LAPORAN PRATIKUM STUKTUR DATA PEKAN 9
IMPLEMENTASI TREE DAN GRAPH TRAVERSAL

MATA KULIAH

DOSEN PENGAMPU : DR. WAHYUDI, S.T, M.T

OLEH:

AUFAN TAUFIQURRAHMAN

NIM 2411532011

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

BAB 1

PENDAHULUAN

1.1. Latar belakang

Dalam ilmu komputer, struktur data adalah cara menyimpan dan mengatur data di dalam komputer sehingga dapat digunakan secara efisien. Struktur data dapat dibagi menjadi dua kategori utama: linier dan non-linier. Struktur data linier, seperti array atau linked list, menyimpan data secara sekuensial. Namun, banyak masalah di dunia nyata yang lebih baik direpresentasikan menggunakan hubungan non-linier.

Dua contoh paling fundamental dari struktur data non-linier adalah Tree (Pohon) dan Graph (Graf). Tree digunakan untuk merepresentasikan data yang memiliki hubungan hierarkis, seperti struktur direktori file, silsilah keluarga, atau DOM (Document Object Model) pada halaman web. Graph, di sisi lain, digunakan untuk memodelkan hubungan jaringan yang kompleks di mana setiap entitas (node) dapat terhubung dengan banyak entitas lainnya, seperti jejaring sosial, jaringan komputer, atau peta jalan.

Memahami cara mengimplementasikan dan menelusuri (traversal) struktur data ini adalah keterampilan dasar yang sangat penting. Traversal adalah proses mengunjungi setiap node dalam struktur data tepat satu kali secara sistematis. Praktikum ini berfokus pada implementasi Binary Tree beserta metode traversal-nya (In-order, Pre-order, Post-order) dan implementasi Graph menggunakan Adjacency List beserta metode traversal-nya (Depth-First Search/DFS dan Breadth-First Search/BFS).

1.2. Tujuan

Adapun tujuan dari pelaksanaan praktikum ini adalah sebagai berikut:

1. Mahasiswa mampu memahami dan mengimplementasikan struktur data Binary Tree (Pohon Biner) menggunakan bahasa pemrograman Java.
2. Mahasiswa mampu mengimplementasikan tiga metode traversal dasar pada Binary Tree: In-order, Pre-order, dan Post-order.
3. Mahasiswa mampu memahami dan mengimplementasikan struktur data Graph (Graf) menggunakan representasi Adjacency List.
4. Mahasiswa mampu mengimplementasikan algoritma traversal pada Graph: Depth-First Search (DFS) dan Breadth-First Search (BFS).
5. Mahasiswa dapat menganalisis perbedaan hasil dan cara kerja dari setiap algoritma traversal, baik pada Tree maupun Graph.

1.3. Dasar Teori

1.3.1. Pohon (tree)

Pohon adalah struktur data non-linier yang merepresentasikan hubungan data yang bersifat hierarkis. Pohon terdiri dari sekumpulan simpul (node) yang saling terhubung oleh sisi (edge). Simpul paling atas disebut akar (root). Setiap simpul selain root memiliki satu simpul induk (parent) dan bisa memiliki nol atau lebih simpul anak (child). Simpul yang tidak memiliki anak disebut daun (leaf).

Pohon Biner (Binary Tree)

Pohon biner adalah jenis pohon khusus di mana setiap simpul dapat memiliki paling banyak dua anak, yang disebut sebagai anak kiri (left child) dan anak kanan (right child).

Traversal pada Pohon Biner

- **Pre-order Traversal:** Prosesnya adalah: kunjungi Root -> telusuri sub-pohon kiri -> telusuri sub-pohon kanan.
- **In-order Traversal:** Prosesnya adalah: telusuri sub-pohon kiri -> kunjungi Root -> telusuri sub-pohon kanan. Pada Binary Search Tree (BST), traversal ini akan menghasilkan urutan data yang tercортир.
- **Post-order Traversal:** Prosesnya adalah: telusuri sub-pohon kiri -> telusuri sub-pohon kanan -> kunjungi Root.

1.3.2. Graf (Graph)

Tentu, ini adalah draf laporan praktikum yang disusun berdasarkan keempat file Java yang Anda berikan. Laporan ini dirancang agar mudah disalin dan ditempelkan ke dalam pengolah kata seperti Microsoft Word.

