LAPORAN HASIL PRAKTIKUM
ALGORITMA DAN STRUKTUR DATA
PRAKTIKUM 13

Oleh:

RANDA HERU KUSUMA

NIM. 2341760009

SIB-1F / 25

D-IV SISTEM INFORMASI BISNIS JURUSAN

TEKNOLOGI INFORMASI POLITEKNIK

NEGERI MALANG

Link github: https://github.com/randaheru/Praktikum13.git

13.2 Kegiatan Praktikum 1

Implementasi Binary Search Tree menggunakan Linked List (45 Menit)

13.2.1 Percobaan 1

1. Buatlah class **NodeNoAbsen**, **BinaryTreeNoAbsen** dan **BinaryTreeMainNoAbsen**

2. Di dalam class **Node**, tambahkan atribut **data**, **left** dan **right**, serta konstruktor default dan berparameter.

```
public class Node25 {
    int data;
    Node25 left;
    Node25 right;

    // Default constructor
    public Node25() {
        this.left = null;
        this.data = 0;  // Initialize data to a default value
        this.right = 
    }
                    public class Node25
                    extends Object
    // Parameterized constructor
    public Node25(int data) {
        this.left = null;
        this.data = data;
        this.right = null;
    }
}
```

3. Di dalam class **BinaryTreeNoAbsen**, tambahkan atribut **root**

```
public class BinaryTree25 {
    Node25 root;
```

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

```
public BinaryTree25() {
    root = null;
}

public 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.

```java
public void add(int data) {
    if (isEmpty()) { // tree is empty
        root = new Node25(data);
    } else {
        Node25 current = root;
        while (true) {
            if (data < current.data) {
                if (current.left == null) {
                    current.left = new Node25(data);
                    break;
                } else {
                    current = current.left;
                }
            } else if (data > current.data) {
                if (current.right == null) {
                    current.right = new Node25(data);
                    break;
                } else {
                    current = current.right;
                }
            } else { // data already exists
                break;
            }
        }
    }
}
```

6. Tambahkan method find()

```java
public boolean find(int data) {
    boolean result = false;
    Node25 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;
}
```

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.

```
public void traversePreOrder(Node25 node) {
    if (node != null) {
        System.out.print(" " + node.data);
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

public void traversePostOrder(Node25 node) {
    if (node != null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.print(" " + node.data);
    }
}

public void traverseInOrder(Node25 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

```
public Node25 getSuccessor(Node25 del) {
    Node25 successor = del.right;
    Node25 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;
}
```

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

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

    Node25 parent = root;
    Node25 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 {
            parent = current;
            current = current.right;
            isLeftChild = false;
        }
    }
```

10. 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;
}

// Case 1: No children
if (current.left == null && current.right == null) {
    if (current == root) {
        root = null;
    } else if (isLeftChild) {
        parent.left = null;
    } else {
        parent.right = null;
    }
}
// Case 2: One child (left)
else if (current.right == null) {
    if (current == root) {
        root = current.left;
    } else if (isLeftChild) {
        parent.left = current.left;
    } else {
        parent.right = current.left;
    }
```

11. Buka class **BinarTreeMain** dan tambahkan method main() kemudian tambahkan kode

```
public class BinaryTreeMain25 {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        BinaryTree25 bt = new BinaryTree25();
        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:"");
    }
```

berikut ini

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

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
```

13.2.2 Pertanyaan Percobaan

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

Pencarian data dalam binary search tree (BST) lebih efektif dibandingkan binary tree biasa karena struktur dan aturan yang mengatur BST

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

Dalam kelas `Node` (atau `Node25` dalam konteks kode), atribut `left` dan `right` memiliki fungsi penting dalam representasi struktur pohon biner, termasuk binary search tree (BST).

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

Atribut `root` dalam kelas `BinaryTree` (atau `BinaryTree25` dalam konteks kode Anda) memiliki peran yang sangat penting dalam mengelola dan mengakses struktur pohon biner

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

Ketika objek pohon biner pertama kali dibuat, nilai dari atribut `root` adalah `null`. Ini menunjukkan bahwa pohon biner tersebut masih kosong dan belum memiliki node apa pun

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

Pengecekan Apakah Tree Kosong: Sistem memeriksa apakah tree kosong dengan melihat apakah atribut root bernilai null.

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;
        }
    }
```

Baris-baris program yang berikan adalah bagian dari metode `add()` dalam kelas `BinaryTree25`. Bagian ini bertanggung jawab untuk menambahkan sebuah node baru ke tree pada posisi yang tepat berdasarkan aturan binary search tree (BST)

13.3 Kegiatan Praktikum 2

Implementasi binary tree dengan array (45 Menit)

13.3.1 Tahapan Percobaan

1. Di dalam percobaan implementasi binary tree dengan array ini, data tree disimpan dalam array dan langsung dimasukan dari method main(), dan selanjutnya akan disimulasikan proses traversal secara inOrder.

