LAPORAN PRAKTIKUM STRUKTUR DATA

PEKAN 9: Graff & Tree

Disusun Oleh

Abdullah Al Ramadhani (2411533016)

DOSEN PENGAMPU

Dr.Wahyudi.ST,MT

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

A. Pendahuluan

Dalam bidang informatika, pemahaman mengenai struktur data sangatlah penting karena menjadi dasar utama dalam merancang program yang efisien dan optimal. Struktur data seperti pohon (tree) dan graf (graph) memiliki peran vital dalam berbagai penerapan, mulai dari pencarian informasi, pemetaan jaringan, hingga pengembangan kecerdasan buatan.

Struktur data pohon biner adalah salah satu bentuk pohon yang setiap nodenya memiliki maksimal dua anak. Jenis pohon ini sering dimanfaatkan dalam pembuatan sistem file, evaluasi ekspresi matematika, serta algoritma pencarian seperti binary search tree. Pada praktikum ini, pohon biner diimplementasikan melalui class Node, BTree, dan TreeMain, yang mendukung pembuatan pohon, penambahan node, perhitungan jumlah simpul, serta traversal seperti preorder, inorder, dan postorder.

Di samping itu, praktikum ini juga membahas penerapan struktur data graf. Graf merupakan kumpulan simpul yang saling terhubung oleh sisi (edge). Implementasi graf tak berarah dilakukan menggunakan class GraphTraversal, yang memiliki fitur penambahan simpul, pencetakan graf dalam bentuk adjacency list, serta penelusuran menggunakan algoritma Depth-First Search (DFS) dan Breadth-First Search (BFS). Kedua metode ini digunakan untuk menelusuri seluruh simpul dalam graf dengan pendekatan yang berbeda.

Melalui praktikum ini, mahasiswa diharapkan mampu memahami cara mengimplementasikan struktur data yang kompleks menggunakan bahasa pemrograman Java, serta mengenal mekanisme kerja dari berbagai jenis traversal baik pada struktur pohon maupun graf.

B. Tujuan

Adapun tujuan dari praktikum ini dibuat adalah sebagai berikut:

1) Memahami prinsip dasar dari struktur data pohon dan graf serta bagaimana cara mengaplikasikannya dalam pemrograman Java.
2) Mampu mengimplementasikan pohon biner dengan berbagai operasi dasar seperti membuat node, menambahkan anak kiri dan kanan, serta melakukan traversal preorder, inorder, dan postorder.
3) Mampu membangun graf tak berarah dengan pendekatan adjacency list dan melakukan penelusuran menggunakan algoritma Depth-First Search (DFS) dan Breadth-First Search (BFS).
4) Mengembangkan kemampuan analitis dan logika pemrograman, khususnya dalam memahami struktur data non-linear serta proses traversal baik secara rekursif maupun iteratif.
5) Membekali mahasiswa dengan keterampilan untuk mengaplikasikan struktur data pohon dan graf pada permasalahan nyata seperti pemetaan jaringan, pencarian rute, dan klasifikasi data.

C. Langkah-langkah

a) Node

1) Deklarasikan kelas beserta variabel-variabelnya, dimulai dari class Node, kemudian atribut int data, Node left, dan Node right.

```
1 package Pekan9;
2
3 public class Node {
4     int data;
5     Node left;
6     Node right;
```

2) Buat konstruktor untuk class Node, yang berfungsi untuk membuat objek simpul baru dengan nilai yang diberikan sebagai parameter. Nilai tersebut akan disimpan dalam variabel data, sementara left dan right diinisialisasi dengan null, karena pada awalnya node belum memiliki anak kiri maupun kanan.

```
 9     public Node (int data) {
10         this.data= data;
11         left = null;
12         right = null;
13     }
```