LAPORAN PRAKTIKUM STRUKTUR DATA
Topik: Implementasi Tree dan Graph Traversal

BAB I: PENDAHULUAN

1.1 Latar Belakang

Dalam ilmu komputer, struktur data adalah cara menyimpan dan mengatur data di dalam komputer sehingga dapat digunakan secara efisien. Struktur data dapat dibagi menjadi dua kategori utama: linier dan non-linier. Struktur data linier, seperti array atau linked list, menyimpan data secara sekuensial. Namun, banyak masalah di dunia nyata yang lebih baik direpresentasikan menggunakan hubungan non-linier.

Dua contoh paling fundamental dari struktur data non-linier adalah Tree (Pohon) dan Graph (Graf). Tree digunakan untuk merepresentasikan data yang memiliki hubungan hierarkis, seperti struktur direktori file, silsilah keluarga, atau DOM (Document Object Model) pada halaman web. Graph, di sisi lain, digunakan untuk memodelkan hubungan jaringan yang kompleks di mana setiap entitas (node) dapat terhubung dengan banyak entitas lainnya, seperti jejaring sosial, jaringan komputer, atau peta jalan.

Memahami cara mengimplementasikan dan menelusuri (traversal) struktur data ini adalah keterampilan dasar yang sangat penting. Traversal adalah proses mengunjungi setiap node dalam struktur data tepat satu kali secara sistematis. Praktikum ini berfokus pada implementasi Binary Tree beserta metode traversal-nya (In-order, Pre-order, Post-order) dan implementasi Graph menggunakan Adjacency List beserta metode traversal-nya (Depth-First Search/DFS dan Breadth-First Search/BFS).

1.2 Tujuan Praktikum

Adapun tujuan dari pelaksanaan praktikum ini adalah sebagai berikut:

Mahasiswa mampu memahami dan mengimplementasikan struktur data Binary Tree (Pohon Biner) menggunakan bahasa pemrograman Java.

Mahasiswa mampu mengimplementasikan tiga metode traversal dasar pada Binary Tree: In-order, Pre-order, dan Post-order.

Mahasiswa mampu memahami dan mengimplementasikan struktur data Graph (Graf) menggunakan representasi Adjacency List.

Mahasiswa mampu mengimplementasikan algoritma traversal pada Graph: Depth-First Search (DFS) dan Breadth-First Search (BFS).

Mahasiswa dapat menganalisis perbedaan hasil dan cara kerja dari setiap algoritma traversal, baik pada Tree maupun Graph.

1.3 Dasar Teori

1. Pohon (Tree)

   Pohon adalah struktur data non-linier yang merepresentasikan hubungan data yang bersifat hierarkis. Pohon terdiri dari sekumpulan simpul (node) yang saling terhubung oleh sisi (edge). Simpul paling atas disebut akar (root). Setiap simpul selain root memiliki satu simpul induk (parent) dan bisa memiliki nol atau lebih simpul anak (child). Simpul yang tidak memiliki anak disebut daun (leaf).

   Pohon Biner (Binary Tree)

   Pohon biner adalah jenis pohon khusus di mana setiap simpul dapat memiliki paling banyak dua anak, yang disebut sebagai anak kiri (left child) dan anak kanan (right child).

   Traversal pada Pohon Biner

   - Pre-order Traversal: Prosesnya adalah: kunjungi Root -> telusuri sub-pohon kiri -> telusuri sub-pohon kanan.
   - In-order Traversal: Prosesnya adalah: telusuri sub-pohon kiri -> kunjungi Root -> telusuri sub-pohon kanan. Pada Binary Search Tree (BST), traversal ini akan menghasilkan urutan data yang tercoртир.
   - Post-order Traversal: Prosesnya adalah: telusuri sub-pohon kiri -> telusuri sub-pohon kanan -> kunjungi Root.

2. Graf (Graph)

   Graf adalah kumpulan simpul (vertices) dan sisi (edges) yang menghubungkan pasangan simpul. Graf digunakan untuk memodelkan jaringan dan hubungan antar objek. Graf dapat bersifat tak berarah (undirected), di mana sisi tidak memiliki arah, atau berarah (directed), di mana sisi memiliki arah dari satu simpul ke simpul lain.

   Representasi Graf

Salah satu cara paling umum untuk merepresentasikan graf adalah dengan Adjacency List (Daftar Ketetanggaan). Dalam representasi ini, untuk setiap simpul, kita menyimpan daftar simpul lain yang terhubung dengannya.

