Algoritme dan Struktur Data

Graph

Putra Pandu Adikara
Universitas Brawijaya

Pengertian

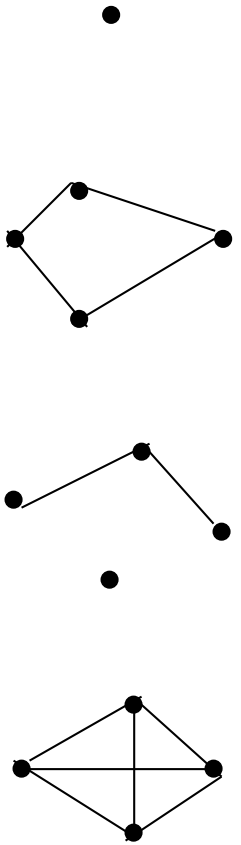
- Merupakan konsep struktur data yang non linier yang setiap node dapat dihubungkan dengan node-node yang lain, tanpa adanya hubungan anak dan parents (tidak berjenjang)
- Sebuah konsep struktur data yang terdiri dari kumpulan node (*vertex*) dan saling berhubungan (*edge*).

Definisi

- Graph adalah struktur data yang memiliki relasi **many to many**, yaitu tiap element dapat memiliki 0 atau lebih dari 1 cabang.
- Graph terbentuk dari 2 bagian, yaitu **node** dan **edge**.
  - Node: digunakan untuk menyimpan data
  - Edge: cabang, untuk menghubungkan node satu dengan node lain.

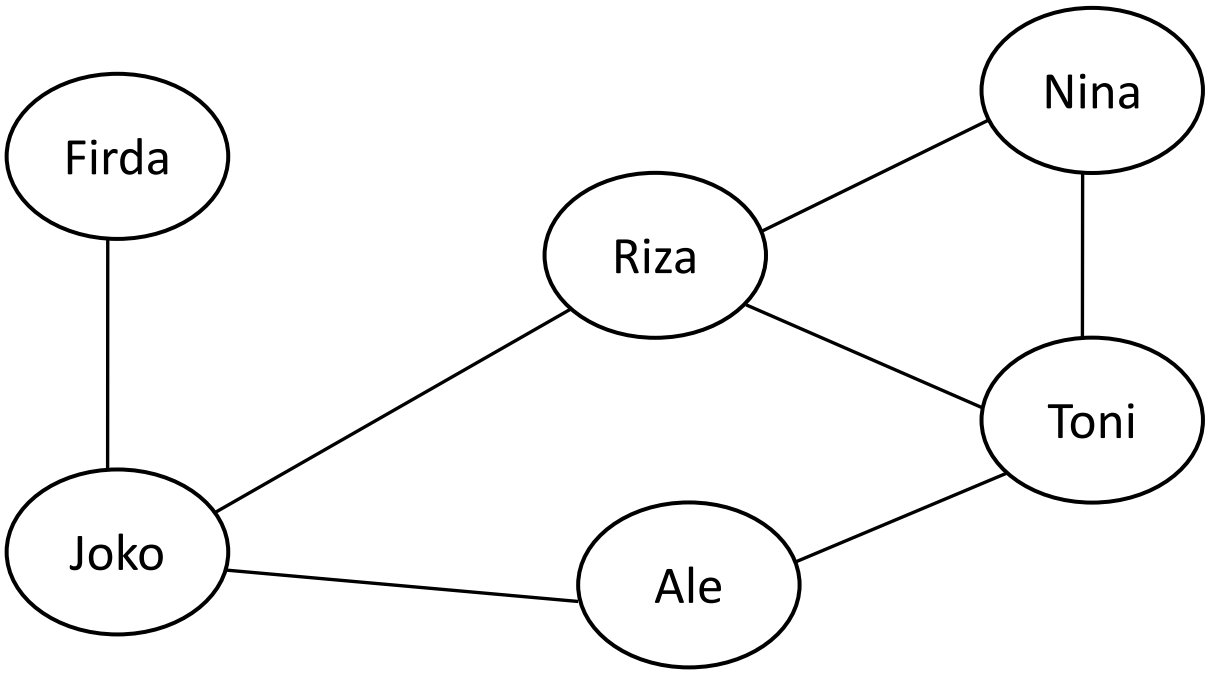
Definisi Graph

- Sebuah graph mungkin hanya terdiri dari satu simpul
- Sebuah graph belum tentu semua simpulnya terhubung dengan busur
- Sebuah graph mungkin mempunyai simpul yang tak terhubung dengan simpul yang lain
- Sebuah graph mungkin semua simpulnya saling berhubungan



Contoh Graph

- Jaringan pertemanan pada Facebook.



Graph dengan 6 node dan 7 edge yang merepresentasikan jaringan pertemanan pada Facebook

Penjabaran

- Jika => G = (N,E)
- G adalah Graph, N adalah Node, dan E adalah Edge.
- Sehingga dari contoh graph facebook tersebut dapat dijabarkan:

  N = {Nina, Toni, Ale, Riza, Joko, Firda}

  E = {{Nina,Toni},{Toni,Riza},{Nina, Riza}, {Toni,Ale},{Ale,Joko},{Riza,Joko},{Firda,Joko}}

  *N: para anggota Facebook

  E: pertemanan antara member satu dengan yang lain.

