PRAKTIKUM 30

GRAPH 2

A. TUJUAN PEMBELAJARAN

1. Memahami mengenai Konsep Graph dan istilah-istilah yang terdapat pada Graph
2. Memahami implementasi Graph ke dalam bahasa pemrograman

B. DASAR TEORI

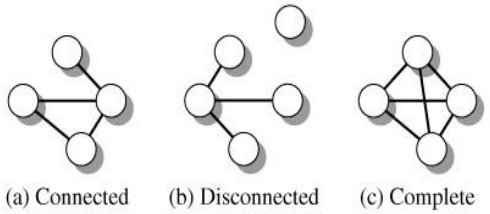
B.1 KONSEP GRAPH

Sebuah graph G=<V,E> terdiri dari sekumpulan titik (*vertices*) V dan sekumpulan garis (*edges*) E. Sebuah garis e = (u,v) menghubungkan dua titik u dan v. Self – loop adalah garis yang menghubungkan ke titik itu sendiri.

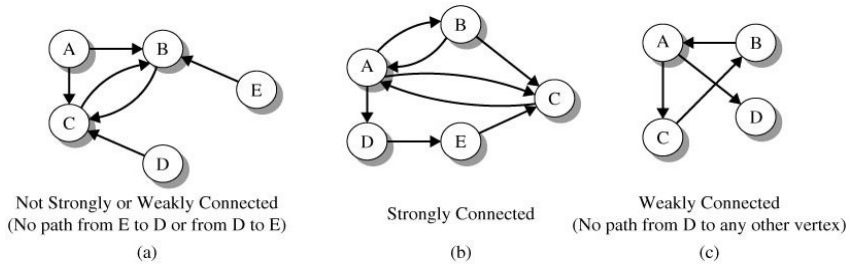
Vertices = {v_1, v_2, v_3, …, v_m}

Edges = {e_1, e_2, e_3, …, e_n}

Dua titik u, v dikatakan *adjacent* to u, jika u dan v dihubungkan dengan garis. Path antara dua titik v dan w adalah sekumpulan garis yang menghubungkan titik v ke titik w. Panjang path adalah jumlah garis dalam path. Graph dikatakan connected/terhubung jika ada sebuah path yang menghubungkan sembarang dua titik berbeda. Complete graph adalah graph yang terhubung (connected graph) dan setiap pasang titik dihubungkan dengan garis. Digraph disebut strongly connected jika ada sebuah path dari sembarang titik ke semua titik. Digraph disebut weakly conneted jika titik u dan v jika terdapat path (u,v) atau path(v,u).

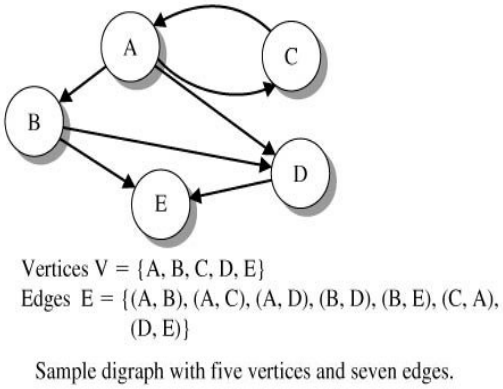


Gambar 30.1 (a) Graph Connected (b) Disconnected (c) Complete



Gambar 30.2 (a) Graph Weakly Connected (b) Strongly Connected (c) Weakly Connected

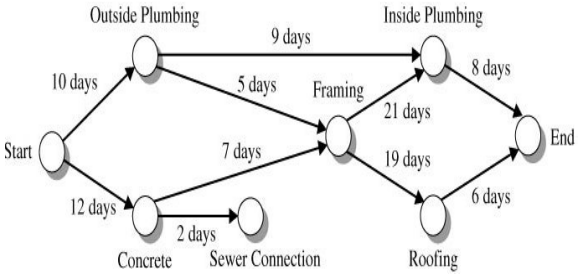
Pada graph berarah (*directed graph/digraph*), setiap garis mempunyai arah dari u ke v dan dituliskan sebagai pasangan <u,v> atau u->v. Pada graph tak berarah (*undirected graph*), garis tidak mempunyai arah dan dituliskan sebagai pasangan {u,v} atau u<->v. Graph tak berarah merupakan graph berarah jika setiap garis tak berarah {u,v} merupakan dua garis berarah <u,v> dan <v,u>.



