

LAPORAN PRAKTIKUM ALGORITMA DAN STRUKTUR DATA



2022



Daftar Isi

PERCOBAAN IMPLEMENTASI GRAPH MENGGUNAKAN LINKED LIST	4
LANGKAH 1.....	4
LANGKAH 2.....	4
LANGKAH 3.....	4
LANGKAH 4.....	5
LANGKAH 5.....	5
LANGKAH 6.....	5
LANGKAH 7.....	6
LANGKAH 8.....	6
LANGKAH 9.....	6
LANGKAH 10.....	7
LANGKAH 11.....	7
LANGKAH 12.....	8
LANGKAH 13.....	8
LANGKAH 14.....	9
<i>Pertanyaan Percobaan.....</i>	<i>10</i>
<i>Jawaban Percobaan.....</i>	<i>10</i>
PERCOBAAN IMPLEMENTASI GRAPH MENGGUNAKAN MATRIKS	12
LANGKAH 1.....	12
LANGKAH 2.....	12
LANGKAH 3.....	12
LANGKAH 4.....	13
LANGKAH 5.....	13
VERIFIKASI HASIL PERCOBAAN	14
<i>Pertanyaan Percobaan.....</i>	<i>14</i>
<i>Jawaban Percobaan.....</i>	<i>14</i>
TUGAS PRAKTIKUM	16
SOAL NOMOR 1.....	16
JAWABAN	16
<i>Source code.....</i>	<i>16</i>
<i>Output.....</i>	<i>16</i>
SOAL NOMOR 2.....	17
JAWABAN	17



PRAKTIKUM ALGORITMA DAN STRUKTUR DATA

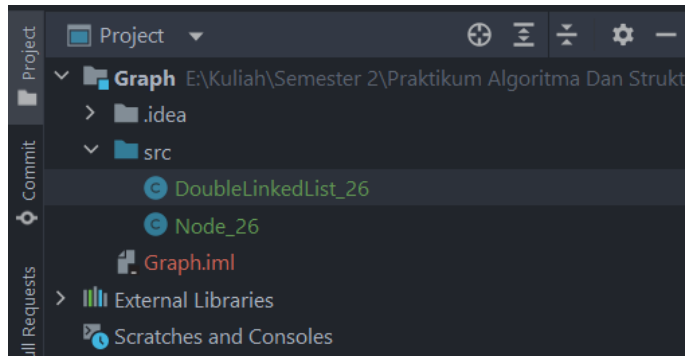
<i>Source code yang dimodifikasi</i>	17
<i>Output Undirected</i>	19
<i>Output Directed</i>	19
SOAL NOMOR 3	19
JAWABAN	20
<i>Source code method remove setelah dimodifikasi</i>	20
<i>Source code method removeEdge setelah direvisi</i>	20
<i>Output</i>	21
SOAL NOMOR 4	21
JAWABAN	21
<i>Source code graph</i>	21
<i>Source code graphArray</i>	23
<i>Output graph</i>	25
<i>Output graphArray</i>	25

Percobaan Implementasi Graph menggunakan Linked List

Pada percobaan ini akan diimplementasikan Graph menggunakan Linked Lists untuk merepresentasikan graph adjacency. Silakan lakukan langkah-langkah praktikum sebagai berikut.

Langkah 1

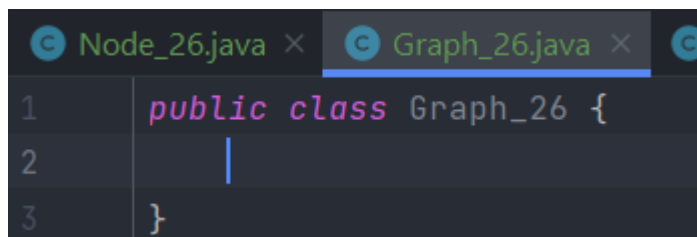
Buatlah class Node, dan class Linked Lists sesuai dengan praktikum Double Linked Lists.



Graph
vertex: int LinkedList: List right: Node
addEdge(source: int, destination: int): void degree(source: int): void removeEdge(source: int, destination: int): void removeAllEdges() printGraph()

Langkah 2

Tambahkan class Graph yang akan menyimpan method-method dalam graph dan juga method main().



