LAPORAN PRATIKUM

"PEKAN 9"

Disusun Untuk Memenuhi Tugas Mata Kuliah Struktur Data

DOSEN PENGAMPU:

Wahyudi, Dr. S.T. M.T.

DISUSUN OLEH:

Karimah Irsyadiyah (2411533018)

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS ANDALAS

T.A 2024/2025

Daftar Pustaka

BAB I
PENDAHULUAN

1.1 Latar Belakang

Struktur data merupakan fondasi penting dalam dunia pemrograman, terutama dalam pengelolaan dan pemrosesan data secara efisien. Salah satu struktur data yang sering digunakan dalam berbagai aplikasi adalah pohon biner (binary tree) dan graf (graph). Pohon biner memungkinkan penyimpanan data secara hierarkis dengan efisiensi tinggi pada operasi pencarian, penyisipan, dan penghapusan. Sementara itu, graf memungkinkan representasi hubungan kompleks antar objek, yang sangat relevan dalam dunia nyata seperti pemetaan jaringan, media sosial, dan sistem transportasi.

Dalam praktikum ini, mahasiswa diminta untuk mengimplementasikan dua struktur data penting tersebut, yaitu pohon biner dan graf, dalam bentuk program Java. Praktikum ini bertujuan untuk melatih keterampilan pemrograman serta memahami prinsip traversal (penelusuran) data menggunakan metode InOrder, PreOrder, PostOrder pada pohon, dan metode DFS dan BFS pada graf.

1.2 Tujuan

- Mengimplementasikan struktur data pohon biner dan graf menggunakan bahasa Java.
- Memahami konsep dan cara kerja penelusuran pohon biner: InOrder, PreOrder, dan PostOrder.
- Menerapkan metode penelusuran graf menggunakan DFS (Depth-First Search) dan BFS (Breadth-First Search).
- Mengetahui bagaimana menghitung jumlah simpul dalam pohon.
- Meningkatkan kemampuan berpikir logis dan pemahaman algoritma traversing melalui pengkodean program.

1.3 Alat dan Bahan

- Perangkat Keras: Laptop/PC
- Perangkat komputer dengan IDE (contoh: IntelliJ IDEA / Eclipse).
- Java Development Kit (JDK).

BAB II
PEMBAHASAN

2.1 Langkah-langkah Praktikum dan Pembahasan Program

A. Class : BTree.java

a) Tujuan Program:

Membuat implementasi struktur data pohon biner (binary tree) dalam bahasa Java yang mendukung:

- Menambahkan node ke dalam pohon.
- Menampilkan urutan simpul dalam **InOrder**, **PreOrder**, dan **PostOrder**.
- Menghitung jumlah simpul dalam pohon.
- Menelusuri dan mencari data.
- Menampilkan bentuk visual pohon secara hierarkis di konsol.

b) Struktur dan Penjelasan Komponen Program:

1. package pekan9;

    - package pekan9;
      Menandakan file berada di dalam folder pekan9.

2. Atribut (variabel):

    - private Node root; → Node akar pohon.
    - private Node currentNode; → Menyimpan node terakhir yang aktif.

3. Method utama:

    - search(int data) → Mencari apakah data tertentu ada dalam pohon.
    - printInorder() → Menampilkan simpul pohon secara InOrder.
    - printPreOrder() → Menampilkan simpul secara PreOrder.
    - printPostOrder() → Menampilkan simpul secara PostOrder.

- countNodes() → Menghitung total simpul dalam pohon.
- print() → Menampilkan struktur pohon secara visual.
- setRoot(Node node) → Menetapkan node sebagai akar pohon.
- getCurrent() dan setCurrent() → Mendapatkan dan mengatur simpul aktif.

4. Fungsi Pendukung:

- Rekursif countNodes(Node node), search(Node node, int data) untuk eksplorasi data pohon.

```
package pekan9;

//Karimah Irsyadiyah
//2411533018

public class BTree {
    private Node root;
    private Node currentNode;

    public BTree() {
        root = null;
    }
    public boolean search(int data) {
        return search(root, data);
    }

    private boolean search(Node node, int data) {
        if (node == null) {
            return false;
        }
        if (node.getData() == data) {
            return true;
        }
        if (node.getLeft() != null) {
            if (search(node.getLeft(), data)) {
                return true;
            }
        }
        if (node.getRight() != null) {
            if (search(node.getRight(), data)) {
                return true;
            }
        }
        return false;
    }
    public void printInorder() {
        root.printInorder(root);
    }
    public void printPreOrder() {
        root.printPreorder(root);
    }
    public void printPostOrder() {
        root.printPostorder(root);
```

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

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

    private int countNodes(Node node) {
        int count = 1;
        if (node == null) {
            return 0;
        } else {
            count += countNodes(node.getLeft());
            count += countNodes(node.getRight());
        }
        return count;
    }

