VERSION 1.1

February 6, 2023

[STRUKTUR DATA]

MODUL 6, GRAPH

DISUSUN OLEH:

MUHAMMAD SYAUQI AMIQ AMRULLAH
GILANG DWI DARMAWAN

DIAUDIT OLEH:
DIDIH RIZKI CHANDRANEGARA, S.KOM., M.KOM.

PRESENTED

BY:

TIM LAB-IT UNIVERSITAS MUHAMMADIYAH MALANG

[STRUKTUR DATA]

PETUNJUK PENGERJAAN MODUL

Perhatikan petunjuk praktikum dibawah ini:

1. Wajib membaca materi modul, sewaktu – waktu dapat direview oleh asisten saat demo
2. Gunakan referensi yang disediakan modul dan referensi lain pada google (yang kredibel)
3. Latihan praktikum wajib dikerjakan pada praktikum minggu pertama secara bersama – sama di laboratorium dan tidak boleh dijadikan pekerjaan rumah
4. Tugas praktikum boleh dijadikan pekerjaan rumah dan di demokan kepada asisten pada praktikum minggu kedua
5. Memperhatikan kerapihan *source code* termasuk aturan penamaan Class, Method, Variable, File, dan lain - lainnya.
6. Segera lapor kepada asisten jika ada kesalahan pada modul praktikum.

PERSIAPAN MATERI

Mahasiswa diharapkan mempelajari materi praktikum dengan baik, sesuai dengan materi yang diberikan oleh dosen pengajar dikelas. Terutama dalam penerapan materi OOP JAVA:

1. Graph

TUJUAN

Mahasiswa mampu menguasai & menjelaskan konsep dari graph.

TARGET MODUL

Mahasiswa mampu memahami:

1. Graph

PERSIAPAN SOFTWARE/APLIKASI

1. Java Development Kit
2. Java Runtime Environment
3. IDE (Intellij IDEA, Eclipse, Netbeans, dll)

REFERENSI MATERI

Geekforgeeks:

https://www.geeksforgeeks.org/implementing-generic-graph-in-java/

Youtube (Indonesia)

https://www.youtube.com/watch?v=2OT-2G0eGnQ&ab_channel=RangkumanHR

Youtube (English)

https://www.youtube.com/watch?v=UhFfdBdHCJM&ab_channel=KartikAgarwal

https://www.youtube.com/watch?v=kkDwvvzTAa8&ab_channel=JavaCodingCommunity-ProgrammingTutorials

https://www.youtube.com/watch?v=x6iO0ZH9h7Q&ab_channel=Geekific

https://www.youtube.com/watch?v=by93qH4ACxo&ab_channel=BroCode

Javapoint

https://www.javatpoint.com/java-graph

https://www.javatpoint.com/bfs-algorithm-in-java

Programiz

https://www.programiz.com/java-programming/examples/graph-implementation

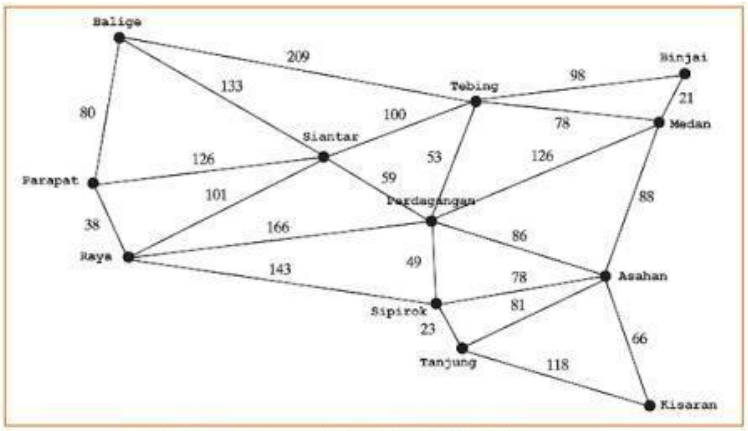
Note:

Dari referensi tersebut mungkin terdapat sedikit perbedaan satu sama yang lain, cukup pahami konsepnya dan terapkan pada kasus di modul ini.