Langkah 3

Di dalam class Graph, tambahkan atribut vertex bertipe integer dan list[] bertipe LinkedList.

```
Node_26.java x Graph_26.java x Doubl
1 public class Graph_26 {
2     int vertex;
3     DoubleLinkedList_26[] list;
4 }
```

Langkah 4

Tambahkan konstruktor default untuk menginisialisasi variabel vertex dan menambahkan perulangan untuk jumlah vertex sesuai dengan jumlah length array yang telah ditentukan.

```
5 public Graph_26(int vertex) {
6     this.vertex = vertex;
7     list = new DoubleLinkedList_26[vertex];
8     for (int i = 0; i < vertex; i++) {
9         list[i] = new DoubleLinkedList_26();
10    }
11 }
```

Langkah 5

Tambahkan method addEdge(). Jika yang akan dibuat adalah graph berarah, maka yang dijalankan hanya baris pertama saja. Jika graph tidak berarah yang dijalankan semua baris pada method addEdge().

```
13 public void addEdge(int source, int destination) {
14     // add edge
15     list[source].addFirst(destination);
16     // add back edge (for undirected)
17     list[destination].addFirst(source);
18 }
```

Langkah 6

Tambahkan method degree() untuk menampilkan jumlah derajat lintasan pada suatu vertex. Di dalam metode ini juga dibedakan manakah statement yang digunakan untuk graph berarah atau graph tidak berarah. Eksekusi hanya sesuai kebutuhan saja

```

20 public void degree(int source) throws Exception {
21     // degree undirected graph
22     System.out.println("degree vertex " + source + " : " + list[source].size());
23
24     // degree directed graph
25     // inDegree
26     int k, totalIn = 0, totalOut = 0;
27     for (int i = 0; i < vertex; i++) {
28         for (int j = 0; j < list[i].size(); j++) {
29             if (list[i].get(j) == source) {
30                 ++totalIn;
31             }
32         }
33         // outDegree
34         for (k = 0; k < list[source].size(); k++) {
35             list[source].get(k);
36         }
37         totalOut = k;
38     }
39     System.out.println("Indegree dari vertex " + source + " : " + totalIn);
40     System.out.println("Outdegree dari vertex " + source + " : " + totalOut);
41     System.out.println("degree vertex " + source + " : " + (totalIn+totalOut));
42 }

```

Langkah 7

Tambahkan method `removeEdge()`. Method ini akan menghapus lintasan ada suatu graph. Oleh karena itu, dibutuhkan 2 parameter untuk menghapus lintasan yaitu `source` dan `destination`.

```

44 public void removeEdge(int source, int destination) throws Exception {
45     for (int i = 0; i < vertex; i++) {
46         if (i == destination) {
47             list[source].remove(destination);
48         }
49     }
50 }

```

Langkah 8

Tambahkan method `removeAllEdges()` untuk menghapus semua vertex yang ada di dalam graph

```

52 public void removeAllEdges() {
53     for (int i = 0; i < vertex; i++) {
54         list[i].clear();
55     }
56     System.out.println("Graph berhasil dikosongkan");
57 }

```

Langkah 9

Tambahkan method `printGraph()` untuk mencetak graph ter-update.

```
59 public void printGraph() throws Exception {
60     for (int i = 0; i < vertex; i++) {
61         if (list[i].size() > 0) {
62             System.out.print("Vertex " + i + " terhubung dengan : ");
63             for (int j = 0; j < list[i].size(); j++) {
64                 System.out.print(list[i].get(j) + " ");
65             }
66             System.out.println("");
67         }
68     }
69     System.out.println("");
70 }
```

Langkah 10

Compile dan jalankan method main() dalam class Graph untuk menambahkan beberapa edge pada graph, kemudian tampilkan. Setelah itu keluarkan hasilnya menggunakan pemanggilan method main().

Keterangan: degree harus disesuaikan dengan jenis graph yang telah dibuat (directed/undirected).

```
72 public static void main(String[] args) throws Exception {
73     Graph_26 graph = new Graph_26( vertex: 6);
74     graph.addEdge( source: 0, destination: 1);
75     graph.addEdge( source: 0, destination: 4);
76     graph.addEdge( source: 1, destination: 2);
77     graph.addEdge( source: 1, destination: 3);
78     graph.addEdge( source: 1, destination: 4);
79     graph.addEdge( source: 2, destination: 3);
80     graph.addEdge( source: 3, destination: 4);
81     graph.addEdge( source: 3, destination: 0);
82     graph.printGraph();
83     graph.degree( source: 2);
84 }
```

Langkah 11

Amati hasil running tersebut

```

"C:\Users\Asus TUF DT\.jdk\openjdk-17.0
Vertex 0 terhubung dengan : 3 4 1
Vertex 1 terhubung dengan : 4 3 2 0
Vertex 2 terhubung dengan : 3 1
Vertex 3 terhubung dengan : 0 4 2 1
Vertex 4 terhubung dengan : 3 1 0

degree vertex 2 : 2
Indegree dari vertex 2 : 2
Outdegree dari vertex 2 : 2
degree vertex 2 : 4

Process finished with exit code 0

```

Langkah 12

Tambahkan pemanggilan method `removeEdge()` sesuai potongan code di bawah ini pada method `main()`. Kemudian tampilkan graph tersebut.

```

84      graph.removeEdge( source: 1, destination: 2);
85      graph.printGraph();

