Nama : Deanissa Sherly Sabilla

Kelas / Absen : SIB 1B / 06

Mata Kuliah : Pratikum Algoritma Struktur Data

Pertemuan : 15

12.1 Implementasi Graph menggunakan Linked List

12.1.1 Tahapan Percobaan

Waktu percobaan (30 menit)

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

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()

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

```
1   public class Graph {
2
3   }
```

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

```
1   public class Graph {
2       int vertex;
3       LinkedList list[];
```

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

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
17         //add back edge (for undirected)
18         list[destination].addFirst(source);
19     }
```

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.

```
public void degree(int source) throws Exception{
    //degree undirected graph
    System.out.println("degree vertex "+source +" : "+list[source].size());

    //degree directed graph
        //inDegree
    int k,totalIn = 0, totalOut = 0;
    for (int i = 0; i <vertex ; i++) {
    for (int j = 0; j < list[i].size(); j++) {
         if(list[i].get(j)==source)
                ++totalIn;
    }
        //outDegree
    for (k = 0; k < list[source].size(); k++) {
        list[source].get(k);
    }
    totalOut = k;
    }
    System.out.println("Indegree dari vertex "+ source +" : "+totalIn);
    System.out.println("Outdegree dari vertex "+ source +" : "+totalOut);
    System.out.println("degree vertex "+source +" : "+(totalIn+totalOut));
}
```

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      }
```

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      }
```

9. Tambahkan method **printGraph()** untuk mencatak graph ter-*update.*

```
public void printGraph() throws Exception{
    for (int i = 0; i <vertex ; i++) {
        if(list[i].size()>0) {
            System.out.print("Vertex " + i + " terhubung dengan: ");
            for (int j = 0; j < list[i].size(); j++) {
                System.out.print(list[i].get(j) + " ");
            }
            System.out.println("");
        }
    }
    System.out.println(" ");
}
```

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 graph = new Graph(6);
74          graph.addEdge(0, 1);
75          graph.addEdge(0, 4);
76          graph.addEdge(1, 2);
77          graph.addEdge(1, 3);
78          graph.addEdge(1, 4);
79          graph.addEdge(2, 3);
80          graph.addEdge(3, 4);
81          graph.addEdge(3, 0);
82          graph.printGraph();
83          graph.degree(2);
84
85      }
```

11. Amati hasil running tersebut.
12. Tambahkan pemanggilan method **removeEdge()** sesuai potongan code di bawah ini pada method main(). Kemudian tampilkan graph tersebut.

```
89          graph.removeEdge(1, 2);
90          graph.printGraph();
```

13. Amati hasil running tersebut.
14. Uji coba penghapusan lintasan yang lain! Amati hasilnya!

2.1.2 Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

Hasil running pada langkah ke-11

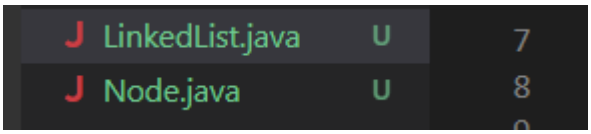
```
--- exec-maven-plugin:1.5.0:exec (default-cli)
Vertex 0 terhubung dengan: 3 4 1
Vertex 1 terhubung dengan: 4 3 2 0
Vertex 2 terhubung dengan: 3 1
Vertex 3 terhubung dengan: 0 4 2 1
Vertex 4 terhubung dengan: 3 1 0

degree vertex 2 : 2
Indegree dari vertex 2 : 2
Outdegree dari vertex 2 : 2
degree vertex 2 : 4
------------------------------------------------
BUILD SUCCESS
------------------------------------------------
```

Hasil running pada langkah ke-13

```
--- exec-maven-plugin:1.5.0:exec (default-cli)
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

------------------------------------------------
BUILD SUCCESS
------------------------------------------------
```

