LAPORAN PRAKTIKUM PEKAN 8 STRUKTUR DATA
GRAF & TREE

OLEH :

FADIL INSANUS SIDDIK

(2411532013)

MATA KULIAH : PRAKTIKUM STRUKTURR DATA

DOSEN PENGAMPU : Dr. Wahyudi, S.T, M.T.

FAKULTAS TEKNOLOGI INFORMASI

PROGRAM STUDI INFORMATIKA

UNIVERSITAS ANDALAS

PADANG, JUNI 2025

A. PENDAHULUAN

Dalam dunia informatika, pemahaman tentang struktur data sangat penting karena menjadi fondasi dalam menyusun program yang efisien dan optimal. Struktur data seperti pohon (tree) dan graf (graph) memiliki peran yang sangat krusial dalam berbagai aplikasi, mulai dari pencarian data, pemetaan jaringan, hingga kecerdasan buatan.

Struktur data pohon biner merupakan jenis pohon yang memiliki maksimal dua anak untuk setiap simpulnya. Pohon ini sering digunakan dalam pengembangan sistem berkas, ekspresi aritmatika, serta algoritma pencarian seperti binary search tree. Dalam praktikum ini, implementasi pohon biner dilakukan melalui class Node, BTree, dan TreeMain, yang memungkinkan pembentukan pohon, penambahan simpul, perhitungan jumlah simpul, serta melakukan traversal seperti preorder, inorder, dan postorder.

Selain itu, struktur data graf juga diimplementasikan dalam praktikum ini. Graf adalah sekumpulan simpul yang saling terhubung melalui sisi (edges). Implementasi graf tak berarah dilakukan dalam class GraphTraversal, yang mendukung penambahan simpul, pencetakan graf dalam bentuk adjacency list, serta penelusuran menggunakan algoritma Depth-First Search (DFS) dan Breadth-First Search (BFS). Kedua algoritma ini digunakan untuk menjelajahi semua simpul dalam graf dengan pendekatan yang berbeda.

Melalui praktikum ini, mahasiswa diharapkan memahami bagaimana struktur data kompleks diimplementasikan secara langsung dalam bahasa pemrograman Java serta mengetahui cara kerja dari masing-masing traversal yang dilakukan baik pada pohon maupun graf.

B. TUJUAN PRAKTIKUM

Adapun tujuan dari dilaksanakannya praktikum ini adalah sebagai berikut:

1. Memahami konsep dasar struktur data pohon dan graf serta penerapannya dalam bahasa pemrograman Java.
2. Mengimplementasikan pohon biner dengan operasi dasar seperti pembuatan simpul, penambahan anak kiri dan kanan, serta traversal preorder, inorder, dan postorder.
3. Mengimplementasikan graf tak berarah menggunakan adjacency list serta menjalankan penelusuran menggunakan algoritma DFS dan BFS.
4. Meningkatkan kemampuan analisis dan logika pemrograman, terutama dalam memahami struktur data non-linear dan traversal rekursif maupun iteratif.
5. Mempersiapkan mahasiswa untuk dapat menerapkan struktur data pohon dan graf dalam permasalahan nyata seperti pemetaan, pencarian jalur, dan pengelompokan data.

C. LANGKAH-LANGKAH

1. Node

```
public class Node {
```

- Mendeklarasikan kelas dengan nama Node.

```
int data;
Node left;
Node right;
```

- Data bertipe int, menyimpan nilai dari node.
- Left dan right adalah referensi ke nod ekiri dan kanan (struktur pohon biner).

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

- Konstruktor Node, menerima data dan menginisialisasi node kiri dan kanan dengan null.

```
 public void setLeft(Node node) {
     if (left == null)
         left = node;
 }
```

- Setter untuk left. Node kiri hanyaakan di-set jika belum ada isinya (null).

```
public void setRight(Node node) {
    if (right == null)
        right = node;
}
```

- Mirip dengan sebelumnya, ini merupakan setter untuk right dengan pengecekan agar tidak menimpa jika sudah ada.

```
public Node getLeft() {
    return left;
}
```

- Getter untuk left.

```
public Node getRight() {
    return right;
}
```

- Getter untuk right.

```
public int getData() {
    return data;
}
```

- Getter untuk data.

```
public void setData(int data) {
    this.data = data;
}
```

- Setter untuk data.

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

- Menampilkan node dalam urutan Preorder: Kunjungi root → kiri → kanan.

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

- Menampilkan node dalam urutan Postorder: kiri → kanan → root.

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

- Menampilkan node dalam urutan Inorder; kiri → root → kanan.

