Laporan Praktikum Alogaritma dan Struktur Data

Jobsheet 14 : TREE

Nama: Aqil Rahmat Alifiandi

NIM: 2341760099

Kelas: SIB 1E

PROGRAM STUDI SISTEM INFORMASI BISNIS

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2023/2024

Praktikum 1

1. Buatlah class Node.05, BinaryTreeNoAbsen dan BinaryTreeMainNoAbsen
2. Di dalam class Node, tambahkan atribut data, left dan right, serta konstruktor default dan berparameter.

```
public class Node05 {
    int data;
    Node05 left;
    Node05 right;

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

3. Di dalam class BinaryTreeNoAbsen, tambahkan atribut root.

```
public class binaryTree05 {
    Node05 root;
```

4. Tambahkan konstruktor default dan method isEmpty() di dalam class BinaryTree05

```
public binaryTree05() {
    root = null;
}

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

5. Tambahkan method add() di dalam class BinaryTreeNoAbsen. Di bawah ini proses penambahan node tidak dilakukan secara rekursif, agar lebih mudah dilihat alur proses penambahan node dalam tree. Sebenarnya, jika dilakukan dengan proses rekursif, penulisan kode akan lebih efisien.

```
void add(int data) {
    if (isEmpty()) {
        root = new Node05(data);
    } else {
        Node05 current = root;
        Node05 parent = null;
        while (true) {
            parent = current;
            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;
            }
        }
    }
}
```

6. Tambahkan method find()

```
boolean find(int data) {
    Node05 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;
}
```

7. Tambahkan method traversePreOrder(), traverseInOrder() dan traversePostOrder(). Method traverse digunakan untuk mengunjungi dan menampilkan node-node dalam tree, baik dalam mode pre-order, in-order maupun post-order

```
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);
    }
}
```

8. Tambahkan method getSuccessor(). Method ini akan digunakan ketika proses penghapusan node yang memiliki 2 child.

```
Node05 getSuccessor(Node05 del) {
    Node05 successor = del.right;
    Node05 successorParent = del;
    while (successor.left!= null) {
        successorParent = successor;
        successor = successor.left;
    }
    if (successorParent!= del) {
        successorParent.left = successor.right;
    } else {
        successorParent.right = successor.right;
    }
    return successor;
}
```

9. Tambahkan method delete(). Di dalam method delete tambahkan pengecekan apakah tree kosong, dan jika tidak cari posisi node yang akan di hapus

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

    Node05 parent = root;
    Node05 current = root;
    boolean isLeftChild = false;
    while (current.data!= data) {
        parent = current;
        if (data < current.data) {
            current = current.left;
            isLeftChild = true;
        } else {
            current = current.right;
            isLeftChild = false;
        }
        if (current == null) {
            System.out.println(x:"Couldn't find data!");
            return;
        }
    }
```

10. Kemudian tambahkan proses penghapusan didalam method delete() terhadap node current yang telah ditemukan.

```
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;
        }
    }
}
```

11. Buka class BinaryTreeMainNoAbsen dan tambahkan method main() kemudian tambahkan kode berikut ini

```
public class binaryTreeMain05 {
    Run | Debug
    public static void main(String[] args) {
        binaryTree05 bt = new binaryTree05();
        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:"");
    }
}
```

12. Compile dan jalankan class BinaryTreeMain untuk mendapatkan simulasi jalannya program tree yang telah dibuat

```
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

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?
   Jawaban: Karena, pada binary search tree telah ditambahkan aturan baru yaitu, “semua left-child harus lebih kecil dibandingkan right-child dan parentnya” sehingga lebih mudah untuk melakukan pencarian data.
2. Untuk apakah di class Node, kegunaan dari atribut left dan right?
   Jawaban: berfungsi untuk menyimpan "left child" atau nilai yang lebih kecil dari root (node induk) dan atribut right berfungsi untuk menyimpan "right child" atau nilai yang lebih besar dari root (node induk)
3. **a**. Untuk apakah kegunaan dari atribut root di dalam class BinaryTree?
   Jawaban: Didalam BinaryTree root digunakan sebagai kepala atau inti, sama dengan head pada linked list yang digunakan sebagai kepala dari setiap linked list atau inti dari sebuah tree.
   b. Ketika objek tree pertama kali dibuat, apakah nilai dari root?
   Jawaban: ketika objek tree pertama kali dibuat nilai dari root bernilai null, karena masih belum ada data yang dimasukan
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
   Jawaban: Penambahan node baru yang akan digunakan sebagai root.
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: untuk mengecek apakah nilai input lebih kecil dari parent atau tidak. Jika iya, dilakukan pengecekan apakah current.left != null atau masih memiliki left child yang dimana memiliki subtree lagi. Jika iya maka dilakukan traversal dengan mengubah nilai current menjadi current.left. lalu, ada pengecekan jika tidak current.left == null atau tidak memiliki subtree atau left child maka posisi current.left tersebut akan menjadi tempat untuk meletakkan data yang diinput.