Gambar 30.3 Contoh Graph Digraph

Jumlah edge yang keluar dari titik/vertex v tersebut adalah out-degree dari vertex v. Jumlah edge yan masuk di vertex v adalah in-degree vertex v.

Baik graph berarah maupun tak berarah dapat diberikan pembobot (*weight*). Pembobot diberikan untuk setiap garis. Hal ini biasanya digunakan untuk menggambarkan jarak antara dua kota, waktu penerbangan, harga tiket, kapasitas elektrik suatu kabel atau ukuran lain yang berhubungan dengan garis.

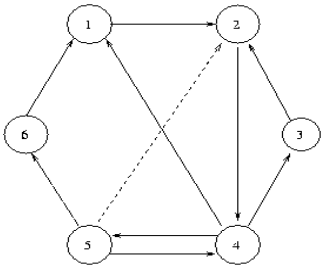


Gambar 30.4 Contoh penggunaan Graph

Graph banyak digunakan untuk menggambarkan jaringan dan peta jalan, jalan kereta api, lintasan pesawat, system perpipaan, saluran telepon, koneksi elektrik, ketergantungan diantara *task* pada sistem manufaktur dan lain-lain. Terdapat banyak hasil dan struktur penting yang didapatkan dari perhitungan dengan graph.

GRAPH DENGAN MATRIK *ADJACENCY*

Sebuah graph G= dapat direpresentasikan dengan matrik *adjacency* A sebagai |V|*|V|. Jika G berarah, A_{ij} = 1 jika dan hanya jika <v_i, v_j> berada pada E. Terdapat paling banyak $|V|^2$ garis pada E.

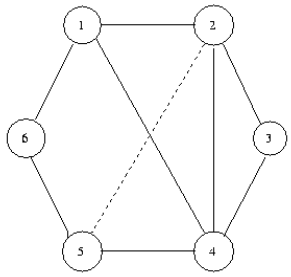


Gambar 30.5 Graph berarah

Matriks *adjacency* dari Graph berarah adalah sebagai berikut :

	1	2	3	4	5	6
1		1				
2				1		
3		1				
4	1		1		1	
5		1		1		1
6	1					

Jika G adalah graph tak berarah, A_{ij}=A_{ji}=1 jika {v_i,v_j} berada pada E dan A_{ij}=A_{ji}=false apabila G adalah graph berarah.



Gambar 30.6 Graph tak berarah

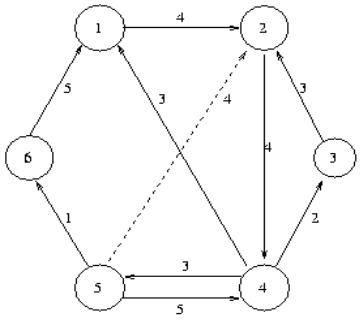
Matriks *adjacency* dari Graph berarah adalah sebagai berikut :

	1	2	3	4	5	6
1		1		1		1
2	1		1	1	1	
3		1		1		
4	1	1	1		1	
5		1		1		1
6	1				1	

Pada graph tak berarah, matriks *adjacency* dapat berupa matriks segitiga atas sebagai berikut:

	1	2	3	4	5	6
1		1		1		1
2			1	1	1	
3				1		
4					1	
5						1
6						

Baik graph berarah maupun tak berarah dapat diberikan pembobot (*weight*). Pembobot diberikan untuk setiap garis. Hal ini biasanya digunakan untuk menggambarkan jarak antara dua kota, waktu penerbangan, harga tiket, kapasitas elektrik suatu kabel atau ukuran lain yang berhubungan dengan garis. Pembobot biasanya disebut panjang garis, khususnya jika graph menggambarkan peta. Pembobot atau panjang dari suatu jalur atau siklus merupakan penjumlahan pembobot atau panjang dari komponen garis.

