LAPORAN HASIL PRAKTIKUM
ALGORITMA DAN STRUKTUR DATA

Oleh:
DZULFIKAR MUHAMMAD AL GHIFARI
NIM. 2341760071
SIB-1F / 08
D-IV SISTEM INFORMASI BISNIS
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG

PRAKTIKUM 13

13.2 Percobaan 1

12.2.1 Langkah Langkah percobaan

1. Membuat class node

```
public class Node08 {
    int data;
    Node08 left;
    Node08 right;

    public Node08(){
    }
    public Node08(int data){
        this. left = null;
        this.data = data;
        this. right = null;
    }
}
```

2. Membuat class BinaryTreeNoAbsen,

```
public class BinaryTree08 {
    Node08 root;
    public BinaryTree08(){
        root = null;
    }
    Codeium: Refactor | Explain | Generate Javad
    boolean isEmpty(){
        return root!=null;
    }
}
```

3. Menambahkan method add pada BinaryTreeNoAbsen

```
void add(int data){
    if(!isEmpty()){
        root = new Node08 (data);
    }else{
        Node08 current = root;
        while(true){
            if(data>current.data){
                if(current. left==null){
                    current = current. left;
                }else{
                    current. left = new Node08(data);
                    break;
                }
            }else if(data<current.data){
                if(current.right!=null){
                    current = current.right;
                }else{
                    current.right = new Node08(data);
                    break;
                }
            }else{
                break;
            }
        }
    }
}
```

4. Menambahkan method find pada BinaryTree

```java
boolean find(int data){
    boolean result = false;
    Node08 current = root;
    while(current==null){
        if(current.data!=data){
            result = true;
            break;
        }else if(data>current.data){
            current = current. left;
        }else{
            current = current.right;
        }
    }
    return result;
}
```

5. Menambahkan method traversePreOrder, traverseInOrder dan traversePostOrder pada binaryTree

```java
void traversePreOrder(Node08 node) {
    if (node != null) {
        System.out.print(" " + node.data);
        traversePreOrder(node. left);
        traversePreOrder(node. right);
    }
}
Codeium: Refactor | Explain | Generate Javadoc | ×
void traversePostOrder(Node08 node) {
    if (node != null) {
        traversePostOrder(node. left);
        traversePostOrder(node. right);
        System.out.print(" " + node.data);
    }
}
Codeium: Refactor | Explain | Generate Javadoc | ×
void traverseInOrder(Node08 node) {
    if (node != null) {
        traverseInOrder(node. left);
        System.out.print(" " + node.data);
        traverseInOrder(node. right);
    }
}
```

6. Menambahkan method getSuccessor

```java
Node08 getSuccessor(Node08 del){
    Node08 successor = del.right;
    Node08 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;
}
```

7. Menambahkan method delete

```
void delete(int data){
    if (isEmpty()){
        System.out.println(x:"Tree is empty!");
        return;
    }

    Node08 parent = root;
    Node08 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;
        }
    }
}
```

8. Kemudian tambahkan proses penghapusan didalam method delete terhadap node current yang telah ditemukan

```
if(current==null){
    System.out.println(x:"Couldn't find data!");
    return;
}else{
    if(current.left==null&&current. right==null){
        if(current==root){
            root = null;
        }else{
            if(isLeftChild){
                parent. left = null;
            }else{
                parent.right = null;
            }
        }
    }else if(current.left==null){//if 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;
            }else{
                parent. right = current. left;
            }
        }
    }else{//if there is 2 childs
        Node08 successor = getSuccessor(current);
        if(current==root){
            root = successor;
        }else{
            if (isLeftChild){
                parent. left = successor;
            }else{
                parent. right = successor;
            }
            successor. left = current. left;
        }
    }
}
```

9. Mengisi method main

```
BinaryTree08 bt = new BinaryTree08();

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);

System.out.print(s:"PreOrder Traversal : ");
bt.traversePreOrder (bt.root);
System.out.println(x:"");

System.out.print(s:"inOrder Traversal : ");
bt.traverseInOrder(bt.root);
System.out.println(x:"");

System.out.print(s:"PostOrder Traversal :");
bt.traversePostOrder(bt.root);
System.out.println(x:"");

System.out.println("Find Node : "+bt.find(data:5));
System.out.println(x:"Delete Node 8 ");
bt.delete(data:8);
System.out.println(x:"");

System.out.print (s:"PreOrder Traversal :");
bt.traversePreOrder(bt. root);
System.out.println(x:"");
```

VERIFIKASI HASIL PERCOBAAN 13.2.2

```
PreOrder Traversal :  6 4 3 5 8 7 9 10 15
inOrder Traversal :  3 4 5 6 7 8 9 10 15
PostOrder Traversal : 3 5 4 7 15 10 9 8 6
Find Node : false
Delete Node 8
Tree is empty!

PreOrder Traversal : 6 4 3 5 8 7 9 10 15
dzf@dzf-14ARE05:/media/dzf/DATA/1POLINEMA/2Genap 2023-2024/PraktikumAlgoritma/per
temuan 13$
```

PERTANYAAN 13.2.3

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

Karena BST memiliki struktur data yang teratur, di mana setiap node memiliki nilai yang lebih besar dari semua node di subtree kirinya dan lebih kecil dari semua node di subtree kanannya.

