true;
        }else if(data > current.data){
            parent = current;
            current = current.right;
            isLeftChild = false;
        }
    }
    //deletion
    if(current == null){
        System.out.println("Couldn't find
data!");
        return;
    }else{
        //if there is no child, simply delete it
        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
there is 1 child(right)
        if(current == root){
            root = current.right;
        }else{
            if(isLeftChild){
                parent.left = current.right;
            }else{
                parent.right =
current.right;
            }
        }
    }else if(current.right == null){//if
there is 1 child (left)
        if(current == root){
            root = current.left;
        }else{
            if(isLeftChild){
                parent.left = current.left;
                parent.right = current.left;
            }
        }
    }else{//if there is 2 childs
        Node successor =
getSuccessor(current);
        if(current == root){
            root = root;
        }else{
            if(isLeftChild){
                parent.left = successor;
            }else{
                parent.right = successor;
            }
            successor.left = current.left;
        }
    }
}
}

void findleaf(Node root){
    if(root == null){
        return;
    }
    if (root.left == null && root.right == null)
{
        System.out.print(root.data + " ");
    }
    findleaf(root.left);
}

```

```

        findleaf(root.right);
    }
}

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

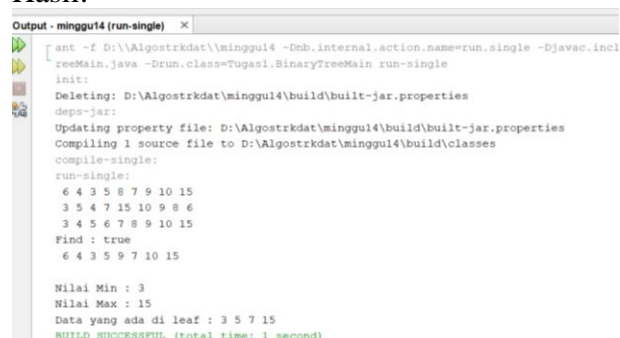
        bt.add(6);
        bt.add(4);
        bt.add(8);
        bt.add(3);
        bt.add(5);
        bt.add(7);
        bt.add(9);
        bt.add(10);
        bt.add(15);

        bt.traversePreOrder(bt.root);
        System.out.println("");
        bt.traversePostOrder(bt.root);
        System.out.println("");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        System.out.println("Find : "+bt.find(5));
        bt.delete(8);
        bt.traversePreOrder(bt.root);
        System.out.println("");
        bt.printMinMax(bt.root);
        System.out.print("Data yang ada di leaf :
");

        bt.findleaf(bt.root);
        System.out.println("");
    }
}

```

Hasil:



```

Output - minggul4 (run-single) X
[ant -f D:\Algostrkdat\minggul4 -Dnb.internal.action.name=run.single -Djavac.incl
reeMain.java -Drun.class=Tugasi.BinaryTreeMain run-single
init:
Deleting: D:\Algostrkdat\minggul4\build\build-jar.properties
deps=jar:
Updating property file: D:\Algostrkdat\minggul4\build\build-jar.properties
Compiling 1 source file to D:\Algostrkdat\minggul4\build\classes
compile-single:
run-single:
6 4 3 5 8 7 9 10 15
3 5 4 7 15 10 9 8 6
3 4 5 6 7 8 9 10 15
Find : true
6 4 3 5 9 7 10 15

Nilai Min : 3
Nilai Max : 15
Data yang ada di leaf : 3 5 7 15
BUILD SUCCESSFUL (total time: 1 second)

```

Penjelasan:

Pada menampilkan data yang ada pada leaf melakukan pengecekan apakah root == null jika tidak maka langsung melakukan pengecekan (root.left == null && root.right == null) jika iya maka akan tampil root.data dan selanjutnya findleaf(root.left); ini berfungsi jika leave ada di sebelah kiri

findleaf(root.right); ini berfungsi jika leaf ada di sebelah kanan
Kesimpulan:

Leaf adalah Node-node dalam tree yang tidak memiliki successor.

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

- **Class Node**

```
package Tugas1;
public class Node {
    int data;
    Node left;
    Node right;

