LAPORAN PRAKTIKUM 9

STRUKTUR DATA

Oleh:
Ibrahim Mousa Dhani
2411532010

Dosen Pengampu : Dr. Wahyudi S.T, M.T.

Mata Kuliah : Struktur Data

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

PADANG

A. Pendahuluan

Pada praktikum kali ini kita akan mencoba memahami dan mengimplementasikan tentang Tree (Pohon) dan Graph (Graf).

Tree adalah struktur data hierarkis yang terdiri dari kumpulan simpul (node) yang saling terhubung. Struktur ini memiliki sebuah simpul puncak yang disebut root dan setiap simpul dapat memiliki anak, membentuk tingkatan-tingkatan layaknya silsilah keluarga. Sifat hierarkisnya membuat Tree sangat ideal untuk merepresentasikan data seperti sistem file direktori, struktur organisasi, atau penguraian ekspresi matematika. Dalam praktikum ini, implementasi Tree difokuskan pada Binary Tree (Pohon Biner).

Graph adalah struktur data yang merepresentasikan kumpulan objek dan hubungan antar objek tersebut. Struktur ini terdiri dari vertices (simpul) dan edges (sisi) yang menghubungkannya. Berbeda dengan Tree, Graph tidak memiliki aturan hierarki yang kaku, sehingga lebih fleksibel dalam memodelkan hubungan kompleks seperti jejaring sosial, peta navigasi jalan, atau ketergantungan antar komponen dalam sebuah jaringan. Pada praktikum ini, Graph diimplementasikan menggunakan representasi Adjacency List.

B. Tujuan Praktikum

1. Memahami dan mengimplementasikan konsep dasar Pohon Biner (Binary Tree).
2. Memahami dan mengimplementasikan konsep dasar Graph (Graf).
3. Mengimplementasikan operasi-operasi fundamental pada Tree , terutama algoritma penelusuran In-order, Pre-order, dan Post-order secara rekursif,
4. Memahami dan mengimplementasikan penelusuran mengggunakan BFS dan DFS pada Graph

C. Langkah Langkah

a. Node

1. Pertama buat package baru dan beri nama pekan9, dan buat kelas baru dan beri nama Node.java. Kelas ini adalah struktur dasar yang nantinya akan digunakan oleh struktur Binary Tree (Pohon Biner) yang lengkap
2. Deklarasikan atribut ‘data’ bertipe int, atribut ini akan menyimpan nilai atau data yang dimiliki oleh node. Setiap node dalam pohon biner perlu menyimpan tiga hal: nilainya sendiri, dan referensi ke anak kiri dan anak kanan, yaitu int data, node left, node right.

```
package pekan9;

// NAMA : IBRAHIM MOUSA DHANI
// NIM  : 2411532010

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

3. Lalu buat sebuah konstruktor public Node(int data). Konstruktor ini menerima satu parameter, yaitu nilai yang akan disimpan di dalam node

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

4. Selanjutnya buat Method Getter dan Setter, yang berfungsi untuk mengakses (get) dan mengubah (set) nilai atribut dari luar kelas. Ini adalah praktik yang baik

```
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;
}
```

dalam pemrograman berorientasi objek. -Setter (setLeft, setRight, setData): Metode untuk menetapkan atau mengubah nilai. -Getter (getLeft, getRight, getData): Metode untuk mengambil atau membaca nilai.

5. Implementasikan Metode Traversal yaitu mengunjungi setiap node dalam pohon tepat satu kali. Ada tiga cara umum: Pre-order (Akar-Kiri-Kanan), Post-order (Kiri-Kanan-Akar), dan In-order(Kiri-Akar-Kanan). Metode ini bekerja secara rekursif. Bisa dilihat pada gambar berikut

```
void printPreorder(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);
}
```

6. Terakhir buat metode untuk mencetak Visual Tree yaitu yang pertama buat metode publik sederhana print() yang tidak memiliki parameter. Metode ini akan memulai proses pencetakan dengan memanggil metode rekursif print dengan nilai awal. Kedua metode Rekursif (print(prefix, isTail, sb)) ini adalah inti dari pencetakan visual. mencetak pohon dengan urutan: Sub-pohon kanan, Node saat ini, Sub-pohon kiri. Ini dilakukan agar akar pohon muncul di kiri atas, dan pohon "tumbuh" ke bawah. prefix: String yang berisi spasi dan garis (│) untuk menciptakan inden dan garis vertikal. isTail: boolean untuk mengetahui apakah node ini adalah anak terakhir dari induknya. Ini menentukan apakah akan menggunakan └── (anak terakhir) atau ┌── (bukan anak terakhir).