Jenis Graph

- Graph dibedakan menjadi beberapa jenis, antara lain:
  - Undirected Graph (Undi-graph)
  - Directed Graph (Di-graph)
  - Weigthed Graph

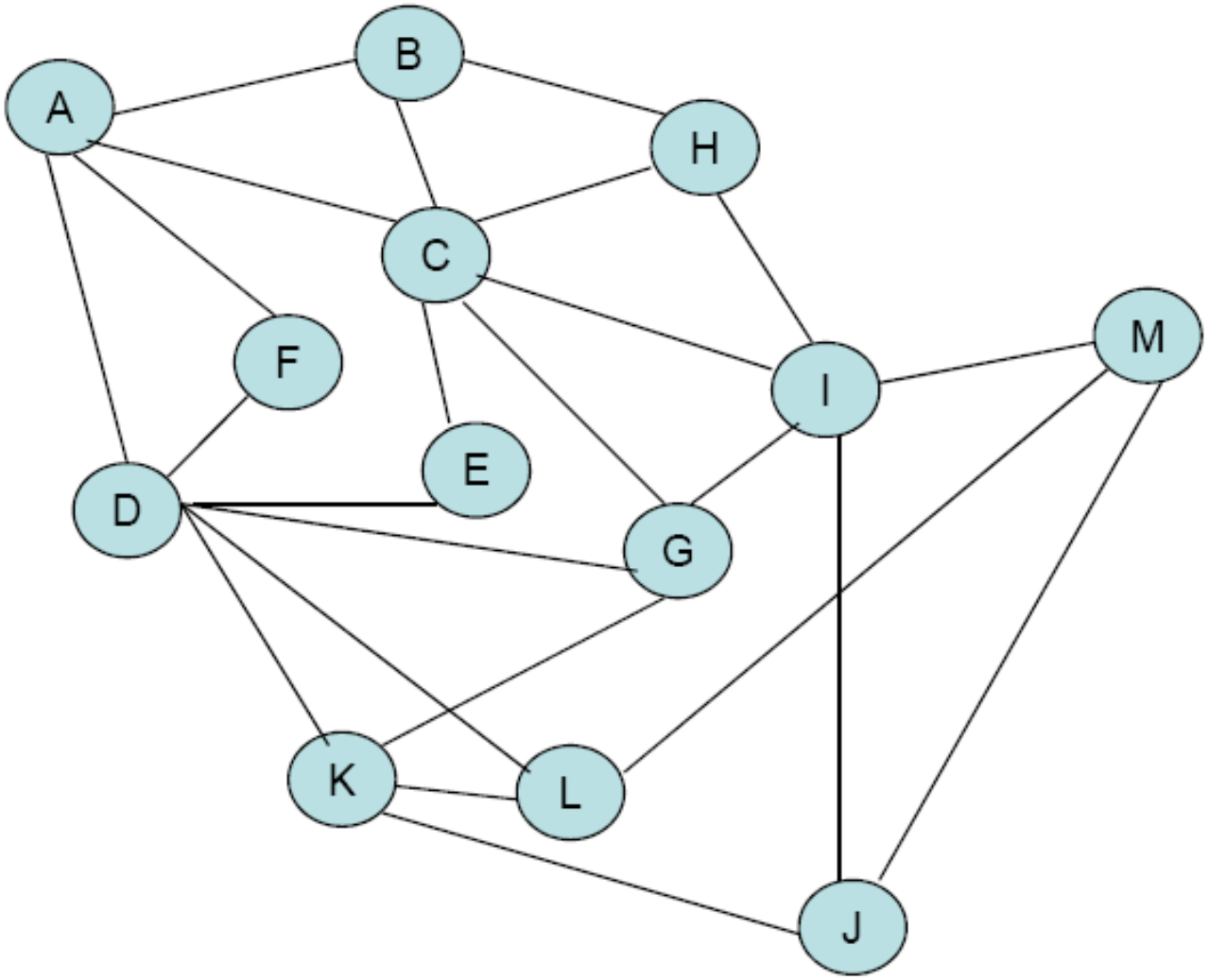
Undirected Graph

- Biasa disingkat: undi-graph.
- Yaitu graph yang tidak memiliki arah.
- Setiap sisi berlaku dua arah.
- Misalkan: {x,y}

  Arah bisa dari x ke y, atau y ke x.
- Secara grafis sisi pada undigraph tidak memiliki mata panah dan secara notasional menggunakan kurung kurawal.