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

- **Class BinaryTree**

```
public class BinaryTree {
    Node root;

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

    void add(int data) {
        root = insertRekusif(root, data);
    }

    //Fungsi Rekursif untuk memasukkan new data
    Node insertRekusif(Node root, int data) {

        if (root == null) {
            root = new Node(data);
            return root;
        }

        //Untuk menaruh leaf apakah di kiri atau
        kanan
        if (data < root.data)
            root.left = insertRekusif(root.left,
data);
        else if (data > root.data)
```

```

        root.right = insertRekusif(root.right,
data);

    return root;
}

int getMin(Node node){
    if (node == null) {
        System.out.println("Tree Empty");
        return 0;
    }
    while (node.left != null) {
        node = node.left;
    }
    return node.data;
}

int getMax(Node node){
    if (node == null) {
        System.out.println("Tree Empty");
        return 0;
    }
    while (node.right != null) {
        node = node.right;
    }
    return node.data;
}

void printMinMax(Node node){
    System.out.println(" ");
    int Min = getMin(node);
    System.out.println("Nilai Min : " + Min);
    int Max = getMax(node);
    System.out.println("Nilai Max : " + Max);
}

boolean find(int data){
    boolean hasil = false;
    Node 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;
}

```



```

    }
    void traversePreOrder(Node node) {
        if(node != null){
            System.out.print(" "+node.data);
            traversePreOrder(node.left);
            traversePreOrder(node.right);
        }
    }
    void traversePostOrder(Node node) {
        if(node != null){
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            System.out.print(" "+node.data);
        }
    }
    void traverseInOrder(Node node) {
        if(node != null){
            traverseInOrder(node.left);
            System.out.print(" "+node.data);
            traverseInOrder(node.right);
        }
    }
}

Node getSuccessor(Node del) {
    Node successor = del.right;
    Node successorParent = del;
    while(successor.left != null){
        successorParent = successor;
        successor = successor.left;
    }
    if(successor != del.right){
        successor.left = successor.right;
        successor.right = del.right;
    }
    return successor;
}
void delete(int data){
    if(isEmpty()){
        System.out.println("Data Tree kosong!");
    }
    //find node(current) itu akan segera dihapus
    Node parent = root;
    Node current = root;
    boolean isLeftChild = false;
    while(current != null){
        if(current.data == data){
            break;
        }else if(data < current.data){
            parent = current;
            current = current.left;
        }
    }
}

```

```

        isLeftChild = true;
    }else if(data > current.data){
        parent = current;
        current = current.right;
        isLeftChild = false;
    }
}
//deletion
if(current == null){
    System.out.println("Couldn't find
data!");
    return;
}else{
    //if there is no child, simply delete it
    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
there is 1 child(right)
        if(current == root){
            root = current.right;
        }else{
            if(isLeftChild){
                parent.left = current.right;
            }else{
                parent.right =
current.right;
            }
        }
        }else if(current.right == null){//if
there is 1 child (left)
        if(current == root){
            root = current.left;
        }else{
            if(isLeftChild){
                parent.left = current.left;
                parent.right = current.left;
            }
        }
        }else{//if there is 2 childs
        Node successor =
getSuccessor(current);
        if(current == root){

```

```

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

}

}

void findleaf(Node root){
    if(root == null){
        return;
    }
    if (root.left == null && root.right == null)
{
        System.out.print(root.data + " ");
    }
    findleaf(root.left);
    findleaf(root.right);
}

int getleaf(Node node){
    if (node==null) {
        return 0;
    }
    if (node.left == null && node.right == null)
{
        return 1;
    }
    return
getleaf(node.left)+getleaf(node.right);
}

}

```

- **Class BinaryTreeMain**

```

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.traversePostOrder(bt.root);
        System.out.println("");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        System.out.println("Find : "+bt.find(5));
        bt.delete(8);
        bt.traversePreOrder(bt.root);
        System.out.println("");
        bt.printMinMax(bt.root);
        System.out.print("Data yang ada di leaf :
");
        bt.findleaf(bt.root);
        System.out.println("\njumlah leaf" +
bt.getleaf(bt.root));
    }
}