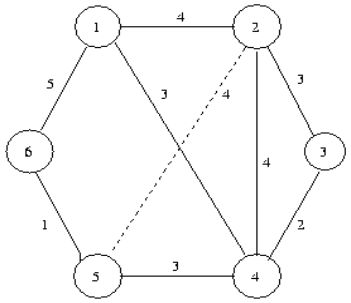


Gambar 30.7 Graph berarah dengan pembobot

Matriks *adjacency* dari graph dengan pembobot dapat digunakan untuk menyimpan pembobot dari setiap garis. Jika garis kehilangan nilai, kemungkinan nilai negative, nol atau bilangan besar menunjukkan nilai yang tak terbatas (*infinity*).

Matriks *adjacency* pada graph berarah dengan pembobot adalah sebagai berikut :

	1	2	3	4	5	6
1		4				
2				4		
3		3				
4	3		2		3	
5		4		5		1
6	5					



Gambar 30.8 Graph tak berarah dengan pembobot

Matriks *adjacency* pada graph tak berarah dengan pembobot adalah sebagai berikut :

	1	2	3	4	5	6
1		4		3		5
2	4		3	4	4	
3		3		2		
4	3	4	2		3	
5		4		3		1
6	5				1	

Atau berupa matriks segitiga atas sebagai berikut :

	1	2	3	4	5	6
1		4		3		5
2			3	4	4	
3				2		
4					3	
5						1
6						

B.2 IMPLEMENTASI GRAPH

INTERFACE GRAPH

Interface Graph berisi method-method dasar dari graph.

Tabel 30.1 Interface Graph

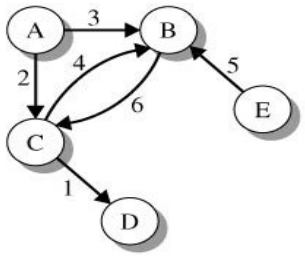
Interface Graph	
`boolean addEdge(T v1, T v2, int w)`	Jika edge (v1,v2) belum terdapat pada graph, tambahkan edge dengan bobot w dan mengembalikan nilai true. Jika edge(v1,v2) sudah ada maka mengembalikan nilai false. Jika v1 dan v2 bukan merupakan vertex di graph maka throws IIlegalArgumentException.
`boolean addVertex(T v)`	Jika vertex v tidak terdapat pada graph maka tambahkan pada graph dan mengembalikan nilai true. Jika vertex v sudah ada maka mengembalikan nilai false.
`void clear()`	Menghapus semua vertex dan edge pada graph
`boolean containsEdge(T v1, T v2)`	Mengembalikan nilai true jika terdapat sebuah edge dari v1 dan v2 di graph dan mengembalikan nilai false jika sebaliknya. Jika v1 atau v2 bukan merupakan vertex pada graph maka throws IIlegalArgumentException.
`boolean containsVertex(Object v)`	Mengembalikan nilai true jika v adalah vertex pada Graph dan mengembalikan nilai false jika v bukan merupakan vertex pada Graph
`Set<T> getNeighbors(T v)`	Mengembalikan vertex-vertex yang terhubung dengan vertex v, dan vertex-vertex tersebut disimpan dalam object S. Jika v bukan merupakan vertex pada graph maka throws IIlegalArgumentException.
`int getWeight(T v1, T v2)`	Mengembalikan bobot dari edge yang menghubungkan vertex v1 dan v2, jika edge (v1,v2) tidak ada maka mengembalikan nilai -1. Jika v1 atau v2 bukan merupakan vertex pada graph maka throws IIlegalArgumentException.
`boolean isEmpty()`	Mengembalikan nilai true jika graph sama sekali tidak mempunyai vertex dan mengembalikan nilai false jika memiliki minimal 1 buah vertex.
`int numberOfEdges()`	Mengembalikan jumlah edge pada graph.
`int numberOfVertices()`	Mengembalikan jumlah vertex pada graph.
`boolean removeEdge()`	Jika (v1,v2) merupakan edge, menghapus edge tersebut dan