U ——— V {U,V} atau {V,U}

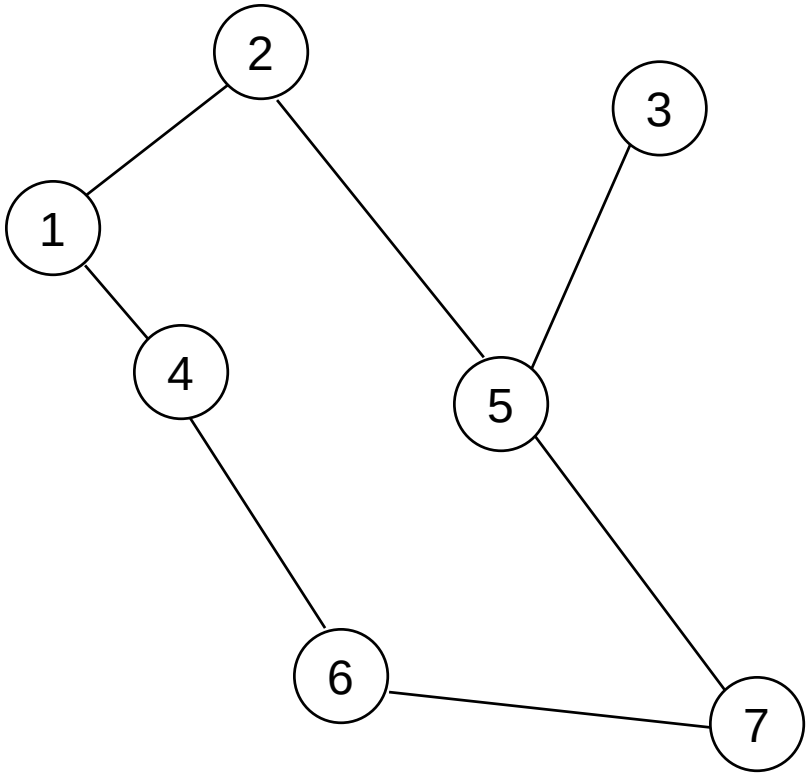
Gambar Undi-Graph



Notasional

- 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} }

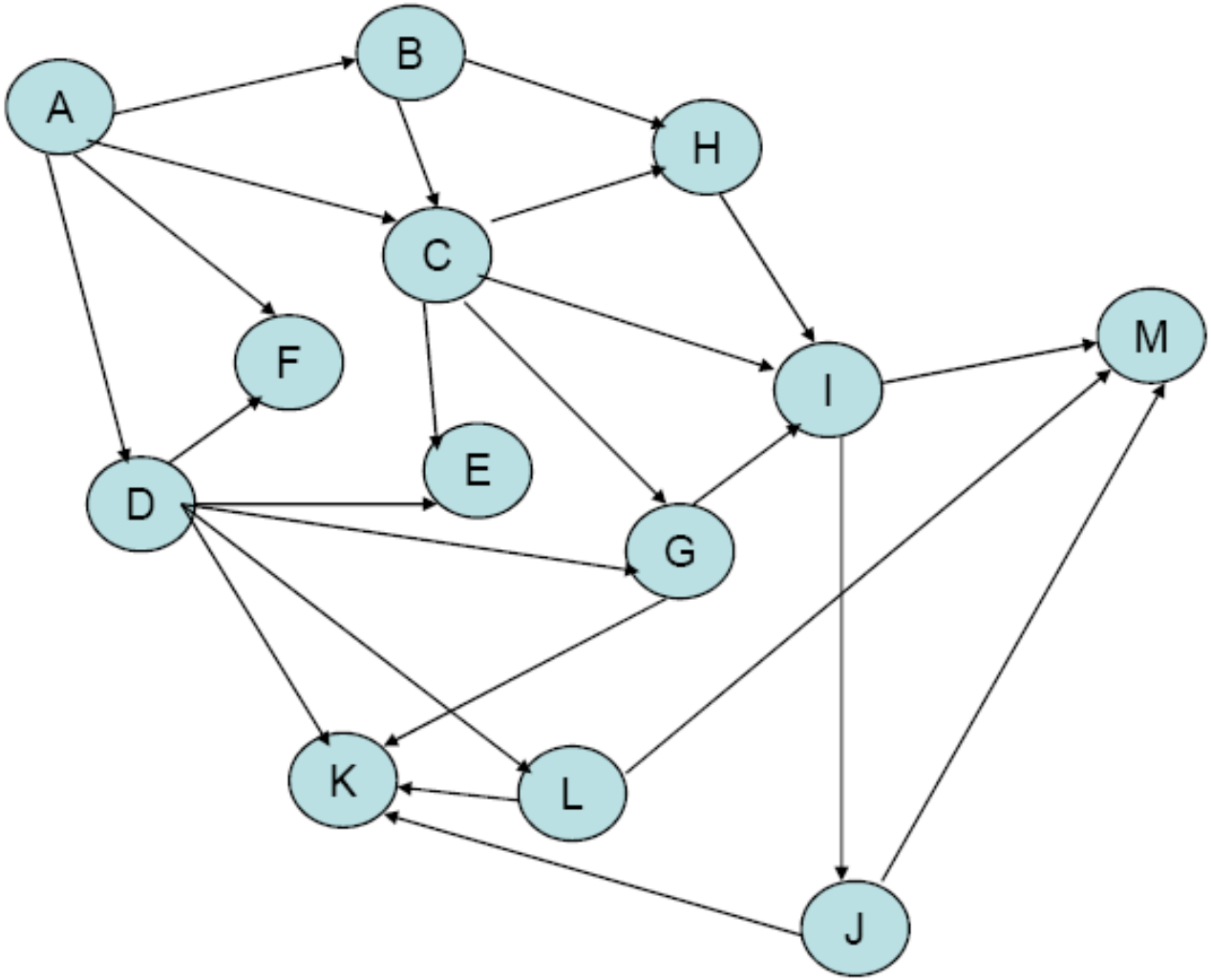
Latihan



Directed Graph

- Biasa disingkat: Di-graph.
- Yaitu graph yang memiliki arah.
- Setiap edge Digraph memiliki anak panah yang mengarah ke node tertentu.
- Secara notasi sisi digraph ditulis sebagai vektor (u, v).
- u = origin (vertex asal)
- v = terminus (vertex tujuan)

