

Laporan Praktikum
Struktur Data



Disusun Oleh :
SASKIA ALIFAH (2411531002)

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

Departemen Informatika
Fakultas Teknologi Informasi
Universitas Andalas
Tahun 2025

A. TUJUAN PRAKTIKUM

1. Memahami konsep dasar dan struktur pohon biner, termasuk istilah node, root, leaf, serta hubungan antar node dalam pohon.
2. Mampu mengimplementasikan operasi-operasi dasar pada pohon biner, seperti penambahan node, traversal (inorder, preorder, postorder), penghitungan jumlah node, dan visualisasi struktur pohon di konsol.
3. Melatih kemampuan analisis dan debugging program pohon biner, termasuk melakukan penanganan error (error handling) agar program berjalan stabil dan efisien.

B. PENDAHULUAN

Struktur data merupakan materi penting dalam dunia pemrograman karena berfungsi sebagai dasar dalam pengelolaan data di berbagai aplikasi. Salah satu struktur data yang sering digunakan adalah pohon biner (binary tree). Pohon biner sangat berguna untuk menyimpan data secara hierarkis, seperti pada proses pencarian data, pengurutan, sistem basis data, dan kompresi file. Dengan menggunakan pohon biner, data dapat diatur sehingga proses pencarian dan pengambilan data menjadi lebih efisien.

Pohon biner terdiri dari node-node yang saling terhubung, di mana setiap node maksimal memiliki dua anak, yaitu anak kiri dan anak kanan. Konsep ini memungkinkan pengembangan pohon biner menjadi berbagai jenis pohon khusus seperti Binary Search Tree (BST), AVL Tree, dan Heap, yang masing-masing memiliki kelebihan dan kegunaan tersendiri. Proses traversal pada pohon biner sangat penting karena menentukan cara data diakses dan diolah, baik secara rekursif maupun iteratif.

Praktikum ini membahas cara membangun dan memanipulasi pohon biner menggunakan bahasa pemrograman Java. Kegiatan praktikum meliputi pembuatan kelas Node, BTree, dan TreeMain, yang digunakan untuk membuat struktur pohon, melakukan traversal dengan berbagai metode (inorder, preorder, postorder), menghitung jumlah node, serta menampilkan bentuk pohon secara visual di konsol. Praktikum ini memberikan pemahaman yang mendalam tentang penerapan struktur data non-linear dan pentingnya pohon biner dalam pengembangan aplikasi yang lebih kompleks.

C. METODE PRAKTIKUM

1. Kelas NODE

a) Deskripsi Kelas

Kelas Node merupakan representasi fundamental dari simpul (node) dalam struktur data pohon biner. Setiap objek Node menyimpan satu nilai data bertipe integer, serta dua referensi yang menunjuk ke simpul anak kiri (left) dan anak kanan (right). Dengan desain ini, kelas Node membentuk dasar bagi pembentukan struktur pohon biner, baik yang bersifat seimbang (balanced) maupun tidak seimbang (unbalanced), serta dapat digunakan untuk berbagai aplikasi seperti binary search tree (BST), expression tree, decision tree, dan lain-lain.

Selain menyimpan data dan referensi anak, kelas ini menyediakan berbagai metode untuk mengatur (setter) dan mengambil (getter) data maupun referensi anak, serta metode traversal pohon (preorder, inorder, postorder) yang penting untuk berbagai operasi pohon. Kelas ini juga dilengkapi dengan metode visualisasi struktur pohon secara hierarkis melalui konsol, sehingga memudahkan proses debugging dan pemahaman bentuk pohon.

b) Penjelasan Fungsi dan Cara Kerja

- i. **Konstruktor Node(int data)**
Konstruktor ini digunakan untuk membuat objek node baru dengan nilai data tertentu. Saat objek dibuat, referensi anak kiri dan kanan secara default diinisialisasi dengan `null`, menandakan bahwa node tersebut belum memiliki anak.
- ii. **Metode setLeft(Node node) dan setRight(Node node)**
Kedua metode ini digunakan untuk menetapkan anak kiri dan kanan pada node. Penetapan hanya dilakukan jika posisi anak masih kosong (`null`). Dengan demikian, metode ini mencegah penimpaan data anak yang sudah ada, sehingga menjaga integritas struktur pohon.
- iii. **Metode getLeft(), getRight(), dan getData()**
Metode-metode ini berfungsi sebagai getter untuk mengambil referensi anak kiri, anak kanan, dan nilai data pada node. Hal ini penting untuk navigasi dan manipulasi pohon secara rekursif maupun iteratif.
- iv. **Metode setData(int data)**
Metode ini memungkinkan perubahan nilai data pada node, sehingga node dapat digunakan kembali tanpa harus membuat objek baru.
- v. **Metode Traversal**
Traversal adalah proses mengunjungi setiap node dalam pohon dengan urutan tertentu. Kelas `Node` menyediakan tiga jenis traversal utama:
printPreorder(Node node)
Melakukan traversal preorder, yaitu mengunjungi node saat ini terlebih

dahulu, kemudian anak kiri, lalu anak kanan. Traversal ini banyak digunakan untuk menyalin pohon atau evaluasi ekspresi.

printInorder(Node node)

Traversal inorder mengunjungi anak kiri terlebih dahulu, kemudian node saat ini, lalu anak kanan. Pada binary search tree, traversal ini akan menghasilkan urutan data yang terurut secara menaik (ascending).

printPostorder(Node node)

Traversal postorder mengunjungi anak kiri, anak kanan, baru kemudian node saat ini. Traversal ini berguna untuk menghapus pohon atau evaluasi postfix expression tree.