```

Langkah 13

Amati hasil running tersebut.

```

Run: Graph_26 x
"C:\Users\Asus TUF DT\.jdk\openjdk-17.0
Vertex 0 terhubung dengan : 3 4 1
Vertex 1 terhubung dengan : 4 3 2 0
Vertex 2 terhubung dengan : 3 1
Vertex 3 terhubung dengan : 0 4 2 1
Vertex 4 terhubung dengan : 3 1 0

degree vertex 2 : 2
Indegree dari vertex 2 : 2
Outdegree dari vertex 2 : 2
degree vertex 2 : 4
Vertex 0 terhubung dengan : 3 4 1
Vertex 1 terhubung dengan : 4 3 0
Vertex 2 terhubung dengan : 3 1
Vertex 3 terhubung dengan : 0 4 2 1
Vertex 4 terhubung dengan : 3 1 0

Process finished with exit code 0

```


Langkah 14

Uji coba penghapusan lintasan yang lain! Amati hasilnya!

```
88 graph.removeEdge( source: 3, destination: 0 );
89 graph.printGraph();
90
```

Run: Graph_26 x

```
degree vertex 2 : 4

Vertex 0 terhubung dengan : 3 4 1
Vertex 1 terhubung dengan : 4 3 0
Vertex 2 terhubung dengan : 3 1
Vertex 3 terhubung dengan : 0 4 2 1
Vertex 4 terhubung dengan : 3 1 0

Vertex 0 terhubung dengan : 3 4 1
Vertex 1 terhubung dengan : 4 3 0
Vertex 2 terhubung dengan : 3 1
Vertex 3 terhubung dengan : 4 2 1
Vertex 4 terhubung dengan : 3 1 0
```

Verifikasi Hasil Percobaan

```
Run: Graph_26 x
```

```
"C:\Users\Asus TUF DT\.jdk\openjdk-17.0.
Vertex 0 terhubung dengan : 3 4 1
Vertex 1 terhubung dengan : 4 3 2 0
Vertex 2 terhubung dengan : 3 1
Vertex 3 terhubung dengan : 0 4 2 1
Vertex 4 terhubung dengan : 3 1 0

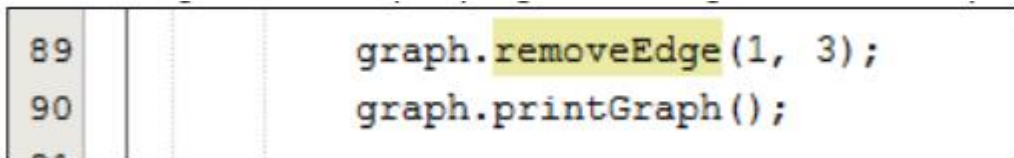
degree vertex 2 : 2
Indegree dari vertex 2 : 2
Outdegree dari vertex 2 : 2
degree vertex 2 : 4

Vertex 0 terhubung dengan : 3 4 1
Vertex 1 terhubung dengan : 4 3 0
Vertex 2 terhubung dengan : 3 1
Vertex 3 terhubung dengan : 0 4 2 1
Vertex 4 terhubung dengan : 3 1 0

Process finished with exit code 0
```

Pertanyaan Percobaan

1. Sebutkan beberapa jenis (minimal 3) algoritma yang menggunakan dasar Graph, dan apakah kegunaan algoritma-algoritma tersebut?
2. Pada class Graph terdapat array bertipe LinkedList, yaitu LinkedList list[]. Apakah tujuan pembuatan variabel tersebut ?
3. Apakah alasan pemanggilan method addFirst() untuk menambahkan data, bukan method add jenis lain pada linked list ketika digunakan pada method addEdge pada class Graph?
4. Bagaimana cara mendeteksi prev pointer pada saat akan melakukan penghapusan suatu edge pada graph ?
5. Kenapa pada praktikum 2.1.1 langkah ke-12 untuk menghapus path yang bukan merupakan lintasan pertama kali menghasilkan output yang salah ? Bagaimana solusinya ?



```
89 graph.removeEdge(1, 3);
90 graph.printGraph();
```

Jawaban Percobaan

1. Algoritma yang menggunakan dasar graph.
 - a. Algoritma Prim
Kegunaan: untuk mencari pohon rentang minimum untuk sebuah graf berbobot yang saling terhubung.
 - b. Algoritma KNN atau K-Nearest Neighbor
Kegunaan: mengklasifikasikan data berdasarkan similarity atau kemiripan atau kedekatannya terhadap data lainnya.
 - c. Algoritma Welch Powell
Kegunaan: algoritma yang digunakan untuk mewarnai sebuah graf secara mangkus, dan tidak selalu memberikan warna minimum untuk sebuah graf, namun keuntungan dari algoritma ini ialah penggunaannya yang lebih sederhana dan lebih mudah.
 - d. Algoritme Dijkstra
Kegunaan: Sebuah algoritma yang dipakai dalam memecahkan permasalahan jarak terpendek (shortest path problem) untuk sebuah graf berarah (directed graph).
2. Variabel array bertipe linked list digunakan untuk menyimpan setiap node vertex pada masing masing index array linkedlist.
3. Karena agar vertex yang dihubungkan oleh edge berada urut didepan vertex source.
4. Pada kasus ini menggunakan pemberian index pada method remove pada class doublelinkedlist dengan parameter destination sebagai acuan.
5. Karena parameter destination ketika pemanggilan method remove dari class doublelinkedlist akan digunakan sebagai index. Sehingga yang dihapus adalah sesuai urutan index tidak sesuai