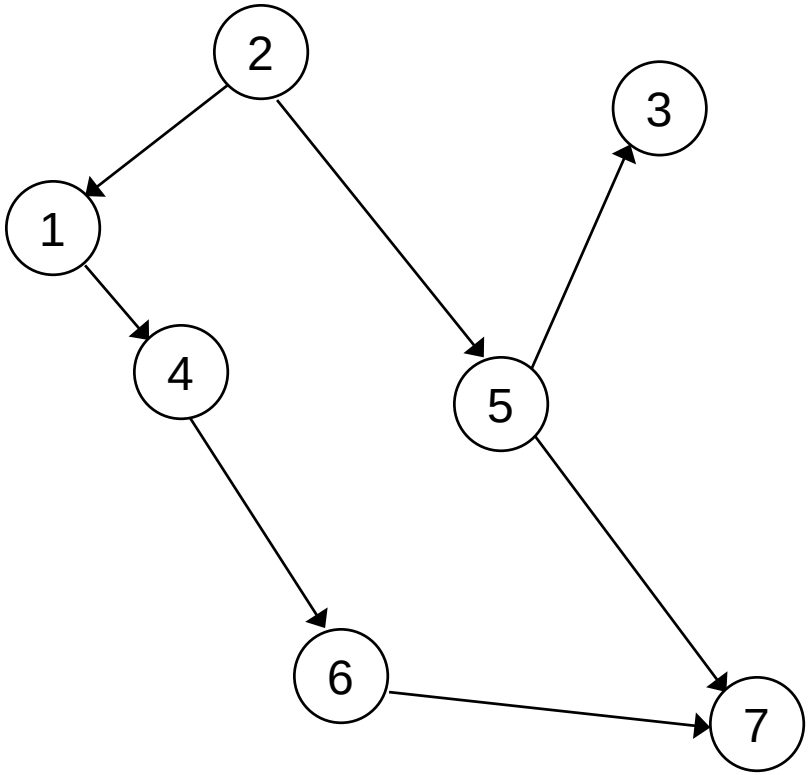
Gambar Digraph



Notasional

- 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) }

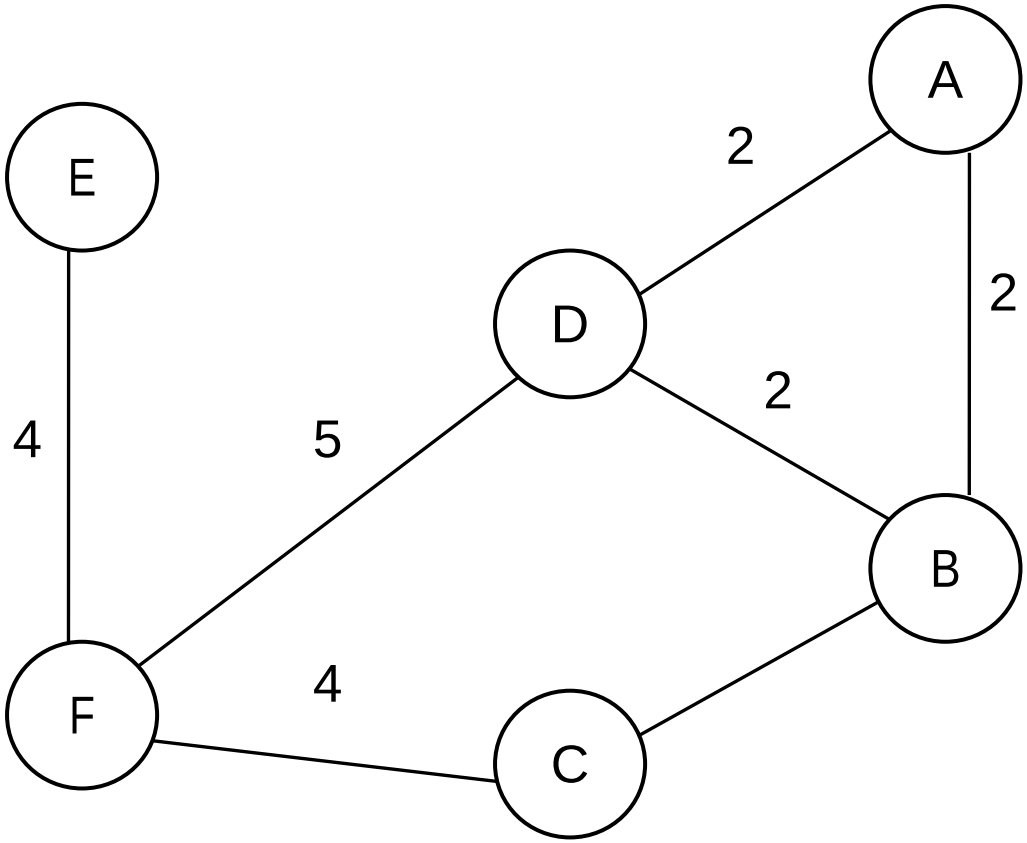
Contoh Digraph



Weigth Graph

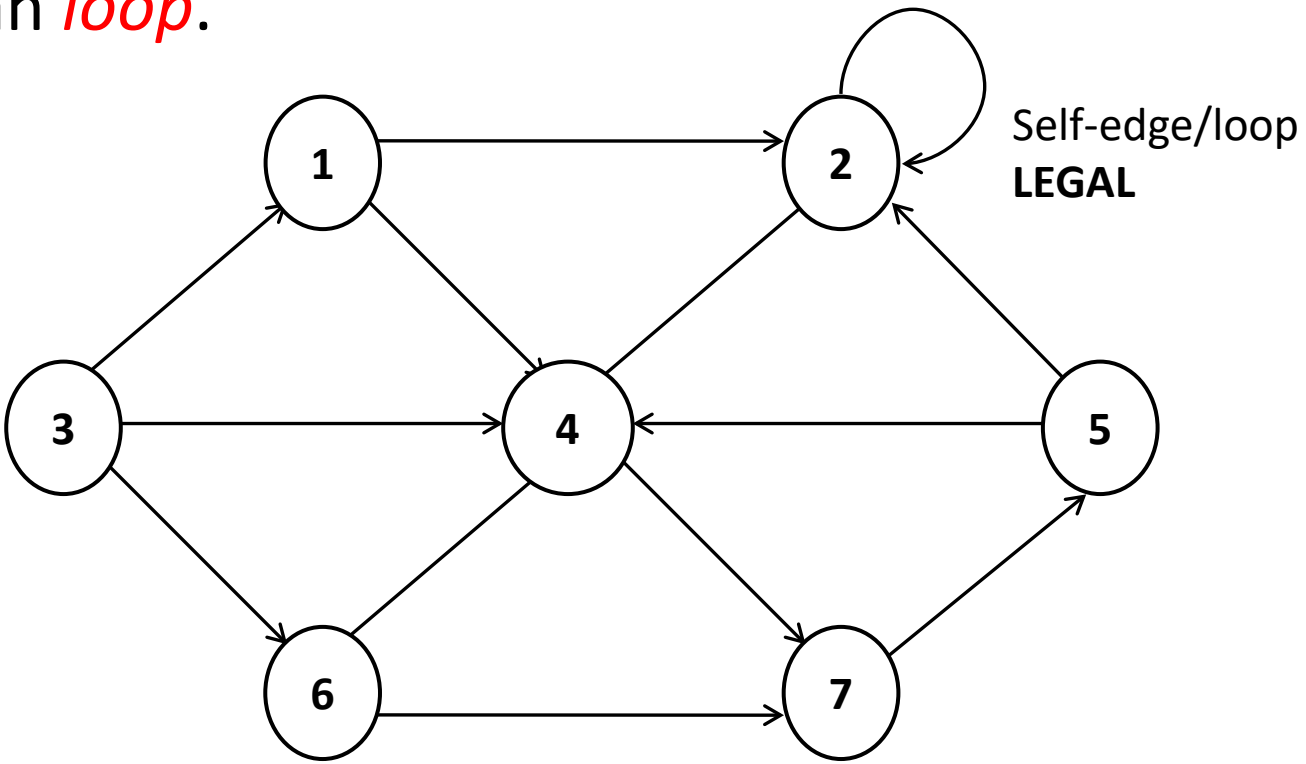
- Graph yang memiliki bobot, yaitu pada tiap edge-nya memiliki nilai.