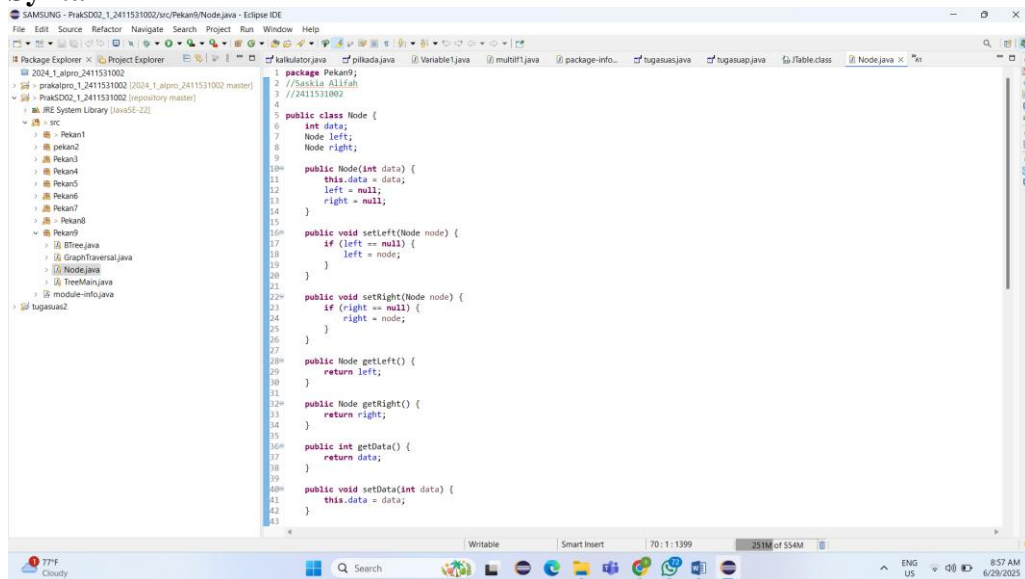
Setiap metode traversal akan mencetak hasil kunjungan node ke layar menggunakan System.out.print.

vi. Metode Visualisasi Struktur Pohon

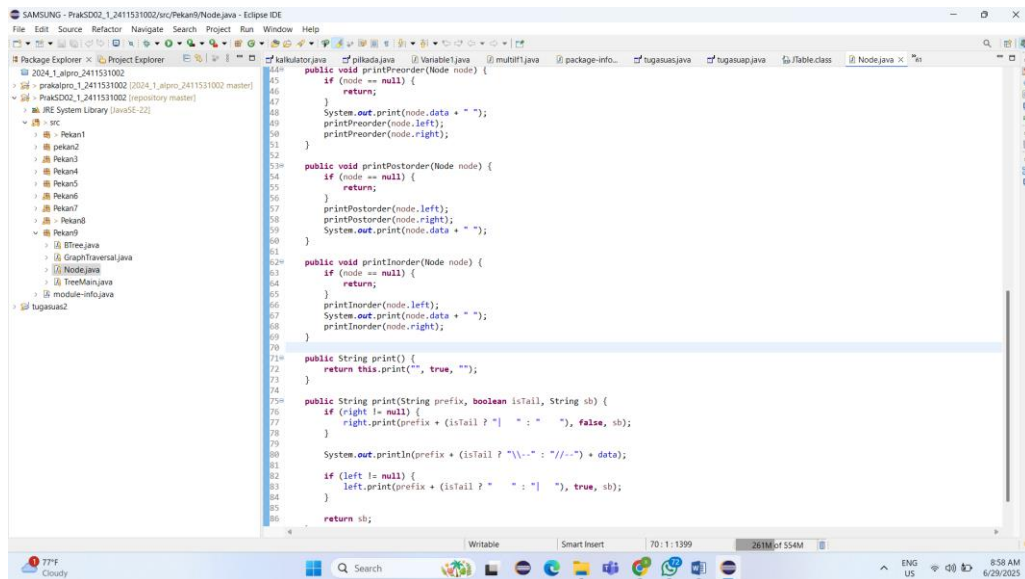
print() dan print(String prefix, boolean isTail, String sb)

Metode ini digunakan untuk mencetak struktur pohon secara hierarkis di konsol. Visualisasi ini sangat membantu untuk memahami bentuk pohon, posisi simpul, dan relasi antar node, terutama saat debugging atau analisis struktur data.

c) Syntax



```
1 package Pekan9;
2 //Saskia Alifrah
3 //2411531002
4
5 public class Node {
6     int data;
7     Node left;
8     Node right;
9
10    public Node(int data) {
11        this.data = data;
12        left = null;
13        right = null;
14    }
15
16    public void setLeft(Node node) {
17        if (left == null) {
18            left = node;
19        }
20    }
21
22    public void setRight(Node node) {
23        if (right == null) {
24            right = node;
25        }
26    }
27
28    public Node getLeft() {
29        return left;
30    }
31
32    public Node getRight() {
33        return right;
34    }
35
36    public int getData() {
37        return data;
38    }
39
40    public void setData(int data) {
41        this.data = data;
42    }
43 }
```



```
44: public void printPreorder(Node node) {
45:     if (node == null) {
46:         return;
47:     }
48:     System.out.print(node.data + " ");
49:     printPreorder(node.left);
50:     printPreorder(node.right);
51: }
52:
53: public void printPostorder(Node node) {
54:     if (node == null) {
55:         return;
56:     }
57:     printPostorder(node.left);
58:     printPostorder(node.right);
59:     System.out.print(node.data + " ");
60: }
61:
62: public void printInorder(Node node) {
63:     if (node == null) {
64:         return;
65:     }
66:     printInorder(node.left);
67:     System.out.print(node.data + " ");
68:     printInorder(node.right);
69: }
70:
71: public String print() {
72:     return this.print("", true, "");
73: }
74:
75: public String print(String prefix, boolean isTail, String sb) {
76:     if (right != null) {
77:         right.print(prefix + (isTail ? " " : " ") + " ", false, sb);
78:     }
79:     System.out.println(prefix + (isTail ? " \\n- " : "//--") + data);
80:
81:     if (left != null) {
82:         left.print(prefix + (isTail ? " " : " ") + " ", true, sb);
83:     }
84:
85:     return sb;
86: }
```

d) Alur Sintaks Program

- **Inisialisasi Node**
Objek node dibuat menggunakan konstruktor, misal: `Node root = new Node(5);`
- **Membangun Pohon**
Mengatur anak kiri dan kanan menggunakan `setLeft()` dan `setRight()`.
- **Traversal Pohon**
Memanggil metode traversal untuk mencetak urutan kunjungan node, misal: `root.printPreorder(root);`
- **Visualisasi**
Memanggil `root.print()` untuk menampilkan bentuk pohon secara visual di konsol.

e) Error Handling

- Metode setter anak (`setLeft`, `setRight`) hanya akan mengatur anak jika posisi masih kosong, sehingga mencegah penimpaan node yang sudah ada.
- Traversal rekursif selalu melakukan pengecekan `if (node == null)` untuk mencegah error stack overflow akibat referensi null.

