

LAPORAN PRAKTIKUM STRUKTUR DATA  
IMPLEMENTASI BINARY TREE DAN GRAPH TRAVERSAL  
PADA PEMROGRAMAN JAVA



Oleh :

DERIEL CHAERAHMAN

NIM 2411533007

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

ASISTEN PRAKTIKUM : RAHMAT DWIRIZKI OLDERS

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

## A. Pendahuluan

Struktur data merupakan komponen penting dalam pemrograman yang berfungsi dalam mengatur, menyimpan dan memproses data secara efisien. Praktikum ini dilakukan untuk mempelajari implementasi dua jenis struktur data non-linear yaitu Binary Tree dan Graph, yang digunakan secara luas dalam aplikasi komputasi seperti searching, pemrosesan, penelusuran (traversal) pada node dari struktur data.

### 1. Binary Tree

Merupakan struktur data non-linear/hierarki yang setiap nodenya hanya memiliki maksimal dua anak, yaitu *left child* dan *right child*. Binary tree sering digunakan untuk menyimpan data dalam struktur hirarki dan memungkinkan pencarian yang efisien. Traversal pada binary tree terbagi menjadi :

- Preorder : kunjungi node saat ini, lalu kiri, lalu kanan.
- Inorder : kunjungi kiri, node saat ini, lalu kanan.
- Postorder : kunjungi kiri, kanan, lalu node saat ini.

### 2. Graph Traversal

Graf (Graph) adalah struktur data yang terdiri dari simpul (node/vertex) dan sisi (edge) yang menghubungkan simpul-simpul tersebut. Graf dapat berbentuk terarah (directed) atau tak terarah (undirected), dan representasinya bisa melalui adjacency list atau adjacency matrix. Traversal graf adalah proses untuk mengunjungi seluruh simpul dalam graf. Dua metode populer yaitu :

- DFS (Depth-First Search): Menelusuri node secara mendalam sebelum kembali (backtrack).
- BFS (Breadth-First Search): Menelusuri semua tetangga pada satu level terlebih dahulu sebelum pindah ke level berikutnya.

### 3. Node dan BTree Class

Node adalah unit dasar dalam pohon yang menyimpan data serta referensi ke node lainnya. Dalam implementasi OOP, node biasanya direpresentasikan dalam class khusus (Node.java). Class BTree.java berfungsi untuk mengelola keseluruhan pohon, termasuk metode insert dan traversal.

## B. Tujuan

Tujuan dari dilakukannya praktikum ini adalah :

1. Memahami Binary Tree serta algoritma operasi traversalnya yaitu preorder, inorder dan postorder. .
2. Memahami Graph Traversal serta algoritma metode penelusuran DFS (Depth-First Search) dan BFS (Breadth-First Search).
3. Menggunakan konsep OOP (Object Oriented Programming).