Contoh Weigth Graph



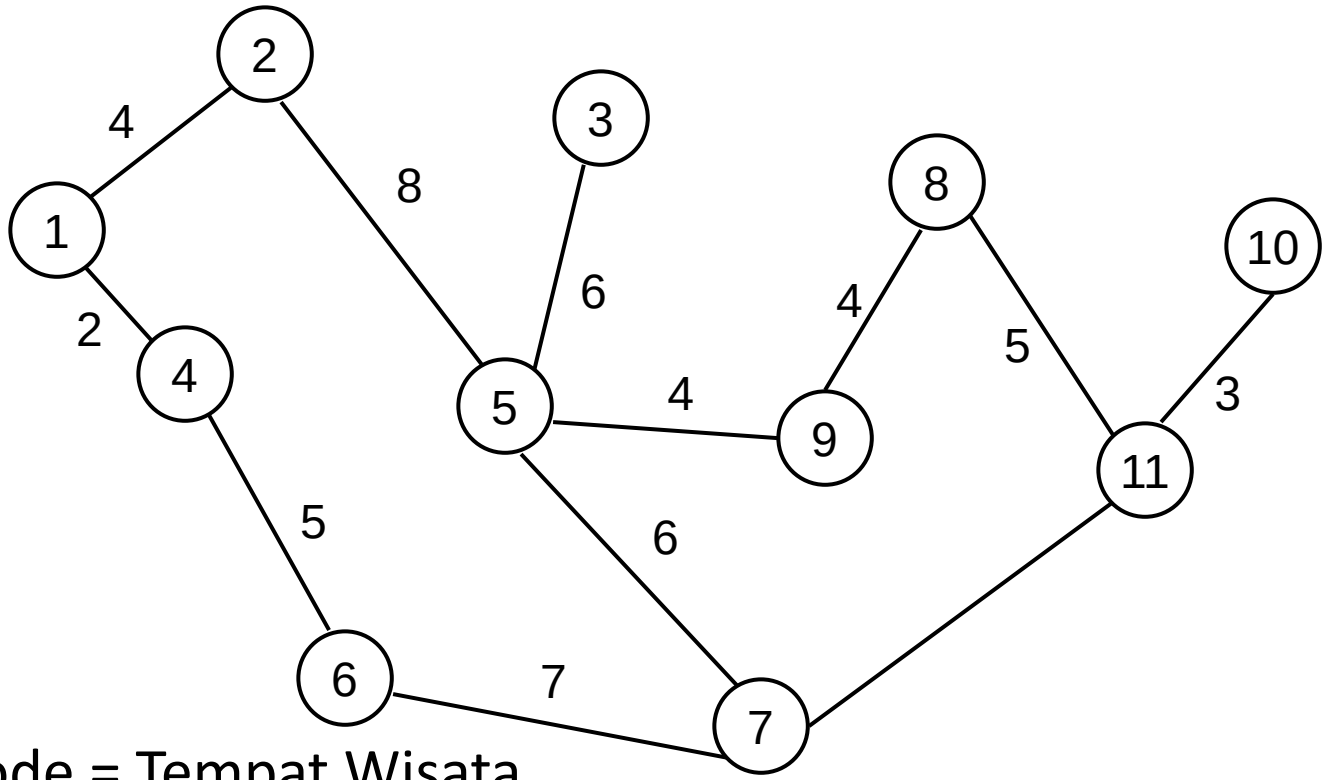
Loop

- Digraph dapat memiliki edge dari dan menuju ke node itu sendiri (*self-edge*). Hal ini dikenal dengan istilah *loop*.