2. Kelas BTree

a) Deskripsi Kelas

Kelas BTree adalah implementasi dari struktur data pohon biner (binary tree) yang berfungsi sebagai wadah utama untuk mengelola dan memanipulasi simpul-simpul (node) yang direpresentasikan oleh kelas Node. Kelas ini bertanggung jawab atas operasi-operasi utama pada pohon, seperti pencarian data, traversal (penelusuran), perhitungan jumlah node, serta pengelolaan root dan node aktif (current node). Dengan desain ini, BTree menjadi fondasi bagi berbagai aplikasi yang membutuhkan struktur pohon, seperti pencarian, pengurutan, ekspresi matematika, dan representasi hierarki data.

b) Penjelasan Fungsi dan Cara Kerja

i. Konstruktor BTree()

Konstruktor ini menginisialisasi pohon dengan root bernilai `null`, menandakan bahwa pohon masih kosong saat pertama kali dibuat.

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

ii. Metode search(int data)

Metode ini digunakan untuk mencari apakah suatu nilai (data) terdapat dalam pohon. Proses pencarian dilakukan secara rekursif mulai dari root, dengan membandingkan nilai pada setiap node. Jika ditemukan, mengembalikan `true`; jika tidak, mengembalikan `false`. Pencarian dilakukan dengan menelusuri seluruh node (traversal), sehingga cocok untuk pohon biner umum (bukan hanya BST).

java

```
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;  
    }  
    return search(node.getLeft(), data) || search(node.getRight(), data);  
}
```

iii. Metode Traversal

Traversal adalah proses mengunjungi setiap node dalam pohon dengan urutan tertentu. Kelas BTree menyediakan tiga jenis traversal utama:

- **printInorder()**

Traversal inorder mengunjungi anak kiri, node saat ini, lalu anak kanan. Pada binary search tree, traversal ini akan menghasilkan urutan data menaik (ascending).

java

```
public void printInorder() {  
    if (root != null) {  
        root.printInorder(root);  
    }  
}
```

- **printPreOrder()**

Traversal preorder mengunjungi node saat ini, lalu anak kiri, kemudian anak kanan.

java

```
public void printPreOrder() {  
    if (root != null) {  
        root.printPreorder(root);  
    }  
}
```

- **printPostOrder()**

Traversal postorder mengunjungi anak kiri, anak kanan, lalu node saat ini.

```

public void printPostOrder() {
    if (root != null) {
        root.printPostorder(root);
    }
}

```

iv. **Metode Pengelolaan Node**

- **getRoot() dan setRoot(Node root)**

Digunakan untuk mengambil dan mengatur node root pohon.

```

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

```

- **getCurrent() dan setCurrent(Node node)**

Digunakan untuk mengambil dan mengatur node yang sedang aktif (current node), misal untuk operasi interaktif atau visualisasi.

```

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

```

- **isEmpty()**

Mengecek apakah pohon kosong (root == null).

```

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

```

v. **Metode countNodes()**

Menghitung jumlah node dalam pohon secara rekursif, dimulai dari root. Jika node bernilai null, mengembalikan 0; jika tidak, menghitung node saat ini ditambah jumlah node di anak kiri dan kanan.

```

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