	mengembalikan nilai true dan mengembalikan nilai false jika sebaliknya. Jika v1 atau v2 bukan merupakan vertex pada graph maka throws IIlegalArgumentException.
`boolean removeVertex(Object v)`	Jika v adalah vertex pada graph, menghapus vertex v dari graph dan mengembalikan nilai true, dan mengembalikan nilai false jika sebaliknya.
`int setWeight(T v1, T v2, int w)`	Jika edge (v1, v2) terdapat pada graph, ubah bobot edge dan mengembalikan bobot sebelumnya jika tidak mengembalikan nilai false. Jika v1 atau v2 bukan vertex di graph, maka throws IIlegalArgumentException.
`Set<T> vertexSet()`	Mengembalikan vertex-vertex yang terdapat pada Graph disimpan dalam object Set.

CLASS DIGRAPH

Class DiGraph merupakan implementasi dari interface Graph dan ada beberapa method tambahan yang berguna dalam aplikasi ini :

- Constructor : membuat graph kosong
- inDegree() : untuk menghitung jumlah inDegree dari vertex v.
- outDegree() : untuk menghitung jumlah outDegree dari vertex v.
- static readGraph() : untuk membangun graph dari inputan file.



Gambar 30.9 Contoh Graph DiGraph

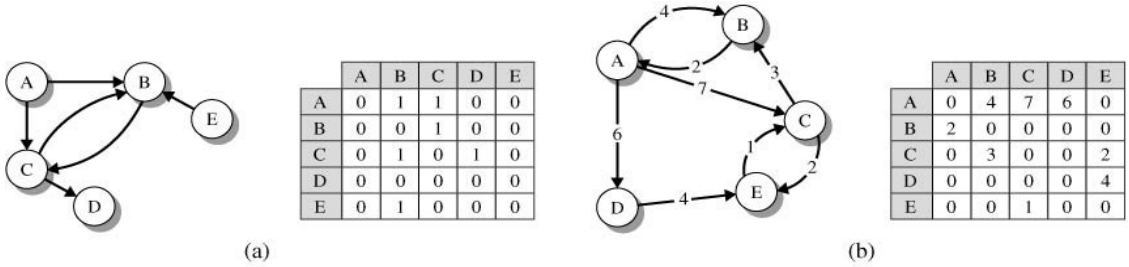
```
File samplegraph.dat
5                   // data for the vertices
A B C D E
6                   // data for the edges
A B 3
A C 2
B C 6
C B 4
C D 1
E B 5
// input vertices, edges, and weights from samplegraph.dat
```

```
DiGraph g = DiGraph.readGraph("samplegraph.dat");
// display the graph
System.out.println(g)
```

```
Output:
A:  in-degree 0  out-degree 2
    Edges: B(3)  C(2)
B:  in-degree 3  out-degree 1
    Edges: C(6)
C:  in-degree 2  out-degree 2
    Edges: B(4)  D(1)
D:  in-degree 1  out-degree 0
    Edges:
E:  in-degree 0  out-degree 1
    Edges: B(5)
```

REPRESENTASI ADJACENCY

Sebuah graph G= dapat direpresentasikan dengan matrik *adjacency*. Berikut adalah graph beserta matrik adjancency.



	A	B	C	D	E
A	0	1	1	0	0
B	0	0	1	0	0
C	0	1	0	1	0
D	0	0	0	0	0
E	0	1	0	0	0

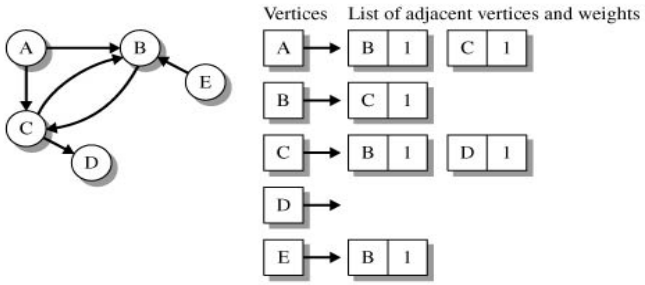
	A	B	C	D	E
A	0	4	7	6	0
B	2	0	0	0	0
C	0	3	0	0	2
D	0	0	0	0	4
E	0	0	1	0	0

Graph representation by means of an adjacency matrix.