MATERI POKOK

A. Graph

Graph adalah struktur data dengan relasi many to many, yaitu tiap element dapat memiliki 0 atau lebih dari 1 cabang. Graph terdiri dari 2 komponen yaitu Node (digunakan untuk menyimpan data) dan Edge (cabang, untuk menghubungkan node satu dengan node lain). Representasi visual dari graph adalah dengan menyatakan objek sebagai noktah, bulatan atau titik (Vertex), sedangkan hubungan antara objek dinyatakan dengan garis (Edge). Lebih Jelasnya dapat dilihat pada gambar dibawah:



1.1 Graph yang menunjukan jarak antar kota

$$G = \{V,E\}$$

Dimana :
G = Graph
V = Simpul atau Vertex, atau Node, atau Titik
E = Busur atau Edge, atau arc

1.2 Representasi Graph

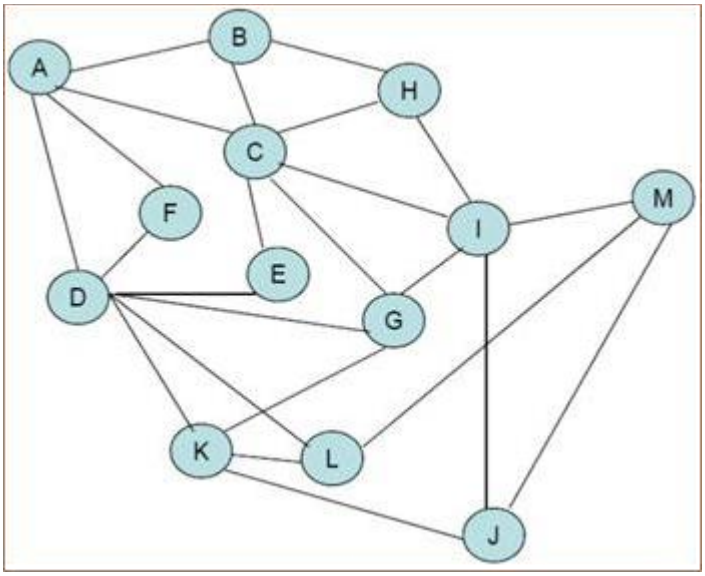
B. Jenis-jenis Graph

a) Undirected Graph

Undirected Graph merupakan graph yang tidak memiliki arah,sehingga dalam penulisannya dapat berlaku 2 arah atau bolak balik. Singkatnya perhatikanlah gambar berikut ini untuk dapat lebih memahami :

DARI GRAPH DIATAS ARAHNYA DAPAT DITULISKAN MANJADI {A,B} ATAU {B,A

Contoh lain sebagai berikut :

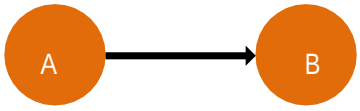


Notasi gambar diatas adalah :

G = ,V, E-
V = ,A, B, C, D, E, F, G, H, I,J, K, L, M-
E = , (A,B),(A,C), (A,D), (A,F), (B,C), (B,H), (C,E), (C,G), (C,H), (C,I), (D,E),(D,F), (D,G), (D,K), (D,L), (E,F), (G,I),(G,K), (H,I), (I,J), (I,M), (J,K),(J,M), (L,K), (L,M)-.

