PRAKTIKUM ALGORITMA DAN STRUKTUR DATA

JOBSHEET 13

NAMA : DIMAS ADI BAYU SAMUDRA

KELAS : 1A

NO. ABSEN : 08

NIM : 2341720169

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2024

Percobaan 1

Kode Node08

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

    public Node08(){

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

}
```

Kode BinaryTree08

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

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

public class BinaryTree08 {
    Node08 root;

    public BinaryTree08() {
        root = null;
    }

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

    void add(int data) {
        if (isEmpty()) {
            root = new Node08(data);
        } else {
            Node08 current = root;
            while (true) {
                if (data < current.data) {  // Corrected the condition
                    if (current.left == null) {
                        current.left = new Node08(data);
                        break;
                    } else {
                        current = current.left;
```

```
                    }
                } else if (data > current.data) {  // Corrected the
condition
                    if (current.right == null) {
                        current.right = new Node08(data);
                        break;
                    } else {
                        current = current.right;
                    }
                } else {
                    break;  // Data already exists in the tree
                }
            }
        }
    }

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

    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);
        }
    }

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

```
    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;
    }

    void delete(int data) {
        if (isEmpty()) {
            System.out.println("Tree is empty!");
            return;
        }
        Node08 parent = null;
        Node08 current = root;
        boolean isLeftChild = false;

        while (current != null && current.data != data) {
            parent = current;
            if (data < current.data) {
                isLeftChild = true;
                current = current.left;
            } else {
                isLeftChild = false;
                current = current.right;
            }
        }

        if (current == null) {
            System.out.println("Couldn't find data!");
            return;
        }

        // Node to be deleted has no children (leaf node)
        if (current.left == null && current.right == null) {
            if (current == root) {
                root = null;
            } else if (isLeftChild) {
                parent.left = null;
            } else {
                parent.right = null;
            }
        }
        // Node to be deleted has only a right child
        else if (current.left == null) {
            if (current == root) {
```

```
                root = current.right;
            } else if (isLeftChild) {
                parent.left = current.right;
            } else {
                parent.right = current.right;
            }
        }
        // Node to be deleted has only a left child
        else if (current.right == null) {
            if (current == root) {
                root = current.left;
            } else if (isLeftChild) {
                parent.left = current.left;
            } else {
                parent.right = current.left;
            }
        }
        // Node to be deleted has two children
        else {
            Node08 successor = getSuccessor(current);
            if (current == root) {
                root = successor;
            } else if (isLeftChild) {
                parent.left = successor;
            } else {
                parent.right = successor;
            }
            successor.left = current.left;
        }
    }
}
```

Kode Main

```
public class BinaryTreeMain08 {
    public static void main(String[] args) {
        BinaryTree08 bt = new BinaryTree08();
        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.print("Delete Node 8 ");
        bt.delete(8);
        System.out.println("");
        System.out.print("PreOrder Traversal : ");
        bt.traversePreOrder(bt.root);
        System.out.println("");

    }
}
```

Hasil Run

```
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
```

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. BST memungkinkan pencarian yang lebih cepat dan efisien karena struktur dan propertinya yang memungkinkan pengurangan ruang pencarian secara sistematis dan cepat.

2. atribut left menunjukan anak sebelah kiri dan atribut right menunjukan node anak sebelah kanan dari node saat ini

3. a. Atribut root dalam kelas BinaryTree berfungsi sebagai titik awal atau referensi utama ke node pertama (atau paling atas) dalam struktur pohon biner.

b. Ketika objek pohon (misalnya objek BinaryTree08) pertama kali dibuat, nilai dari root adalah null. Ini menandakan bahwa pohon biner tersebut masih kosong dan belum ada node yang ditambahkan ke dalamnya.

4. pertama akan di cek apakah tree kosong, jika kosong akan menambahkan node baru sebagai root

5. a. if (data < current.data):

- Kondisi ini memeriksa apakah nilai data yang akan ditambahkan lebih kecil dari nilai data pada node saat ini (current).Jika benar (true), maka data baru harus ditempatkan di subpohon kiri dari node saat ini.

b. if (current.left != null):

Kondisi ini memeriksa apakah node kiri dari node saat ini (current.left) tidak kosong (null).Jika node kiri ada (current.left tidak null), berarti kita perlu melanjutkan pencarian tempat yang tepat untuk data baru di subpohon kiri.

3. current = current.left:

Jika kondisi sebelumnya benar (node kiri ada), maka kita mengupdate current untuk menjadi node kiri tersebut (current.left).Ini berarti kita sekarang memproses subpohon kiri dari node saat ini dalam iterasi berikutnya.

4. else:

Jika kondisi if (current.left != null) salah (node kiri tidak ada atau null), kita masuk ke blok else.

5. current.left = new Node(data):

Jika kita mencapai blok else, artinya node kiri dari node saat ini kosong (null). Di sini, kita membuat node baru dengan nilai data baru (new Node(data)) dan menempatkannya sebagai anak kiri dari node saat ini (current.left).

6. break:

Setelah menambahkan node baru, kita menggunakan break untuk keluar dari loop.Ini menghentikan proses penambahan karena data baru telah ditempatkan di lokasi yang sesuai.

Percobaan 2

Kode BinaryTreeArray