dengan value vertex sehingga muncul error indeks diluar batas ketika ingin menghapus edge dengan source 1 dan destination 3.

Solusi:

Disini saya sedikit memodifikasi method remove yang semula menggunakan indeks disini saya menggunakan destination parameter sebagai acuan node mana yang akan dihapus sehingga tidak ada indeks melebihi batas.

```

44 public void removeEdge(int source, int destination) throws Exception {
45     for (int i = 0; i < vertex; i++) {
46         if (i == destination) {
47             list[source].remove(destination);
48         }
49     }
50     list[source].removeRevisi(destination);
51 }

```

```

142 public void removeRevisi(int destination) throws Exception {
143     if (isEmpty()) {
144         throw new Exception("Linked list kosong");
145     } else if (head.data == destination) {
146         removeFirst();
147     } else {
148         Node_26 current = head;
149         int i = 0;
150         while (i < size) {
151             if (current.data == destination) {
152                 if (current.next == null) {
153                     current.prev.next = null;
154                 } else if (current.prev == null) {
155                     current = current.next;
156                     current.prev = null;
157                     head = current;
158                 } else {
159                     current.prev.next = current.next;
160                     current.next.prev = current.prev;
161                 }
162             }
163             current = current.next;
164             i++;
165         }
166         size--;
167     }
168 }

```

Percobaan Implementasi Graph menggunakan Matriks

Pada praktikum 2.2 ini akan diimplementasikan Graph menggunakan matriks untuk merepresentasikan graph adjacency. Silakan lakukan langkah-langkah praktikum sebagai berikut.

Langkah 1

Uji coba graph bagian 2.2 menggunakan array 2 dimensi sebagai representasi graph. Buatlah class graphArray yang didalamnya terdapat variabel vertices dan array twoD_array!

```
Node_26.java × Graph_26.java × graphArray_26.java
1      public class graphArray_26 {
2          private final int vertices;
3          private final int [][] twoD_array;
4      }
```

Langkah 2

Buatlah konstruktor graphArray sebagai berikut!

```
5      public graphArray_26(int v) {
6          vertices = v;
7          twoD_array = new int[vertices + 1][vertices + 1];
8      }
```

Langkah 3

Untuk membuat suatu lintasan maka dibuat method makeEdge() sebagai berikut.

```
10     public void makeEdge(int to, int from, int edge) {
11         try {
12             twoD_array[to][from] = edge;
13         }
14         catch (ArrayIndexOutOfBoundsException index) {
15             System.out.println("Vertex tidak ada");
16         }
17     }
```

Untuk menampilkan suatu lintasan diperlukan pembuatan method getEdge() berikut

```

19     public int getEdge(int to, int from) {
20         try {
21             return twoD_array[to][from];
22         }
23         catch (ArrayIndexOutOfBoundsException index) {
24             System.out.println("Vertex tidak ada");
25         }
26         return -1;
27     }

```

Langkah 4

Kemudian buatlah method main() seperti berikut ini.

```

public static void main(String[] args) {
    int v, e, count = 1, to = 0, from = 0;
    Scanner sc = new Scanner(System.in);
    graphArray_26 graph;
    try {
        System.out.println("Masukkan jumlah vertices: ");
        v = sc.nextInt();
        System.out.println("Masukkan jumlah edges: ");
        e = sc.nextInt();

        graph = new graphArray_26(v);

        System.out.println("Masukkan edges: <to> <from>");
        while (count <= e) {
            to = sc.nextInt();
            from = sc.nextInt();

            graph.makeEdge(to, from, 1);
            count++;
        }
        System.out.println("Array 2D sebagai representasi graph sbb: ");
        System.out.print(" ");
        for (int i = 1; i <= v; i++) {
            System.out.print(i + " ");
        }
        System.out.println();

        for (int i = 1; i <= v; i++) {
            System.out.print(i + " ");
            for (int j = 1; j <= v; j++) {
                System.out.print(graph.getEdge(i, j) + " ");
            }
            System.out.println();
        }
    } catch (Exception E) {
        System.out.println("Error. Silakan cek kembali\n" + E.getMessage());
    }
    sc.close();
}

```

Langkah 5

Jalankan class graphArray dan amati hasilnya!