Traversal pada Graf

- Depth-First Search (DFS): DFS adalah algoritma penelusuran yang menjelajahi graf sejauh mungkin di sepanjang setiap cabang sebelum melakukan backtracking. Algoritma ini secara alami dapat diimplementasikan menggunakan rekursi atau tumpukan (stack).
- Breadth-First Search (BFS): BFS adalah algoritma penelusuran yang menjelajahi semua simpul tetangga pada level saat ini sebelum pindah ke simpul di level berikutnya. Algoritma ini menggunakan antrian (queue) untuk melacak simpul yang akan dikunjungi.

BAB 2

PROSEDUR KERJA DAN ANALISIS KODE

2.1. Implementasi Pohon Biner (Binary Tree)

2.1.1. File Node.java

File ini mendefinisikan struktur dasar dari sebuah simpul (node) yang akan digunakan dalam pohon biner.

```java
package pekan9;

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

```java
void printPreOrder( Node node) {
    if (node == null) {
        return;
    }
    System.out.print(node.getData() + " ");
    printPreOrder(node.getLeft());
    printPreOrder(node.getRight());
}
void printInOrder( Node node) {
    if (node == null) {
        return;
    }
    printInOrder(node.getLeft());
    System.out.print(node.getData() + " ");
    printInOrder(node.getRight());
}
void printPostOrder( Node node) {
    if (node == null) {
        return;
    }
    printPostOrder(node.getLeft());
    printPostOrder(node.getRight());
    System.out.print(node.getData() + " ");
}
```

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

Penjelasan Kode:

- **Atribut**: Setiap Node memiliki tiga atribut: data (nilai integer), left (referensi ke anak kiri), dan right (referensi ke anak kanan).
- **Konstruktor**: Node(int data) menginisialisasi sebuah simpul baru dengan data yang diberikan dan mengatur anak kiri dan kanan menjadi null.
- **Setter dan Getter**: Menyediakan metode untuk mengakses dan memodifikasi anak kiri, anak kanan, dan data.
- **Metode Traversal**: printPreOrder, printInOrder, dan printPostOrder adalah implementasi rekursif dari algoritma traversal. Setiap metode memeriksa apakah node saat ini null (kasus dasar). Jika tidak, metode tersebut akan mencetak data dan memanggil dirinya sendiri untuk anak kiri dan/atau kanan sesuai dengan urutan traversal yang benar.

2.1.2. File BTree.java

```java
package pekan9;

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.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;
    }
```

```java
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);
}
```

```java
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;
}
```

Penjelasan Kode:

- **Atribut**: root adalah referensi ke simpul akar pohon.
- **Konstruktor**: Menginisialisasi pohon sebagai pohon kosong dengan root bernilai null.
- **Metode Wrapper Traversal**: Metode seperti printInOrder() di kelas ini berfungsi sebagai "pintu masuk" yang mudah digunakan. Mereka memanggil metode traversal rekursif yang sebenarnya yang ada di kelas Node, dimulai dari root.
- **countNodes()**: Metode ini menghitung jumlah total simpul di pohon secara rekursif. Jika simpul saat ini null, ia mengembalikan 0. Jika tidak, ia mengembalikan 1 (untuk simpul itu sendiri) ditambah jumlah simpul di sub-pohon kiri dan sub-pohon kanan.
- **search()**: Mencari sebuah nilai di dalam pohon. Metode ini melakukan traversal sederhana (mirip pre-order) untuk menemukan data.

2.1.3. TreeMain.java

File ini berfungsi sebagai driver untuk membuat objek pohon, membangun strukturnya, dan mendemonstrasikan semua fungsionalitas yang telah dibuat.

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

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

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

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

```java
tree.setCurrent(tree.getRoot());
System.out.println(x:"menampilkan simpul terakhir: ");
System.out.println(tree.getCurrent().getData());
System.out.println(x:"Jumlah simpul; setelah simpul 7 ditambahkan");
System.out.println(tree.countNodes());
System.out.println(x:"InOrder: ");
tree.printInOrder();
System.out.println(x:"\nPreorder: ");
tree.printPreOrder();
System.out.println(x:"\nPostorder : ");
tree.printPostOrder();
System.out.println(x:"\nMenampilkan simpul dalam bentuk pohon");
tree.print();
```

Penjelasan Kode:

- **main method**: Titik masuk program.

- **Instansiasi**: Sebuah objek BTree dibuat.
- **Pembangunan Pohon**: Beberapa objek Node dibuat. Kemudian, metode setRoot(), setLeft(), dan setRight() digunakan untuk menghubungkan node-node ini secara manual untuk membentuk struktur pohon yang spesifik.
- **Demonstrasi**: Kode ini memanggil semua metode utama (countNodes, printInOrder, printPreOrder, printPostOrder, print) untuk menunjukkan bahwa implementasi pohon dan traversalnya bekerja sesuai harapan dan menampilkan hasilnya ke konsol.

2.1.4. File GraphTraversal.java

File ini mengimplementasikan struktur data graf menggunakan Adjacency List dan menyediakan metode untuk traversal DFS dan BFS.

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

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