```

vi. **Metode Visualisasi**

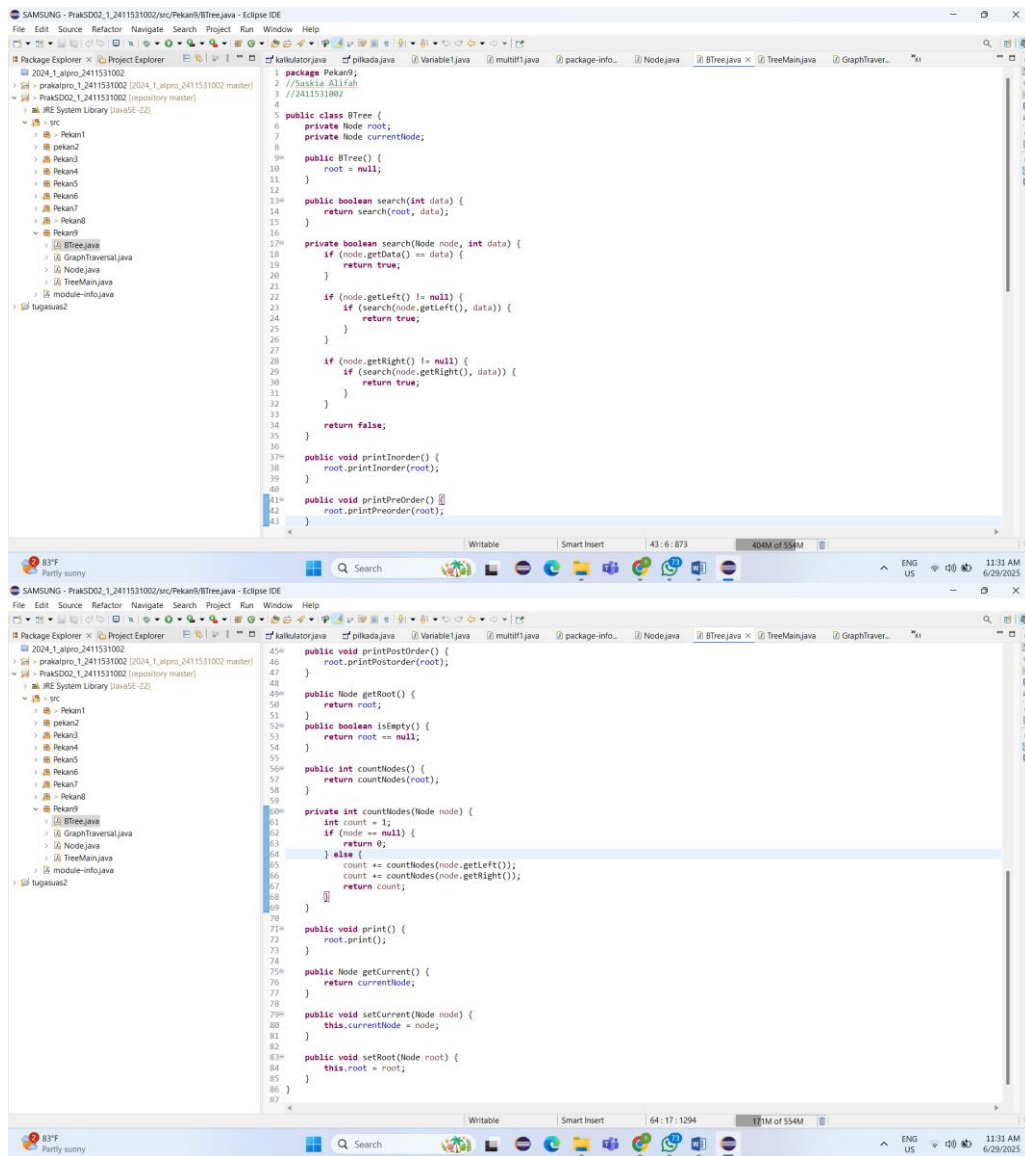
Metode ini digunakan untuk mencetak struktur pohon secara hierarkis di konsol, sehingga bentuk pohon dapat dilihat secara visual.

```

public void print() {
    if (root != null) {
        root.print();
    }
}

```

c) **Syntax:**



d) Alur Program

- Program dimulai, objek pohon biner (BTree) dibuat kosong.
- Jumlah simpul awal pohon dicek dan ditampilkan (seharusnya 0).
- Node root dibuat dan di-set sebagai root pohon.
- Jumlah simpul setelah penambahan root dicek dan ditampilkan (seharusnya 1).
- Node-node anak (2–7) dibuat satu per satu.
- Struktur pohon dibangun dengan menghubungkan node-node menggunakan setLeft() dan setRight().
- Current node di-set ke root, lalu data current node ditampilkan.
- Jumlah simpul setelah pohon lengkap dicek dan ditampilkan (seharusnya 7).
- Traversal pohon dilakukan dengan tiga metode:
InOrder (kiri, root, kanan)
PreOrder (root, kiri, kanan)
PostOrder (kiri, kanan, root)
- Hasil traversal dicetak ke layar.
- Struktur pohon divisualisasikan secara hierarkis di konsol.

- Identitas mahasiswa (nama dan NIM) dicetak di akhir program.

3. Kelas TreeMain

a) Deskripsi Kelas

Kelas TreeMain merupakan kelas utama (main class) dalam package Pekan9 yang berfungsi untuk mendemonstrasikan pembuatan, manipulasi, dan penelusuran struktur data pohon biner menggunakan kelas BTree dan Node. Melalui kelas ini, mahasiswa dapat mempraktikkan secara langsung konsep pohon biner, mulai dari penambahan simpul (node), penghitungan jumlah simpul, traversal (penelusuran) dengan tiga metode (inorder, preorder, postorder), hingga visualisasi struktur pohon secara hierarkis di konsol.

b) Penjelasan Fungsi dan Cara Kerja

i. Package dan Deklarasi Kelas

Program ini berada dalam package Pekan9, yang menandakan file ini merupakan bagian dari modul praktikum ke-9. Penggunaan package bertujuan untuk mengelompokkan file Java sesuai topik atau minggu praktikum, sehingga struktur project menjadi lebih rapi dan maintainable.

package Pekan9;

Kelas utama dideklarasikan sebagai berikut:

```
public class TreeMain {
    public static void main(String[] args) {
        // ...
    }
}
```

Kelas ini berperan sebagai entry point program, di mana seluruh proses pembuatan, pengisian, dan penelusuran pohon biner dijalankan secara berurutan.

ii. Inisialisasi dan Pembuatan Pohon

• Membuat objek pohon biner kosong

Langkah awal adalah membuat objek BTree yang akan menampung struktur pohon. Pada tahap ini, pohon masih kosong sehingga jumlah simpul adalah 0. Hal ini dapat digunakan untuk menguji fungsi countNodes() pada kondisi pohon kosong.

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

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

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

Output: 0

- **Membuat dan menetapkan root**

Node dengan data 1 dibuat dan di-set sebagai root pohon. Setelah root di-set, jumlah simpul menjadi 1. Ini membuktikan bahwa penambahan root berjalan dengan benar.

```
Node root = new Node(1);
tree.setRoot(root);
System.out.println("Jumlah simpul jika hanya ada root:");
System.out.println(tree.countNodes());
Output: 1
```

iii. Penambahan Simpul dan Penyusunan Pohon

- **Membuat simpul-simpul lain**

Simpul dengan data 2 hingga 7 dibuat satu per satu. Setiap node dideklarasikan secara eksplisit untuk memudahkan pengelolaan referensi dan penyusunan struktur pohon secara manual.

```
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);
```

- **Menyusun struktur pohon secara manual**

Setiap node dihubungkan sesuai urutan untuk membentuk pohon biner penuh. Penempatan node dilakukan dengan metode setLeft() dan setRight(), sehingga struktur pohon dapat dikontrol dengan jelas dan mudah dievaluasi.