## C. Langkah kerja praktikum

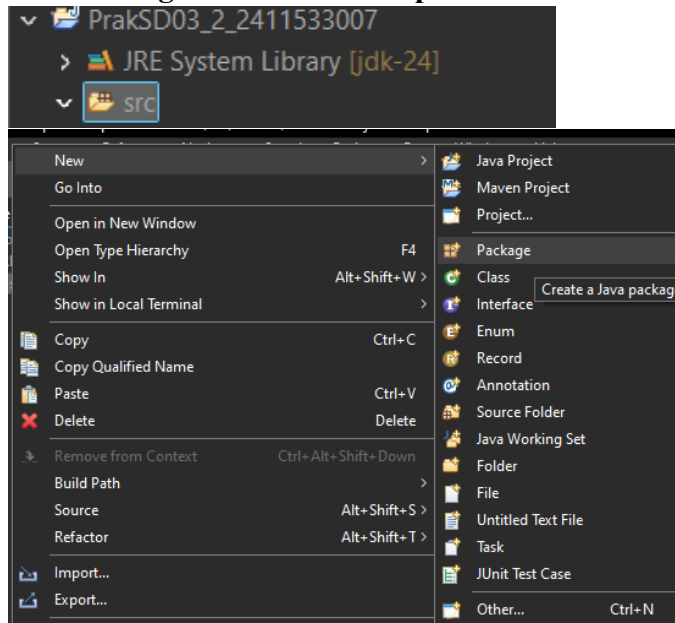
### a. Alat dan Bahan

1. Perangkat computer atau laptop
2. Jaringan internet

3. IDE (Integrated Development Environment) direkomendasikan Eclipse IDE
4. Java JDK (Java Development Kit)

#### b. Package pekan 9

1. Buat new package. **Buka java project** yang telah dibuat sebelumnya, lalu **klik kanan** pada folder 'src', setelahnya akan muncul list option, pilih 'new', lalu 'Package' dan beri nama 'pekan9'.



#### c. Program (basic class) Node

1. Buat new class dengan klik kanan pada package pekan 9 dan beri nama "Node".
2. Membuat class publik (dapat diakses dari luar kelas) bernama "node" dengan attribut int data yang berfungsi menyimpan data dalam node tersebut, lalu attribut Node left and right berfungsi menyimpan referensi ke node anak left and right, hal ini memungkinkan node menjadi struktur Binary tree.

```

3 public class Node {
4     int data;
5     Node left;
6     Node right;

```

Konstruktor yang akan dijalankan saat objek node dibuat, menerima parameter "int data" yang lalu di setter nilai dari parameter ini ke attribut class sebagai nilai dari node, lalu set attribut class left and right menjadi null (pointernya tidak menunjuk ke node manapun)

```

7 public Node(int data) {
8     this.data = data;
9     left = null;
10    right = null;
11 }

```

3. Method setter void (tidak mengembalikan nilai). Mengecek apakah left/right masih kosong (null), jika ya maka atur node yang dikirim sebagai anak kiri. Mencegah penimpanan node left atau right yang sudah terisi.

```

13•   public void setLeft (Node node) {
14       if (left == null)
15           left = node;
16   }
17•   public void setRight (Node node) {
18       if (right == null)
19           right = node;
20   }

```

Method getter utk megembalikan nilai node anak right/left dari akhir node ini.

```

21•   public Node getLeft () {
22       return left;
23   }
24•   public Node getRight () {
25       return right;
26   }

```

Method getter unutm mendapatkan nilai data yang disimpan dalam node ini.

```

27•   public int getData() {
28       return data;
29   }

```

Method setter dengan parameter int data, berfungsi mengubah nilai “data” dari node. Jika ingin mengganti nilainya setelah node dibuat.

```

30•   public void setData (int data) {
31       this.data = data;
32   }

```

4. **Method printPreorder** berfungsi untuk mencetak isi pohon dengan traversal preorder. If statement memeriksa baris rekursi, jika node null, maka berhenti. Flow : Cetak nilai **node sekarang** terlebih dahulu, lalu rekursif ke anak **kiri**, lalu anak **kanan**.

```

34•   void printPreorder (Node node) {
35       if (node == null)
36           return;
37       System.out.println(node.data + " ");
38       printPreorder(node.left);
39       printPreorder(node.right);
40   }

```

5. **Method printPostorder** berfungsi mencetak isi pohon dengan traversal postorder (anak kiri → anak kanan → node sekarang).

Flow : Cetak isi pohon secara rekursif dari kiri ke kanan, lalu cetak node saat ini terakhir.

```

41•   void printPostorder (Node node) {
42       if (node == null)
43           return;
44       printPostorder(node.left);
45       printPostorder(node.right);
46       System.out.println(node.data + " ");
47   }

```

6. **Method printInorder** berfungsi mencetak isi pohon dengan traversal Inorder (anak kiri → node sekarang → anak kanan).

```
48 void printInorder (Node node) {
49     if (node == null)
50         return;
51     printInorder(node.left);
52     System.out.println(node.data + " ");
53     printInorder(node.right);
54 }
```

