PRAKTIKUM ALGORITMA DAN STRUKTUR DATA

JOBSHEET PERTEMUAN KE-12

NAMA : ALVINO VALERIAN D.R

KELAS : 1A

NO. ABSEN : 05

NIM : 2341720027

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2024

13. Amati hasil running tersebut.

```
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 : true
delete node 8

preOrder traversal : 6 4 3 5 9 7 10 15
```

Binarytree05

```
package praktikum1;
public class BinaryTree05 {
    Node05 root;
    public BinaryTree05(){
        root =null;
    }
    boolean isEmpty(){
        return root == null;
    }
    void add(int data) {
        if (isEmpty()) {
            root = new Node05(data);
        } else {
            Node05 current = root;
            while (true) {
                if (data < current.data) {
                    if (current.left == null) {
                        current.left = new Node05(data);
                        break;
                    } else {
                        current = current.left;
                    }
                } else if (data > current.data) {
                    if (current.right == null) {
                        current.right = new Node05(data);
                        break;
                    } else {
                        current = current.right;
                    }
                } else {
```

```
                    break;
                }
            }
        }
    }
    boolean find (int data){
        boolean result = false;
        Node05 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;
    }
    void traversePreOrder(Node05 node){
        if (node != null) {
            System.out.print(" "+node.data);
            traversePreOrder(node.left);
            traversePreOrder(node.right);
        }
    }
    void traversePostOrder(Node05 node){
        if (node != null) {
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            System.out.print(" "+node.data);
        }
    }
    void traverseInOrder(Node05 node){
        if (node != null) {
            traverseInOrder(node.left);
```

```
            System.out.print(" "+node.data);
            traverseInOrder(node.right);
        }
    }
    Node05 getSuccessor(Node05 del){
        Node05 successor = del.right;
        Node05 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()) {
            System.out.println("tree is empty!");
            return;
        }
        Node05 parent = root;
        Node05 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 (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;
                    }
                }
            }else{
                Node05 successor = getSuccessor(current);
```

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

Binarytreemain

```
package praktikum1;
public class BinaryTreeMain05 {
    public static void main(String[] args) {
        BinaryTree05 bt = new BinaryTree05();
        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);
        System.out.print("preOrder traversal");
        bt.traversePreOrder(bt.root);
        System.out.println("");
        System.out.print("InOrder traversal");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        System.out.print("postOrder traversal");
        bt.traversePostOrder(bt.root);
        System.out.println("");
```

```
        System.out.println("find node : "+bt.find(5));
        System.out.println("delete node 8 ");
        bt.delete(8);
        System.out.println("");
        System.out.print("preOrder traversal :");
        bt.traversePreOrder(bt.root);
        System.out.println("");
    }
}
```

Node05

```
package praktikum1;
public class Node05 {
    int data;
    Node05 left;
    Node05 right;
    public Node05(){}
    public Node05(int data){
        this.left =null;
        this.data =data;
        this.right=null;
    }
}
```