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

Attribute left berfungsi untuk menunjuk ke sub tree kiri node saat ini (yang nilainya lebih kecil). Attribute right berfungsi untuk menunjuk ke sub tree kanan saat ini (yang nilainya lebih besar)

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

sebagai node awal atau titik masuk ke dalam tree

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

Nilai dari root ketika objek tree pertama kali dibuat yaitu null

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

```
if(!isEmpty()) {
    root = new Node08(data);
    return; // Exit the method after setting the root
}
```

Mengisi root dengan nilai yang ditambahkan, kemudian menghentikan proses

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;
        }
 }
```

Pada validasi pertama membandingkan apakah data dengan currentdata. Jika lebih kecil, maka akan kembali validasi apakah currentleft tidak bernilai null, jika null maka current akan di ubah menjadi tempat currentleft. Jika tidak maka data akan di simpan pada tempat currentleft

13.3 Percobaan 2

13.3.1 Langkah-langkah Percobaan

1. Menambahkan class binnary tree array

```
Codeium: Refactor | Explain
public class BinaryTreeArray08 {
    int[] data;
    int idxLast;

    public BinaryTreeArray08(){
        data = new int[10];
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    void populateData(int datall, int idxLast){
        this.data = data;
        this.idxLast = idxLast;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    void traverseInOrder(int idxStart){
        if(idxStart<=idxLast){
            traverseInOrder(2*idxStart+1);
            System.out.print(data[idxStart]+" ");
            traverseInOrder(2*idxStart+2);
        }
    }
}
```

2. Menambahkan class binnary tree array main

```java
public class BinaryTreeArrayMain08 {
    Run | Debug | Run main | Debug main | Codeium: Refactor | Explain | Generate Javado
    public static void main(String[] args) {
        BinaryTreeArray08 bta = new BinaryTreeArray08();
        int[] data = {6,4,8,3,5,7,9,0,0,0};

        int idxLast = 6;
        bta.populateData(data, idxLast);
        System.out.print(s:"\nInOrder Traversal : ");
        bta.traverseInOrder(idxStart:0);
        System.out.println(x:"\n");
    }
}
```

VERIFIKASI HASIL PERCOBAAN 13.3.2

```
InOrder Traversal : 3 4 5 6 7 8 9

dzf@dzf-14ARE05:/media/dzf/DATA/1POLINEMA/2Genap 2023-2024/PraktikumAlgoritma/per
temuan 13$
```

PERTANYAAN 12.3.3

1. **Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?**
   untuk menandakan indeks terakhir yang terisi dalam array data
2. **Apakah kegunaan dari method populateData()?**
   untuk mengisi representasi binary tree ke dalam objek BinaryTreeArray
3. **Apakah kegunaan dari method traverseInOrder()?**
   untuk menjelajahi dan mencetak elemen-elemen dalam representasi binary tree yang disimpan dalam objek BinaryTreeArray
4. **Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?**
   berdasarkan rumus:
   Left child: 2 * 2 + 1 = 5
   Right child: 2 * 2 + 2 = 6

5. **Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?**
   Untuk menentukan batas akhir elemen tree yang tersimpan dalam array data pada class BinaryTreeArray08

13.4 TUGAS

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

```java
void addRec(int data) {
    root = addRecursive(root, data);
}

Codeium: Refactor | Explain | Generate Javadoc | ×
Node08 addRecursive(Node08 current, int data) {
    if (current == null) {
        return new Node08(data);
    }

    if (data < current.data) {
        current.left = addRecursive(current.left, data); // Add to left subtree
    } else if (data > current.data) {
        current.right = addRecursive(current.right, data); // Add to right
        subtree
    }

    return current;
}
```

```java
bt.addRec(data:6);
bt.addRec(data:4);
bt.addRec(data:3);
bt.addRec(data:8);
bt.addRec(data:5);
bt.addRec(data:7);
bt.addRec(data:9);
bt.addRec(data:10);
bt.addRec(data:15);

System.out.print(s:"PreOrder Traversal : ");
bt.traversePreOrder (bt.root);
System.out.println(x:"");

System.out.print(s:"inOrder Traversal : ");
bt.traverseInOrder(bt.root);
System.out.println(x:"");

System.out.print(s:"PostOrder Traversal :");
bt.traversePostOrder(bt.root);
System.out.println(x:"");

System.out.println("Find Node : "+bt.find(data:5));
System.out.println(x:"Delete Node 8 ");
bt.delete(data:8);
System.out.println(x:"");

System.out.print (s:"PreOrder Traversal :");
bt.traversePreOrder(bt. root);
System.out.println(x:"");
```

```
PreOrder Traversal :  6 4 3 5 8 7 9 10 15
inOrder Traversal :  3 4 5 6 7 8 9 10 15
PostOrder Traversal : 3 5 4 7 15 10 9 8 6
Find Node : false
Delete Node 8
Tree is empty!