7. **Method String** pembantu untuk mencetak struktur tree dalam bentuk diagram hierarki di konsol, menggunakan versi method print yang menerima 3 parameter.

```
55 public String print() {
56     return this.print("", true, "");
57 }
```

8. **Method String** (rekursif) untuk mencetak bentuk visual pohon. Memiliki parameter prefix : indentasi visual, isTail: apakah node ini anak terakhir, sb: string builder (tidak dipakai untuk menyusun string di sini, hanya sebagai placeholder).

```
58 public String print (String prefix, boolean isTail, String sb) {
```

Memeriksa node kanan jika tidak null, maka cetak bagian kanan terlebih dahulu.

```
59     if (right != null) {
60         right.print(prefix + (isTail ? "| " : " "), false, sb);
61     }
```

Cetak node sekarang dengan indentasi dan simbol cabang sesuai posisinya.

```
62     System.out.println(prefix + (isTail ? "\\--" : "/--") + data);
```

Setelah mencetak node kanan dan node sekarang, baru cetak node kiri.

```
63     if (left != null) {
64         left.print (prefix + (isTail ? " " : "| "), true, sb);
65     }
```

“Return sb” digunakan untuk konsistensi method, tapi string-nya tidak dimodifikasi.

```
66     return sb;
67 }
68 }
```

9. Penjelasan program : Program Node sebagai struktur dasar dari satu simpul (node) dalam pohon biner, yang menyimpan sebuah nilai (data) serta referensi ke anak kiri dan kanan. Selain itu, kelas ini menyediakan metode untuk mengatur dan mengambil data maupun anak node, serta tiga jenis traversal pohon yaitu preorder, inorder, dan postorder. Kelas ini juga dilengkapi dengan metode untuk mencetak struktur pohon secara visual di konsol, sehingga memudahkan pengguna dalam melihat bentuk hierarki pohon secara menyeluruh.

#### d. Program BTree

1. Buat new class dengan klik kanan pada package pekan 9 dan beri nama “BTree”.
2. Mendefinisikan public class bernama “BTree”, dengan instance variabel root bertipe Node, yaitu simpul akar dari Biner tree. Instance “currentNode” untuk menyimpan node yang sedang aktif/dipilih.

```
3 public class BTree {  
4     private Node root;  
5     private Node currentNode;
```

Konstruktor dari kelas BTree. Saat pohon dibuat, root diinisialisasi sebagai null (pohon kosong).

```
6 public BTree() {  
7     root = null;  
8 }
```

3. Method pencarian nilai dalam pohon, dimulai dari akar (root). Ini adalah method publik yang memanggil versi rekursif.

```
10 public boolean search (int data) {  
11     return search(root, data);  
12 }
```

4. Method rekursif utk mencari nilai data pada node tertentu dan anak anaknya.

```
13 private boolean search(Node node, int data) {
```

Memeriksa node menggunakan method getter untuk menganbil nilai pada node tersebut apakah equivalen data yang ingin dicari, jika cocok kembalikan true.

```
14     if (node.getData() == data)  
15         return true;
```

Memeriksa jika node kiri tidak null, maka lanjut ke pencarian ke node kiri.

```
16     if (node.getLeft() != null)  
17         if (search (node.getLeft(), data))  
18             return true;
```

Jika tidak ditemukan di kiri dan ada anak kanan, lanjut cari ke kanan.

```
19     if (node.getRight() != null)  
20         if (search(node.getRight(), data))  
21             return true;
```

Jika semua jalur sudah diperiksa dan tidak ditemukan, maka kembalikan false.

```
22     return false;  
23 }
```

5. Memanggil method **printInorder** milik Node dari root untuk **menampilkan data** secara **inOrder** traversal.

```
24 public void printInorder() {  
25     root.printInorder(root);  
26 }
```

6. Memanggil method **printPreorder** milik Node dari root untuk **menampilkan data** secara **PreOrder** traversal.

```

27 public void printPreorder() {
28     root.printPreorder(root);
29 }