Verifikasi Hasil Percobaan

```
Run: graphArray_26 x
"C:\Users\Asus TUF DT\.jdk\openjdk-17.0.2\bin\java.exe"
Masukkan jumlah vertices:
5
Masukkan jumlah edges:
6
Masukkan edges: <to> <from>
1 2
1 5
2 3
2 4
2 5
3 4
Array 2D sebagai representasi graph sbb:
  1 2 3 4 5
1 0 1 0 0 1
2 0 0 1 1 1
3 0 0 0 1 0
4 0 0 0 0 0
5 0 0 0 0 0
Process finished with exit code 0
```

Pertanyaan Percobaan

1. Apakah perbedaan degree/derajat pada directed dan undirected graph?
2. Pada implementasi graph menggunakan adjacency matriks. Kenapa jumlah vertices harus ditambahkan dengan 1 pada indeks array berikut?

```
8 public graphArray(int v)
9 {
10     vertices = v;
11     twoD_array = new int[vertices + 1][vertices + 1];
12 }
```

3. Apakah kegunaan method getEdge() ?
4. Termasuk jenis graph apakah uji coba pada praktikum 2.2?
5. Mengapa pada method main harus menggunakan try-catch Exception ?

Jawaban Percobaan

1. Degree pada undirected graph jika ada edge menghubungkan antar vertex maka dihitung 1 degree sedangkan degree pada directed graph edge akan dihitung 1 jika menghubungkan antar vertex tidak secara bolak-balik, jika edge menghubungkan vertex A dengan vertex B secara bolak-balik maka akan dihitung 2 degree.
2. Karena vertex dimulai dari angka 1 jadi array harus ditambah 1.

3. Untuk menampilkan nilai array pada indeks ke[to][from] namun jika indeks out of bound maka akan muncul pemberitahuan vertex tidak ada.
4. Directed graph.
5. Agar ketika ada error program tidak langsung close namun ada pemberitahuan untuk mengecek errornya.

Tugas Praktikum

Soal Nomor 1

Ubahlah lintasan pada praktikum 2.1 menjadi inputan!

Jawaban

Source code

```
75 ▶ public static void main(String[] args) throws Exception {
76     //      Graph_26 graph = new Graph_26(6);
77     // soal nomor 1
78     Scanner input = new Scanner(System.in);
79     int source, destination;
80     System.out.print("Masukkan jumlah vertices : ");
81     Graph_26 graph = new Graph_26(input.nextInt());
82     System.out.print("Masukkan jumlah edges : ");
83     int edges = input.nextInt();
84     System.out.println("Masukkan edges: <source> <destination>");
85     for (int i = 0; i < edges; i++) {
86         source = input.nextInt();
87         destination = input.nextInt();
88         graph.addEdge(source, destination);
89     }
90     graph.printGraph();
```

Output

```
▶ ↑ "C:\Users\Asus TUF DT\.jdk\openjdk-17.0.
  ↓ Masukkan jumlah vertices : 6
  ⏏ Masukkan jumlah edges : 8
  ⏏ Masukkan edges: <source> <destination>
  ⏏ 0 1
  ⏏ 0 4
  ⏏ 1 2
  ⏏ 1 3
  ⏏ 1 4
  ⏏ 2 3
  ⏏ 3 4
  ⏏ 3 0
  ⏏ Vertex 0 terhubung dengan : 3 4 1
  ⏏ Vertex 1 terhubung dengan : 4 3 2 0
  ⏏ Vertex 2 terhubung dengan : 3 1
  ⏏ Vertex 3 terhubung dengan : 0 4 2 1
  ⏏ Vertex 4 terhubung dengan : 3 1 0
  ⏏ Process finished with exit code 0
```


Soal Nomor 2

Tambahkan method `graphType` dengan tipe boolean yang akan membedakan graph termasuk directed atau undirected graph. Kemudian update seluruh method yang berelasi dengan method `graphType` tersebut (hanya menjalankan statement sesuai dengan jenis graph) pada praktikum 2.1

Jawaban

Source code yang dimodifikasi

Penambahan variabel jenis

```
1  import java.util.Scanner;
2
3  public class Graph_26 {
4      int vertex;
5      DoubleLinkedList 26[] list;
6      public static String jenis;
```

Input value variabel jenis

```
107 public static void main(String[] args) throws Exception {
108     // Graph_26 graph = new Graph_26(6);
109     // soal nomor 1
110     Scanner input = new Scanner(System.in);
111     int source, destination;
112     System.out.println("Masukkan jenis graph : (directed / undirected)");
113     jenis = input.next();
```

Method mengecek jenis graph

```
103 @ public boolean graphType(String jenis) {
104     return jenis.equalsIgnoreCase("undirected");
105 }
```