- **INPUT :**

Class Node dan Linked List mengambil dari Pratikum Double LinkedList

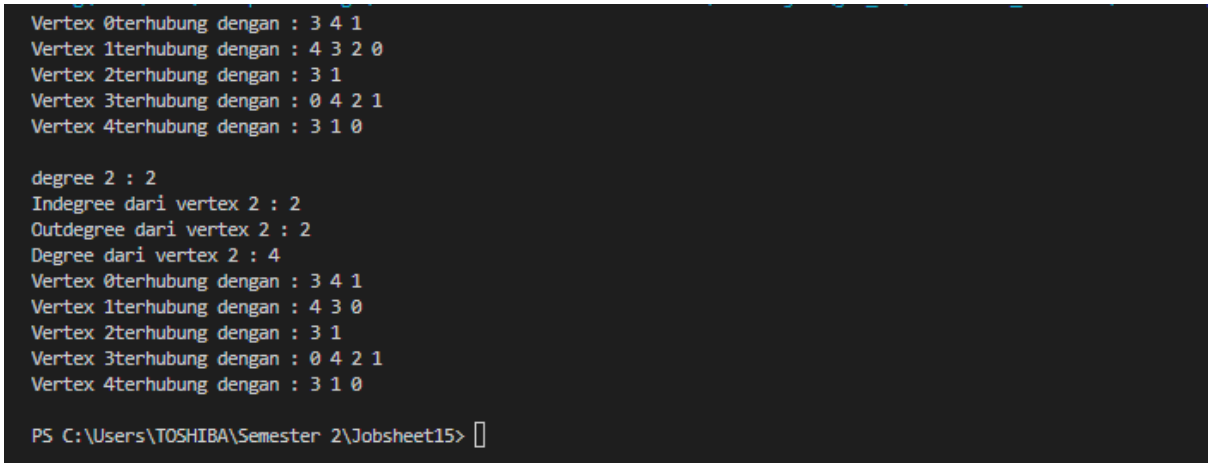


Class Graph

```
4   public class Graph {
19      public void degree (int source) throws Exception {
33          System.out.println("Indegree dari vertex " +source+ " : " +totalIn);
34          System.out.println("Outdegree dari vertex " +source+ " : " +totalOut);
35          System.out.println("Degree dari vertex " +source+ " : " +(totalIn+totalOut));
36      }
37      public void removeEdge (int source, int destination) throws Exception {
38          for (int i = 0; i < vertex; i++) {
39              if(i==destination) {
40                  list[source].remove(destination);
41              }
42          }
43      }
44      public void removeAllEdges() {
45          for (int i = 0; i < vertex; i++) {
46              list[i].clear();
47          }
48          System.out.println(("Graph berhasil dikosongkan"));
49      }
50      public void printGraph() throws Exception {
51          for (int i = 0; i < vertex; i++) {
52              if(list[i].size()>0) {
53                  System.out.print("Vertex "+i+ "terhubung dengan : ");
54                  for (int j = 0; j < list[i].size(); j++) {
55                      System.out.print(list[i].get(j) + " ");
56                  }
57                  System.out.println(x:" ");
58              }
59          }
60          System.out.println(x:" ");
61      }
```

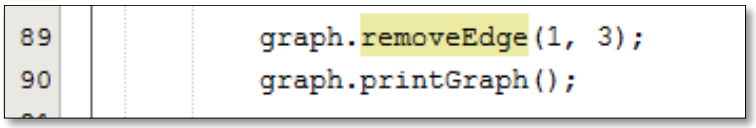
```
    Run | Debug
62      public static void main(String[] args) throws Exception {
63          Graph graph = new Graph(vertex:6);
64          graph.addEdge (source:0, destination:1);
65          graph.addEdge (source:0, destination:4);
66          graph.addEdge (source:1, destination:2);
67          graph.addEdge (source:1, destination:3);
68          graph.addEdge (source:1,destination:4);
69          graph.addEdge (source:2, destination:3);
70          graph.addEdge (source:3, destination:4);
71          graph.addEdge (source:3, destination:0);
72          graph.printGraph();
73          graph.degree (source:2);
74          graph.removeEdge(source:1, destination:2);
75          graph.printGraph();
76      }
77  }
```

- **HASIL OUTPUT :**



2.1.3 Pertanyaan Percobaan

1. Sebutkan beberapa jenis (minimal 3) algoritma yang menggunakan dasar Graph, dan apakah kegunaan algoritma-algoritma tersebut?
   - **DFS (Depth-First Search)**: untuk melakukan pencarian dalam graph.
   - **BFS (Breadth-First Search):** untuk melakukan pencarian dalam graph secara menyeluruh dengan menjelajahi semua tetangga dari node saat ini sebelum melanjutkan ke level berikutnya.
   - **Dijkstra:** untuk menemukan jalur terpendek dari satu node ke node lainnya dalam sebuah graph yang berbobot dan berarah atau tidak berarah.
2. Pada class Graph terdapat array bertipe LinkedList, yaitu LinkedList list[]. Apakah tujuan pembuatan variabel tersebut ?
   - Untuk menyimpan adjacency list dari setiap vertex dalam graph.
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**
   - Yaitu agar efisien untuk menambahkan elemen dalam adjacency list pada struktur graph, mendukung kesederhanaan dalam implementasi graph tersebut.
4. Bagaimana cara mendeteksi prev pointer pada saat akan melakukan penghapusan suatu edge pada graph ?
   - Menggunakan Iterator untuk menghapus elemen dari LinkedList saat iterasi, yang menghindari kebutuhan untuk pointer sebelumnya (prev pointer).
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 ?