```

7. Memanggil method **printPostOrder** milik Node dari root untuk menampilkan data secara **PostOrder** traversal.

```

30 public void printPostorder() {
31     root.printPostorder(root);
32 }

```

8. **Method Getter** untuk mengembalikan nilai root dari tree.

```

34 public Node getRoot() {
35     return root;
36 }

```

9. **Method boolean** untuk memeriksa apakah tree kosong (tidak terdapat root).

```

38 public boolean isEmpty() {
39     return root == null;
40 }

```

10. **Method Integer** yang memanggil versi rekursif dari method **countNodes** untuk menghitung semua simpul dalam pohon.

```

42 public int countNodes() {
43     return countNodes(root);
44 }

```

11. **Method countNodes(Node node)**, berfungsi mulai menghitung dari 1 node (yang sedang dikunjungi), jika node kosong, kembalikan 0. Jika tidak, hitung total node dari anak kiri dan kanan secara rekursif.

```

46 private int countNodes (Node node) {
47     int count = 1;
48     if (node == null) {
49         return 0;
50     } else {
51         count += countNodes(node.getLeft());
52         count += countNodes(node.getRight());
53         return count;
54     }
55 }

```

12. **Method print()**, digunakan untuk memanggil method **print()** dari objek Node menampilkan diagram visual pohon.

```

57 public void print() {
58     root.print();
59 }

```

13. **Method getter** terhadap node yang sedang aktif/dipilih(**currentNode**), mengembalikan nilai **currentNode**.

```

61 public Node getCurrent() {
62     return currentNode;
63 }

```

14. Method setter untuk set nilai currentNode (instance class Btree) dengan nilai dari parameter method ini.

```
65 public void setCurrent (Node node) {  
66     this.currentNode = node;  
67 }
```

15. Method setter untuk menentukan node mana yang akan menjadi akar pohon (root).

```
69 public void setRoot(Node root) {  
70     this.root = root;  
71 }  
72 }
```

16. Output program : Kelas BTree berfungsi sebagai struktur utama dari sebuah pohon biner (Binary Tree) yang menyimpan referensi ke simpul akar dan menyediakan berbagai metode operasi seperti pencarian data secara rekursif, menampilkan isi pohon menggunakan traversal inorder, preorder, dan postorder, mencetak bentuk visual pohon, serta menghitung jumlah total simpul dalam pohon. Dengan menyediakan getter dan setter untuk node aktif (currentNode) dan akar (root), kelas ini juga dirancang agar fleksibel untuk pengembangan lebih lanjut seperti manipulasi node dinamis atau pembuatan struktur pohon interaktif.

#### e. Program TreeMain

1. Buat new class dengan klik kanan pada package pekan 9 dan beri nama "TreeMain".
2. Membuat object baru dari kelas BTree dengan identifier "tree", yang digunakan sebagai struktur binary tree.

```
4 public static void main(String[] args) {  
5     //Membuat pohon  
6     BTree tree = new BTree();
```

3. Menampilkan informasi jumlah simpul awal pohon menggunakan method countNodes() terhadap objek "tree" (yang seharusnya mengembalikan nol).

```
7     System.out.println("Jumlah simpul awal pohon : ");  
8     System.out.println(tree.countNodes());
```

4. Menambahkan simpul data dengan nilai int 1 dan set sebagai akar menggunakan method setRoot(root).

```
9         //menambahkan simpul data 1  
10        Node root = new Node (1);  
11        //menjadikan simpul 1 sebagai root  
12        tree.setRoot(root);
```

5. Menampilkan jumlah simpul setelah penambahan root, cek menggunakan method countNodes() terhadap object "tree".

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

6. Menambahkan node baru (node 2 sampai node 7) bernilai integer 2 sampai 7.



```

15         Node node2 = new Node(2);
16         Node node3 = new Node(3);
17         Node node4 = new Node(4);
18         Node node5 = new Node(5);
19         Node node6 = new Node(6);
20         Node node7 = new Node(7);