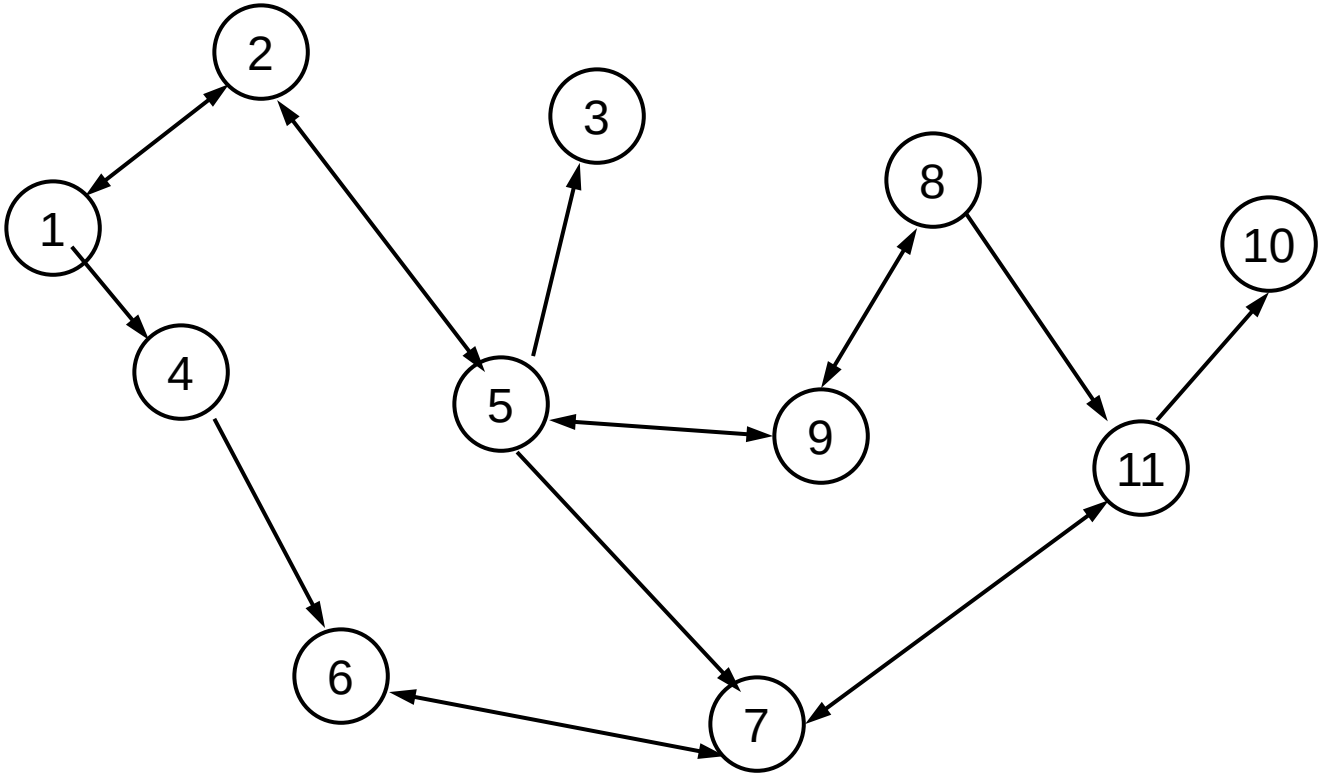
Contoh Penerapan Graph

Peta Penelusuran Kota
(Berdasarkan Jarak)



- Node = Tempat Wisata
- edge = jalur
- Weight = jarak

Peta kota



- Beberapa jalan hanya boleh 1 arah

Graph Property

Property

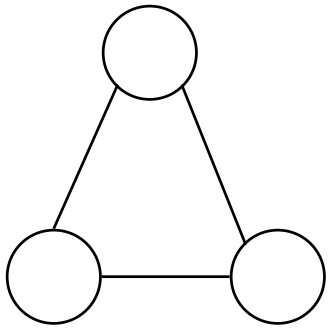
- Jumlah Edge
- Degree
- Jumlah Vertex Degree
- In-Degree
- Out-Degree

Jumlah Edge

- Jumlah pasangan edge yang mungkin (banyak maksimal edge) dapat dilihat dari jumlah node (n).
- Dibedakan menjadi 2: untuk undi-graph dan di-graph.
- Undi-graph: (lebih kecil sama dengan) <= n(n-1)/2
- Di-graph: (lebih kecil sama dengan) <= n(n-1)
- Dengan *n* adalah jumlah node.

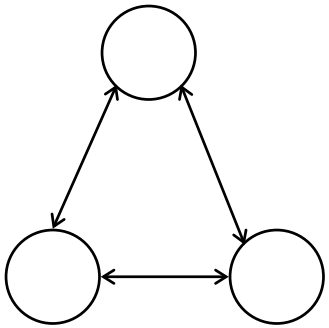
Contoh

- Undi-graph
  - jumlah maks edge: 3



n = 3

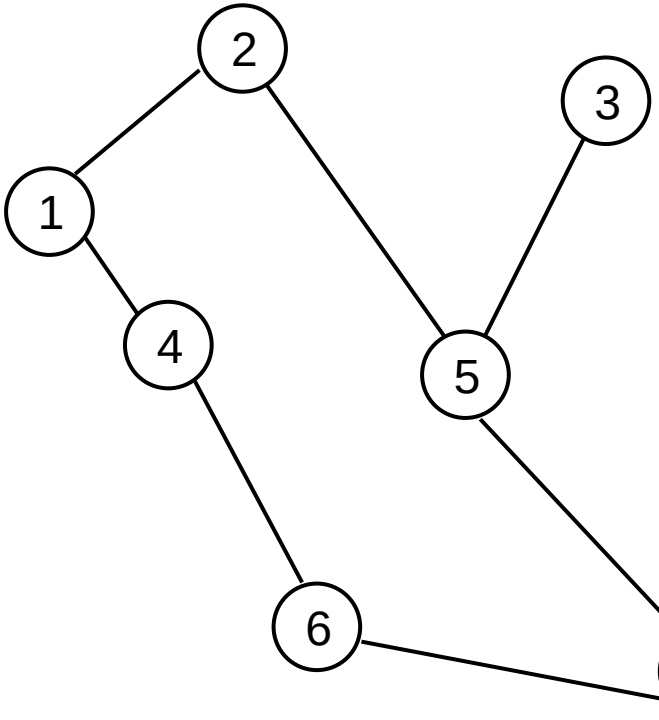
- Di-graph
  - jumlah maks edge: 6
  - dengan loop: 9