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

- Fungsi ini digunakan untuk mencetak struktur pohon secara visual.
- Memanggil overload method print(String, Boolean, String) dengan argument awal kosong.

```
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;
}
```

- Tujuan: Menampilkan node dan anak-anaknya dalam bentuk visual seperti struktur pohon.
- Parameter:
  - Prefix : Indentasi atau awalan baris.
  - isTail : Apakah node ini adalah anak terakhir (berpengaruh pada symbol seperti \--).
  - Sb : Placeholder string builder, meskipun sebenarnya tidak digunakan untuk membangun string (mungkin ditinggalkan dari versi sebelumnya).
- Jika ada anak kanan, rekursi ke anak kanan terlebih dahulu (supaya kanan tampil di atas saat ditampilkan seperti pohon).
- Menampilkan node saat ini dengan awalan dan symbol cabang.
- Jika ada anak kiri, rekursi ke anak kiri.
- Mengembalikan sb (meskipun nilai ini sebenarnya tidak dipakai secara nyata).

2. BTree

```
public class BTree {
```

- Mendeklarasikan kelas dengna nama BTree yang merepresentasikan pohon biner.

```
private Node root;
private Node currentNode;
```

- root adalah node akar dari pohon.
- currentNode disiapkan untuk keperluan tambahan (belum digunakan di bagian awal ini).

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

- Konstruktor BTree, menginisialisasi pohon dengan root = null, artinya pohon awalnya kosong.

```
public boolean search(int data) {
    return search(root, data);
}
```

- Metode public search() dipanggil dari luar memulai pencarian root.

```
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;
}
```

- Tujuan: Mencari nilai data di dalam poson secara rekursif.
- Jika data cocok, kembalikan true.
- Cek anak kiri, lanjutkan pencarian jika tidak null.
- Cek anak kanan.
- Jika tidak ditemukan di keduanya, kembalikan false.

```
public void printInOrder() {
    root.printInorder(root);
}
```

- Mencetak isi pohon menggunakan traversal in-order (kiri → root → kanan).

```
public void printPreOrder() {
    root.printPreorder(root);
}
```

- Mencetak isi pohon menggunakan pre-order (root → kiri → kanan).

```
public void printPostOrder() {
    root.printPostorder(root);
}
```

- Mencetak isi pohon mengunakan post-order (kiri → kanan → root).

```
public Node getRoot() {
    return root;
}
```

- Getter untuk node root.

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

- Mengecek apakah pohon masih kosong.

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

- Menghitung jumlah semua node dalam pohon, dimulai dari root.

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

- Fungsi rekursif untuk menghitung jumlah node.
- count = 1 artinya node saat ini dihitung.
- Menambahkan jumlah node dari subtree kiri dan kanan secara rekursif.

```
        return count;
    }
}
```

- Mengembalikan total hitungna dari subtree yang telah dihitung.
- Penutup metode countNodes.

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

- Memanggil fungsi print() pada root, yang biasanya mencetak struktur pohon secara visual.
- Ini mengandalkan method print() dalam kelas Node.

```
public Node getCurrent() {
    return currentNode;
}
```

- Mengembalikan currentNode, yaitu node yang sedang aktif atau terpilih untuk operasi saat ini.

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

- Menetapkan currentNode ke node yang diberikan.

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

- Menetapkan root pohon ke node yang diberikan dari luar (biasanya digunakan dalam manipulasi manual).

3. GraphTraversal

```
public class GraphTraversal {
```

- Mendeklarasikan kelas dengan nama GraphTraversal.

```
private Map<String, List<String>> graph = new HashMap<>();
```

- Membuat atribut graph bertipe Map<String, List<String>, dimana:
  - String adalah nama node.
  - List<String> adalah daftar node tetangga dari node tersebut.
- graph diinisialisasi sebagai HashMap.

```
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);
}
```

- Menambahkan edge antara node1 dan node2.
- Jika node belum ada dalam graph, tambahkan dengan list kosong.
- Menambahkan hubungan dua arah → graf tak berarah.

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

- Menampilkan representasi adjacency list dari graf.
- Melakukan iterasi terhadap semua node (graph.keySet()), lalu mencetak node dan daftar tetangganya.
- Menggabungkan elemen list dengna koma.