```

7. Menjadikan node 2 sebagai cabang kiri dari akar.

```

21         root.setLeft(node2);

```

Set node 4 sebagai cabang kiri dari node 2

```

22         node2.setLeft(node4);

```

Set node 5 sebagai cabang kiri dari node 2

```

23         node2.setRight(node5);

```

Set node 6 sebagai cabang kiri dari node 3

```

24         node3.setLeft(node6);

```

Set node 3 sebagai cabang kanan dari root

```

25         root.setRight(node3);

```

Set node 7 sebagai cabang kanan dari node 3

```

26         node3.setRight(node7);

```

8. Menandai currnetNode dengan nilai dari root saat ini (mengambil nilai dari class tree menggunakan method getter “getRoot”) sebagai root tree.

```

27         //set root
28         tree.setCurrent(tree.getRoot());

```

9. Menampilkan simpul terakhir (simpul currentNode yang saat ini adalah root bernilai integer 1) menggunakan method getter.

```

29         System.out.println("Menampilkan simpul terakhir : ");
30         System.out.println(tree.getCurrent().getData());

```

10. Menampilkan jumlah node setelah semua simpul (2 samapi 7) ditambahkan, menggunakan method countNodes() yang akan mencetak 7 karena tree sudah memiliki total 7 simpul.

```

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

```

11. Menampilkan traversal Inorder dari tree menggunakan method printInorder().

```

33         System.out.println("Inorder : ");
34         tree.printInorder();

```

12. Menampilkan traversal PreOrder dari tree pakai method printPreOrder().

```

35         System.out.println("\nPreOrder : ");
36         tree.printPreorder();

```

13. Menampilkan traversal PostOrder dari tree pakai method printPostOrder().

```

37         System.out.println("\nPostOrder : ");
38         tree.printPostorder();

```

14. Menampilkan struktur visual dari pohon di konsol, berupa bentuk bercabang seperti diagram.

```

39         System.out.println("\nMenampilkan simpul dalam bentuk pohon");
40         tree.print();
41     }
42 }

```

15. Output program : Program TreeMain bertujuan menguji implementasi struktur data Binary Tree dengan menggunakan kelas Node dan Btree (yang telah dibuat sebelumnya). Program ini menunjukkan bagaimana membuat dan

menyusun node dalam struktur pohon, mengatur simpul akar (root), menambahkan simpul anak kiri dan kanan, serta mencetak jumlah simpul yang ada. Selain itu, program menampilkan isi pohon dengan tiga jenis traversal: inorder, preorder, dan postorder, serta memberikan tampilan visual struktur pohon secara hierarkis. Ini berfungsi sebagai demonstrasi praktis dari penggunaan struktur pohon dalam pemrograman Java.

```
Console X
<terminated> TreeMain [Java Application] C:\Program Files\Java\jdk-24\bin\javaw.exe (Jun 30, 2025, 7:1
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
```

#### f. Program GraphTraversal

1. Buat new class dengan klik kanan pada package pekan 9 dan beri nama "GraphTraversal".
2. Import semua class dari java util yang diperlukan. Class Map dan HashMap untuk representasi graf. Class set dan HashSet untuk menandai node yang telah dikunjungi. Class list dan arrayList untuk menyimpan tetangga (adjacency). Class Queue dan LinkedList untuk penelusuran BFS.

```
1 package pekan9;  
2 import java.util.*;
```

3. Public class (main class) GraphTraversal yang berisi struktur dan metode untuk manipulasi graf. Private Map<String, List<String>> digunakan untuk menyimpan representasi adjacency list, yaitu struktur graf dengan key sebagai simpul dan value-nya adalah daftar tetangga. Key String yaitu nama node/simpul misal simpul A, B, dll. Nilainya List<String> berfungsi sebagai daftar node yang terhubung langsung (tetangga).

```
4 public class GraphTraversal {  
5     private Map<String, List<String>> graph = new HashMap<>();
```