Gambar 30.10 Graph dan Matrik Adjacency

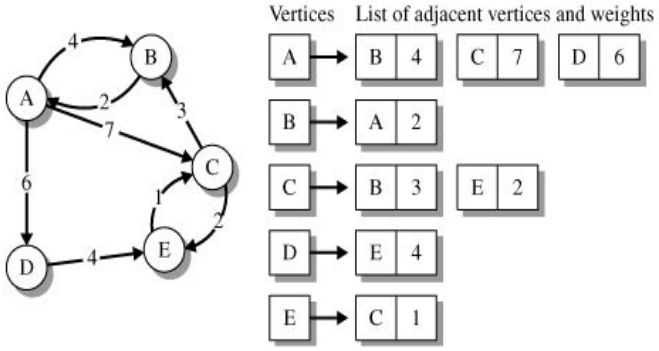
Bentuk lain dari matrik adjacency adalah sebagai berikut :

Sebagai contoh vertex A memiliki keterhubungan dengan vertex B dengan bobot 1 dan vertex C dengan bobot 1.



Gambar 30.11 Graph 1 dan List Adjacency

Sebagai contoh vertex A memiliki keterhubungan dengan vertex B dengan bobot 4, vertex C dengan bobot 7 dan vertex D dengan bobot 6.



Gambar 30.12 Graph 2 dan List Adjacency

REPRESENTASI EDGE

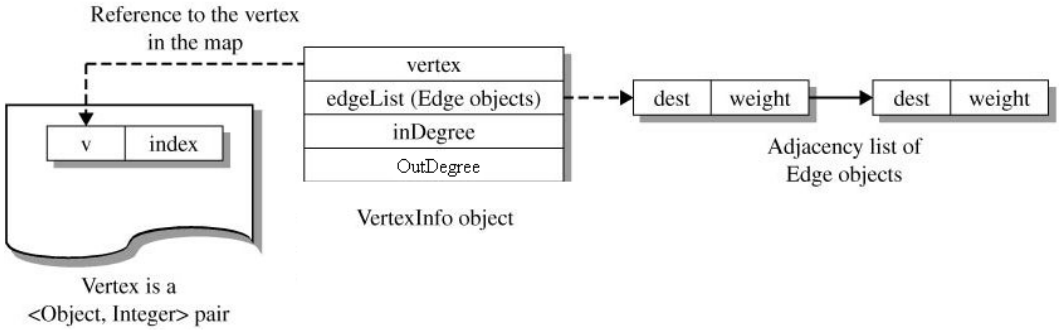
Class Edge merupakan representasi edge yang menghubungkan vertex u dan vertex v pada Graph.

```
class Edge
{
   // index of the destination vertex in the ArrayList
   // vInfo of vertex properties
   public int dest;
   // weight of this edge
   public int weight;
   public Edge(int dest, int weight)
   {
      this.dest = dest;
      this.weight = weight;
   }
   public boolean equals(Object obj)
   {
      return ((Edge)obj).dest == this.dest;
   }
}
```

REPRESENTASI INFORMASI VERTEX

Class VertexInfo merepresentasikan sebuah vertex v beserta informasinya yaitu

- vertex : Nama vertex
- edgeList : Vertex-vertex yang terhubung dengan vertex v
- inDegree : Jumlah inDegree
- outDegree : Jumlah outDegree