Modifikasi pada `addEdge`

```
16 public void addEdge(int source, int destination) {
17     if (graphType(jenis)) {
18         list[source].addFirst(destination);
19         list[destination].addFirst(source);
20     } else {
21         list[source].addFirst(destination);
22     }
```

Modifikasi pada `removeEdge`

```
75 public void removeEdge(int source, int destination) throws Exception {
76     // for (int i = 0; i < vertex; i++) {
77     //     if (i == destination) {
78     //         list[source].remove(destination);
79     //     }
80     // }
81     if (graphType(jenis)) {
82         list[source].removeRevisi(destination);
83         list[destination].removeRevisi(source);
84     } else {
85         list[source].removeRevisi(destination);
86     }
87 }
```

Modifikasi pada degree

```
29 public void degree(int source) throws Exception {
30     if (graphType(jenis)) {
31         System.out.println("degree vertex " + source + " : " + list[source].size());
32     } else {
33         //indegree
34         int k, totalIn = 0, totalOut = 0;
35         for (int i = 0; i < vertex; i++) {
36             for (int j = 0; j < list[i].size(); j++) {
37                 if (list[i].get(j) == source) {
38                     ++totalIn;
39                 }
40             }
41             // outDegree
42             for (k = 0; k < list[source].size(); k++) {
43                 list[source].get(k);
44             }
45             totalOut = k;
46         }
47         System.out.println("Indegree dari vertex " + source + " : " + totalIn);
48         System.out.println("Outdegree dari vertex " + source + " : " + totalOut);
49         System.out.println("degree vertex " + source + " : " + (totalIn+totalOut));
50     }
}
```

Output Undirected

```
"C:\Users\Asus TUF DT\.jdk\openjdk-17.0.2\bin\j
Masukkan jenis graph : (directed / undirected)
undirected
Masukkan jumlah vertices : 6
Masukkan jumlah edges : 8
Masukkan edges: <source> <destination>
0 1
0 4
1 2
1 3
1 4
2 3
3 4
3 0
Vertex 0 terhubung dengan : 3 4 1
Vertex 1 terhubung dengan : 4 3 2 0
Vertex 2 terhubung dengan : 3 1
Vertex 3 terhubung dengan : 0 4 2 1
Vertex 4 terhubung dengan : 3 1 0
Process finished with exit code 0
```

Output Directed

```
"C:\Users\Asus TUF DT\.jdk\openjdk-17.0.2\bin\ja
Masukkan jenis graph : (directed / undirected)
directed
Masukkan jumlah vertices : 6
Masukkan jumlah edges : 8
Masukkan edges: <source> <destination>
0 1
0 4
1 2
1 3
1 4
2 3
3 4
3 0
Vertex 0 terhubung dengan : 4 1
Vertex 1 terhubung dengan : 4 3 2
Vertex 2 terhubung dengan : 3
Vertex 3 terhubung dengan : 0 4
Process finished with exit code 0
```

Soal Nomor 3

Modifikasi method `removeEdge()` pada praktikum 2.1 agar tidak menghasilkan output yang salah untuk path selain path pertama kali!

Jawaban

Source code method remove setelah dimodifikasi

```

142     public void removeRevisi(int destination) throws Exception {
143         if (isEmpty()) {
144             throw new Exception("Linked list kosong");
145         } else if (head.data == destination) {
146             removeFirst();
147         } else {
148             Node_26 current = head;
149             int i = 0;
150             while (i < size) {
151                 if (current.data == destination) {
152                     if (current.next == null) {
153                         current.prev.next = null;
154                     } else if (current.prev == null) {
155                         current = current.next;
156                         current.prev = null;
157                         head = current;
158                     } else {
159                         current.prev.next = current.next;
160                         current.next.prev = current.prev;
161                     }
162                 }
163                 current = current.next;
164                 i++;
165             }
166             size--;
167         }
168     }

```

Source code method removeEdge setelah direvisi

```

75     public void removeEdge(int source, int destination) throws Exception {
76         for (int i = 0; i < vertex; i++) {
77             // if (i == destination) {
78             //     list[source].remove(destination);
79             // }
80             // }
81             if (graphType(jenis)) {
82                 list[source].removeRevisi(destination);
83                 list[destination].removeRevisi(source);
84             } else {
85                 list[source].removeRevisi(destination);
86             }
87         }

```

Output

Dengan inputan yang sama seperti praktikum 2.1 namun sudah bisa menghapus edge selain pertama kali.