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

- Memulai definisi fungsi DFS rekursif dari node start.
- visited: Set untuk menyimpan node yang sudah dikunjungi agar tidak dikunjungi lagi.
- Menampilkan teks sebagai heading sebelum proses DFS dimulai.
- Memanggil fungsi bantu rekursif dsHelper() untuk memulai DFS dari node start.
- Pindah baris setelah DFS selesai.

```
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);
    }
}
```

- dfsHelper(): Fungsi rekursif untuk penelusuran DFS.
- Jika node current sudah dikunjungi → hentikan rekursi (return).
- Jika belum:
  - Tandai sebagai dikunjungi.
  - Cetak node.
  - Rekursif untuk setiap neighbor.

```
public void bfs(String start) {
    Set<String> visited = new HashSet<>();
    Queue<String> queue = new LinkedList<>();
```

- Mulai deklarasi fungsi BFS.
- visited: Menyimpan node yang sudah dikunjungi.
- queue: Menyimpan antrian node yang akan dieksplorasi, menggunakan algoritma FIFO.

```
queue.add(start);
visited.add(start);
```

- Tambahkan node awal ke dalam antrian dan visited.

```
System.out.println("Penelusuran BFS:");
```

- Menampilkan heading sebelum proses 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);
        }
    }
}
```

- Selama antrian tidak kosong:
  - Ambil node dari antrian.
  - Cetak node.
  - Untuk setiap neighbor yang belum dikunjungi:
    Tambahkan ke antrian.
    Tandai sebagai dikunjungi.

```
System.out.println();
```

- Baris baru setelah DFS selesai.

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

- Fungsi main untuk menjalankan program.
- Membuat objek GraphTraversal.

```
graph.addEdge("A", "B");
graph.addEdge("A", "C");
graph.addEdge("B", "D");
graph.addEdge("B", "E");
```

- Menambahkan beberapa edge antar node untuk membuat graf:
  - A ←→ B
  - A ←→ C
  - B ←→ D
  - B ←→ E

```
System.out.println("Graf Awal adalah:");
graph.printGraph();
```

- Mencetak representasi adjacency list dari graf.

```
graph.dfs("A");
graph.bfs("A");
```

- Melakukan penelusuran:
  - DFS mulai dari A
  - BFS mulai dari A

4. TreeMain

```
public class TreeMain {
```

- Mendeklarasikan kelas dengna nama TreeMain

```
public static void main(String[] args) {
```

- Titik masuk utama dari program Java.

```
BTree tree = new BTree();
```

- Membuat objek baru bertipe BTree

```
System.out.print("Jumlah Simpul awal pohon: ");
System.out.println(tree.countNodes());
```

- Menampilkan jumlah simpul (node) awal, yang seharusnya 0 karena pohon belum diisi.
- tree.countNodes() memanggil method untuk menghitung total node dalam pohon.

```
Node root = new Node(1);
```

- Membuat objek Node dengan data 1. Ini akan dijadikan root.

```
tree.setRoot(root);
System.out.println("Jumlah simpul jika hanya ada root");
```

- setRoot(root) → mengatur simpul root sebagai akar pohon.
- Menampilkan informasi bahwa hanya root yang ada.

```
System.out.println(tree.countNodes());
```

- Menampilkan jumlah simpul saat ini (harusnya 1).

```
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);
```

- Membuat node tambahan dari 2-7.

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

- Menyusun struktur pohon:
  - Node 1 → root.
  - Node 2&3 → anak kiri & kanan dari 1.
  - Node 4&5 → anak dari 2.
  - Node 6&7 → anak dari 3.

```
tree.setCurrent(tree.getRoot());
```

- Mengatur node “current” menjadi root.
- Fungsi ini bisa berguna jika navigasi pohon dilakukan secara dinamis.

```
System.out.println("menampilkan simpul terakhir: ");
System.out.println(tree.getCurrent().getData());
```

- Menampilkan data dari simpul current, yaitu simpul root (1).

```
System.out.println("Jumlah simpul setelah simpul 7 ditambahkan");
System.out.println(tree.countNodes());
```

- Menampilkan jumlah simpul setelah pohon lengkap terbentuk (harusnya 7 simpul).

```
System.out.println("\nInOrder: ");
tree.printInOrder();
```

- Traversal inorder → kiri → root → kanan.
- Menampilkan urutan simpul sesuai inorder.

```
System.out.println("\nPreOrder: ");
tree.printPreOrder();
```

- Traversal preorder → root → kiri → kanan.

```
System.out.println("\nPostOrder : ");
tree.printPostOrder();
```

- Traversal postorder → kiri → kanan → root.

```
System.out.println("\nmenampilkan simpul dalam bentuk pohon");
tree.print();
```

- Menampilkan pohon secara visual.
- tree.print() bisa jadi mencetak struktur berbentuk pohon (tergantung implementasi BTree).