   - Disebabkan oleh cara iterasi dan penghapusan elemen dari LinkedList. Jika edge yang dihapus bukan lintasan pertama, iterasi mungkin tidak menemukan dan menghapus elemen dengan benar karena LinkedList di Java dapat berisi elemen yang sama lebih dari sekali. Solusinya dengan memastikan setiap occurrence dari edge yang ingin dihapus ditemukan dan dihapus.

2.2 Implementasi Graph menggunakan Matriks

Kegiatan praktikum 2 merupakan implementasi Graph dengan Matriks. Silakan lakukan langkah-langkah percobaan praktikum berikut ini, kemudian verifikasi hasilnya. Setelah itu jawablah pertanyaan terkait percobaan yang telah Anda lakukan.

2.2.1 Tahapan Percobaan

Waktu percobaan: 30 menit

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

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**!

```
public class graphArray {

    private final int vertices;
    private final int[][] twoD_array;
```

2. Buatlah konstruktor **graphArray** sebagai berikut!

```
public graphArray(int v)
{
    vertices = v;
    twoD_array = new int[vertices + 1][vertices + 1];
}
```

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

```
14      public void makeEdge(int to, int from, int edge)
15      {
16          try
17          {
18              twoD_array[to][from] = edge;
19          }
20          catch (ArrayIndexOutOfBoundsException index)
21          {
22              System.out.println("Vertex tidak ada");
23          }
24      }
```

Untuk menampilkan suatu lintasan diperlukan pembuatan method **getEdge()** berikut.

```
26          public int getEdge(int to, int from)
27          {
28              try
29              {
30                  return twoD_array[to][from];
31              }
32              catch (ArrayIndexOutOfBoundsException index)
33              {
34                  System.out.println("Vertex tidak ada");
35              }
36              return -1;
37          }
```

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 graph;
    try {
        System.out.println("Masukkan jumlah vertices: ");
        v = sc.nextInt();
        System.out.println("Masukkan jumlah edges: ");
        e = sc.nextInt();

        graph = new graphArray(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();
}
```

5. Jalankan class **graphArray** dan amati hasilnya!

2.2.2 Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

```
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
BUILD SUCCESSFUL (total time: 47 seconds)
```

- **INPUT :**

GraphArray > GraphArray.java > GraphArray > main(String[])

```java
package GraphArray;
import java.util.Scanner;

public class GraphArray {
    private final int vertices;
    private final int[] [] twoD_array;

    public GraphArray( int v) {
        vertices = v;
        twoD_array = new int [vertices + 1] [vertices + 1];
    }
    public void makeEdge (int to, int from, int edge) {
        try {
            twoD_array [to] [from] = edge;
        }
        catch (ArrayIndexOutOfBoundsException index) {
            System.out.println(x:"Vertex tidak ada");
        }
    }
    public int getEdge (int to, int from) {
        try {
            return twoD_array [to][from];
        }
        catch (ArrayIndexOutOfBoundsException index) {
            System.out.println(x:"Vertex tidak ada");
        }
        return -1;
    }
```

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

        System.out.println("Masukkan edges : <to> <from>");
        while (count <= e) {
            to = sc.nextInt();
            from = sc.nextInt();
            graph.makeEdge(to, from, edge:1);
            count++;
        }
        System.out.println("Array 2D sebagai represntasi graph sbb : ");
        System.out.print(" ");
        for (int i = 0; i <= v; i++) {
            System.out.print(i+ " ");
        }
        System.out.println();

        for (int i = 0; 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 Silahkan cek kembali\n" +E.getMessage());
    }
    sc.close();
}
```

➢ HASIL OUTPUT :

```
Masukkan jumlah verticas :
5
Masukkan jumlah edges :
6
Masukkan edges : <to> <from>
1 2
1 5
2 3
2 4
2 5
3 4
Array 2D sebagai represntasi graph sbb :
 0 1 2 3 4 5
0 0 0 0 0 0
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
PS C:\Users\TOSHIBA\Semester 2\Jobsheet15>
```