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

Struktur pohon yang terbentuk:

```

  1
 / \
2   3
/ \ / \
4 5 6 7
```

Penyusunan manual seperti ini sangat berguna untuk eksperimen, debugging, dan demonstrasi prinsip dasar pohon biner.

iv. Pengaturan dan Akses Current Node

Mengatur current node ke root

Fitur current node (currentNode) digunakan untuk menandai simpul yang sedang aktif atau difokuskan, misal untuk operasi traversal interaktif, pengeditan, atau visualisasi lanjutan. Pada tahap ini, current node di-set ke root, lalu data current node ditampilkan.

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

```
System.out.println("Menampilkan simpul terakhir:");
```

```
System.out.println(tree.getCurrent().getData());
```

Output: 1

(karena current node di-set ke root)

v. Penghitungan Jumlah Simpul

Menampilkan jumlah simpul setelah pohon lengkap

Setelah seluruh simpul ditambahkan, jumlah simpul dihitung kembali menggunakan fungsi rekursif. Hal ini penting untuk memastikan seluruh node sudah terhubung dengan benar.

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

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

Output: 7

vi. Traversal Pohon

Traversal adalah proses mengunjungi seluruh simpul dalam pohon dengan urutan tertentu. Tiga metode traversal utama didemonstrasikan:

- **InOrder Traversal (kiri, root, kanan):**

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

```
tree.printInorder();
```

Output: 4 2 5 1 6 3 7

(menunjukkan urutan nilai secara ascending untuk pohon biner penuh)

- **PreOrder Traversal (root, kiri, kanan):**

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

```
tree.printPreOrder();
```

Output: 1 2 4 5 3 6 7

(mengunjungi root lebih dulu, lalu seluruh subtree kiri dan kanan)

- **PostOrder Traversal (kiri, kanan, root):**

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

```
tree.printPostOrder();
```

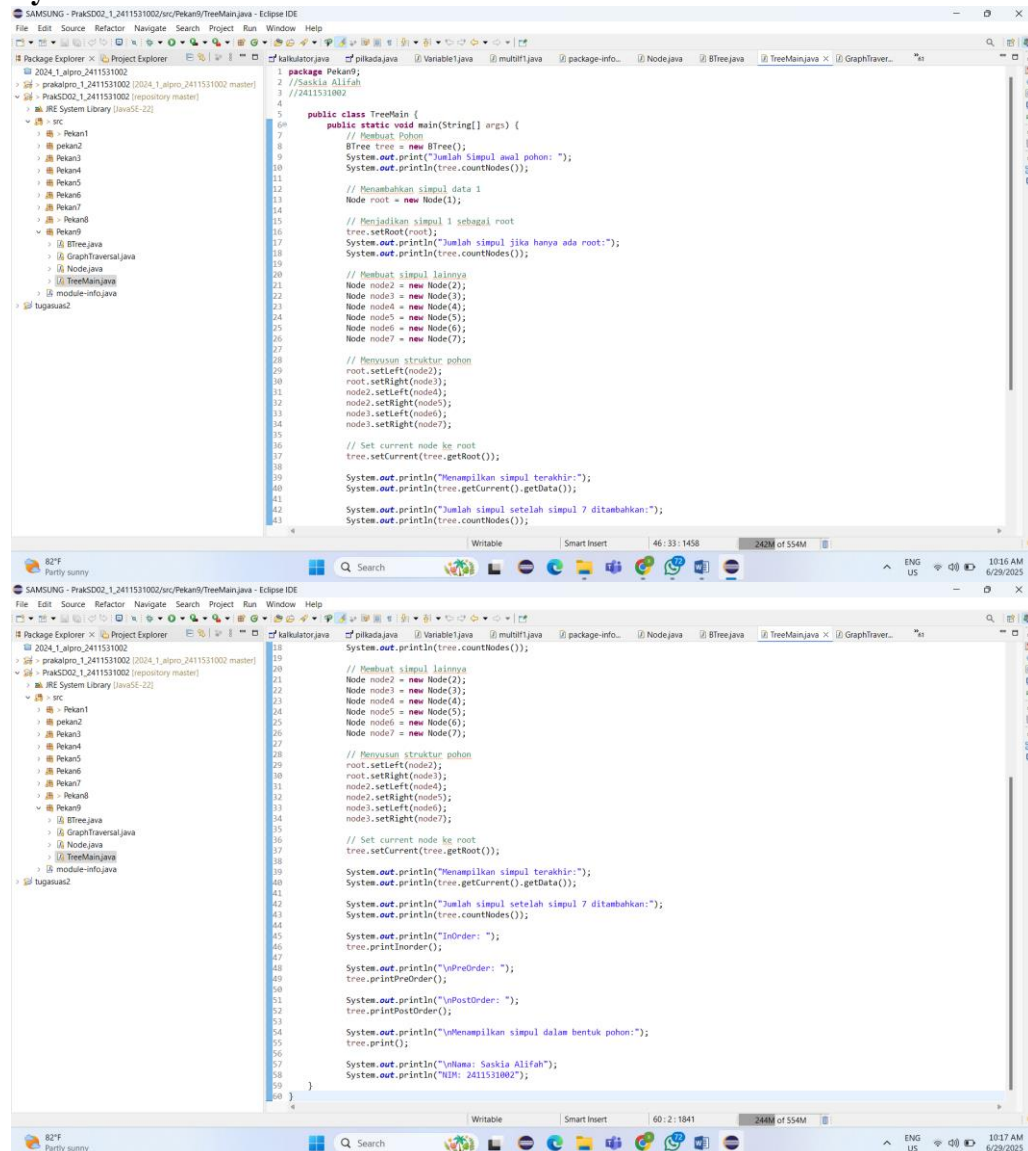
Output: 4 5 2 6 7 3 1

(berguna untuk operasi penghapusan node atau evaluasi ekspresi postfix)

vii. Visualisasi Struktur Pohon

Menampilkan struktur pohon secara visual

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



- **Mulai**
Program dimulai dari fungsi main().
- **Inisialisasi objek BTree**
Membuat objek pohon biner kosong.
- **Hitung & tampilkan jumlah simpul awal pohon**
Mengecek jumlah node pada pohon yang masih kosong (hasilnya 0).