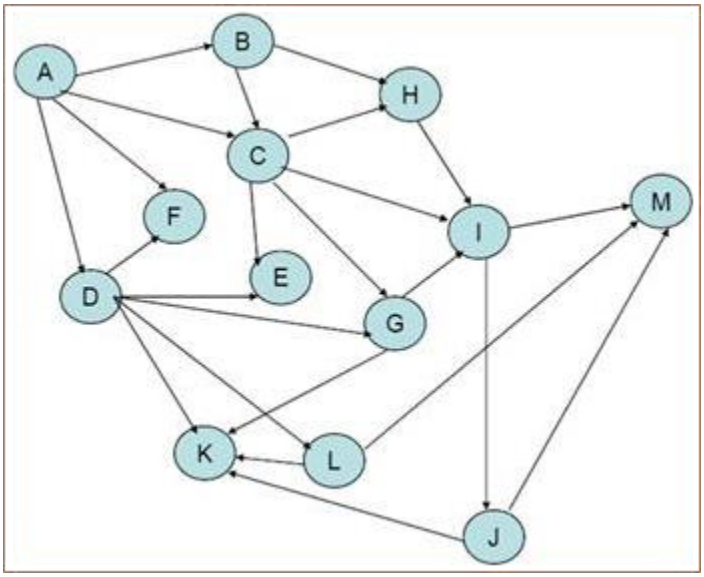
b) Directed Graph

Dari graph dibawah arahnya dapat dituliskan menjadi ,A,B- dan bukan ,B,A-. Berbeda dengan Undirected graph yang tidak memiliki arah, direted graph memiliki arah yang tertuju pada node/vertex tertentu yang biasa dinotasikan dengan gambar anak panah. Untuk lebih jelasnya perhatikan contoh dibawah ini :



DARI GRAPH DIATAS ARAHNYA DAPAT DITULISKAN MANJADI {A,B} BUKAN {B,A}

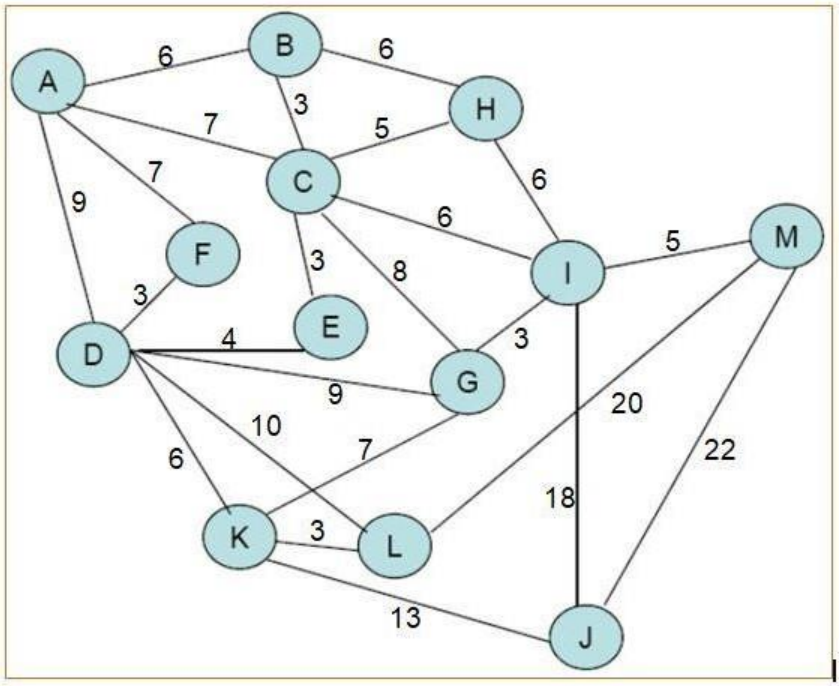
Adapun untuk contoh lainnya seperti gambar dibawah ini :



Notasi dari gambar diatas:
G = ,V, E-
V = ,A, B, C, D, E, F, G, H, I,J, K, L, M-
E = ,(A,B),(A,C), (A,D), (A,F), (B,C), (B,H), (C,E), (C,G), (C,H), (C,I), (D,E),
(D,F), (D,G), (D,K), (D,L), (E,F), (G,I), (G,K), (H,I), (I,J), (I,M), (J,K), (J,M), (L,K), (L,M)-.

c) Weight Graph

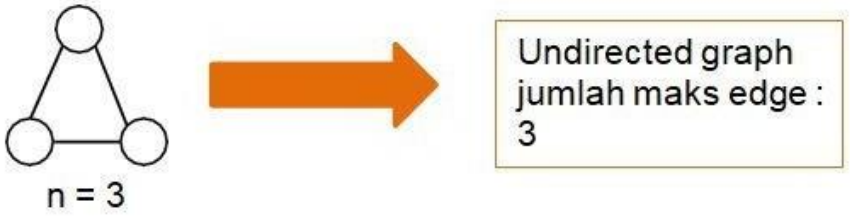
Sesuai namanya, Weight graph merupakan yang memiliki bobot atau nilai pada tiap - tiap nodenya. Untuk Lebih Jelasnya dapat memperhatikan gambar berikut:

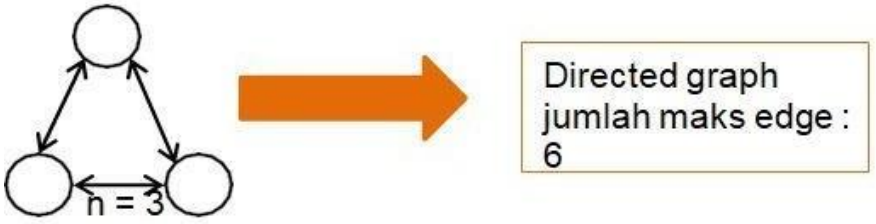


C. Istilah – istilah pada Graph

a) Jumlah Edge

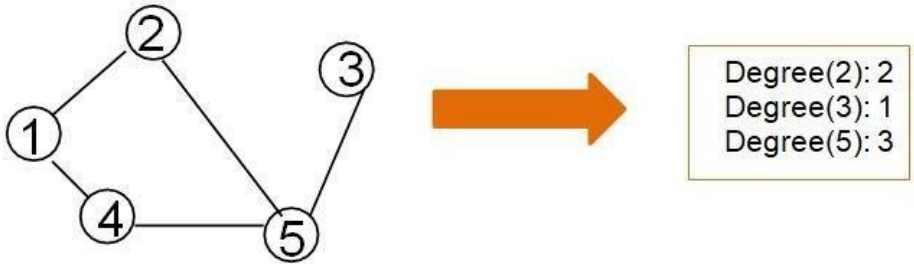
Jumlah pasangan edge yang mungkin (banyak maksimal edge) dapat dilihat dari jumlah node (n). Dapat dihitung menggunakan formula Undirected Graph : n(n-1)/2 dan untuk Directed Graph :n(n-1)





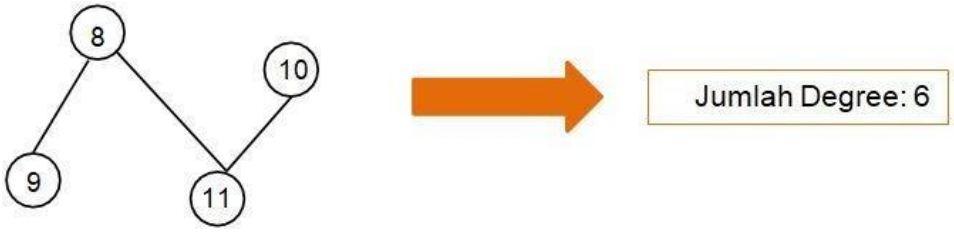
b) Degree

Jumlah cabang atau jumlah edge yang dimiliki node.



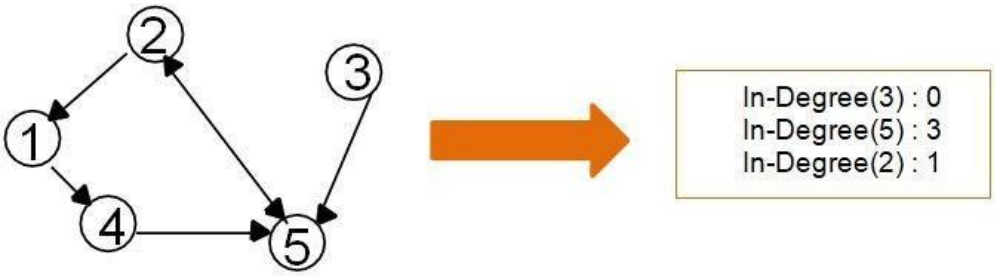
c) Jumlah Degree

Jumlah degree adalah jumlah total cabang/degree yang ada pada graph. Dapat dirumuskan dengan notasi : 2e (e merupakan jumlah edge).