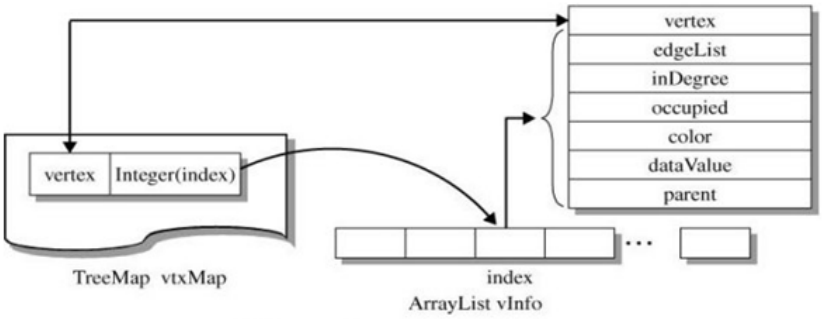


Gambar 30.13 Cara penyimpanan vertex dalam object VertexInfo

```
import java.util.LinkedList;

public class VertexInfo<T> {
    public T vertex ;
    public LinkedList<Edge> edgeList ;

    public int inDegree ;
    public int outDegree;

    public VertexInfo(T v) {
        vertex = v ;
        edgeList = new LinkedList<Edge>() ;
        inDegree = 0 ;
        outDegree = 0;
    }


    public boolean equals(Object obj) {
        VertexInfo<T> vertex2 = (VertexInfo<T>) obj ;
        return this.vertex.equals(vertex2.vertex);
    }
}
```

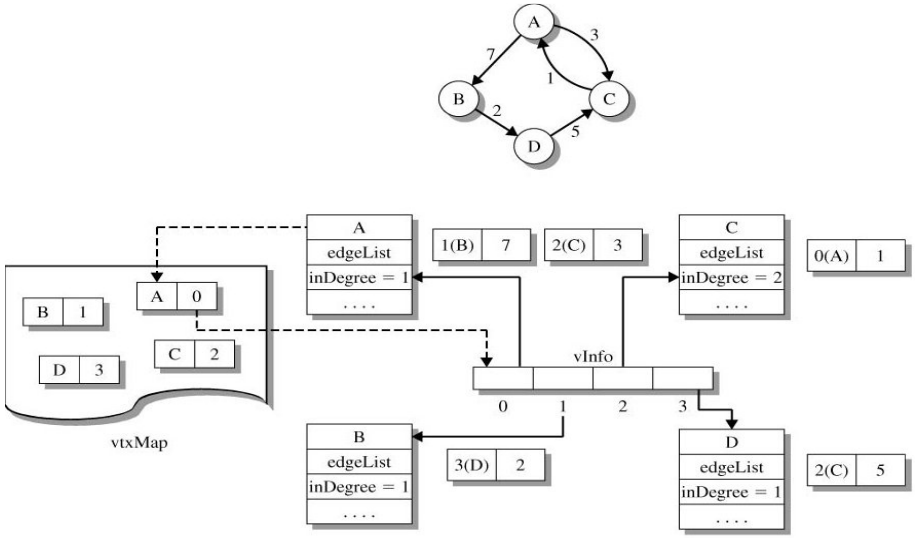
VERTEX MAP vtxMap DAN ARRAYLIST VertexInfo

VERTEX vtxMap terdiri dari data yang berpasangan(key-value) sebagai key adalah Vertex dan sebagai value adalah index pada ArrayList VertexInfo



Gambar 30.14 Keterkaitan HashMap vtxMap dan ArrayList vInfo

Di bawah ini terdapat Graph, beserta data yang tersimpan di vtxMap dan ArrayList VertexInfo.



Gambar 30.15 Graph, beserta data yang tersimpan di vtxMap dan ArrayList vInfo.

C. TUGAS PENDAHULUAN

Jelaskan mengenai Graph di bawah ini :