Disini sudah terlihat bahwa pada vertex 1 sudah tidak terhubung ke vertex 2 dan 3 dan sebaliknya karena menggunakan graph undirected.

```

145 graph.printGraph();
146
147 graph.addEdge( source: 0, destination: 1);
148 graph.addEdge( source: 0, destination: 4);
149 graph.addEdge( source: 1, destination: 2);
150 graph.addEdge( source: 1, destination: 3);
151 graph.addEdge( source: 1, destination: 4);
152 graph.addEdge( source: 2, destination: 3);
153 graph.addEdge( source: 3, destination: 4);
154 graph.addEdge( source: 3, destination: 0);
155 graph.printGraph();
156 graph.degree( source: 2);
157 System.out.println();
158 graph.removeEdge( source: 1, destination: 2);
159 graph.printGraph();
160 System.out.println();
161 graph.removeEdge( source: 1, destination: 3);
162 graph.printGraph();
163 }
164 }
165
166
167
168
169
170
171

```

Run: Graph_26

```

"C:\Users\Asus TUF DT\.jdk\openjdk-17.0.2\bin\
Masukkan jenis graph : (directed / undirected)
undirected
Vertex 0 terhubung dengan : 3 4 1
Vertex 1 terhubung dengan : 4 3 2 0
Vertex 2 terhubung dengan : 3 1
Vertex 3 terhubung dengan : 0 4 2 1
Vertex 4 terhubung dengan : 3 1 0

degree vertex 2 : 2

Vertex 0 terhubung dengan : 3 4 1
Vertex 1 terhubung dengan : 4 3 0
Vertex 2 terhubung dengan : 3
Vertex 3 terhubung dengan : 0 4 2 1
Vertex 4 terhubung dengan : 3 1 0

Vertex 0 terhubung dengan : 3 4 1
Vertex 1 terhubung dengan : 4 0
Vertex 2 terhubung dengan : 3
Vertex 3 terhubung dengan : 0 4 2
Vertex 4 terhubung dengan : 3 1 0

Process finished with exit code 0

```

Soal Nomor 4

Ubahlah tipe data vertex pada seluruh graph pada praktikum 2.1 dan 2.2 dari Integer menjadi tipe generic agar dapat menerima semua tipe data dasar Java! Misalnya setiap vertex yang awalnya berupa angka 0,1,2,3, dst. selanjutnya ubah menjadi suatu nama daerah seperti Gresik, Bandung, Yogya, Malang, dst.

Jawaban

Source code graph

Membuat class generic

```

1 public class DataGeneric<T> {
2     T data;
3     public DataGeneric(T data) {
4         this.data = data;
5     }
6
7     public T getData() {
8         return data;
9     }
10 }

```

Modifikasi method main

```

108 ▶ public static void main(String[] args) throws Exception {
109     daerah[0] = new DataGeneric<>("Gresik ");
110     daerah[1] = new DataGeneric<>("Bandung");
111     daerah[2] = new DataGeneric<>("Yogyakarta ");
112     daerah[3] = new DataGeneric<>("Malang ");
113     daerah[4] = new DataGeneric<>("Kediri ");
114     daerah[5] = new DataGeneric<>("Jakarta");
115     // Graph_26 graph = new Graph_26(6);
116     Scanner input = new Scanner(System.in);
117     String source, destination;
118     int jumVertex, idxSource = 0, idxDestination = 0;
119     System.out.println("Masukkan jenis graph : (directed / undirected)");
120     jenis = input.next();
121     System.out.print("Masukkan jumlah vertices : ");
122     jumVertex = input.nextInt();
123     Graph_26 graph = new Graph_26(jumVertex);
124     System.out.print("Masukkan jumlah edges : ");
125     int edges = input.nextInt();
126     System.out.println("Masukkan edges: <source> <destination>");
127     for (int i = 0; i < edges; i++) {
128         source = input.next();
129         destination = input.next();
130         for (int j = 0; j < jumVertex; j++) {
131             if (source.equalsIgnoreCase(daerah[j].getData().trim())) {
132                 idxSource = j;
133             }
134             if (destination.equalsIgnoreCase(daerah[j].getData().trim())) {
135                 idxDestination = j;
136             }
137         }
138         graph.addEdge(idxSource, idxDestination);
139     }
140     graph.printGraph();

```

Modifikasi method printGraph

```

91 public void printGraph() throws Exception {
92     for (int i = 0; i < vertex; i++) {
93         if (list[i].size() > 0) {
94             System.out.print("Vertex " + daerah[i].getData() + " terhubung dengan : ");
95             for (int j = 0; j < list[i].size(); j++) {
96                 System.out.print(daerah[list[i].get(j)].getData() + " ");
97             }
98             System.out.println("");
99         }
100     }
101     System.out.println("");
102 }

```

Modifikasi method degree

```

30 public void degree(int source) throws Exception {
31     if (graphType(jenis)) {
32         System.out.println("degree vertex " + daerah[source].getData() + " : " + list[source].size());
33     } else {
34         //indegree
35         int k, totalIn = 0, totalOut = 0;
36         for (int i = 0; i < vertex; i++) {
37             for (int j = 0; j < list[i].size(); j++) {
38                 if (list[i].get(j) == source) {
39                     ++totalIn;
40                 }
41             }
42             // outDegree
43             for (k = 0; k < list[source].size(); k++) {
44                 list[source].get(k);
45             }
46             totalOut = k;
47         }
48         System.out.println("Indegree dari vertex " + daerah[source].getData() + " : " + totalIn);
49         System.out.println("Outdegree dari vertex " + daerah[source].getData() + " : " + totalOut);
50         System.out.println("degree vertex " + daerah[source].getData() + " : " + (totalIn+totalOut));
51     }