2. Buatlah class **BinaryTreeArrayNoAbsen** dan **BinaryTreeArrayMainNoAbsen**

3. Buat atribut **data** dan **idxLast** di dalam class **BinaryTreeArrayNoAbsen**. Buat juga method

   populateData() dan **traverseInOrder().**

```
public class BinaryTreeArray25 {
    int[] data;
    int idxLast;

    public BinaryTreeArray25() {
        data = new int[10];
    }

    public void populateData(int[] data, int idxLast) {
        this.data = data;
        this.idxLast = idxLast;
    }

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

4. Kemudian dalam class **BinaryTreeArrayMainNoAbsen** buat method main() dan tambahkan kode seperti gambar berikut ini di dalam method Main

```
public class BinaryTreeArrayMain25 {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray25 bta = new BinaryTreeArray25();
        int[] data = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0};
        int idxLast = 6;
        bta.populateData(data, idxLast);

        System.out.print(s:"InOrder Traversal: ");
        bta.traverseInOrder(idxStart:0);
        System.out.println();
    }
}
```

5. Jalankan class **BinaryTreeArrayMain** dan amati hasilnya!

```
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
```

13.3.2 Pertanyaan Percobaan

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

Data digunakan untuk menyimpan elemen-elemen dari binary tree. Dan dapat membatasi traversal agar hanya berlangsung hingga node terakhir yang sebenarnya menyimpan data, tanpa harus melintasi elemen-elemen yang tidak terisi atau hanya sebagai padding.

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

Method populateData() memiliki tujuan untuk mengisi data ke dalam struktur binary tree yang direpresentasikan menggunakan array.

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

Method traverseInOrder() memiliki tujuan untuk melakukan penelusuran atau traversal pada binary tree dalam urutan in-order

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

Jika suatu node disimpan pada indeks 2, maka left child akan berada pada indeks (2 * 2) + 1 = 5, dan right child akan berada pada indeks (2 * 2) + 2 = 6.

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

Digunakan untuk menunjukkan bahwa node terakhir yang berisi data yang valid dalam binary tree adalah node pada indeks ke-6 dalam array data

13.4 Tugas Praktikum

Waktu pengerjaan: 90 menit

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

```
private Node25 addRecursive(Node25 current, int data) {
    if (current == null) {
        return new Node25(data);
    }

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

    return current;
}

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

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

```
public int findMin() {
    if (isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return Integer.MIN_VALUE; // Return the smallest integer value if the tree is empty
    }

    Node25 current = root;
    while (current.left != null) {
        current = current.left;
    }

    return current.data;
}

// Method untuk menampilkan nilai paling besar dalam tree
public int findMax() {
    if (isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return Integer.MAX_VALUE; // Return the largest integer value if the tree is empty
    }

    Node25 current = root;
    while (current.right != null) {
        current = current.right;
    }

    return current.data;
}
```

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

```
public void displayLeafNodes(Node25 node) {
    if (node == null) {
        return;
    }

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

    displayLeafNodes(node.left);
    displayLeafNodes(node.right);
}
```

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

```
public int countLeafNodes(Node25 node) {
    if (node == null) {
        return 0;
    }

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

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

5. Modifikasi class **BinaryTreeArray**, dan tambahkan :
   - method **add(int data)** untuk memasukan data ke dalam tree
   - method **traversePreOrder()** dan **traversePostOrder()**

```
public void add(int value) {
    if (idxLast >= data.length - 1) {
        System.out.println(x:"Tree is full, can't add more elements.");
        return;
    }
    idxLast++;
    data[idxLast] = value;
}
```

```
// Method untuk pre-order traversal
public void traversePreOrder(int idxStart) {
    if (idxStart <= idxLast) {
        System.out.print(data[idxStart] + " ");
        traversePreOrder(2 * idxStart + 1);
        traversePreOrder(2 * idxStart + 2);
    }
}

// Method untuk post-order traversal
public void traversePostOrder(int idxStart) {
    if (idxStart <= idxLast) {
        traversePostOrder(2 * idxStart + 1);
        traversePostOrder(2 * idxStart + 2);
        System.out.print(data[idxStart] + " ");
    }
}
```