```

Hasil:

```

Output - minggu14 (run-single) x
ant -f D:\Algostrkdat\minggu14 -Dmb.internal.action.name=run-single -Djavac.includes=Tugas1\
aryTreeMain.java -Drun.class=Tugas1.BinaryTreeMain run-single
init:
Deleting: D:\Algostrkdat\minggu14\build\build-jar.properties
deps-jar:
Updating property file: D:\Algostrkdat\minggu14\build\build-jar.properties
Compiling 1 source file to D:\Algostrkdat\minggu14\build\classes
compile-single:
run-single:
6 4 3 5 8 7 9 10 15
3 5 4 7 15 10 9 8 6
3 4 5 6 7 8 9 10 15
Find : true
6 4 3 5 9 7 10 15

Nilai Min : 3
Nilai Max : 15
Data yang ada di leaf : 3 5 7 15
jumlah leaf 4

BUILD SUCCESSFUL (total time: 1 second)

```

Penjelasan:

Pada menampilkan data yang ada pada leaf melakukan pengecekan apakah `root == null` jika iya maka akan langsung mereturn 0 langsung melakukan pengecekan `(root.left == null && root.right == null)` jika iya maka akan mereturn 1 dan selanjutnya `return getleaf(node.left)+getleaf(node.right);` ini berfungsi untuk menambahkan jumlah leaf yang disebelah kiri dan kanan

Kesimpulan:

Leaf adalah Node-node dalam tree yang tidak memiliki successor.

5. Modifikasi class `BinaryTreeArray`, dan tambahkan :

- method `add(int data)` untuk memasukan data ke dalam tree
 - Class `BinaryTreeArray`

```

package Tugas5;

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 addData(int value, int temp){
    data[temp] = value;
}

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

```

➤ **Class BinaryTreeArrayMain**

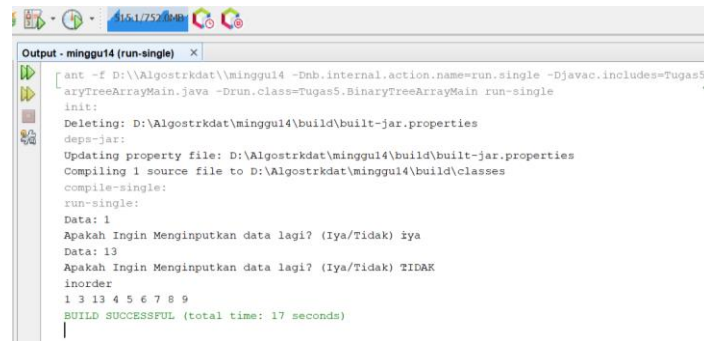
```

import java.util.Scanner;
public class BinaryTreeArrayMain {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Scanner input = new Scanner(System.in);
        BinaryTreeArray bta = new
BinaryTreeArray();
        int[] data = {6,4,8,3,5,7,9,0,0,0};
        int idxLast = 6;
        while (idxLast < data.length-1){
            idxLast +=1;
            System.out.print("Data: ");
            int dat = sc.nextInt();
            bta.populateData(data, idxLast);
            bta.addData(dat, idxLast);
            System.out.print("Apakah Ingin
Menginputkan data lagi? (Iya/Tidak) : ");
            String pilihan = input.nextLine();
            if
(pilihan.equalsIgnoreCase("tidak")) {
                break;
            }
        }
        System.out.println("inorder");
        bta.traverseInOrder(0);
    }
}

```

}

Hasil:



```
Output - minggui4 (run-single) x
[ant -f D:\Algostrkdat\minggui4\build\build-jar.properties -Dnb.internal.action.name=run.single -Djavac.includes=Tugas5\
aryTreeArrayMain.java -Druntime.class=Tugas5.BinaryTreeArrayMain run-single
init:
Deleting: D:\Algostrkdat\minggui4\build\build-jar.properties
deps-jar:
Updating property file: D:\Algostrkdat\minggui4\build\build-jar.properties
Compiling 1 source file to D:\Algostrkdat\minggui4\build\classes
compile-single:
run-single:
Data: 1
Apakah Ingin Menginputkan data lagi? (Iya/Tidak) iya
Data: 13
Apakah Ingin Menginputkan data lagi? (Iya/Tidak) TIDAK
inorder
1 3 13 4 5 6 7 8 9
BUILD SUCCESSFUL (total time: 17 seconds)
```