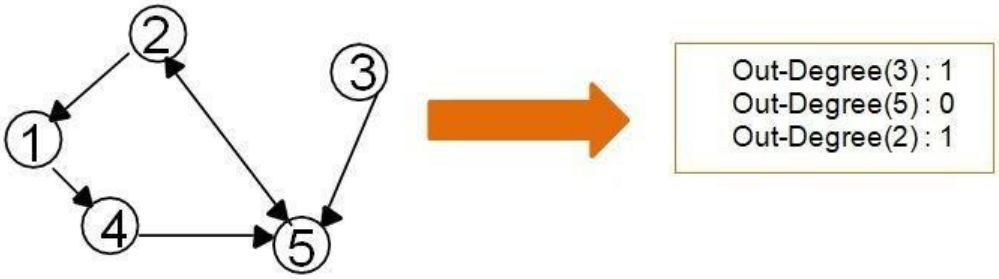
d) In-Degree

Jumlah edge yang masuk atau mengarah ke Node.



e) Out-Degree

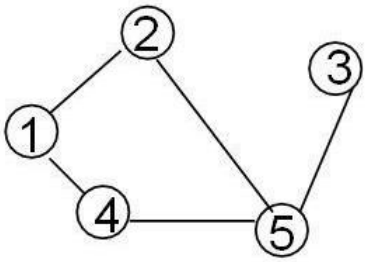
Jumlah edge yang keluar dari Node.



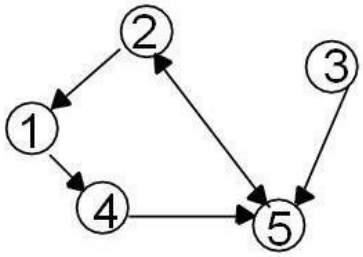
D. Adjency Matriks

Adjacency Matriks pada graph merupakan representasi graph kedalam betuk matriks dengan ordo n x n, dimana n adalah node. Dalam Adjacency Matriks pada graph ini, baris pada matriks berisi node asal sedangkan kolom pada matriks berisi node tujuan. Jika terdapat edge antar

node maka diisi dengan nilai 1 sedangkan jika tidak ada node antar node diisi dengan nilai 0.



	1	2	3	4	5
1	0	1	0	1	0
2	1	0	0	0	1
3	0	0	0	0	1
4	1	0	0	0	1
5	0	1	1	1	0



	1	2	3	4	5
1	0	0	0	1	0
2	1	0	0	0	1
3	0	0	0	0	0
4	0	0	0	0	1
5	0	1	1	0	0

E. Graph Traversal

Graph Traversal merupakan metode penelusuran yang digunakan untuk mengunjungi tiap simpul/node secara sistematik. Metode penelusuran pada graph dibagi menjadi 2, yaitu:

- DFS (Depth First Search), melakukan kunjungan ke node-node dengan cara mengunjungi node terdalam/kebawah ,setelah itu mencari ketempat yang lainnya , dan sistemnya menggunakan stack.
- BFS (Breadth First Search), melakukan visit ke node- node dengan cara melebar kesamping, dan sistemnya menggunakan queue.

F. Interface Graph

Interface Graph berisi method-method dasar dari graph.

Method	Deskripsi
`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 IllegalArgumentException.
`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 IllegalArgumentException.

`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 IllegalArgumentException.
`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 IllegalArgumentException.
`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 IllegalArgumentException.
`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 IllegalArgumentException.
`Set<T> vertexSet()`	Mengembalikan vertex-vertex yang terdapat pada Graph disimpan dalam object Set.