2.2.3 Pertanyaan Percobaan

1. Apakah perbedaan degree/derajat pada *directed* dan *undirected graph*?
   - Degree (derajat) sebuah node adalah jumlah sisi yang bersebelahan dengan node tersebut atau jumlah garis yang keluar dari node, sedangkan undirected setiap sisinya (edge) tidak memiliki arah.
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      }
```

   - Untuk mempermudah manipulasi dan pengaksesan elemen-elemen dalam adjacency matrix dengan indeks yang lebih intuitif.
3. Apakah kegunaan method **getEdge()** ?
   - Untuk memeriksa dan mendapatkan nilai yang menunjukkan ada atau tidaknya edge (sisi) antara dua vertex tertentu. Nilai yang dikembalikan oleh metode ini menunjukkan apakah ada koneksi (edge) dari vertex from ke vertex to.
4. Termasuk jenis graph apakah uji coba pada praktikum 2.2?
   - Termasuk jenis menggunakan graph yang diimplementasikan dengan adjacency matrix. Dalam pratikum ini, graph yang diuji coba adalah graph tidak berarah (undirected graph). Ini ditunjukkan oleh metode makeEdge yang hanya mengatur satu arah dari to ke from tanpa pengaturan arah sebaliknya.
5. Mengapa pada method main harus menggunakan *try-catch Exception* ?
   - Untuk menangani potensi kesalahan atau pengecualian yang mungkin terjadi selama eksekusi program

3. Tugas Praktikum

1. Tambahkan scanner pada method AddEdge praktikum 2.1 untuk menerima input Edge

Menambahkan scanner input untuk AddEdge :

```
19      public void addEdge (int source, int destination) {
20          list[source].addFirst(destination);
21          list[destination].addFirst(source);
22      }
23      public void addEdgeInput() {
24          Scanner sc = new Scanner(System.in);
25          System.out.print(s:"Masukkan jumlah vertex: ");
26          int source = sc.nextInt();
27          System.out.print(s:"Masukkan jumlah destination vertex: ");
28          int destination = sc.nextInt();
29          addEdge(source, destination);
30      }
        Run | Debug
74      public static void main(String[] args) throws Exception {
75          Graph graph = new Graph(vertex:6);
76
77          Scanner sc = new Scanner(System.in);
78          System.out.print(s:"Masukkan jumlah edges: ");
79          int edges = sc.nextInt();
80
81          for (int i = 0; i < edges; i++) {
82              graph.addEdgeInput();
83          }
84          graph.printGraph();
85          graph.degree (source:2);
86          graph.removeEdge(source:1, destination:2);
87          graph.printGraph();
```

Maka, hasil output :

```
Masukkan jumlah edges: 8
Masukkan jumlah vertex: 0
Masukkan jumlah destination vertex: 1
Masukkan jumlah vertex: 0
Masukkan jumlah destination vertex: 4
Masukkan jumlah vertex: 1
Masukkan jumlah destination vertex: 2
Masukkan jumlah vertex: 1
Masukkan jumlah destination vertex: 2
Masukkan jumlah vertex: 1
Masukkan jumlah destination vertex: 4
Masukkan jumlah vertex: 2
Masukkan jumlah destination vertex: 3
Masukkan jumlah vertex: 3
Masukkan jumlah destination vertex: 4
Masukkan jumlah vertex: 3
Masukkan jumlah destination vertex: 0
Vertex 0terhubung dengan : 3 4 1
Vertex 1terhubung dengan : 4 2 2 0
Vertex 2terhubung dengan : 3 1 1
Vertex 3terhubung dengan : 0 4 2
Vertex 4terhubung dengan : 3 1 0

degree 2 : 3
Indegree dari vertex 2 : 3
Outdegree dari vertex 2 : 3
Degree dari vertex 2 : 6
Vertex 0terhubung dengan : 3 4 1
Vertex 1terhubung dengan : 4 2 0
Vertex 2terhubung dengan : 3 1 1
Vertex 3terhubung dengan : 0 4 2
Vertex 4terhubung dengan : 3 1 0
```

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

Menambahkan graphType dengan tipe Boolean :