- **Buat node root**
Membuat node dengan data 1.
- **Set root ke pohon**
Menetapkan node tersebut sebagai root pohon.
- **Hitung & tampilkan jumlah simpul (hanya root)**
Jumlah node sekarang 1.
- **Buat node 2-7**
Membuat node dengan data 2 sampai 7.
- **Susun struktur pohon**
Menghubungkan node-node tersebut ke root dan ke anak-anaknya membentuk pohon biner penuh.
- **Set currentNode ke root**
Menetapkan currentNode ke root.
- **Tampilkan data currentNode**
Menampilkan data pada currentNode (yaitu root).
- **Hitung & tampilkan jumlah simpul setelah lengkap**
Jumlah node sekarang menjadi 7.
- **Traversal InOrder**
Menampilkan urutan node dengan traversal inorder.
- **Traversal PreOrder**
Menampilkan urutan node dengan traversal preorder.
- **Traversal PostOrder**
Menampilkan urutan node dengan traversal postorder.
- **Visualisasi struktur pohon**
Menampilkan bentuk pohon secara visual di konsol.
- **Tampilkan identitas**
Menampilkan nama dan NIM mahasiswa.
- **Selesai**
Program berakhir.

e) Error Handling

- **Null Checking pada Traversal dan Penghitungan Node**
Setiap metode traversal (printInorder, printPreOrder, printPostOrder) dan penghitungan node (countNodes) selalu melakukan pengecekan apakah node yang sedang diproses adalah null.
Contoh:

```
if (node == null) {  
    return 0; // atau return; pada traversal  
}
```


Hal ini mencegah terjadinya NullPointerException saat pohon masih kosong atau saat mencapai daun pohon.
- **Pengaturan Node secara Eksplisit**
Penambahan node dilakukan secara eksplisit dan tidak otomatis, sehingga tidak ada risiko node tertimpa secara tidak sengaja.
Jika ingin menambah atau menghapus node, programmer harus memodifikasi kode secara manual, sehingga struktur pohon tetap terkontrol.
- **Akses Current Node**

Sebelum mengakses data pada current node, program memastikan current node sudah di-set (misal ke root) sehingga tidak terjadi error saat pemanggilan getData().

- **Tidak Ada Input dari User**

Karena semua data node di-hardcode, tidak ada risiko input tidak valid dari user. Namun, jika program dikembangkan untuk menerima input dari user, perlu ditambahkan validasi input agar tidak terjadi error parsing.

f) Alur Sintaks Program

- Program dijalankan, objek pohon biner dibuat kosong.
- Node root dibuat dan di-set sebagai root pohon.
- Node-node lain dibuat dan dihubungkan ke root dan anak-anaknya.
- Current node diatur ke root, lalu data current node ditampilkan.
- Jumlah simpul dicek dan ditampilkan sebelum dan sesudah pohon lengkap.
- Traversal pohon dilakukan dengan tiga metode: inorder, preorder, dan postorder.
- Struktur pohon divisualisasikan di konsol.
- Identitas mahasiswa dicetak di akhir program.

g) Output Program

```
<terminated> TreeMain [Java Application] C:\Users\SAMSUNG\
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

Nama: Saskia Alifah
NIM: 2411531002
```

Setelah program dijalankan, berikut adalah penjelasan dan interpretasi hasil output yang dihasilkan oleh setiap langkah pada kelas TreeMain:

i. Jumlah Simpul Awal Pohon

Jumlah Simpul awal pohon: 0

Penjelasan:

Pada awal program, objek pohon BTree masih kosong (belum ada node yang dimasukkan sebagai root), sehingga fungsi countNodes() mengembalikan nilai 0. Ini menandakan pohon benar-benar baru dibuat dan belum berisi data apapun.

ii. Jumlah Simpul Setelah Root Ditambahkan

Jumlah simpul jika hanya ada root:

1

Penjelasan:

Setelah node dengan data 1 dibuat dan di-set sebagai root, fungsi countNodes() kini mengembalikan nilai 1. Ini menunjukkan bahwa pohon telah memiliki satu simpul utama (root) dan belum ada anak/cabang lain.

iii. Menampilkan Current Node

Menampilkan simpul terakhir:

1

Penjelasan:

Current node di-set ke root, sehingga saat dipanggil getCurrent().getData(), yang tampil adalah data pada root, yaitu 1. Ini berguna untuk memastikan current node sudah terhubung ke root dengan benar.

iv. Jumlah Simpul Setelah Pohon Lengkap

Jumlah simpul setelah simpul 7 ditambahkan:

7

Penjelasan:

Setelah seluruh node (2 sampai 7) dihubungkan ke pohon, fungsi countNodes() menghitung semua node yang terhubung dari root, menghasilkan total 7. Ini menandakan seluruh struktur pohon telah terbentuk sempurna.

v. Traversal Pohon

• InOrder Traversal

InOrder:

4 2 5 1 6 3 7

Penjelasan:

Traversal inorder mengunjungi node secara urut: anak kiri, root, anak kanan.

Untuk pohon ini, hasilnya adalah 4 2 5 1 6 3 7. Traversal ini sering digunakan untuk mendapatkan data terurut dari pohon binary search tree (BST), meskipun pohon ini bukan BST.

• PreOrder Traversal

PreOrder:

1 2 4 5 3 6 7

Penjelasan:

Traversal preorder mengunjungi node: root, anak kiri, anak kanan. Hasilnya adalah 1 2 4 5 3 6 7, sesuai dengan urutan penambahan node ke dalam pohon.