4. Method yang berfungsi menambahkan edge (graf tidak berarah) antar 2 node. Jika node belum ada di map, akan dibuat daftar baru terlebih dahulu. Menambahkan node 2 ke tetangga node 1 dan sebaliknya (node1 ke node2), karena graf tidak berarah. "putIfAbsent" hanya menambahkan key jika belum ada (untuk menghindari null). Dua baris terakhir berfungsi menambahkan koneksi dua arah — node1 ke node2 dan node2 ke node1.

```
7     //Menambahkan edge (graf toial 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    }
```

5. Method printGraph untuk menampilkan graf awal, dengan mencetak isi graf dalam bentuk adjacency list. Setiap simpul ditampilkan beserta tetangganya (daftar simpul yang terhubung dengannya).

graph.keySet(): mendapatkan semua node, get(node) untuk mendapatkan list tetangga dari node tersebut, String.join() untuk menggabungkan isi list jadi String (dengan pemisah koma).

```
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    }
```

6. Method dfs() untuk penelusuran DFS, Memanggil fungsi rekursif dfsHelper() dengan parameter simpul awal dan set visited kosong. Memulai penelusuran

(traversal) dari simpul 'start'. Object visited untuk menyimpan node yang sudah dikunjungi agar tidak dikunjungi lagi.

```
26 //DFS rekursif
27 public void dfs(String start) {
28     Set<String> visited = new HashSet<>();
29     System.out.println("Penelusuran DFS : ");
30     dfsHelper(start, visited);
31     System.out.println();
32 }
```

7. **Method dfsHelper()** berfungsi mengambil input array dari user, memprosesnya, dan menampilkannya secara visual. Modifier "private" artinya method ini hanya bisa dipanggil di dalam kelas ini. "void" method tidak mengembalikan nilai. Mengecek apakah 'current' sudah pernah dikunjungi, jika belum maka cetak dan tandai sudah dikunjungi, lanjut kunjungi semua tetangga secara rekursif.

```
34 private void dfsHelper(String current, Set<String> visited) {
35     if (visited.contains(current))
36         return;
37     visited.add(current);
38     System.out.print(current + " ");
39     for (String neighbor : graph.getDefault(current, new ArrayList<>())) {
40         dfsHelper(neighbor, visited);
41     }
42 }
```

Penjelasan algoritma DFS : Penelusuran dilakukan sedalam mungkin, dari simpul ke tetangga pertama, lalu ke tetangga dari tetangga, dst., sampai mentok, lalu backtrack.

8. Method penelusuran BFS(), membuat object visited untuk menghindari simpul dikunjungi dua kali dan object queue untuk menyimpan simpul yang akan dikunjungi berdasarkan urutan level.

```
44 //BFS iteratif
45 public void bfs(String start) {
46     Set<String> visited = new HashSet<>();
47     Queue<String> queue = new LinkedList<>();
48     queue.add(start);
49     visited.add(start);
50     System.out.println("Penelusuran BFS : ");
```

Iterasi penelusuran BFS, queue.poll() berfungsi mengambil simpul dari antrian dan menyimpan nilainya pada variabel "current". Gunakan for loop untuk menambahkan semua tetangga yang belum dikunjungi ke dalam antrian.

```
51 while (!queue.isEmpty()) {
52     String current = queue.poll();
53     System.out.print(current + " ");
54     for (String neighbor : graph.getDefault(current, new ArrayList<>())) {
55         if (!visited.contains(neighbor)) {
56             queue.add(neighbor);
57             visited.add(neighbor);
58         }
59     }
60 }
61 System.out.println();
62 }
```

Penjelasan algoritma BFS : BFS melakukan penelusuran berdasarkan level, artinya mengunjungi semua tetangga dari simpul dulu sebelum masuk ke level berikutnya.

9. **Main method**, Membuat object dari class GraphTraversal bernama “graph”.