```java
public class Graph {
        int vertex;
        LinkedList list[];
        boolean isDirected;

        public Graph(int vertex, boolean isDirected) {
            this.vertex = vertex;
            this.isDirected = isDirected;
            list = new LinkedList[vertex];
            for (int i = 0; i < vertex; i++) {
                list[i] = new LinkedList();
            }
        }
    public boolean graphType() {
        return isDirected;
    }

    public void addEdge(int source, int destination) {
        list[source].addFirst(destination);
        if (!isDirected) {
            list[destination].addFirst(source);
        }
    }
  public static void main(String[] args) throws Exception {
     Scanner sc = new Scanner(System.in);
     System.out.print(s:"Masukkan jumlah vertices: ");
     int vertices = sc.nextInt();
     System.out.print(s:"Apakah grafiknya berarah? (true/false) ");
     boolean isDirected = sc.nextBoolean();

     Graph graph = new Graph(vertices, isDirected);

     System.out.print(s:"Masukkan jumlah edges: ");
     int edges = sc.nextInt();
     for (int i = 0; i < edges; i++) {
         graph.addEdgeInput();
     }
     graph.printGraph();
     graph.degree (source:2);
     graph.removeEdge(source:1, destination:2);
     graph.printGraph();
```

Maka, hasil output :

False

```
Masukkan jumlah vertices: 6
Apakah grafiknya berarah? (true/false) false
Masukkan jumlah edges: 8
Masukkan jumlah vertex: 0
Masukkan jumlah destination vertex: 1
Masukkan jumlah vertex: 0
Masukkan jumlah destination vertex: 4
Masukkan jumlah vertex: 1
Masukkan jumlah destination vertex: 2
Masukkan jumlah vertex: 1
Masukkan jumlah destination vertex: 3
Masukkan jumlah vertex: 1
Masukkan jumlah destination vertex: 4
Masukkan jumlah vertex: 2
Masukkan jumlah destination vertex: 3
Masukkan jumlah vertex: 3
Masukkan jumlah destination vertex: 4
Masukkan jumlah vertex: 3
Masukkan jumlah destination vertex: 0
Vertex 0terhubung dengan : 3 4 1
Vertex 1terhubung dengan : 4 3 2 0
Vertex 2terhubung dengan : 3 1
Vertex 3terhubung dengan : 0 4 2 1
Vertex 4terhubung dengan : 3 1 0

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

True

```
Masukkan jumlah vertices: 6
Apakah grafiknya berarah? (true/false) true
Masukkan jumlah edges: 8
Masukkan jumlah vertex: 0
Masukkan jumlah destination vertex: 1
Masukkan jumlah vertex: 0
Masukkan jumlah destination vertex: 4
Masukkan jumlah vertex: 1
Masukkan jumlah destination vertex: 2
Masukkan jumlah vertex: 1
Masukkan jumlah destination vertex: 3
Masukkan jumlah vertex: 1
Masukkan jumlah destination vertex: 4
Masukkan jumlah vertex: 2
Masukkan jumlah destination vertex: 3
Masukkan jumlah vertex: 3
Masukkan jumlah destination vertex: 4
Masukkan jumlah vertex: 3
Masukkan jumlah destination vertex: 0
Vertex 0terhubung dengan : 4 1
Vertex 1terhubung dengan : 4 3 2
Vertex 2terhubung dengan : 3
Vertex 3terhubung dengan : 0 4

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

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

Menambah pada method removeEdge() :

```java
    public void removeEdge(int source, int destination) throws Exception {
        list[source].remove((Integer) destination);
        if (!isDirected) {
            list[destination].remove((Integer) source);
        }
    }
    public void removeAllEdges() {
        for (int i = 0; i < vertex; i++) {
            list[i].clear();
        }
        System.out.println(x:"Graph berhasil dikosongkan");
    }
```

```java
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        System.out.print(s:"Masukkan jumlah vertices: ");
        int vertices = sc.nextInt();
        System.out.print(s:"Apakah grafiknya berarah? (true/false) ");
        boolean isDirected = sc.nextBoolean();

        Graph graph = new Graph(vertices, isDirected);

        System.out.print(s:"Masukkan jumlah edges: ");
        int edges = sc.nextInt();
        for (int i = 0; i < edges; i++) {
            graph.addEdgeInput();
        }
        graph.printGraph();
        try {
            graph.degree(source:2);
        } catch (Exception e) {
            e.printStackTrace();
        }
        graph.removeEdge(source:1, destination:2);
        graph.printGraph();
```

Maka, hasil output :

```
Masukkan jumlah vertices: 6
Apakah grafiknya berarah? (true/false) true
Masukkan jumlah edges: 8
Masukkan jumlah vertex: 0
Masukkan jumlah destination vertex: 1
Masukkan jumlah vertex: 0
Masukkan jumlah destination vertex: 4
Masukkan jumlah vertex: 1
Masukkan jumlah destination vertex: 2
Masukkan jumlah vertex: 1
Masukkan jumlah destination vertex: 3
Masukkan jumlah vertex: 1
Masukkan jumlah destination vertex: 4
Masukkan jumlah vertex: 2
Masukkan jumlah destination vertex: 3
Masukkan jumlah vertex: 3
Masukkan jumlah destination vertex: 4
Masukkan jumlah vertex: 3
Masukkan jumlah destination vertex: 0
Vertex 0terhubung dengan : 4 1
Vertex 1terhubung dengan : 4 3 2
Vertex 2terhubung dengan : 3
Vertex 3terhubung dengan : 0 4

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

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.