• PostOrder Traversal

PostOrder:

4 5 2 6 7 3 1

Penjelasan:

Traversal postorder mengunjungi node: anak kiri, anak kanan, root. Hasilnya adalah 4 5 2 6 7 3 1. Traversal ini sering digunakan untuk operasi penghapusan pohon atau evaluasi ekspresi postfix.

vi. Visualisasi Struktur Pohon

Menampilkan simpul dalam bentuk pohon:

```
--1
 |  --3
 |   |  --7
 |   |   --6
 --2
 |   --5
   --4
```

Penjelasan:

Visualisasi ini memberikan gambaran hierarki pohon secara jelas. Simbol -- menunjukkan node utama, sedangkan | -- menunjukkan cabang ke anak kanan atau kiri. Dengan visualisasi ini, mahasiswa dapat melihat struktur dan relasi antar node secara intuitif, yang sangat membantu dalam memahami konsep pohon biner.

vii. Identitas Mahasiswa

Nama: Saskia Alifah

NIM: 2411531002

Penjelasan:

Bagian ini menampilkan identitas mahasiswa sebagai penanda kepemilikan hasil praktikum.

Kesimpulan Output:

Setelah seluruh proses dijalankan, program berhasil membangun dan menampilkan struktur pohon biner lengkap. Setiap langkah output memberikan insight tentang status pohon:

Jumlah simpul menunjukkan keberhasilan penambahan node.

Traversal memperlihatkan urutan kunjungan node yang berbeda sesuai metode yang digunakan.

Visualisasi membantu memahami bentuk dan relasi pohon secara hierarkis.

Identitas memastikan hasil praktikum terdokumentasi jelas.

Dengan demikian, output program ini sangat efektif untuk mendemonstrasikan konsep dasar pohon biner, traversal, dan visualisasi struktur data di Java, serta dapat dijadikan referensi dalam pembelajaran dan pengembangan aplikasi berbasis pohon di masa mendatang.

4. Kelas GraphTraversal

a) Deskripsi Kelas

Kelas GraphTraversal adalah implementasi struktur data graf tak berarah menggunakan adjacency list berbasis HashMap di Java. Kelas ini menyediakan operasi penambahan edge (sisi), pencetakan graf, serta dua metode penelusuran graf yang sangat penting dalam ilmu komputer, yaitu DFS (Depth-First Search)

dan BFS (Breadth-First Search). Penelusuran dilakukan mulai dari node tertentu, dan hasil urutan kunjungan node dicetak ke layar. Kelas ini sangat bermanfaat untuk memahami konsep dasar graf, traversal, serta implementasi algoritma pencarian pada graf.

b) Penjelasan Fungsi dan Cara Kerja

i. Package dan Deklarasi Kelas

Program berada dalam package Pekan9, yang menandakan file ini merupakan bagian dari modul praktikum ke-9.

```
package Pekan9;  
import java.util.*;
```

Kelas utama dideklarasikan sebagai berikut:

```
public class GraphTraversal {  
    // ...  
}
```

ii. Inisialisasi Struktur Data Graf

Deklarasi Adjacency List:

Graf direpresentasikan dengan Map<String, List<String>> graph yang menyimpan setiap node beserta daftar tetangganya.

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

iii. Penambahan Edge (Sisi)

addEdge(String node1, String node2):

Menambahkan sisi antara dua node. Karena graf tak berarah, kedua node saling menambahkan satu sama lain ke daftar tetangganya.

```
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);  
}
```

iv. Mencetak Adjacency List

printGraph():

Menampilkan seluruh isi adjacency list graf ke layar.

```
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();
    }

```

v. **Penelusuran DFS (Depth-First Search)**

dfs(String start):

Melakukan penelusuran graf secara mendalam (rekursif), mencetak urutan kunjungan node.

```

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.getDefault(current, new ArrayList<>())) {
        dfsHelper(neighbor, visited);
    }
}

```

vi. **Penelusuran BFS (Breadth-First Search)**

bfs(String start):

Melakukan penelusuran graf secara melebar (iteratif dengan queue), mencetak urutan kunjungan node.

```

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.getDefault(current, new ArrayList<>())) {
            if (!visited.contains(neighbor)) {
                queue.add(neighbor);
                visited.add(neighbor);
            }
        }
    }
    System.out.println();
}

```

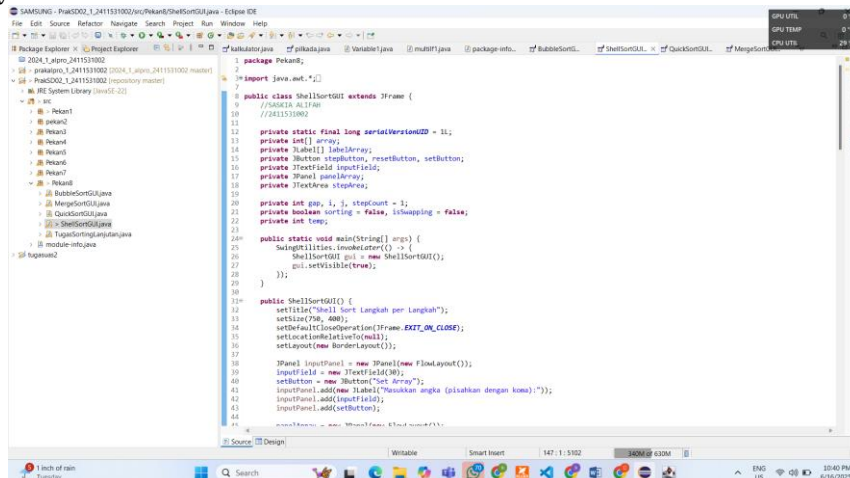
vii. Main Method

main(String[] args):