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

    public Node getCurrent() {
        return currentNode;
    }

    public void setCurrent(Node node) {
        this.currentNode = node;
    }

    public void setRoot(Node root) {
        this.root = root;
    }

}
```

B. Class : Node.java

a) Tujuan Program:

Sebagai representasi struktur simpul dari pohon biner, yang menyimpan data serta referensi ke anak kiri dan kanan.

b) Struktur dan Penjelasan Komponen Program:

1. Atribut:

- int data; → Menyimpan nilai integer.
- Node left, right; → Menyimpan referensi ke simpul anak.

2. Method Utama:

- Setter dan Getter → Mengatur dan mengambil nilai data, left, dan right.
- printInorder(Node node) → Penelusuran inorder: kiri → akar → kanan.
- printPreorder(Node node) → Penelusuran preorder: akar → kiri → kanan.
- printPostorder(Node node) → Penelusuran postorder: kiri → kanan → akar.
- print() → Menampilkan pohon secara visual di konsol (dengan garis dan simbol).

```
package pekan9;

//Karimah Irsyadiyah
//2411533018

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

    public Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
    public void setLeft(Node node) {
        if (left == null) {
            left = node;
        }
    }
    public void setRight(Node node) {
        if (right == null) {
            right = node;
        }
    }
    public Node getLeft() {
        return left;
    }

    public Node getRight() {
        return right;
    }
    public int getData() {
        return data;
    }
    public void setData(int data) {
        this.data = data;
    }
    void printPreorder(Node node) {
        if (node == null) {
            return;
        }
```

```
        System.out.print(node.getData() + " ");
        printPreorder(node.getLeft());
        printPreorder(node.getRight());
    }
    void printPreorder1(Node node) {
        if (node == null) {
            return;
        }
        System.out.print(node.data + " ");
        printPreorder(node.left);
        printPreorder(node.right);
    }

    void printPostorder(Node node) {
        if (node == null) {
            return;
        }
        printPostorder(node.left);
        printPostorder(node.right);
        System.out.print(node.data + " ");
    }

    void printInorder(Node node) {
        if (node == null) {
            return;
        }
        printInorder(node.left);
        System.out.print(node.data + " ");
        printInorder(node.right);
    }

    public String print() {
        return this.print("", true, "");
    }

    public String print(String prefix, boolean isTail, String sb) {
        if (right != null) {
            right.print(prefix + (isTail ? "|   " : "  "), false, sb);
        }
        System.out.println(prefix + (isTail ? "\\--" : "|--") +
data);
        if (left != null) {
            left.print(prefix + (isTail ? "    " : "|   "), true, sb);
        }
        return sb;
    }
}
```

C. Class : TreeMain.java

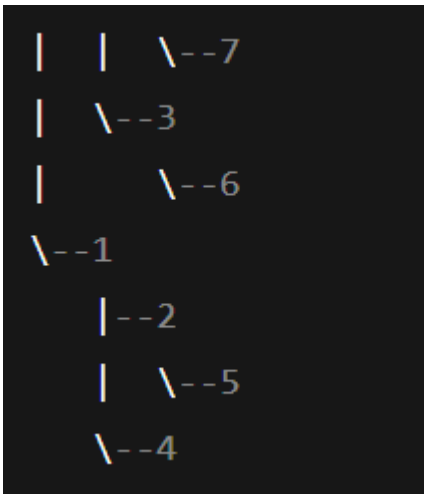
a) Tujuan Program:

Menguji implementasi pohon biner dengan menyusun struktur pohon secara manual, menghitung simpul, dan mencetak urutan penelusuran simpul.

b) Langkah-langkah Praktikum:

1. Membuat objek BTree sebagai struktur pohon.
2. Menambahkan node:
    - Akar: 1
    - Anak kiri akar: 2, dengan anak: 4 (kiri), 5 (kanan)
    - Anak kanan akar: 3, dengan anak: 6 (kiri), 7 (kanan)
3. Menetapkan node1 sebagai root.
4. Menghitung jumlah simpul dengan countNodes().
5. Menampilkan urutan simpul:
    - printInorder()
    - printPreOrder()
    - printPostOrder()
6. Menampilkan pohon secara visual di terminal.

c) Contoh Output Visual Pohon:




```
package pekan9;

//Karimah Irsyadiyah
//2411533018

public class TreeMain {
    public static void main(String[] args) {
        // Membuat Pohon
        BTree tree = new BTree();
        System.out.print("Jumlah Simpul awal pohon: ");
        System.out.println(tree.countNodes());
```

```
        // menambahkan simpul data 1
        Node root = new Node(1);
        // menjadikan simpul 1 sebagai root
        tree.setRoot(root);
        System.out.println("Jumlah simpul jika hanya ada root");
        System.out.println(tree.countNodes());

        Node node2 = new Node(2);
        Node node3 = new Node(3);
        Node node4 = new Node(4);
        Node node5 = new Node(5);
        Node node6 = new Node(6);
        Node node7 = new Node(7);

        root.setLeft(node2);
        node2.setLeft(node4);
        node2.setRight(node5);
        node3.setLeft(node6);
        root.setRight(node3);
        node3.setRight(node7);