```

Menambahkan variabel generic dan modifikasi class addEdge

```

1 import java.util.Scanner;
2
3 public class Graph_26 {
4     int vertex;
5     DoubleLinkedList_26[] list;
6     public static String jenis;
7     private static DataGeneric <String>[] daerah = new DataGeneric[6];
8
9     public Graph_26(int vertex) {
10         this.vertex = vertex;
11         list = new DoubleLinkedList_26[vertex];
12         for (int i = 0; i < vertex; i++) {
13             list[i] = new DoubleLinkedList_26();
14         }
15     }
16
17     public void addEdge(int source, int destination) {
18         if (graphType(jenis)) {
19             list[source].addFirst(destination);
20             list[destination].addFirst(source);
21         } else {
22             list[source].addFirst(destination);
23         }

```

Source code graphArray

Menambah variabel generic

```

1 import java.util.Scanner;
2
3 public class graphArray_26 {
4     private final int vertices;
5     private final int [][] twoD_array;
6     private static DataGeneric<String>[] daerah = new DataGeneric[5];
7

```

Modifikasi method main

```

public static void main(String[] args) {
    daerah[0] = new DataGeneric<>("Gresik ");
    daerah[1] = new DataGeneric<>("Bandung");
    daerah[2] = new DataGeneric<>("Yogya ");
    daerah[3] = new DataGeneric<>("Malang ");
    daerah[4] = new DataGeneric<>("Kediri ");
    int v, e, count = 1, indexTo = 0, indexFrom = 0;
    String to, from;
    Scanner sc = new Scanner(System.in);
    graphArray_26 graph;
    try {
        System.out.println("Masukkan jumlah vertices: ");
        v = sc.nextInt();
        System.out.println("Masukkan jumlah edges: ");
        e = sc.nextInt();
        graph = new graphArray_26(v);
        System.out.println("Masukkan edges: <to> <from>");
        while (count <= e) {
            to = sc.next();
            from = sc.next();
            for (int i = 0; i < v; i++) {
                if (to.equalsIgnoreCase(daerah[i].getData().trim())) {
                    indexTo = i;
                }
                if (from.equalsIgnoreCase(daerah[i].getData().trim())) {
                    indexFrom = i;
                }
            }
            graph.makeEdge(indexTo, indexFrom + 1, 1);
            count++;
        }
        System.out.println("Array 2D sebagai representasi graph sbb: ");
        System.out.print("\t\t");
        for (int i = 0; i < v; i++) {
            System.out.print(daerah[i].getData() + " ");
        }
        System.out.println();

        for (int i = 0; i < v; i++) {
            System.out.print(daerah[i].getData() + " \t");
            for (int j = 1; j <= v; j++) {
                System.out.print(graph.getEdge(i, j) + " \t\t");
            }
            System.out.println();
        }
    } catch (Exception E) {
        System.out.println("Error. Silakan cek kembali\n" + E.getMessage());
    }
    sc.close();
}

```


Output graph

```
"C:\Users\Asus TUF DT\.jdk\openjdk-17.0.2\bin\java.exe" "-javaagent
Masukkan jenis graph : (directed / undirected)
undirected
Masukkan jumlah vertices : 6
Masukkan jumlah edges : 8
Masukkan edges: <source> <destination>
gresik bandung
gresik kediri
bandung yogya
bandung malang
bandung kediri
yogya malang
malang kediri
malang gresik
Vertex Gresik terhubung dengan : Malang Kediri Bandung
Vertex Bandung terhubung dengan : Kediri Malang Yogya Gresik
Vertex Yogya terhubung dengan : Malang Bandung
Vertex Malang terhubung dengan : Gresik Kediri Yogya Bandung
Vertex Kediri terhubung dengan : Malang Bandung Gresik

Process finished with exit code 0
```

Output graphArray

```
"C:\Users\Asus TUF DT\.jdk\openjdk-17.0.2\bin\java.exe"
Masukkan jumlah vertices:
5
Masukkan jumlah edges:
6
Masukkan edges: <to> <from>
gresik bandung
gresik kediri
bandung yogya
bandung malang
bandung kediri
yogya malang
Array 2D sebagai representasi graph sbb:
      Gresik  Bandung  Yogya  Malang  Kediri
Gresik    0        1      0       0       1
Bandung    0        0      1       1       1
Yogya      0        0      0       1       0
Malang     0        0      0       0       0
Kediri     0        0      0       0       0

Process finished with exit code 0
```