n = 3

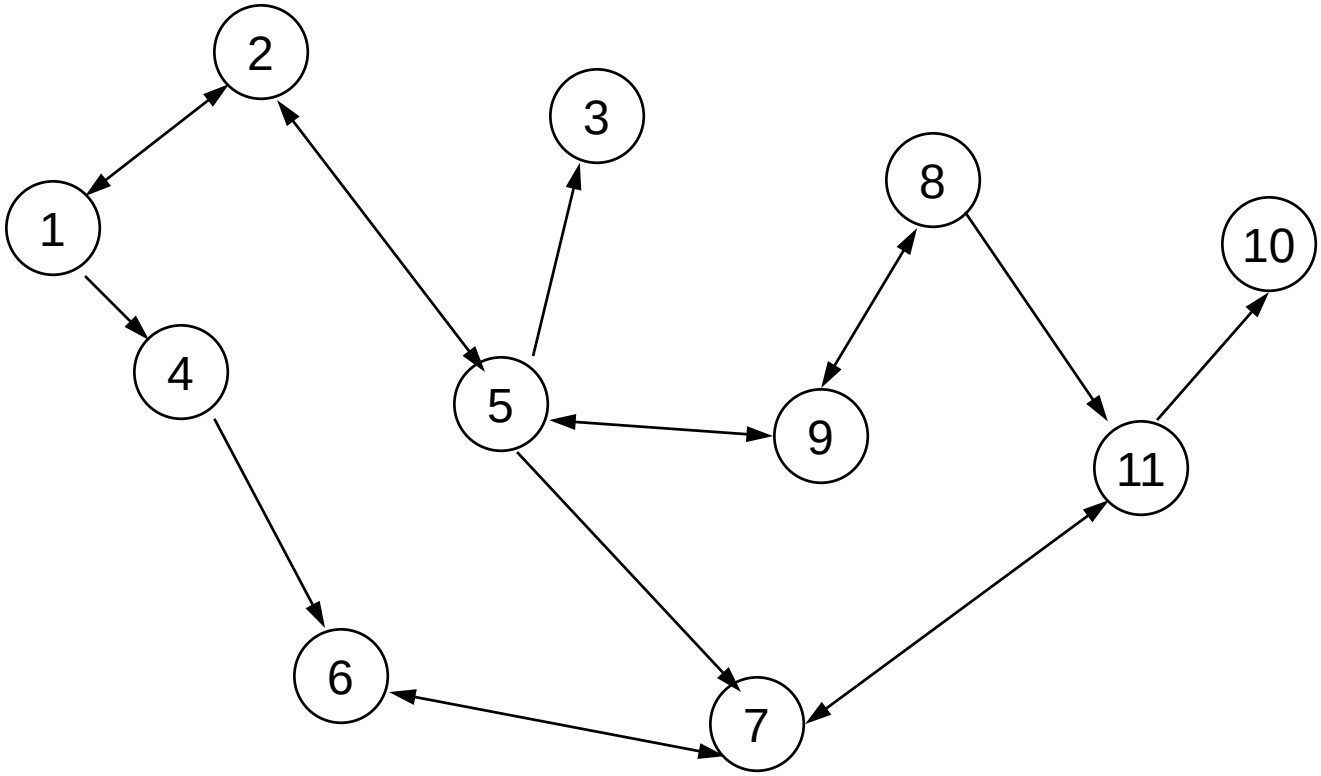
Degree

- Degree: jumlah cabang atau jumlah edge yang dimiliki node.
- Contoh undi-graph:



degree(2) = 2,
degree(5) = 3,
degree(3) = ???
degree(7) = ???

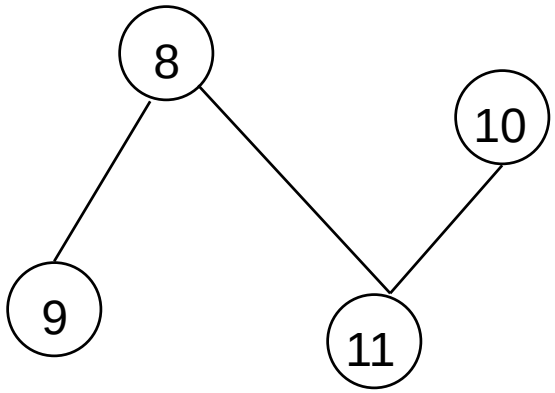
Contoh (Di-graph)



- Degree (4) =???
- Degree (7) = ???

Jumlah Degree

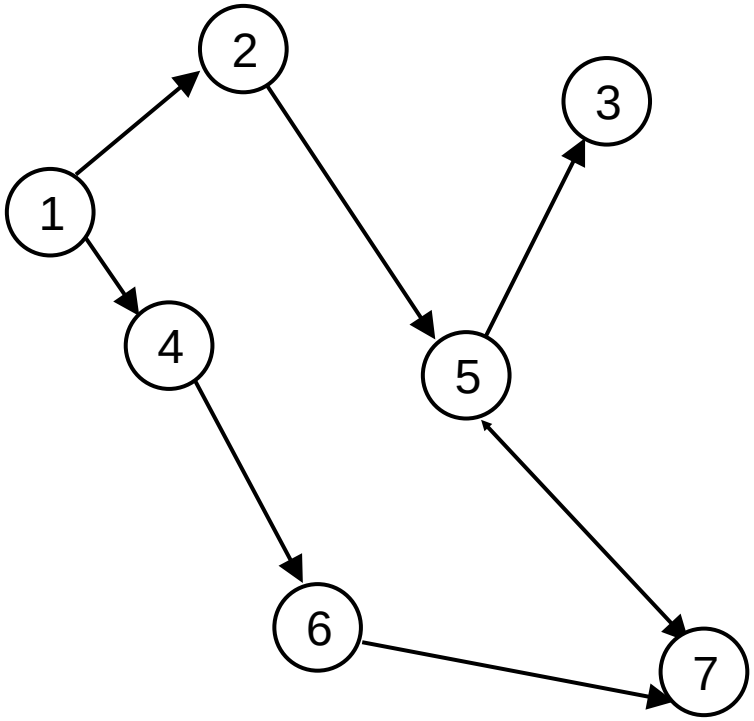
- Jumlah degree adalah jumlah total cabang/degree yang ada pada graph.
- Rumus = 2e (dengan e adalah jumlah edge)



- Hanya berlaku untuk undi-graph!

In-Degree

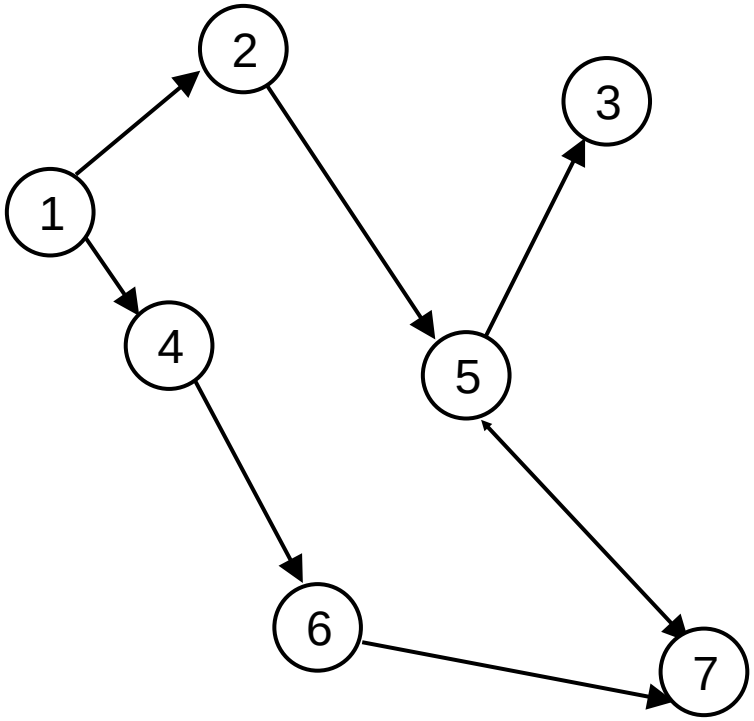
- Jumlah edge yang masuk atau mengarah ke Node.