LATIHAN PRAKTIKUM

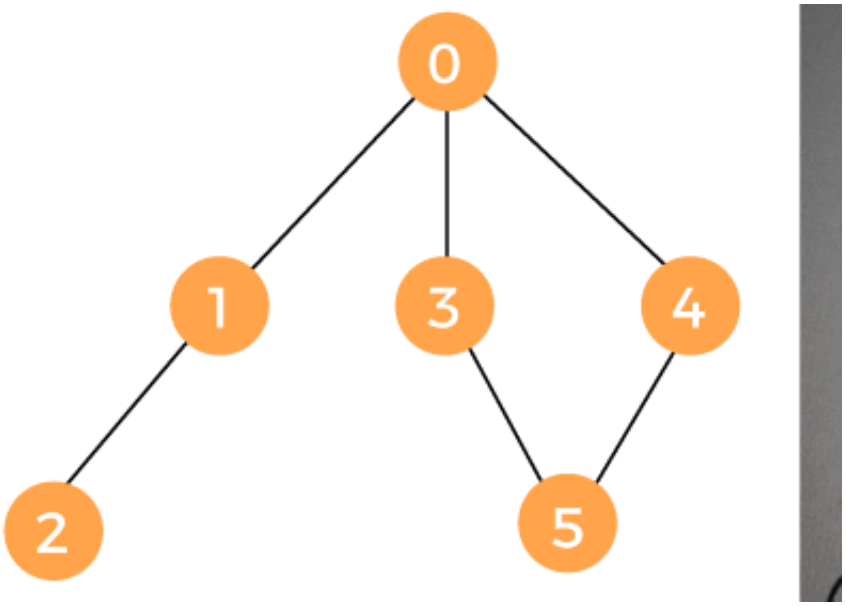
LATIHAN 1

A. BFS (Breadth-First Search)

Breadth-First Search (BFS) didasarkan pada traversal node dengan menambahkan tetangga

dari setiap node ke antrian traversal mulai dari root node. BFS untuk graph mirip dengan Tree, dengan pengecualian bahwa graph mungkin memiliki siklus. Berbeda dengan pencarian depth-first, semua node tetangga pada kedalaman tertentu diselidiki sebelum melanjutkan ke level berikutnya.

Mari kita coba, perhatikan contoh graph berikut:



Dari gambar diatas mari kita membuat kode nya:

```java
import java.io.*;
import java.util.*;
public class BFSTraversal
{
    private int node;       /* total number number of nodes in the graph */
    private LinkedList<Integer> adj[];      /* adjacency list */
    private Queue<Integer> que;           /* maintaining a queue */
    BFSTraversal(int v)
    {
        node = v;
        adj = new LinkedList[node];
        for (int i=0; i<v; i++)
        {
            adj[i] = new LinkedList<>();
        }
        que = new LinkedList<Integer>();
    }
    void insertEdge(int v,int w)
    {
        adj[v].add(w);      /* adding an edge to the adjacency list (edges are bidirectional in this example) */
    }
    void BFS(int n)
    {
        boolean nodes[] = new boolean[node];       /* initialize boolean array for holding the data */
        int a = 0;
        nodes[n]=true;
        que.add(n);       /* root node is added to the top of the queue */
        while (que.size() != 0)
        {
            n = que.poll();        /* remove the top element of the queue */
            System.out.print(n+" ");    /* print the top element of the queue */
            for (int i = 0; i < adj[n].size(); i++)  /* iterate through the linked list and push all neighbors into queue */
            {
                a = adj[n].get(i);
                if (!nodes[a])      /* only insert nodes into queue if they have not been explored already */
                {
                    nodes[a] = true;
                    que.add(a);
                }
            }
        }
    }
    public static void main(String args[])
    {
        BFSTraversal graph = new BFSTraversal(6);
        graph.insertEdge(0, 1);
        graph.insertEdge(0, 3);
        graph.insertEdge(0, 4);
        graph.insertEdge(4, 5);
        graph.insertEdge(3, 5);
        graph.insertEdge(1, 2);
        graph.insertEdge(1, 0);
        graph.insertEdge(2, 1);
        graph.insertEdge(4, 1);
        graph.insertEdge(3, 1);
        graph.insertEdge(5, 4);
        graph.insertEdge(5, 3);
        System.out.println("Breadth First Traversal for the graph is:");
        graph.BFS(0);
    }
}
```

Maka outputnya akan seperti berikut:

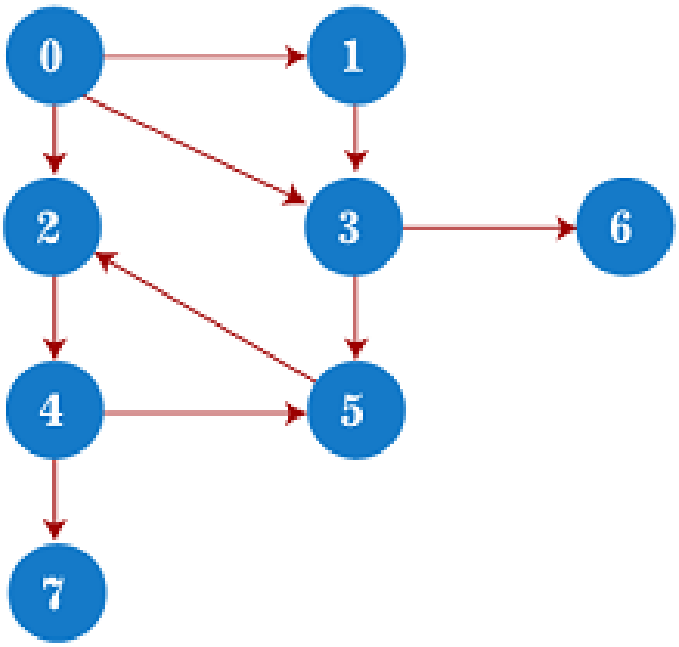
```
Breadth First Traversal for the graph is:
0 1 3 4 2 5
```

Dari output tersebut terlihat rute dari algoritma BFS ini berjalan, dimulai dari Node 0 ke 1 lalu ke 3 dan seterusnya.

B. DFS (Depth-First Search)

Kita akan membahas algoritma DFS pada struktur data. Ini adalah algoritma rekursif untuk mencari semua simpul dari struktur data Tree atau Graph. Algoritma depth-first search (DFS) dimulai dengan node awal dari masuk lebih dalam hingga kita menemukan node tujuan atau node tanpa anak.

Mari kita coba, perhatikan contoh graph berikut:



Mari kita buat kode nya:

```java
import java.util.*;

class DFSTraversal {
  private LinkedList<Integer> adj[]; /*adjacency list representation*/
  private boolean visited[];

  /* Creation of the graph */
  DFSTraversal(int V) /*'V' is the number of vertices in the graph*/
  {
    adj = new LinkedList[V];
    visited = new boolean[V];

    for (int i = 0; i < V; i++)
      adj[i] = new LinkedList<Integer>();
  }

  /* Adding an edge to the graph */
  void insertEdge(int src, int dest) {
    adj[src].add(dest);
  }

  void DFS(int vertex) {
    visited[vertex] = true; /*Mark the current node as visited*/
    System.out.print(vertex + " ");

    Iterator<Integer> it = adj[vertex].listIterator();
    while (it.hasNext()) {
      int n = it.next();
      if (!visited[n])
        DFS(n);
    }
  }

  public static void main(String args[]) {
    DFSTraversal graph = new DFSTraversal(8);

        graph.insertEdge(0, 1);
        graph.insertEdge(0, 2);
        graph.insertEdge(0, 3);
        graph.insertEdge(1, 3);
        graph.insertEdge(2, 4);
        graph.insertEdge(3, 5);
        graph.insertEdge(3, 6);
        graph.insertEdge(4, 7);
        graph.insertEdge(4, 5);
        graph.insertEdge(5, 2);

        System.out.println("Depth First Traversal for the graph is:");
        graph.DFS(0);
  }
}
```

Maka outputnya akan seperti berikut:

```
Depth First Traversal for the graph is:
0 1 3 5 2 4 7 6
```