Pratikum 2.1

Modifikasi bagian class Graph :

Graph > Graph.java > Graph<T> > addEdge(T, T)

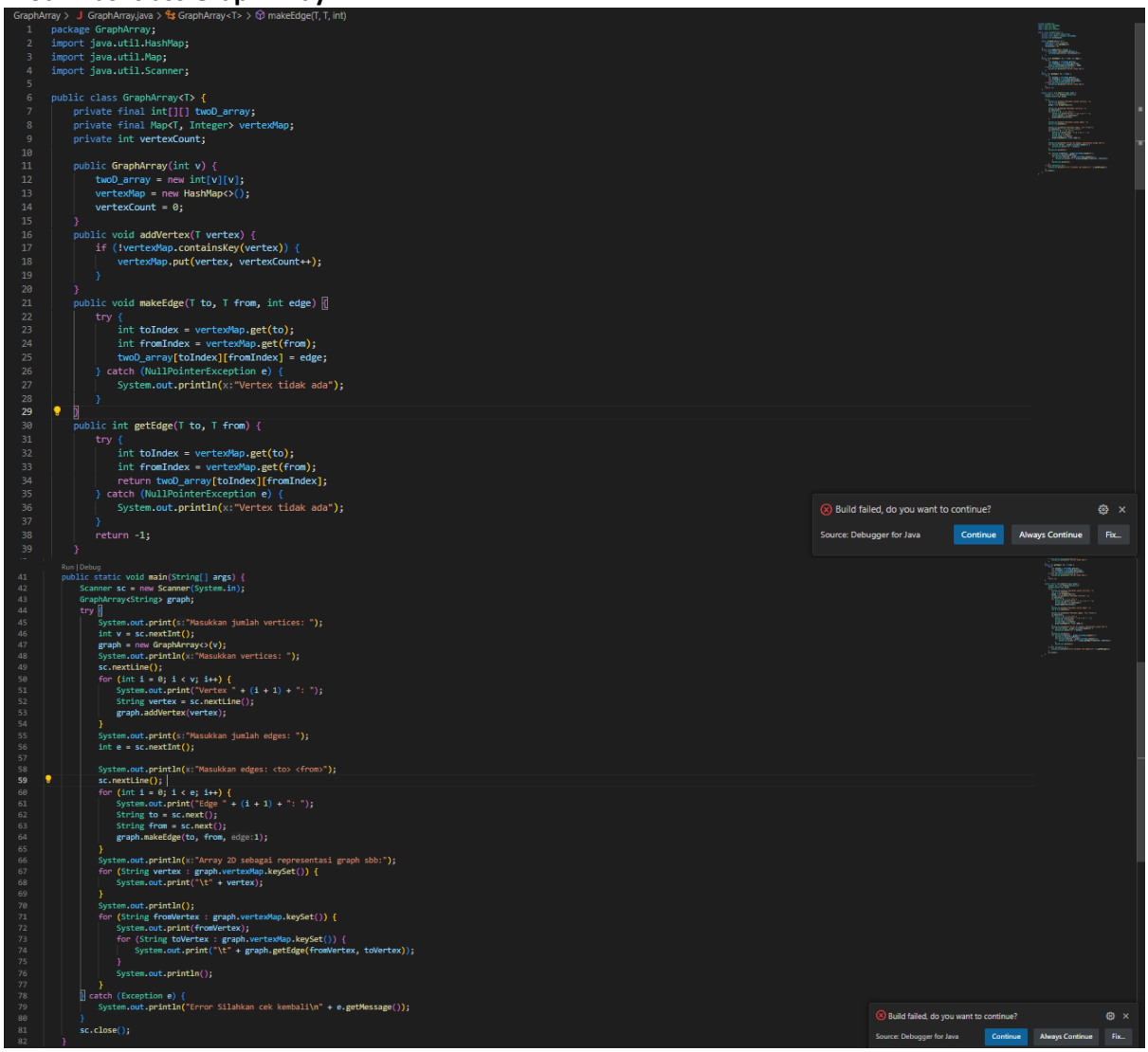
```java
package Graph;

import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;
import java.util.Scanner;

public class Graph<T> {
    private Map<T, LinkedList<T>> adjacencyList;
    private boolean isDirected;

    public Graph(boolean isDirected) {
        this.isDirected = isDirected;
        adjacencyList = new HashMap<>();
    }
    public boolean isDirected() {
        return isDirected;
    }
    public void addVertex(T vertex) {
        adjacencyList.putIfAbsent(vertex, new LinkedList<>());
    }
    public void addEdge(T source, T destination) {
        adjacencyList.get(source).addFirst(destination);
        if (!isDirected) {
            adjacencyList.get(destination).addFirst(source);
        }
    }
    public void addEdgeInput() {
        Scanner sc = new Scanner(System.in);
        System.out.print(s:"Masukkan vertex: ");
        T source = (T) sc.next();
        System.out.print(s:"Masukkan destination vertex: ");
        T destination = (T) sc.next();
        addEdge(source, destination);
    }
    public void degree(T vertex) {
        int inDegree = 0, outDegree = adjacencyList.get(vertex).size();

        for (T key : adjacencyList.keySet()) {
            for (T value : adjacencyList.get(key)) {
                if (value.equals(vertex)) {
                    inDegree++;
                }
            }
        }
        System.out.println("Indegree of vertex " + vertex + ": " + inDegree);
        System.out.println("Outdegree of vertex " + vertex + ": " + outDegree);
        System.out.println("Degree of vertex " + vertex + ": " + (inDegree + outDegree));
    }
    public void removeEdge(T source, T destination) {
        adjacencyList.get(source).remove(destination);
        if (!isDirected) {
            adjacencyList.get(destination).remove(source);
        }
    }
    public void removeAllEdges() {
        for (T vertex : adjacencyList.keySet()) {
            adjacencyList.get(vertex).clear();
        }
        System.out.println(x:"Graph berhasil dikosongkan");
    }
    public void printGraph() {
        for (T vertex : adjacencyList.keySet()) {
            System.out.print("Vertex " + vertex + " terhubung dengan : ");
            for (T neighbor : adjacencyList.get(vertex)) {
                System.out.print(neighbor + " ");
            }
            System.out.println();
        }
    }

    Run | Debug
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print(s:"Apakah graphnya berarah  (true/false)? ");
        boolean isDirected = sc.nextBoolean();

        Graph<String> graph = new Graph<>(isDirected);

        System.out.print(s:"Masukkan jumlah vertices: ");
        int vertices = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < vertices; i++) {
            System.out.print("Masukkan vertex " + (i + 1) + ": ");
            String vertex = sc.nextLine();
            graph.addVertex(vertex);
        }

        System.out.print(s:"Masukkan jumlah edges: ");
        int edges = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < edges; i++) {
            graph.addEdgeInput();
        }
        graph.printGraph();
        System.out.print(s:"Masukkan vertex untuk mencari degree : ");
        String vertex = sc.next();
        graph.degree(vertex);
        System.out.print(s:"Masukkan vertex untuk menghapus edges : ");
        String source = sc.next();
        System.out.print(s:"Masukkan destination vertex untuk menghapus edges : ");
        String destination = sc.next();
        graph.removeEdge(source, destination);
        graph.printGraph();
    }
}
```