13.2.2 Pertanyaan Percobaan

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

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

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

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

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

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;
}
```

JAWABAN

1.Dalam binary search tree (BST), setiap node memiliki aturan tertentu: nilai di node sebelah kiri selalu lebih kecil dari nilai di node tersebut, dan nilai di node sebelah kanan selalu lebih besar. Karena aturan ini, BST memungkinkan proses pencarian data dilakukan secara efisien dengan membandingkan nilai yang dicari dengan nilai di node saat ini dan kemudian hanya perlu mencari ke kiri atau ke kanan, bukan keduanya. Ini menghasilkan waktu pencarian yang rata-rata O(log n), yang jauh lebih cepat dibandingkan dengan binary tree biasa yang tidak memiliki aturan khusus dan bisa memerlukan pencarian pada setiap node, dengan waktu pencarian rata-rata O(n).

2. Atribut left dan right dalam class Node digunakan untuk menyimpan referensi ke node anak kiri dan kanan. left digunakan untuk node dengan nilai lebih kecil dan right untuk node dengan nilai lebih besar.

3.

a. Atribut root di dalam class BinaryTree digunakan untuk menyimpan referensi ke node akar (node paling atas) dari pohon. root adalah titik awal untuk semua operasi pohon, seperti penambahan, penghapusan, dan penelusuran data.

b. Ketika objek BinaryTree pertama kali dibuat, nilai dari root adalah null, menandakan bahwa pohon tersebut masih kosong dan belum memiliki node.

4. Ketika tree masih kosong (artinya root adalah null), dan sebuah node baru akan ditambahkan, node baru tersebut akan menjadi root dari tree. Ini dilakukan dengan menginisialisasi root dengan node baru yang dibuat.

5.

if(data < current.data): Memeriksa apakah data yang akan ditambahkan lebih kecil dari data di node saat ini (current). Jika ya, kita perlu menambahkan data di subtree kiri.

if(current.left != null): Memeriksa apakah node anak kiri dari node saat ini (current.left) tidak null (artinya ada node di sebelah kiri).

current = current.left;: Jika ada node di sebelah kiri, berpindah ke node anak kiri tersebut dan lanjutkan pemeriksaan di iterasi berikutnya dari loop.

else: Jika tidak ada node di sebelah kiri (current.left adalah null), buat node baru dengan data yang akan ditambahkan dan letakkan sebagai anak kiri dari node saat ini (current.left = new Node(data)).

break;: Mengakhiri loop setelah menambahkan node baru.

Jalankan class **BinaryTreeArrayMain** dan amati hasilnya!

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


D:\alvino\Semester 2\Prak algorita & struktur data\jobsheet12>
```

BinarryTreeArray05

```
package praktikum2;

public class BinarryTreeArray05 {
    int[] data;
    int idxLast;

    public BinarryTreeArray05(){
        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);
        }
    }
}
```

BinarryTreeArrayMain05

```
package praktikum2;

public class BinaryTreeArrayMain05 {
    public static void main(String[] args) {
        BinarryTreeArray05 bta = new BinarryTreeArray05();
        int[] data = {6,4,8,3,5,7,9,0,0,0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        System.out.print("\nInorder Traversal : ");
        bta.traverseInOrder(0);
        System.out.println("\n");
    }
}
```

13.3.2 Pertanyaan Percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?
2. Apakah kegunaan dari method populateData()?
3. Apakah kegunaan dari method traverseInOrder()?
4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?
5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?

JAWABAN

1. data: Atribut ini adalah array yang digunakan untuk menyimpan elemen-elemen dari binary tree. Elemen-elemen dalam tree disimpan dalam bentuk array di mana posisi elemen mengikuti aturan tertentu (misalnya, untuk node pada indeks i, anak kiri di 2*i+1 dan anak kanan di 2*i+2).

idxLast: Atribut ini menyimpan indeks dari elemen terakhir yang ada dalam binary tree. Ini digunakan untuk membatasi operasi agar hanya bekerja pada elemen-elemen yang valid dalam array.

2. Method populateData(int[] data, int idxLast) digunakan untuk menginisialisasi array data dan nilai idxLast pada objek BinarryTreeArray05. Ini memungkinkan pengguna untuk mengisi tree dengan data yang diinginkan dan menentukan elemen terakhir dalam array.

3. Method traverseInOrder(int idxStart) digunakan untuk melakukan penelusuran in-order pada binary tree yang disimpan dalam array. Penelusuran in-order adalah salah satu metode untuk mengunjungi semua node dalam tree, di mana node dikunjungi dalam urutan:
Subtree kiri, Node saat ini, Subtree kanan

4. Left child dari node pada indeks 2 berada di indeks `2*2 + 1 =5`.
Right child dari node pada indeks 2 berada di indeks 2*2 + 2 = 6.

5. Statement int idxLast = 6 digunakan untuk menunjukkan bahwa elemen terakhir dalam binary tree yang disimpan dalam array berada pada indeks 6. Ini berarti tree memiliki 7 elemen (indeks 0 sampai 6). idxLast membantu metode traverseInOrder dan operasi lain pada tree untuk menentukan batas akhir dari elemen-elemen yang valid dalam array sehingga mencegah akses ke indeks yang tidak valid.