Praktikum 2

1. Buatlah class BinaryTreeArrayNoAbsen dan BinaryTreeArrayMainNoAbsen.
2. Buat atribut data dan idxLast di dalam class BinaryTreeArrayNoAbsen. Buat juga method populateData() dan traverseInOrder().

```
public class binaryTreeArray05 {
    int[] data;
    int idxLast;

    public binaryTreeArray05() {
        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);
        }
    }
}
```

3. Kemudian dalam class BinaryTreeArrayMain05 buat method main() dan tambahkan kode seperti gambar berikut ini di dalam method Main

```
public class binaryTreeArrayMain05 {
    Run | Debug
    public static void main(String[] args) {
        binaryTreeArray05 bta = new binaryTreeArray05();
        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");
    }
}
```

4. Jalankan class BinaryTreeArrayMain dan amati hasilnya!

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

Pertanyaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?
   Jawaban: Atribut data adalah sebuah array yang menyimpan elemen-elemen pohon biner, idxLast menyimpan indeks dari elemen terakhir yang valid dalam array data.
2. Apakah kegunaan dari method populateData()?
   Jawaban: Menggunakan method populateData() dapat dengan mudah mengisi atau memperbarui data pohon biner yang diwakili oleh array, dan menentukan hingga mana elemen-elemen dalam array tersebut valid untuk digunakan dalam operasi pada pohon biner.
3. Apakah kegunaan dari method traverseInOrder()?
   Jawaban: Kegunaan method traverseInOrder() adalah untuk mengunjungi dan mencetak elemen-elemen dari pohon biner dalam urutan yang teratur, yang sangat berguna untuk analisis data dan verifikasi struktur pohon biner.
4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?
   Jawaban: Jika suatu node binary tree disimpan dalam array pada indeks 2, maka posisi left child adalah pada indeks 5, dan posisi right child adalah pada indeks 6.
5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
   Jawaban: Statement int idxLast = 6; sangat penting untuk menentukan batas elemen-elemen yang valid dalam array data, memungkinkan traversal yang benar dan efisien pada pohon biner.

Tugas Praktikum

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(

Jawaban:

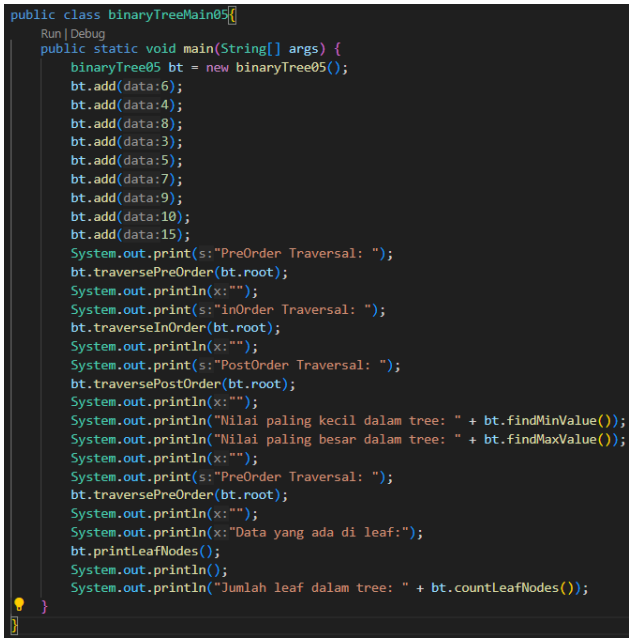
- BinaryTree

```java
public class binaryTree05 {
    Node05 root;

    public binaryTree05() {
        this.root = null;
    }

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

    public void add(int data) {
        root = addRecursive(root, data);
        Node05 newNode = new Node05(data);
        if (isEmpty()) {
            root = newNode;
        } else {
            Node05 current = root;
            Node05 parent;
            while (true) {
                parent = current;
                if (data < current.data) {
                    current = current.left;
                    if (current == null) {
                        parent.left = newNode;
                        return;
                    }
                } else {
                    current = current.right;
                    if (current == null) {
                        parent.right = newNode;
                        return;
                    }
                }
            }
        }
    }

    public 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);
        } else {
            return current;
        }
        return current;
    }
```

```java
public boolean find(int data) {
    Node05 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;
}

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

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

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

public Node05 getSuccessor(Node05 delNode) {
    Node05 successorParent = delNode;
    Node05 successor = delNode;
    Node05 current = delNode.right;
    while (current != null) {
        successorParent = successor;
        successor = current;
        current = current.left;
    }
    if (successor != delNode.right) {
        successorParent.left = successor.right;
        successor.right = delNode.right;
    }
    return successor;
}
```

```java
public boolean delete(int data) {
    Node05 parent = root;
    Node05 current = root;
    boolean isLeftChild = false;
    while (current.data != data) {
        parent = current;
        if (current.data > data) {
            isLeftChild = true;
            current = current.left;
        } else {
            isLeftChild = false;
            current = current.right;
        }
        if (current == null) {
            return false;
        }
    }

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

    else if (current.right == null) {
        if (current == root) {
            root = current.left;
        } else if (isLeftChild) {
            parent.left = current.left;
        } else {
            parent.right = current.left;
        }
    } 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.left != null && current.right != null) {
        Node05 successor = getSuccessor(current);
        if (current == root) {
            root = successor;
        } else if (isLeftChild) {
            parent.left = successor;
        } else {
            parent.right = successor;
        }
        successor.left = current.left;
    }
    return true;
}

public int findMinValue() {
    if (isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return Integer.MIN_VALUE;
    }
    Node05 current = root;
    while (current.left != null) {
        current = current.left;
    }
    return current.data;
}

public int findMaxValue() {
    if (isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return Integer.MAX_VALUE;
    }
    Node05 current = root;
    while (current.right != null) {
        current = current.right;
    }
    return current.data;
}

public void printLeafNodes() {
    displayLeafData(root);
}

public void displayLeafData(Node05 node) {
    if (node == null) {
        return;
    }
    if (node.left == null && node.right == null) {
        System.out.print(node.data + " ");
    }
    displayLeafData(node.left);
    displayLeafData(node.right);
}

public int countLeafNodes() {
    return countLeafNodesRekursif(root);
}

public int countLeafNodesRekursif(Node05 node) {
    if (node == null) {
        return 0;
    }
    if (node.left == null && node.right == null) {
        return 1;
    }
    return countLeafNodesRekursif(node.left) + countLeafNodesRekursif(node.right);

}
}
```

- BinaryTreeMain



```
public class binaryTreeMain05{
    Run | Debug
    public static void main(String[] args) {
        binaryTree05 bt = new binaryTree05();
        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("Nilai paling kecil dalam tree: " + bt.findMinValue());
        System.out.println("Nilai paling besar dalam tree: " + bt.findMaxValue());
        System.out.println(x:"");
        System.out.print(s:"PreOrder Traversal: ");
        bt.traversePreOrder(bt.root);
        System.out.println(x:"");
        System.out.println(x:"Data yang ada di leaf:");
        bt.printLeafNodes();
        System.out.println();
        System.out.println("Jumlah leaf dalam tree: " + bt.countLeafNodes());
    }
}
```

- BinaryTreeArray

```
public class binaryTreeArray05 {
    int[] data;
    int idxLast;

    public binaryTreeArray05() {
        data = new int[10];
        idxLast = -1;
    }
    public void add(int newData) {
        if (idxLast + 1 < data.length) {
            data[++idxLast] = newData;
        } else {
            System.out.println(x:"Tree penuh");
        }
    }

    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);
        }
    }
    public void traversePreOrder() {
        traversePreOrder(idxStart:0);
    }
    private void traversePreOrder(int idxStart) {
        if (idxStart <= idxLast) {
            System.out.print(data[idxStart] + " ");
            traversePreOrder(2 * idxStart + 1);
            traversePreOrder(2 * idxStart + 2);
        }
    }
    public void traversePostOrder() {
        traversePostOrder(idxStart:0);
    }

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

- BinaryTreeArrayMain

```java
public class binaryTreeArrayMain05 {
    Run | Debug
    public static void main(String[] args) {
        binaryTreeArray05 bta = new binaryTreeArray05();
        int[] data = {6,4,8,3,5,7,9,0,0,0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        System.out.println(x:"\nInOrder Traversal : ");
        bta.traverseInOrder(idxStart:0);
        System.out.println();
        System.out.println(x:"PreOrder Traversal: ");
        bta.traversePreOrder();
        System.out.println();
        System.out.println(x:"PostOrder Traversal: ");
        bta.traversePostOrder();
    }
}
```

- Hasil:

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