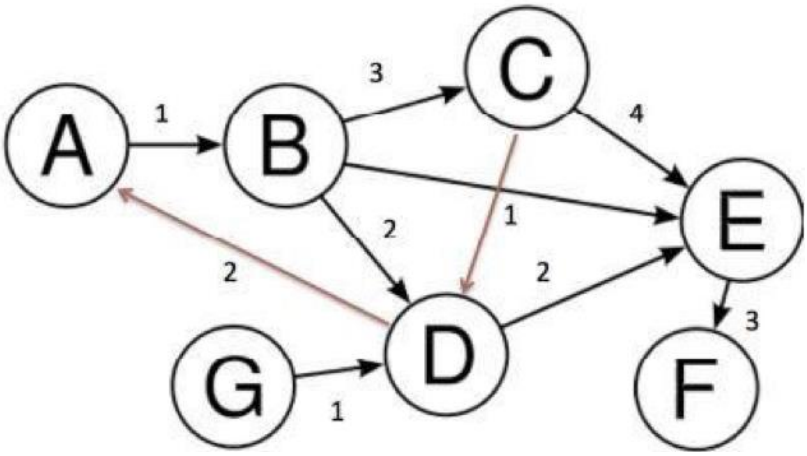
NB:

Contoh ini didapat dari sumber Javapoint silahkan kunjungi website nya jika masih kurang paham.

TUGAS PRAKTIKUM

KEGIATAN 1

1. Jelaskan kepada asisten tentang apa yang anda ketahui tentang graph dan jelaskan pula perbedaan antara Struktur data Tree dan Struktur data Graph.
2. Perhatikan gambar dibawah ini :

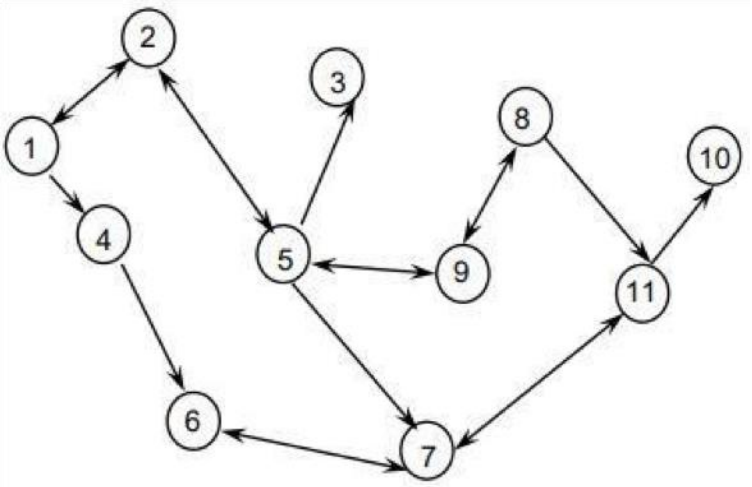


Jelaskan kepada Asisten :

a. Notasi dari graph diatas (G=,V,E-)
b. Jumlah in-degree, jumlah out-degree, dan jumlah Edge
c. Adjacency Matriks dari gambar tersebut
d. Tentukan weight pada tiap-tipa node

Kerjakan di word dan tunjukakan kepada asisten saat demo.

3. Buatlah program yang dapat menentukan rute DFS (Depth First Search) dan BFS (Breadth First Search) dari gambar dibawah ini :



4. Kemudian jelaskan rute dan proses dari program anda.

CATATAN

Aturan umum penulisan JAVA agar mudah dikoreksi oleh asisten:

1. Untuk nama class, enum, dan yang lainnya biasakan menggunakan gaya CamelCase (diawali dengan huruf besar pada tiap kata untuk mengganti spasi) seperti: Kursi, JalanRaya, ParkiranGedung, dan lain seterusnya.
2. Untuk penulisan nama method dan attribute diawali dengan huruf kecil di awal kata dan menggunakan huruf besar untuk kata setelahnya, seperti: getNamaJalan, namaJalan, harga, setNamaJalan, dan lain seterusnya.
3. Jika menggunakan IDE Intellij jangan lupa untuk memformat kode agar terlihat rapi menggunakan menu code -> show reformat file dialog -> centang semua field dan klik ok.

DETAIL PENILAIAN TUGAS

Kriteria	Nilai
Semua Ketentuan Pada Tugas Praktikum Terpenuhi Saat Demo dan Kerapihan Code Serta Tidak Ada Plagiasi	40
Mengerjakan Latihan Praktikum	40
Presensi Kehadiran	20