Hasil Output :

```
Apakah graphnya berarah  (true/false)? false
Masukkan jumlah vertices: 5
Masukkan vertex 1: Gresik
Masukkan vertex 2: Bandung
Masukkan vertex 3: Yogya
Masukkan vertex 4: Malang
Masukkan vertex 5: Surabaya
Masukkan jumlah edges: 6
Masukkan vertex: Gresik
Masukkan destination vertex: Bandung
Masukkan vertex: Gresik
Masukkan destination vertex: Yogya
Masukkan vertex: Bandung
Masukkan destination vertex: Malang
Masukkan vertex: Yogya
Masukkan destination vertex: Malang
Masukkan vertex: Malang
Masukkan destination vertex: Surabaya
Masukkan vertex: Surabaya
Masukkan destination vertex: Gresik
Vertex Surabaya terhubung dengan : Gresik Malang
Vertex Yogya terhubung dengan : Malang Gresik
Vertex Gresik terhubung dengan : Surabaya Yogya Bandung
Vertex Bandung terhubung dengan : Malang Gresik
Vertex Malang terhubung dengan : Surabaya Yogya Bandung
Masukkan vertex untuk mencari degree : Malang
Indegree of vertex Malang: 3
Outdegree of vertex Malang: 3
Degree of vertex Malang: 6
Masukkan vertex untuk menghapus edges : Malang
Masukkan destination vertex untuk menghapus edges : Surabaya
Vertex Surabaya terhubung dengan : Gresik
Vertex Yogya terhubung dengan : Malang Gresik
Vertex Gresik terhubung dengan : Surabaya Yogya Bandung
Vertex Bandung terhubung dengan : Malang Gresik
Vertex Malang terhubung dengan : Yogya Bandung
PS C:\Users\TOSHIBA\Semester 2\Jobsheet15>
```