```
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;
}
```

b. BTree

1. Buat kelas baru dan beri nama BTree, yang berfungsi berfungsi sebagai pengontrol dari objek-objek Node yang sudah dibuat sebelumnya.
2. Deklarasikan atribut Node root yang menyimpan referensi ke Node paling atas (akar) dari seluruh pohon. Dan Node currentNode yang digunakan untuk melacak node yang sedang aktif atau dipilih. Serta buat Konstruktor untuk Pohon Kosong

```
package pekan9;

//NAMA : IBRAHIM MOUSA DHANI
//NIM  : 2411532010

public class BTree {
    private Node root;
    private Node currentNode;
    public BTree() {
        root = null;
    }
```

3. Implementasikan Metode Pencarian (search), untuk memeriksa apakah sebuah nilai ada di dalam pohon. Implementasinya menggunakan rekursi dan dibagi menjadi dua bagian. Pertama buat metode public boolean search(int data). Metode ini yang akan dipanggil oleh pengguna. Tugasnya hanya satu: memulai proses rekursif dari root. Kedua buat metode private boolean search(Node node, int data) yang melakukan pekerjaan sebenarnya.

```
  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;
  }
```

4. Buat metode untuk menambahkan fungsi Traversal. Metode ini akan memanggil fungsi traversal dari class Node yang sudah dibuat sebelumnya, digunakan untuk menampilkan isi pohon.

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

5. Buat fungsi countNodes() yang berfungsi untuk menghitung jumlah seluruh simpul (node) dalam pohon.

```
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;
    }
```

6. Buat fungsi print() untuk menampilkan struktur pohon

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

7. Tambahkan Getter dan Setter, get/setRoot() berfungsi untuk mengatur akar pohon.get/setCurrent() berfungsi untuk menunjuk node aktif

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

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

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

8. Berikut kode lengkapnya

```
package pekan9;

//NAMA : IBRAHIM MOUSA DHANI
//NIM  : 2411532010

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;
    }
    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;
    }

```

c. TreeMain

1. Buat kelas baru dan beri nama TreeMain yang berfungsi sebagai program utama (main class) untuk menguji dan menjalankan struktur pohon biner (BTree) dan (Node.java) yang telah dibuat sebelumnya.

```
package pekan9;

// NAMA : IBRAHIM MOUSA DHANI
// NIM  : 2411532010

public class TreeMain {
```

2. Buat Objek Pohon Biner (BTree). Membuat objek tree dari class BTree. Pada awalnya root masih null, artinya pohon kosong. Dan tampilkan jumlah simpul saat pohon kosong. countNodes() dipanggil untuk menghitung jumlah node. Karena belum ada node, hasilnya 0.

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

3. Buat simpul (node) pertama dengan nilai 1 Menjadikannya sebagai akar pohon dengan setRoot(). Dan tampilkan jumlah simpul setelah ditambahkan. Setelah ada satu node (root), hasil countNodes() akan menjadi 1. Lalu buat simpul-simpul baru dengan data 2 hingga 7.

```
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);
```

4. Hubungkan Node secara manual, gunakan metode setLeft() dan setRight() dari kelas Node untuk merangkai strukturnya.

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

5. Panggil metode-metode yang sudah ada di kelas BTree untuk menampilkan informasi tentang pohon yang sudah jadi.
   tree.setCurrent(tree.getRoot()) digunakan untuk mendemonstrasikan fungsi setCurrent. Kemudian tree.getCurrent().getData() dipanggil untuk mencetak data dari currentNode tersebut, yaitu 1. tree.countNodes() untuk memastikan semua 7 simpul sudah terhitung dengan benar.

```
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());
```

6. Lalu panggil metode Traversal.
   tree.printInorder();: Mencetak dengan urutan Kiri-Akar-Kanan.
   tree.printPreOrder();: Mencetak dengan urutan Akar-Kiri-Kanan.
   tree.printPostOrder();: Mencetak dengan urutan Kiri-Kanan-Akar.
   Terkahir panggil metode cetak tree.print() yang memanggil metode visual untuk menggambar struktur pohon di konsol.

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

7. Setelah semua benar dan tidak ada eror, maka jalankan program, lalu akan menghasilkan output sebagai berikut