```java
// 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(x:"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();
}
```

```java
// Main
Run main | Debug main | Run | Debug
public static void main(String[] args) {
    GraphTraversal graph = new GraphTraversal();

    // Contoh graf: A-B, A-C, B-D, B-E
    graph.addEdge(node1:"A", node2:"B");
    graph.addEdge(node1:"A", node2:"C");
    graph.addEdge(node1:"B", node2:"D");
    graph.addEdge(node1:"B", node2:"E");

    // Cetak graf awal
    System.out.println(x:"Graf Awal adalah: ");
    graph.printGraph();

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

Penjelasan Kode:

- **Representasi Graf**: private Map<String, List<String>> graph digunakan sebagai Adjacency List. Map ini memetakan setiap nama node (sebuah String) ke sebuah List yang berisi semua tetangganya.
- **addEdge**: Metode ini menambahkan sebuah sisi antara node1 dan node2. Karena ini adalah implementasi untuk graf tak berarah, node2 ditambahkan ke daftar tetangga node1, dan node1 ditambahkan ke daftar tetangga node2.
- **dfs(String start)**: Implementasi DFS.
    - Menggunakan Set<String> visited untuk memastikan setiap node hanya dikunjungi sekali, mencegah loop tak terbatas.
    - Implementasinya rekursif. Fungsi dfsHelper mengunjungi node saat ini, mencetaknya, lalu secara rekursif memanggil dirinya untuk setiap tetangga yang belum dikunjungi. Ini menciptakan efek penelusuran "mendalam".
- **bfs(String start)**: Implementasi BFS.
    - Menggunakan Queue<String> queue untuk menyimpan node yang akan dikunjungi.
    - Prosesnya iteratif. Dimulai dengan menambahkan node awal ke antrian. Selama antrian tidak kosong, ia mengambil node dari depan (poll), memprosesnya (mencetaknya), lalu menambahkan semua tetangganya yang belum dikunjungi ke belakang antrian. Ini menciptakan efek penelusuran "melebar" atau per level.
- **main method**: Membuat sebuah objek GraphTraversal, menambahkan beberapa sisi untuk membentuk graf contoh, lalu memanggil dfs("A") dan bfs("A") untuk mendemonstrasikan kedua algoritma traversal dari node "A".

BAB 3

PENUTUP

3.1 Kesimpulan

Dari praktikum yang telah dilaksanakan, dapat ditarik beberapa kesimpulan sebagai berikut:

1. Struktur data **Binary Tree** berhasil diimplementasikan menggunakan kelas Node untuk representasi simpul dan kelas BTree untuk manajemen pohon secara keseluruhan.
2. Tiga metode traversal pohon biner—**Pre-order (Root-Kiri-Kanan)**, **In-order (Kiri-Root-Kanan)**, dan **Post-order (Kiri-Kanan-Root)**—berhasil diimplementasikan secara rekursif. Setiap metode menghasilkan urutan kunjungan simpul yang berbeda sesuai dengan aturannya masing-masing, yang berguna untuk tujuan yang berbeda pula.

3. Struktur data **Graph** berhasil diimplementasikan menggunakan representasi **Adjacency List** dengan Map<String, List<String>>, yang efisien untuk menyimpan graf dengan jumlah sisi yang tidak terlalu padat.
4. Algoritma traversal graf, **DFS (Depth-First Search)** dan **BFS (Breadth-First Search)**, berhasil diimplementasikan.
    - **DFS**, yang diimplementasikan secara rekursif, menjelajahi satu cabang sepenuhnya sebelum beralih ke cabang lain.
    - **BFS**, yang diimplementasikan menggunakan Queue, menjelajahi simpul berdasarkan level atau jarak dari simpul awal.
5. Praktikum ini memberikan pemahaman praktis mengenai cara kerja, implementasi, dan perbedaan fundamental antara struktur data hierarkis (Tree) dan struktur data jaringan (Graph) serta algoritma-algoritma traversal yang menjadi dasar dari banyak algoritma yang lebih kompleks.