    // Set root
        tree.setCurrent(tree.getRoot());
        System.out.println("menampilkan simpul terakhir: ");
        System.out.println(tree.getCurrent().getData());
        System.out.println("Jumlah simpul, setelah simpul 7 ditambahkan");
        System.out.println(tree.countNodes());

        tree.printInorder();
        System.out.println("\nPreorder: ");
        tree.printPreOrder();
        System.out.println("\nPostorder: ");
        tree.printPostOrder();
        System.out.println("\nMenampilkan simpul dalam bentuk pohon");
        tree.print();

    }
}
```

D. Class : GraphTraversal.java

a) Tujuan Program:

Membuat graf tak berarah sederhana menggunakan HashMap dan melakukan penelusuran DFS (Depth First Search) dan BFS (Breadth First Search).

b) Komponen Program:

1. Struktur Data:

Map<String, List<String>> graph → Menyimpan representasi graf adjacency list.

2. Method Utama:

- addEdge(String node1, String node2) → Menambahkan sisi (edge) antara dua node.

- printGraph() → Menampilkan representasi adjacency list.
- dfs(String start) → Penelusuran mendalam menggunakan rekursi.
- bfs(String start) → Penelusuran lebar menggunakan antrian (queue).

3. Contoh Penelusuran:
   - Input:

     A - B
     A - C
     B - D
     B - E

   - DFS Output:

     Penelusuran DFS:
     A B D E C

   - BFS Output:

     Penelusuran BFS:
     A B C D E

c) Fungsi Utama Program:

- Menampilkan struktur pohon biner dengan urutan penelusuran.
- Menyediakan fungsi pencarian data dan perhitungan jumlah simpul.
- Menampilkan visualisasi graf dan melakukan traversals (DFS dan BFS).

```
package pekan9;
import java.util.*;
public class GraphTraversal {
    private Map<String, List<String>> graph = new HashMap<>();

    // Menambahkan edge (graf tak berarah)
    public void addEdge(String node1, String node2) {
        graph.putIfAbsent(node1, new ArrayList<>());
        graph.putIfAbsent(node2, new ArrayList<>());
        graph.get(node1).add(node2);
        graph.get(node2).add(node1);
    }

    // Menampilkan graf
    public void printGraph() {
        System.out.println("Graf Asal (Adjacency List):");
        for (String node : graph.keySet()) {
            System.out.print(node + " -> ");
            List<String> neighbors = graph.get(node);
            System.out.println(String.join(", ", neighbors));
        }
        System.out.println();
```

```
    }

    // DFS rekursif
    public void dfs(String start) {
        Set<String> visited = new HashSet<>();
        System.out.println("Penelusuran DFS:");
        dfsHelper(start, visited);
        System.out.println();
    }

    private void dfsHelper(String current, Set<String> visited) {
        if (visited.contains(current)) return;
        visited.add(current);
        System.out.print(current + " ");
        for (String neighbor : graph.getOrDefault(current, new
ArrayList<>())) {
            dfsHelper(neighbor, visited);
        }
    }
 // BFS Iteratif
    public void bfs(String start) {
        Set<String> visited = new HashSet<>();
        Queue<String> queue = new LinkedList<>();
        queue.add(start);
        visited.add(start);

        System.out.println("Penelusuran BFS:");
        while (!queue.isEmpty()) {
            String current = queue.poll();
            System.out.print(current + " ");
            for (String neighbor : graph.getOrDefault(current, new
ArrayList<>())) {
                if (!visited.contains(neighbor)) {
                    queue.add(neighbor);
                    visited.add(neighbor);
                }
            }
        }

        System.out.println();
    }

    // Main
    public static void main(String[] args) {
        GraphTraversal graph = new GraphTraversal();

        // Contoh graf: A-B, A-C, B-D, B-E
        graph.addEdge("A", "B");
        graph.addEdge("A", "C");
        graph.addEdge("B", "D");
        graph.addEdge("B", "E");
        System.out.println("Graf Asal adalah: ");
        graph.printGraph();

        // Lakukan penelusuran
        graph.dfs("A");
        graph.bfs("A");
    }
}
```

BAB III
PENUTUP

3.1 Kesimpulan

Berdasarkan hasil praktikum yang telah dilaksanakan, dapat disimpulkan beberapa hal sebagai berikut:

- Struktur data pohon biner berhasil diimplementasikan menggunakan kelas Node, BTree, dan diuji dalam TreeMain. Penelusuran data dapat dilakukan menggunakan metode InOrder, PreOrder, dan PostOrder.
- Struktur data graf tak berarah juga berhasil dibangun dengan menggunakan kelas GraphTraversal yang mendukung penelusuran menggunakan DFS (rekursif) dan BFS (iteratif).
- Program mampu menghitung jumlah simpul dalam pohon serta menampilkan struktur pohon dalam bentuk visual hierarkis pada konsol.
- Praktikum ini memperkuat pemahaman mahasiswa terhadap struktur data non-linear dan traversal data, yang menjadi dasar penting dalam pemrograman lanjutan dan pengembangan aplikasi kompleks.
- Secara keseluruhan, praktikum berjalan lancar dan memberikan pengalaman implementasi nyata terhadap konsep-konsep abstrak dalam teori struktur data.