Pratikum 2.2

Modifikasi class GraphArray :



```
GraphArray > GraphArray.java > GraphArray<T> > makeEdge(T, T, int)
```

```java
package GraphArray;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class GraphArray<T> {
    private final int[][] twoD_array;
    private final Map<T, Integer> vertexMap;
    private int vertexCount;

    public GraphArray(int v) {
        twoD_array = new int[v][v];
        vertexMap = new HashMap<>();
        vertexCount = 0;
    }
    public void addVertex(T vertex) {
        if (!vertexMap.containsKey(vertex)) {
            vertexMap.put(vertex, vertexCount++);
        }
    }
    public void makeEdge(T to, T from, int edge) {
        try {
            int toIndex = vertexMap.get(to);
            int fromIndex = vertexMap.get(from);
            twoD_array[toIndex][fromIndex] = edge;
        } catch (NullPointerException e) {
            System.out.println(x:"Vertex tidak ada");
        }
    }
    public int getEdge(T to, T from) {
        try {
            int toIndex = vertexMap.get(to);
            int fromIndex = vertexMap.get(from);
            return twoD_array[toIndex][fromIndex];
        } catch (NullPointerException e) {
            System.out.println(x:"Vertex tidak ada");
        }
        return -1;
    }
```

Build failed, do you want to continue?
Source: Debugger for Java
Continue | Always Continue | Fix...

```
    Run | Debug
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        GraphArray<String> graph;
        try {
            System.out.print(s:"Masukkan jumlah vertices: ");
            int v = sc.nextInt();
            graph = new GraphArray<>(v);
            System.out.println(x:"Masukkan vertices: ");
            sc.nextLine();
            for (int i = 0; i < v; i++) {
                System.out.print("Vertex " + (i + 1) + ": ");
                String vertex = sc.nextLine();
                graph.addVertex(vertex);
            }
            System.out.print(s:"Masukkan jumlah edges: ");
            int e = sc.nextInt();

            System.out.println(x:"Masukkan edges: <to> <from>");
            sc.nextLine();
            for (int i = 0; i < e; i++) {
                System.out.print("Edge " + (i + 1) + ": ");
                String to = sc.next();
                String from = sc.next();
                graph.makeEdge(to, from, edge:1);
            }
            System.out.println(x:"Array 2D sebagai representasi graph sbb:");
            for (String vertex : graph.vertexMap.keySet()) {
                System.out.print("\t" + vertex);
            }
            System.out.println();
            for (String fromVertex : graph.vertexMap.keySet()) {
                System.out.print(fromVertex);
                for (String toVertex : graph.vertexMap.keySet()) {
                    System.out.print("\t" + graph.getEdge(fromVertex, toVertex));
                }
                System.out.println();
            }
        } catch (Exception e) {
            System.out.println("Error Silahkan cek kembali\n" + e.getMessage());
        }
        sc.close();
    }
}
```

Build failed, do you want to continue?
Source: Debugger for Java
Continue | Always Continue | Fix...

Hasil Output :

```
Masukkan jumlah vertices: 4
Masukkan vertices:
Vertex 1: Gresik
Vertex 2: Bandung
Vertex 3: Yogya
Vertex 4: Malang
Masukkan jumlah edges: 4
Masukkan edges: <to> <from>
Edge 1: Gresik Bandung
Edge 2: Bandung Yogya
Edge 3: Yogya Malang
Edge 4: Malang Gresik
Array 2D sebagai representasi graph sbb:
        Yogya   Gresik  Bandung Malang
Yogya   0       0       0       1
Gresik  0       0       1       0
Bandung 1       0       0       0
Malang  0       1       0       0
PS C:\Users\TOSHIBA\Semester 2\Jobsheet15>
```