```
public class BinaryTreeArray08 {
        int[] data;
        int idxLast;

        public BinaryTreeArray08() {
            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);
            }
        }
    }
```

Kode BinnaryTreeArrayMain

```
public class BinaryTreeArrayMain08 {
    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("\nInOrder Traversal : ");
        bta.traverseInOrder(0);
        System.out.println("\n");
    }
}
```

Hasil run

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

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. Atribut data adalah sebuah array yang menyimpan elemen-elemen dari binary tree. Dan Atribut idxLast menyimpan indeks dari elemen terakhir yang valid dalam array data.

2. Method populateData() digunakan untuk mengisi atribut data dan idxLast dari objek BinaryTreeArray08 dengan nilai-nilai yang diberikan sebagai argumen.

3. Method traverseInOrder() digunakan untuk melakukan traversal secara in-order pada sebuah pohon biner.

4. Jika suatu node binary tree disimpan dalam array pada indeks i, maka:

- Posisi anak kiri (left child) berada pada indeks 2 * i + 1.

- Posisi anak kanan (right child) berada pada indeks 2 * i + 2.

5. Statement int idxLast = 6 pada praktikum 2 percobaan nomor 4 digunakan untuk menyimpan indeks dari elemen terakhir (atau node terakhir) yang ada dalam array yang merepresentasikan binary tree.

Tugas

Kode Node08

```
package Tugas;

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

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

Kode BinnaryTree08

```
package Tugas;

public class BinaryTree08 {
    Node08 root;

    public BinaryTree08() {
        this.root = null;
    }

    private Node08 addRekursif(Node08 current, int data) {
        if (current == null) {
            return new Node08(data);
        }
```

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

        return current;
    }

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

    public int findMin() {
        return findMin(root);
    }

    private int findMin(Node08 node) {
        if (node == null) {
            return Integer.MAX_VALUE;
        }

        int min = node.data;
        int leftMin = findMin(node.left);
        int rightMin = findMin(node.right);

        if (leftMin < min) {
            min = leftMin;
        }
        if (rightMin < min) {
            min = rightMin;
        }

        return min;
    }

    public int findMax() {
        return findMax(root);
    }

    private int findMax(Node08 node) {
        if (node == null) {
            return Integer.MIN_VALUE;
        }

        int max = node.data;
        int leftMax = findMax(node.left);
        int rightMax = findMax(node.right);

        if (leftMax > max) {
            max = leftMax;
        }
```

```
        if (rightMax > max) {
            max = rightMax;
        }

        return max;
    }

    public void print() {
        printLeaf(root);
    }

    void printLeaf(Node08 node) {
        if (node == null) {
            return;
        }

        if (node.left == null && node.right == null) {
            System.out.print(node.data + " ");
        }

        printLeaf(node.left);
        printLeaf(node.right);
    }

    public int jmlLeaf() {
        return jmlLeaf(root);
    }

    int jmlLeaf(Node08 node) {
        if (node == null) {
            return 0;
        }

        if (node.left == null && node.right == null) {
            return 1;
        }

        return jmlLeaf(node.left) + jmlLeaf(node.right);
    }
    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);
        }
```

```
    }
}
```

Kode BinnaryTreeMain08

```
package Tugas;

public class BinaryTreeMain08 {
    public static void main(String[] args) {
        BinaryTree08 tree = new BinaryTree08();
        tree.add(50);
        tree.add(30);
        tree.add(70);
        tree.add(20);
        tree.add(40);
        tree.add(60);
        tree.add(80);

        System.out.println("Nilai paling kecil dalam tree: " +
tree.findMin());
        System.out.println("Nilai paling besar dalam tree: " +
tree.findMax());

        System.out.print("Data pada leaf nodes: ");
        tree.print();
        System.out.println();

        System.out.println("Jumlah leaf nodes: " + tree.jmlLeaf());
        System.out.print("PreOrder Traversal  : ");
        tree.traversePreOrder(tree.root);
        System.out.println("");
        System.out.print("PostOrder Traversal : ");
        tree.traversePostOrder(tree.root);
        System.out.println("");
    }
}
```

Hasil Run Program

```
Nilai paling kecil dalam tree: 20
Nilai paling besar dalam tree: 80
Data pada leaf nodes: 20 40 60 80
Jumlah leaf nodes: 4
PreOrder Traversal  :  50 30 20 40 70 60 80
PostOrder Traversal :  20 40 30 60 80 70 50
```