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

10. Memanggil object grap lalu instasiasi menggunakan method addEdge() untuk menambahkan simpul dan sisi dengan pola berpasangan antar node.

```
68 //contoh graf : A-B, A-C, B-D, B-E  
69 graph.addEdge("A", "B");  
70 graph.addEdge("A", "C");  
71 graph.addEdge("B", "D");  
72 graph.addEdge("B", "E");
```

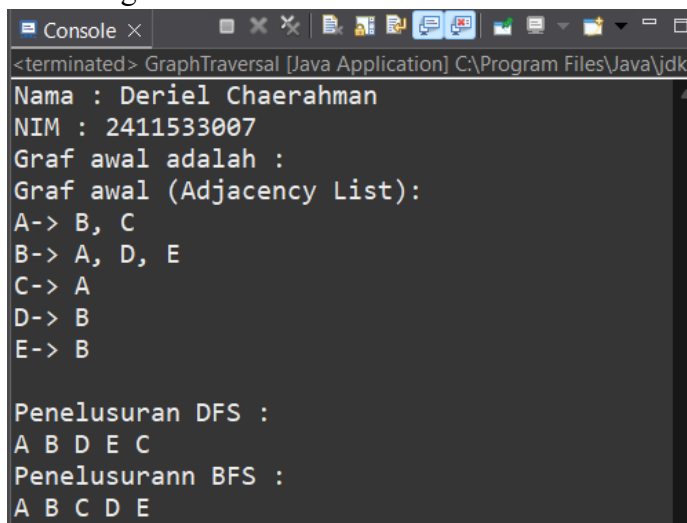
11. Menampilkan seluruh isi graf, menggunakan method printGraph().

```
74 //cetak graf awal  
75 System.out.println("Graf awal adalah : ");  
76 graph.printGraph();  
77
```

12. Lakukan penelusuran pada object graph menggunakan method dfs dan bfs dengan acuan simpul awal “A”.

```
78 //lakukan penelusuran  
79 graph.dfs("A");  
80 graph.bfs("A");  
81 }  
82 }
```

13. Output program : Program GraphTraversal.java bertujuan mengimplementasi kan struktur graf tidak berarah menggunakan representasi adjacency list dan menyediakan dua metode penelusuran utama, yaitu Depth-First Search (DFS) dan Breadth-First Search (BFS). Program ini memungkinkan pengguna untuk menambahkan sisi antar simpul, mencetak isi graf dalam bentuk daftar ketetanggaan, dan melakukan penelusuran dari simpul awal untuk menunjukkan urutan kunjungan simpul sesuai dengan metode DFS (menggunakan rekursi) dan BFS (menggunakan antrian). Dengan pendekatan ini, program memberikan gambaran yang jelas tentang perbedaan strategi penelusuran pada graf serta cara menyusun dan menavigasi data yang saling terhubung.



```
<terminated> GraphTraversal [Java Application] C:\Program Files\Java\jdk  
Nama : Deriel Chaerahan  
NIM : 2411533007  
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  
Penelusurann BFS :  
A B C D E
```

#### **D. Kesimpulan**

Setelah melakukan praktikum disimpulkan implementasi konsep struktur data Binary Tree dan Graph Traversal menggunakan konsep OOP menggunakan bahasa Java. Penerapan Binary Tree pada file Node.java yang mendefinisikan struktur dasar simpul/node pada Binary tree lengkap dengan metode penelusurannya yaitu preOrder, inOrder, dan postOrder, serta memvisualkan struktur dari Binary Tree. Program Btree.java sebagai class untuk manajemen tree seperti method searching, penghitung simpul dan akses setter/getter terhadap object Binary tree yang dibuat. Program treeMain.java berfungsi sebagai program utama untuk membuat object Binary Tree, menambahkan simpul, melakukan traversal dan mencetak struktur tree.

Untuk materi graf implementasinya tentang penelusuran menggunakan algoritma DFS dan BFS pada program GraphTraversal.java, dengan graf tidak berarah menggunakan adjacency list dan metode penelusurannya berupa rekursif untuk DFS dan Antrian (Queue) untuk BFS.