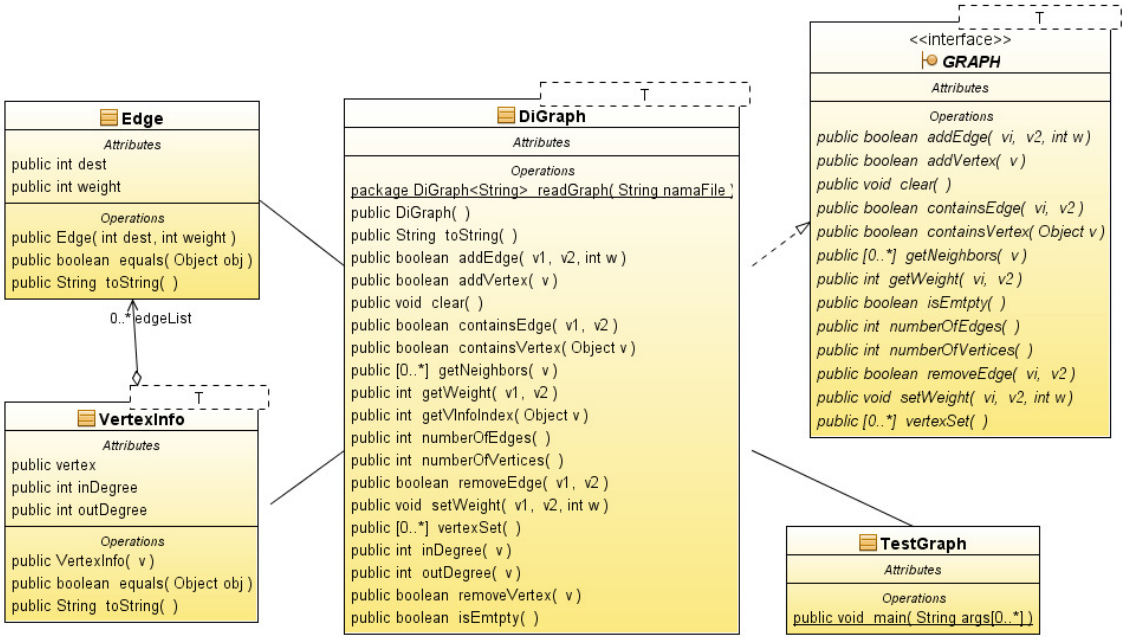
- Kegunaan Graph
- Vertex
- Edge
- Path
- Connected dan Disconnected
- Complete Graph
- Undirected Graph dan DiGraph
- StronglyConnected dan Weakly Connected

D. PERCOBAAN

Untuk praktikum Graph 2 ini melanjutkan praktikum dari Graph 1, mengimplementasikan method-method yang belum dikerjakan di praktikum Graph 1.

Untuk mengimplementasikan Graph, maka perlu dibuat :

- Interface GRAPH
- Class VertexInfo : merepresentasikan sebuah vertex pada Graph
- Class Edge : merepresentasikan sebuah edge/link pada Graph
- Class DiGraph : merepresentasikan Directed Graph/ DiGraph
- Class TestGraph : menguji method-method yang terdapat pada class DiGraph



Gambar 30.16 UML Graph

Percobaan 1 : method getNeighbors() melakukan pembacaan list vertex dan mencari neighbors dari sebuah vertex dan disimpan dalam object Set

```
public Set<T> getNeighbors(T v)
{
   // find the VertexInfo object for index v
   int index = getVInfoIndex(v);
   // check for an error and throw exception
   // if vertices not in graph
   if (index == -1)
      throw new IllegalArgumentException(
            "DiGraph getNeighbors(): vertex not in graph");

   // create HashSet object to hold vertices,
   // obtain the VertexInfo object, and initialize
   // an iterator to scan the adjacency list of
   // the VertexInfo object
   HashSet<T> edgeSet = new HashSet<T>();
   VertexInfo<T> vtxInfo = vInfo.get(index);
   Iterator<Edge> iter = vtxInfo.edgeList.iterator();
   Edge e = null;
   while (iter.hasNext())
   {
      e = iter.next();
      edgeSet.add(vInfo.get(e.dest).vertex);
   }
   return edgeSet;
}
```

E. LATIHAN

Latihan 1 : Lengkapi method-method pada class DiGraph seperti UML di bawah ini.

Tabel 30.1 Interface method-method pada class DiGraph

Method	Kegunaan
public void clear()	Untuk menghapus data dari Vertex
public Set<T> getNeighbors(T v)	Untuk mendapatkan neighbor dari Vertex tertentu.
public int getWeight(T v1, T v2)	Untuk mendapatkan bobot dari sebuah edge dari Vertex v1 ke Vertex v2
public boolean removeEdge(T v1, T v2)	Untuk meghapus edge dari Vertex v1 ke Vertex v2
public void setWeight(T v1, T v2, int w)	Untuk memberikan bobot w dari sebuah edge dari Vertex v1 ke Vertex v2
public boolean removeVertex(T v)	Untuk menghapus Vertex
public boolean isEmtpty()	Untuk mengecek apakah Graph kosong atau tidak.