3) Selanjutnya, buat metode setter dan getter, yaitu setLeft, setRight, getLeft, getRight, getData, dan setData. Masing-masing metode ini memiliki fungsi tertentu dalam pengolahan dan pengambilan data pada simpul pohon.

```
16     public void setLeft(Node node) {
17         if (left == null)
18             left = node;
19     }
20     public void setRight (Node node) {
21         if (right == null)
22             right = node;
23     }
24     public Node getLeft() {
25         return left;
26     }
27     public Node getRight() {
28         return right;
29     }
30     public int getData() {
31         return data;
32     }
33     public void setData(int data) {
34         this.data = data;
35     }
```

4) Buat juga metode traversal yang mencakup tiga jenis: Pre-order, In-order, dan Post-order. Traversal Pre-order dilakukan dengan mengunjungi simpul terlebih dahulu, lalu anak kiri, kemudian anak kanan. Traversal Post-order dilakukan dengan mengunjungi anak kiri dan kanan terlebih dahulu, kemudian simpul utama. Sedangkan traversal In-order dimulai dengan anak kiri, lalu simpul utama, dan terakhir anak kanan.

```
39    void printPreorder(Node node) {
40        if (node == null)
41            return;
42        System.out.print(node.data + " ");
43        printPreorder(node.left);
44        printPreorder(node.right);
45    }
46    void printPostorder(Node node) {
47        if (node == null)
48            return;
49        printPostorder(node.left);
50        printPostorder(node.right);
51        System.out.print(node.data + " ");
52    }
53    void printInorder(Node node) {
54        if (node == null)
55            return;
56        printPostorder(node.left);
57        System.out.print(node.data + " ");
58        printPostorder(node.right);
59    }
```

5) Terakhir, buatlah metode print yang berfungsi untuk mencetak atau menampilkan struktur dari pohon biner yang telah dibuat.

```
62    public String print() {
63        return this.print("", true, "");
64        }
65    public String print(String prefix, boolean isTail, String sb) {
66        if (right != null) {
67            right.print(prefix + (isTail ? "|   " : "    "), false, sb);
68        }
69        System.out.println( prefix+(isTail ? "\\--" : "/--")+data);
70        if (left != null) {
71        left.print(prefix+(isTail ? "    " : "   "), true, sb);
72        }
73        return sb;
74    }
75 }
```

b) Tree

1) Buat class BTree dan deklarasikan dua variabel, yaitu private Node root yang berfungsi sebagai akar dari pohon, serta Node currentNode yang digunakan untuk menyimpan referensi simpul yang sedang aktif atau sedang diproses dalam pohon.

```
1 package Pekan9;
2
3 public class BTree {
4     private Node root;
5     private Node currentNode;
6
```

2) Tambahkan konstruktor untuk class BTree, yang bertugas menginisialisasi pohon baru dengan root = null, menandakan bahwa pohon masih kosong ketika pertama kali dibuat.

```
8       public BTree() {
9           root = null;
10      }
```

3) Buatlah metode pencarian, yaitu search(int data) untuk mencari nilai dalam pohon. Metode ini akan memulai pencarian dari akar pohon dan kemudian memanggil metode pencarian rekursif. Selanjutnya, buat juga metode search(Node node, int data) yang merupakan versi rekursif untuk mencari data pada simpul tertentu.

```
12      public boolean search(int data) {
13          return search(root, data);
14      }
15
16      private boolean search(Node node, int data) {
17          if (node.getData() == data)
18              return true;
19          if (node.getLeft() != null)
20              if (search(node.getLeft(), data))
21                  return true;
22          if (node.getRight() != null)
23              if (search(node.getRight(), data))
24                  return true;
25          return false;
26      }
```

4) Tambahkan tiga metode untuk melakukan traversal pohon biner dalam urutan yang berbeda, yaitu in-order, pre-order, dan post-order. Setiap metode ini akan memanggil fungsi printInorder, printPreorder, dan printPostorder yang telah dibuat di dalam class Node, dengan parameter berupa akar dari pohon.

```
28      public void printInorder() {
29          root.printInorder(root);
30      }
31
32      public void printPreOrder() {
33          root.printPreorder(root);
34      }
35
36      public void printPostOrder() {
37          root.printPostorder(root);
38      }
```