```
<terminated> TreeMain [Java Application] C:\Users\cibur\
Jumlah Simpul awal pohon: 0
Jumlah simpul jika hanya ada root
1
menampilkan simpul terakhir:
1
Jumlah simpul; setelah simpul 7 ditambahkan
7
InOrder:
4 2 5 1 6 3 7
Preorder:
1 2 4 5 3 6 7
Postorder :
4 5 2 6 7 3 1
Menampilkan simpul dalam bentuk pohon
        ┌── 7
    ┌── 3
    │   └── 6
└── 1
    │   ┌── 5
    └── 2
        └── 4
```

8. Berikut kode program versi lengkapnya

```java
package pekan9;

// NAMA : IBRAHIM MOUSA DHANI
// NIM  : 2411532010

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());
        System.out.println("InOrder: ");
        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. GraphTraversal

1. Buat kelas baru dan beri nama GraphTraversal, dan import semua kelas yang dibutuhkan pada java.util. serta inisilaisasi graph

```java
package pekan9;

// NAMA : IBRAHIM MOUSA DHANI
// NIM  : 2411532010

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

2. Buat Fungsi addEdge() yang berfungsi menambahkan edge dua arah karena graf tidak berarah. Jika node belum ada dalam graf, maka akan dibuat dulu (putIfAbsent). Lalu, node tetangga dimasukkan ke daftar masing-masing node.

```
// 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);
```

3. Lalu buat Fungsi printGraph() yang menampilkan struktur graf dalam bentuk adjacency list.

```
// Menampilkan graf awal
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));
    }
    System.out.println();
```

4. Implementasi Depth-First Search (DFS), terdapat dua fungsi yaitu dfs dan dfsHelper. DFS dilakukan secara rekursif, pertama kunjungi node saat ini lalu tandai sebagai sudah dikunjungi dan rekursif ke semua tetangganya

```
// 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);
```

5. Implementasi Breadth-First Search (BFS), BFS dilakukan secara iteratif menggunakan antrean pertama masukkan simpul awal ke antrean, lalu selama antrean tidak kosong: ambil simpul dari depan, kunjungi dan cetak, dan masukkan semua tetangga yang belum dikunjungi ke antrean

```
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();
```

6. Terakhir buat fungsi main unntuk menjalankan program
   Buat Objek Graf: GraphTraversal graph = new GraphTraversal();
   Panggil metode addEdge beberapa kali untuk membuat struktur graf yang diinginkan.
   Panggil printGraph() untuk memverifikasi bahwa struktur graf sudah benar.
   Panggil graph.dfs("A") dan graph.bfs("A") untuk menjalankan kedua algoritma penelusuran, dimulai dari node A

```
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");
    // Cetak graf awal
    System.out.println("Graf Awal adalah: ");
    graph.printGraph();
    // Lakukan penelusuran
    graph.dfs("A");
    graph.bfs("A");
}
```

7. Setelah semua benar dan tidak ada eror, maka jalankan program, lalu akan menghasilkan output sebagai berikut

```
<terminated> GraphTraversal [Ja
Graf Awal adalah:
Graf Awal (Adjacency List):
A -> B, C
B -> A, D, E
C -> A
D -> B
E -> B

Penelusuran DFS:
A B D E C
Penelusuran BFS:
A B C D E
```

8. Berikut kode program versi lengkapnya

```
package pekan9;

// NAMA : IBRAHIM MOUSA DHANI
// NIM  : 2411532010

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 awal
    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));
        }
        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");
        // Cetak graf awal
        System.out.println("Graf Awal adalah: ");
        graph.printGraph();
        // Lakukan penelusuran
        graph.dfs("A");
        graph.bfs("A");
    }
}
```

D. Kesimpulan

Berdasarkan praktikum yang telah dikakukan, dapat disimpulkan bahwa implementasi struktur data pohon biner dan graf sangat penting dalam pengelolaan data yang kompleks. Dengan membangun class Node, BTree, dan TreeMain, mahasiswa dapat memahami bagaimana menyusun pohon secara manual, melakukan traversal (inorder, preorder, postorder), serta menampilkan struktur pohon secara visual. Traversal pohon membantu memahami urutan akses data dalam berbagai konteks seperti pencarian, pengurutan, dan ekspresi.

Sementara itu, penerapan struktur graf tak berarah menggunakan adjacency list memungkinkan hubungan antar simpul direpresentasikan secara efisien. Melalui algoritma DFS (rekursif) dan BFS (iteratif), kita dapat menjelajahi graf secara menyeluruh dengan pendekatan yang berbeda. DFS mendalam lebih dahulu, sedangkan BFS melebar terlebih dahulu. Praktikum ini tidak hanya mengasah keterampilan pemrograman, tetapi juga memperkuat pemahaman konsep dasar struktur data graf dan pohon dalam pemecahan masalah.