Penjelasan:

Pada menambahkan data pada array dengan cara `data[temp] = value` selanjutnya pada main class perulangan while `idxlast` kurang dari `data.length-1` karena array mulai dari 0 maka akan `idxlast +=1` berguna untuk menambahkan data

Kesimpulan:

Bisa menambahkan data jika pada inisialisasi awal masih tercukupi

- method `traversePreOrder()` dan `traversePostOrder()`

➤ Class `BinaryTreeArray`

```
package Tugas5;
```

```
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 addData(int value, int temp){
```

```
        data[temp] = value;
```

```
    }
```

```
    void traverseInOrder(int idxStart){
```

```
        if(idxStart <= idxlast)
```

```
        {
```

```
            traverseInOrder(2 * idxStart + 1);
```

```
            System.out.print(data[idxStart]+"
```

```
");
```

```
            traverseInOrder(2 * idxStart + 2);
```

```
        }
```

```
    }
```

```
    void preOrder(int idxStart) {
```

```

        if(idxStart <= idxlast)
        {
            System.out.print(data[idxStart]+"
");
            preOrder(2 * idxStart + 1);
            preOrder(2 * idxStart + 2);
        }
    }

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

```

➤ **Class BinaryTreeArrayMain**

```

import java.util.Scanner;
public class BinaryTreeArrayMain {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Scanner input = new Scanner(System.in);
        BinaryTreeArray bta = new
BinaryTreeArray();
        int[] data = {6,4,8,3,5,7,9,0,0,0};
        int idxLast = 6;
        while (idxLast < data.length-1){
            idxLast +=1;
            System.out.print("Data: ");
            int dat = sc.nextInt();
            bta.populateData(data, idxLast);
            bta.addData(dat, idxLast);
            System.out.print("Apakah Ingin
Menginputkan data lagi? (Iya/Tidak) : ");
            String pilihan = input.nextLine();
            if
(pilihan.equalsIgnoreCase("tidak")) {
                break;
            }
        }
        System.out.println("inorder");
        bta.traverseInOrder(0);
        System.out.println("\nPreoder");
        bta.preOrder(0);
        System.out.println("\npostorder");
        bta.postOrder(0);
        System.out.println(" ");
    }
}

```

}

}

Hasil:



```
Output - minggu14 (run-single) x
[ant -f D:\Algostrkdat\minggu14 -Dnb.internal.action.name=run.single -Djavac.includes=Tugas5\
aryTreeArrayMain.java -Drun.class=Tugas5.BinaryTreeArrayMain run-single
init:
Deleting: D:\Algostrkdat\minggu14\build\build-jar.properties
deps-jar:
Updating property file: D:\Algostrkdat\minggu14\build\build-jar.properties
Compiling 1 source file to D:\Algostrkdat\minggu14\build\classes
compile-single:
run-single:
Data: 1
Apakah Ingin Menginputkan data lagi? (Iya/Tidak) iYa
Data: 12
Apakah Ingin Menginputkan data lagi? (Iya/Tidak) TiDaK
inorder
1 3 12 4 5 6 7 8 9
Preoder
6 4 3 1 12 5 8 7 9
postorder
1 12 3 5 4 7 9 8 6
BUILD SUCCESSFUL (total time: 20 seconds)
```

Penjelasan:

Pada menambahkan method traverse preorder itu menampilkan data root awal dulu diikuti dengan anak kiri dan selanjutnya anak kanan ini diasumsikan dimulai dari indeks ke 0

Pada menambahkan method traverser postorder dengan anak kiri dan selanjutnya anak kanan ini diasumsikan dan selanjutnya menampilkan root awal ini diasumsikan dimulai dari indeks ke 0

Kesimpulan:

Representasi array harus tau rumusnya.