5) Buat metode getRoot() untuk mengambil referensi dari akar pohon, dan isEmpty() untuk mengecek apakah pohon dalam keadaan kosong atau tidak.

```
42      public Node getRoot() {
43          return root;
44      }
45
46      public boolean isEmpty() {
47          return root == null;
48      }
```

6) Untuk menghitung jumlah simpul dalam pohon, buatlah metode countNodes() yang akan memanggil metode rekursif countNodes(Node node), di mana fungsi ini akan menghitung total node yang ada dalam struktur pohon.

```
52      public int countNodes() {
53          return countNodes(root);
54      }
55
56      private int countNodes(Node node) {
57          int count = 1;
58          if (node == null) {
59              return 0;
60          } else {
61              count += countNodes(node.getLeft());
62              count += countNodes(node.getRight());
63              return count;
64          }
65      }
```

7) Tambahkan juga metode print() yang bertugas untuk mencetak atau menampilkan struktur pohon biner.

```
67      public void print() {
68          root.print();
69      }
```

8) Buat metode tambahan untuk mengakses dan mengatur nilai dari currentNode, yang merupakan simpul aktif saat ini dalam pohon.

```
72      public Node getCurrent() {
73          return currentNode;
74      }
75
76      public void setCurrent(Node node) {
77          this.currentNode = node;
78      }
```

9) Tambahkan fungsi untuk mengatur nilai root pohon berdasarkan simpul yang diberikan.

```
81    public void setRoot(Node root) {
82        this.root = root;
83    }
84 }
```

c) GraphTraversal

1) Deklarasikan class GraphTraversal, yang berfungsi sebagai struktur data untuk merepresentasikan graf. Graf ini menggunakan sebuah objek Map, di mana setiap key merepresentasikan nama simpul (node), dan setiap value berupa List<String> yang berisi daftar simpul lain yang terhubung langsung dengan simpul tersebut.

```
1 package Pekan9;
2 import java.util.*;
3
4 public class GraphTraversal {
5     private Map<String, List<String>> graph = new HashMap<>();
```

2) Tambahkan metode addEdge untuk menambahkan sisi (edge) pada graf. Metode ini menghubungkan dua simpul, yaitu node1 dan node2. Karena graf yang digunakan bersifat tak berarah, maka hubungan antar simpul ditambahkan ke kedua arah.

```
7     // Menambahkan edge (graf tak berarah)
8     public void addEdge(String node1, String node2) {
9         graph.putIfAbsent(node1, new ArrayList<>());
10        graph.putIfAbsent(node2, new ArrayList<>());
11        graph.get(node1).add(node2);
12        graph.get(node2).add(node1);
13    }
```

3) Buat metode untuk menampilkan graf dalam bentuk Adjacency List. Metode ini akan melakukan iterasi terhadap setiap simpul pada graf (graph.keySet()), kemudian mencetak nama simpul beserta daftar simpul yang terhubung dengannya.

```
15    // Menampilkan graf awal
16    public void printGraph() {
17        System.out.println("Graf Awal (Adjacency List):");
18        for (String node : graph.keySet()) {
19            System.out.print(node + " -> ");
20            List<String> neighbors = graph.get(node);
21            System.out.println(String.join(", ", neighbors));
22        }
23        System.out.println();
24    }
```

4) Implementasikan algoritma penelusuran Depth-First Search (DFS) untuk menjelajahi simpul-simpul dalam graf secara mendalam.

```
    // 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) Implementasikan juga penelusuran Breadth-First Search (BFS) untuk menjelajahi simpul-simpul graf secara melebar dari simpul awal.

```
    // 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();
    }
```

6) Buat method main() sebagai titik awal program, tempat di mana graf dibentuk, simpul dan sisi ditambahkan, serta penelusuran dijalankan.

```
    // 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");
    }
}
```

7) Jalankan program dan perhatikan hasil output yang dihasilkan untuk memastikan graf dan algoritma penelusuran bekerja sebagaimana mestinya.

```
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
```

d) TreeMain

1) Deklarasikan class TreeMain sebagai class utama yang akan menjalankan program pohon biner.