13.4 Tugas Praktikum

Waktu pengerjaan: 90 menit

1. Buat method di dalam class BinaryTree yang akan menambahkan node dengan cara rekursif.
2. Buat method di dalam class BinaryTree untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.
3. Buat method di dalam class BinaryTree untuk menampilkan data yang ada di leaf.
4. Buat method di dalam class BinaryTree untuk menampilkan berapa jumlah leaf yang ada di dalam tree.
5. Modifikasi class BinaryTreeArray, dan tambahkan :

• method add(int data) untuk memasukan data ke dalam tree

• method traversePreOrder() dan traversePostOrder()

1.

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

    Node05 addRecursive(Node05 current, int data) {
        if (current == null) {
            return new Node05(data);
        }

        if (data < current.data) {
            current.left = addRecursive(current.left, data);
        } else if (data > current.data) {
            current.right = addRecursive(current.right, data);
        }
        return current;
    }
```

2.

```
int findMinimal() {
        if (isEmpty()) {
            System.out.println("Tree is empty");
        }
        Node05 current = root;
        while (current.left != null) {
            current = current.left;
        }
        System.out.println(current.data);
        return current.data;

    }
    int findMaximal() {
        if (isEmpty()) {
            System.out.println("Tree is empty");
        }
        Node05 current = root;
        while (current.right != null) {
            current = current.right;
        }
        System.out.println(current.data);
        return current.data;
    }
```

3.

```
void printLeaves(Node05 node) {
        if (node != null) {
            if (node.left == null && node.right == null) {
                System.out.print(node.data + " ");
            }
            printLeaves(node.left);
            printLeaves(node.right);
        }
    }
    void printLeaves() {
        printLeaves(root);
    }
```

4.

```
int countLeaves(Node05 node) {
        if (node == null) {
            return 0;
        }
        if (node.left == null && node.right == null) {
            return 1;
        }
        return countLeaves(node.left) + countLeaves(node.right);
    }
    int countLeaves() {
        int count = countLeaves(root);
        System.out.println(count);
        return count;
    }
```

```
preOrder traversal 5 3 2 4 7 6 8
InOrder traversal 2 3 4 5 6 7 8
postOrder traversal 2 4 3 6 8 7 5
find node : true
delete node 8

preOrder traversal : 5 3 2 4 7 6
maximal: 7

minimal: 2

data yang ada di leaf: 2 4 6
berapa jumlah leaf: 3


D:\alvino\Semester 2\Prak algorita & struktur data\jobsheet12>
```

5.

Method ADD

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

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

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

```
PreOrder traversal: 6 4 3 0 0 5 0 8 7 10
PostOrder traversal: 0 0 3 0 5 4 7 10 8 6
```

Binarytree05

```
package praktikum1;
public class BinaryTree05 {
    Node05 root;
    public BinaryTree05(){
        root =null;
    }
    boolean isEmpty(){
        return root == null;
    }
    void add(int data) {
        if (isEmpty()) {
            root = new Node05(data);
        } else {
            Node05 current = root;
            while (true) {
                if (data < current.data) {
                    if (current.left == null) {
                        current.left = new Node05(data);
                        break;
```

```
                } else {
                    current = current.left;
                }
            } else if (data > current.data) {
                if (current.right == null) {
                    current.right = new Node05(data);
                    break;
                } else {
                    current = current.right;
                }
            } else {
                break;
            }
        }
    }
}
boolean find (int data){
    boolean result = false;
    Node05 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;
}

void traversePreOrder(Node05 node){
    if (node != null) {
        System.out.print(" "+node.data);
        traversePreOrder(node.left);
        traversePreOrder(node.right);
```

```
        }
    }
    void traversePostOrder(Node05 node){
        if (node != null) {
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            System.out.print(" "+node.data);
        }
    }
    void traverseInOrder(Node05 node){
        if (node != null) {
            traverseInOrder(node.left);
            System.out.print(" "+node.data);
            traverseInOrder(node.right);
        }
    }
    Node05 Successor(Node05 del){
        Node05 successor = del.right;
        Node05 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()) {
            System.out.println("tree is empty!");
            return;
        }
        Node05 parent = root;
        Node05 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 (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;
                    }
                }
            }else{
                Node05 successor = Successor(current);
                if (current==root) {
                    root = successor;
                }else{
                    if (isLeftChild) {
                        parent.left = successor;
                    }else{
                        parent.right = successor;
                    }
                    successor.left = current.left;
                }
            }
        }
    }
    void Recursive(int data) {
        root = addRecursive(root, data);
    }