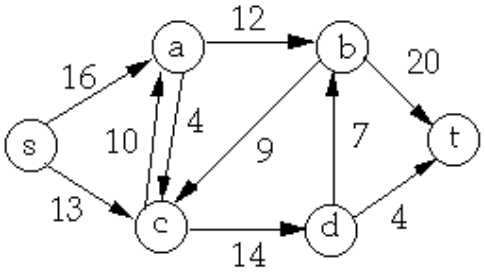
Selanjutnya ujilah method-method yang sudah diimplementasikan dengan program berikut:

```
public class TestGraph2 {

    public static void main(String[] args)
            throws FileNotFoundException, IOException {
        // construct graph with vertices of type
        // String by reading from the file "graphIO.dat"
        DiGraph<String> g = DiGraph.readGraph("graphIO.txt");
        System.out.println(g.toString());
        String vtxName;
        // sets for vertexSet() and adjacent
        // vertices (neighbors)
        Set<String> vtxSet, neighborSet;
        // output number of vertices and edges
        System.out.println("Number of vertices: " + g.numberOfVertices());
        System.out.println("Number of edges: "+ g.numberOfEdges());

        // delete edge with weight 2
        g.removeEdge("A", "C");

        System.out.println("Setelah dihapus e(A,C)");
        System.out.println("Weight e(A,C): "+ g.getWeight("A", "C"));

        System.out.println(g.toString());
        // add and update attributes of the graph
        // increase weight from 4 to 8
        g.setWeight("A", "B", 8);

        System.out.println("Weight e(A,B): "+ g.getWeight("A", "B"));

        // add vertex F
        g.addVertex("F");
        // add edge (F,D) with weight 3
        g.addEdge("F", "D", 3);
```

```
        System.out.println("Weight e(F,D): "+ g.getWeight("F", "D"));

        System.out.println(g.toString());
        // after all updates, output the graph
        // and its properties
        System.out.println("After all the graph updates");
        System.out.println(g);

        // get the vertices as a Set and
        // create set iterator
        vtxSet = g.vertexSet();
        Iterator vtxIter = vtxSet.iterator();
        // scan the vertices and display
        // the set of neighbors
        while (vtxIter.hasNext()) {
            vtxName = (String) vtxIter.next();
            neighborSet = g.getNeighbors(vtxName);
            System.out.println("   Neighbor set for "
                    + "vertex " + vtxName + " is "
                    + neighborSet);
        }
        g.removeVertex("E");
    }
}
```

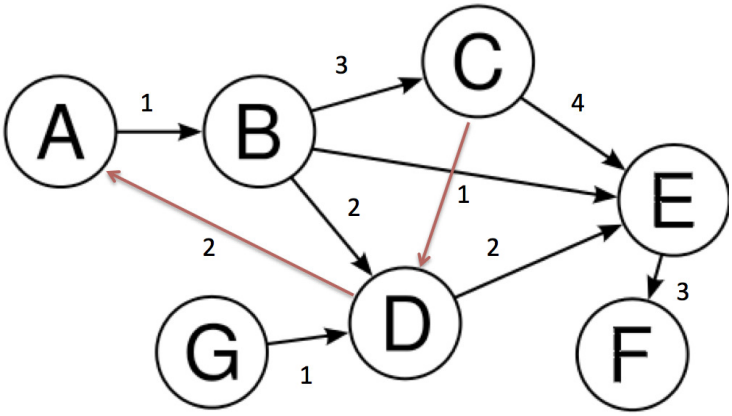
Latihan 3 : Studi kasus graph 1.

- Tentukan output pembacaan graph
- Ubah link AC dengan 7
- Hapus link CA, tampilkan output graph
- Tambahkan link CB dengan bobot 2, selanjutnya tampilkan output graph.



Latihan 4 : Studi kasus graph 2

- Tentukan output pembacaan graph
- Ubah link BE dengan 7
- Hapus link DE, tampilkan output graph
- Tambahkan link EB dengan bobot 5, tampilkan output graph



F. LAPORAN RESMI

Kerjakan hasil percobaan(D) dan latihan(E) di atas dan tambahkan analisa.