```
1 package Pekan9;
2
3 public class TreeMain {
```

2) Buat objek pohon biner dan lakukan perhitungan jumlah simpul (node) yang ada di dalamnya.

```
5         //Membuat Pohon
6         BTree tree = new BTree();
7         System.out.print("Jumlah Simpul awal pohon: ");
8         System.out.println(tree.countNodes());
```

3) Tambahkan simpul akar (root) ke dalam pohon sebagai simpul pertama.

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

4) Buat simpul-simpul lainnya dan tambahkan ke dalam struktur pohon.

```
18            Node node2 = new Node(2);
19            Node node3 = new Node(3);
20            Node node4 = new Node(4);
21            Node node5 = new Node(5);
22            Node node6 = new Node(6);
23            Node node7 = new Node(7);
```

5) Tambahkan simpul sebagai anak dari akar maupun simpul lain di dalam pohon sesuai dengan strukturnya.

```
25          root.setLeft(node2);
26          node2.setLeft(node4);
27          node2.setRight(node5);
28          node3.setLeft(node6);
29          root.setRight(node3);
30          node3.setRight(node7);
```

6) Atur simpul yang sedang diproses menggunakan tree.setCurrent(tree.getRoot()), untuk menjadikan akar sebagai simpul aktif. Gunakan tree.getCurrent().getData() untuk menampilkan data dari simpul yang sedang aktif, yaitu simpul akar (dalam contoh bernilai 1).

```
32          //set root
33          tree.setCurrent(tree.getRoot());
34          System.out.println("menampilkan simpul terakhir: ");
35          System.out.println(tree.getCurrent().getData());
```

7) Setelah penambahan simpul-simpul baru, tampilkan kembali jumlah total simpul yang ada di dalam pohon.

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

8) Cetak hasil traversal dari pohon, baik preorder, inorder, maupun postorder.

```
41          System.out.println("InOrder: ");
42          tree.printInorder();
43          System.out.println("\nPreorder: ");
44          tree.printPreOrder();
45          System.out.println("\nPostorder: ");
46          tree.printPostOrder();
```

9) Tampilkan struktur pohon dalam bentuk visual agar susunan simpul dan relasinya dapat terlihat lebih jelas.

```
48          System.out.println("\nDmenampilkan simpul dalam bentuk pohon");
49          tree.print();
50      }
51  }
```

10) Jalankan program dan perhatikan output yang ditampilkan untuk memastikan bahwa semua fungsi berjalan dengan benar.

```
<terminated> TreeMain [Java Application] C:\Program Files\Java
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
```

D. Kesimpulan

Berdasarkan pembahasan mengenai struktur data pohon biner dan graf, dapat disimpulkan bahwa keduanya memiliki peranan yang sangat penting dalam dunia pemrograman komputer, khususnya dalam hal pengelolaan serta pencarian data secara efisien. Pohon biner menyediakan pendekatan yang efektif untuk menyusun dan menelusuri data dalam bentuk hierarki melalui operasi dasar seperti traversal inorder, preorder, dan postorder. Sementara itu, struktur graf memungkinkan representasi hubungan antar simpul, serta memfasilitasi penelusuran menggunakan algoritma Depth-First Search (DFS) dan Breadth-First Search (BFS).

Penerapan struktur data pohon dan graf menggunakan bahasa pemrograman Java memperlihatkan bagaimana teori dapat diubah menjadi praktik nyata dalam bentuk kode program. Dengan mengimplementasikan algoritma penelusuran dan pencarian pada kedua struktur ini, mahasiswa dapat memahami cara kerjanya dalam konteks permasalahan komputasi secara lebih mendalam.

Oleh karena itu, penguasaan terhadap konsep serta implementasi pohon dan graf merupakan kemampuan dasar yang sangat penting bagi mahasiswa di bidang Teknologi Informasi, terutama pada tahap awal perkuliahan. Pemahaman ini menjadi landasan penting dalam mengembangkan keterampilan pemrograman yang lebih kompleks di masa mendatang.