    Node05 addRecursive(Node05 current, int data) {
        if (current == null) {
            return new Node05(data);
        }

        if (data < current.data) {
            current.left = addRecursive(current.left, data);
        } else if (data > current.data) {
```

```
            current.right = addRecursive(current.right, data);
        }
        return current;
    }
    // cari max min
    int findMinimal() {
        if (isEmpty()) {
            System.out.println("Tree is empty");
        }
        Node05 current = root;
        while (current.left != null) {
            current = current.left;
        }
        System.out.println(current.data);
        return current.data;

    }
    int findMaximal() {
        if (isEmpty()) {
            System.out.println("Tree is empty");
        }
        Node05 current = root;
        while (current.right != null) {
            current = current.right;
        }
        System.out.println(current.data);
        return current.data;
    }
    // menampilkan data leaf
    void printLeaves(Node05 node) {
        if (node != null) {
            if (node.left == null && node.right == null) {
                System.out.print(node.data + " ");
            }
            printLeaves(node.left);
            printLeaves(node.right);
        }
```

```
    }

    void printLeaves() {
        printLeaves(root);
    }
    // jumlah data leaf
    int countLeaves(Node05 node) {
        if (node == null) {
            return 0;
        }
        if (node.left == null && node.right == null) {
            return 1;
        }
        return countLeaves(node.left) + countLeaves(node.right);
    }
    int countLeaves() {
        int count = countLeaves(root);
        System.out.println(count);
        return count;
    }

}
```

Main

```
package praktikum1;
public class BinaryTreeMain05 {
    public static void main(String[] args) {
        BinaryTree05 bt = new BinaryTree05();
        bt.Recursive(5);
        bt.Recursive(3);
        bt.Recursive(7);
        bt.Recursive(2);
        bt.Recursive(4);
        bt.Recursive(6);
        bt.Recursive(8);
        System.out.print("preOrder traversal");
        bt.traversePreOrder(bt.root);
        System.out.println("");
```

```
        System.out.print("InOrder traversal");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        System.out.print("postOrder traversal");
        bt.traversePostOrder(bt.root);
        System.out.println("");
        System.out.println("find node : "+bt.find(6));
        System.out.println("delete node 8 ");
        bt.delete(8);
        System.out.println("");
        System.out.print("preOrder traversal :");
        bt.traversePreOrder(bt.root);
        System.out.println("");
        System.out.print("maximal: ");
        bt.findMaximal();
        System.out.println("");
        System.out.print("minimal: ");
        bt.findMinimal();
        System.out.println("");
        System.out.print("data yang ada di leaf: ");
        bt.printLeaves();;
        System.out.println("");
        System.out.print("berapa jumlah leaf: ");
        bt.countLeaves();
        System.out.println("");
    }
}
```

BinarryArray

```
package praktikum2;

public class BinarryTreeArray05 {
    int[] data;
    int idxLast;

    public BinarryTreeArray05(){
        data = new int[10];
    }
```

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

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

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

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

Main

```
package praktikum2;

public class BinaryTreeArrayMain05 {
    public static void main(String[] args) {
        BinarryTreeArray05 bta = new BinarryTreeArray05();
        int[] data = {6,4,8,3,5,7,10,0,0,0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        bta.populateData(data, data.length - 1);
        System.out.print("\nInorder Traversal : ");
        bta.traverseInOrder(0);
        System.out.println("\n");

        System.out.print("InOrder traversal: ");
        bta.traverseInOrder(0);
        System.out.println();

        System.out.print("PreOrder traversal: ");
        bta.traversePreOrder(0);
        System.out.println();

        System.out.print("PostOrder traversal: ");
        bta.traversePostOrder(0);
        System.out.println();

        bta.add(9);
        System.out.print("InOrder traversal after 9: ");
        bta.traverseInOrder(0);
        System.out.println();
    }
}
```