Menyusun graf contoh, mencetak adjacency list, lalu melakukan penelusuran DFS dan BFS 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");  
}
```

c) Syntax:



d) Diagram Alur

- **Mulai**
Program dimulai dari fungsi main().
- **Inisialisasi objek Graph**
Membuat objek GraphTraversal dan menyiapkan struktur data graf (HashMap).
- **Tambahkan edge**
Menambahkan sisi-sisi graf: A-B, A-C, B-D, B-E menggunakan method addEdge.

- **Cetak adjacency list graf**
Menampilkan struktur graf dalam bentuk adjacency list ke layar dengan method `printGraph`.
- **Lakukan DFS dari node A**
Melakukan penelusuran graf secara mendalam (Depth-First Search) mulai dari node "A" dengan method `dfs`.
- **Lakukan BFS dari node A**
Melakukan penelusuran graf secara melebar (Breadth-First Search) mulai dari node "A" dengan method `bfs`.
- **Selesai**
Program berakhir setelah seluruh proses traversal selesai.

e) **Error Handling**

- **Null Checking pada Traversal**
Pada traversal DFS dan BFS, jika node tidak memiliki tetangga (tidak ditemukan di graph), digunakan `getOrDefault(current, new ArrayList<>())` untuk mencegah `NullPointerException`.
- **Visited Set**
Setiap node yang sudah dikunjungi ditandai dalam `Set<String> visited` untuk mencegah infinite loop pada graf yang memiliki siklus.
- **Queue pada BFS**
BFS menggunakan queue dan pengecekan `visited` agar tidak terjadi pengulangan kunjungan node yang sama.
- **Input Hardcoded**
Karena node dan edge dimasukkan secara hardcoded di main, tidak ada risiko input tidak valid dari user. Namun, jika dikembangkan untuk menerima input user, perlu validasi nama node dan edge.

f) **Alur Sintaks Program**

- **Inisialisasi**
Membuat objek `GraphTraversal`.
- **Penambahan Edge**
Menambahkan edge A-B, A-C, B-D, B-E menggunakan `addEdge`.
- **Cetak Graf**
Menampilkan adjacency list graf dengan `printGraph`.
- **DFS Traversal**
Melakukan DFS dari node "A" dengan `dfs("A")`.
- **BFS Traversal**
Melakukan BFS dari node "A" dengan `bfs("A")`.
- **Program selesai**
Semua output traversal dan graf sudah dicetak ke layar.

a) **Output Program**

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

Setelah program dijalankan, berikut adalah penjelasan dari setiap bagian output yang dihasilkan oleh kelas GraphTraversal:

- **Adjacency List (Graf Awal)**

Graf Awal adalah:

Graf Awal (Adjacency List):

A -> B, C

B -> A, D, E

C -> A

D -> B

E -> B

Penjelasan:

Bagian ini menampilkan struktur graf dalam bentuk adjacency list.

Setiap baris menunjukkan node dan daftar tetangganya.

Misal, A -> B, C berarti node A terhubung ke B dan C.

Karena graf tak berarah, keterhubungan bersifat dua arah (misal, B juga mencantumkan A sebagai tetangga).

Struktur ini memudahkan visualisasi dan pemahaman relasi antar node sebelum dilakukan traversal.

- **Penelusuran DFS (Depth-First Search)**

Penelusuran DFS:

A B D E C

Penjelasan:

Penelusuran DFS dimulai dari node A.

Urutan kunjungan:

Mulai dari A, lanjut ke B (tetangga A), dari B ke D (tetangga B), kembali ke B, lalu ke E (tetangga B berikutnya), kembali ke A, lalu ke C (tetangga A yang belum dikunjungi).

DFS menelusuri satu cabang hingga paling dalam sebelum kembali dan menjelajah cabang lain.

Hasil traversal ini bisa berbeda tergantung urutan tetangga pada adjacency list, namun prinsipnya selalu "sedalam mungkin".

- **Penelusuran BFS (Breadth-First Search)**

Penelusuran BFS:

A B C D E

Penjelasan:

Penelusuran BFS juga dimulai dari node A.

Urutan kunjungan:

Mulai dari A, kunjungi semua tetangga langsung (B dan C).

Lanjutkan ke tetangga berikutnya dari B (D dan E).

BFS menelusuri node berdasarkan level/kedalaman dari node awal, sehingga semua node pada level yang sama dikunjungi lebih dulu sebelum lanjut ke level berikutnya.

Hasil traversal ini membantu memahami penyebaran/penjajakan node secara melebar.

- **Interpretasi dan Kesimpulan Output**

Adjacency List memperjelas struktur graf yang dibangun, sehingga mahasiswa dapat memastikan edge sudah benar.

DFS menunjukkan urutan kunjungan node secara mendalam, cocok untuk pencarian jalur, pengujian keterhubungan, atau eksplorasi graf secara rekursif.

BFS menunjukkan urutan kunjungan node secara melebar, sangat bermanfaat untuk pencarian jalur terpendek, penelusuran level, atau penyebaran informasi.

Output traversal memperlihatkan perbedaan nyata antara strategi DFS dan BFS, meskipun struktur graf sama.

Kesimpulan Output:

Program GraphTraversal berhasil membangun graf tak berarah, menampilkan struktur adjacency list, dan melakukan penelusuran DFS serta BFS dari node awal yang sama. Output traversal membuktikan bahwa urutan kunjungan node sangat tergantung pada strategi penelusuran yang digunakan, sehingga pemilihan metode harus disesuaikan dengan kebutuhan aplikasi graf di dunia nyata.

D. KESIMPULAN

Dari praktikum ini, konsep dasar pohon biner telah berhasil dipahami dan diimplementasikan, mulai dari pembuatan node, penyusunan struktur pohon, hingga operasi traversal dan visualisasi. Berbagai metode traversal seperti inorder, preorder, dan postorder memberikan gambaran nyata mengenai cara data dalam pohon biner diakses dan diproses secara efisien.

Selain itu, praktikum ini juga menekankan pentingnya penerapan error handling dan modularitas dalam pemrograman struktur data. Dengan membangun program yang terstruktur dan melakukan pengecekan error secara cermat, program dapat berjalan dengan stabil dan mudah dikembangkan. Pemahaman ini menjadi dasar yang kuat untuk menghadapi pengembangan aplikasi dan algoritma yang lebih kompleks di masa mendatang.