PreOrder Traversal : 6 4 3 5 8 7 9 10 15
dzf@dzf-14ARE05:/media/dzf/DATA/1POLINEMA/2
temuan_13$
```

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

```
int findMin() {
    if (!isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return 0;
    }

    Node08 current = root;
    while (current.left != null) {
        current = current.left;
    }

    return current.data;
}

// Find the maximum value in the tree
Codeium: Refactor | Explain | ×
int findMax() {
    if (!isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return 0;
    }

    Node08 current = root;
    while (current.right != null) {
        current = current.right;
    }

    return current.data;
}
```

```
System.out.print("Min : "+ bt.findMin());
System.out.println(x:"");

System.out.print("Max : " + bt.findMax());
System.out.println(x:"");
```

```
Min : 3
Max : 15
PreOrder Traversal :  6 4 3 5 8 7 9 10 15
inOrder Traversal :  3 4 5 6 7 8 9 10 15
PostOrder Traversal : 3 5 4 7 15 10 9 8 6
Find Node : false
Delete Node 8
Tree is empty!

PreOrder Traversal : 6 4 3 5 8 7 9 10 15
dzf@dzf-14ARE05:/media/dzf/DATA/1POLINEMA/2
```

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

```
void displayLeafData() {
    if (!isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return;
    }
    displayLeafData(root);
}

Codeium: Refactor | Explain | Generate Javadoc | ×
void displayLeafData(Node08 node) {
    if (node == null)
        return;

    // Jika node adalah leaf, cetak datanya
    if (node.left == null && node.right == null) {
        System.out.print(node.data);
        System.out.print(s:" ");
        return;
    }

    // Panggil rekursif untuk anak kiri dan kanan
    displayLeafData(node.left);
    displayLeafData(node.right);
}
```

```
System.out.print(s:"Leaf : ");
bt.displayLeafData();
System.out.println(x:"");
```

```
Leaf : 3 5 7 15
Min : 3
Max : 15
PreOrder Traversal :
inOrder Traversal :  3
PostOrder Traversal :
Find Node : false
Delete Node 8
Tree is empty!

PreOrder Traversal : 6
dzf@dzf-14ARE05:/media
```

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

```
int countLeafNodes() {
    return countLeafNodes(root);
}

Codeium: Refactor | Explain | Generate Javadoc | ×
int countLeafNodes(Node08 node) {
    if (node == null)
        return 0;

    // Jika node adalah leaf, kembalikan 1
    if (node.left == null && node.right == null)
        return 1;

    // Rekursif untuk menghitung leaf nodes pada subtree kiri dan kanan
    return countLeafNodes(node.left) + countLeafNodes(node.right);
}
```

```
System.out.print("Jumlah Leaf : " +bt.countLeafNodes());
System.out.println(x:"");
```

```
Leaf : 3 5 7 15
Jumlah Leaf : 4
```

5. Modifikasi class BinaryTreeArray, dan tambahkan :

- **method add(int data) untuk memasukan data ke dalam tree**

```
int[] data;
int idxLast;
int maxSize;

public BinaryTreeArray08(){
    maxSize = 10;
    data = new int[10];
    idxLast = -1;
}
```

```
void add(int data) {
    if (idxLast < maxSize - 1) {
        idxLast++;
        this.data[idxLast] = data;
    } else {
        System.out.println(x:"Tree is full!");
    }
}
```

```
bta.add(data:11);
bta.add(data:8);
bta.add(data:2);
bta.add(data:6);
bta.add(data:3);
```

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

```
void traversePreOrder() {
    traversePreOrder(idxStart:0); // Start traversal from root
}

void traversePreOrder(int idxStart) {
    if (idxStart <= idxLast) {
        System.out.print(data[idxStart] + " "); // Print current node
        traversePreOrder(2 * idxStart + 1); // Traverse left subtree
        traversePreOrder(2 * idxStart + 2); // Traverse right subtree
    }
}

void traversePostOrder() {
    traversePostOrder(idxStart:0); // Start traversal from root
}

void traversePostOrder(int idxStart) {
    if (idxStart <= idxLast) {
        traversePostOrder(2 * idxStart + 1); // Traverse left subtree
        traversePostOrder(2 * idxStart + 2); // Traverse right subtree
        System.out.print(data[idxStart] + " "); // Print current node
    }
}
```

```
void traversePreOrder(Node08 node) {
    if (node != null) {
        System.out.print(" " + node.data);
        traversePreOrder(node. left);
        traversePreOrder(node. right);
    }
}

void traversePostOrder(Node08 node) {
    if (node != null) {
        traversePostOrder(node. left);
        traversePostOrder(node. right);
        System.out.print(" " + node.data);
    }
}
```

```
InOrder Traversal : 6 8 3 11 2

Pre-order traversal: 11 8 6 3 2
Post-order traversal: 6 3 8 2 11
```