- indegree(2) = 1, indegree(7) = 2, indegree(1)=???

Out-Degree

- Jumlah edge yang keluar dari Node.



outdegree(2) = 1, outdegree(7) = 0, outdegree(1) = ???

Representasi Graph

Representasi Graph

Representasi graph dibedakan menjadi 2:

1. Adjacency Matrix

   dapat direpresentasikan dengan matriks (array 2 dimensi).

2. Adjacency Lists

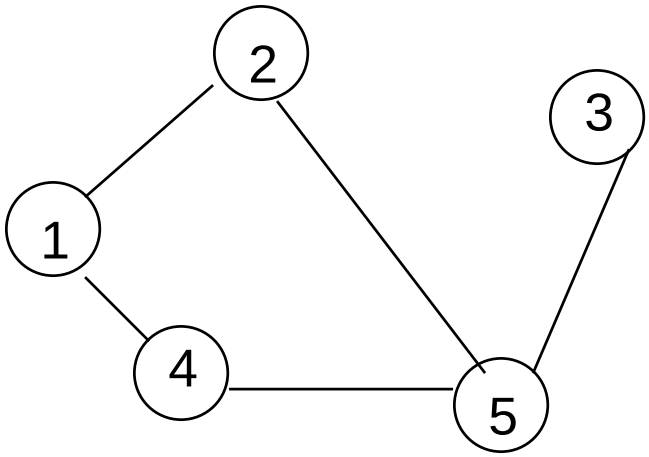
   dapat direpresentasikan dengan array (bukan berupa matriks) maupun linked list.

Adjacency Matrix

- Representasi Graph berupa Matrik ordo $n \times n$. dengan n = node.
- Baris berisi Node asal, sedangkan kolom berisi Node tujuan.
- **Jika graph tidak berbobot**,maka nilai matriks diisi dengan 1 atau 0. nilai 1 jika ada edge, dan 0 jika tidak ada edge antar node.

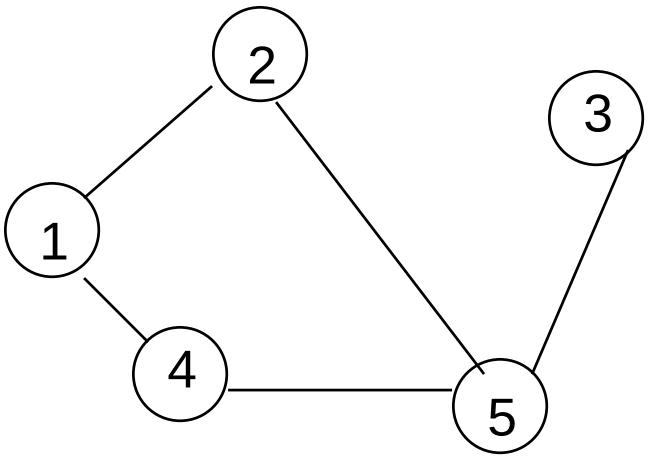
  ***A(i, j)* = 1**, jika antara node i dan node j terdapat edge/terhubung.
- **Jika graph berbobot,** maka nilai matriks diisi dengan bobot dari edge. ***A(i, j)* = nilai bobot.**

Adjacency Matrix



	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

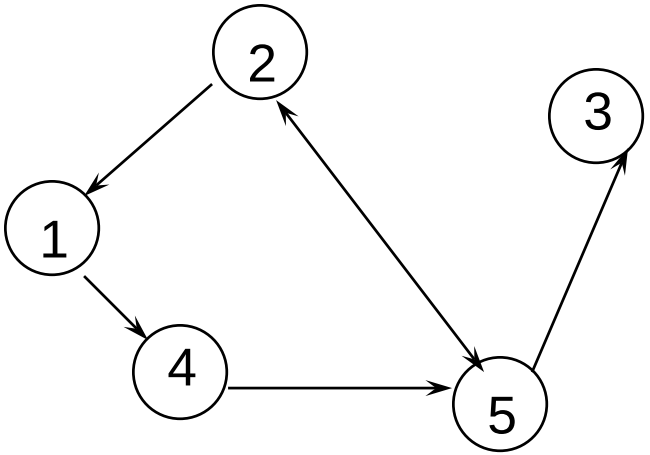
Undi-graph



	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

- Bagian diagonal berisi nol (0)
- Adjacency matrix dari undirected graph adalah simetris
  - A(i,j) = A(j,i) for all i and j.

Di-graph



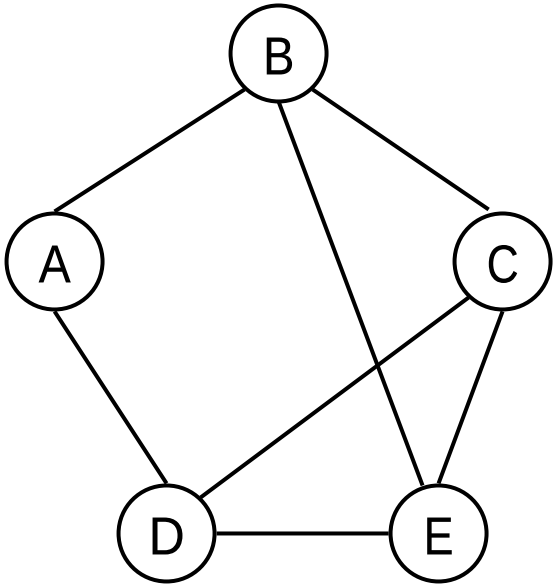
	1	2	3	4	5
1	0	0	0	1	0
2	1	0	0	0	1
3	0	0	0	0	0
4	0	0	0	0	1
5	0	1	1	0	0

- Dimungkinkan tidak simetris jika terdapat loop.

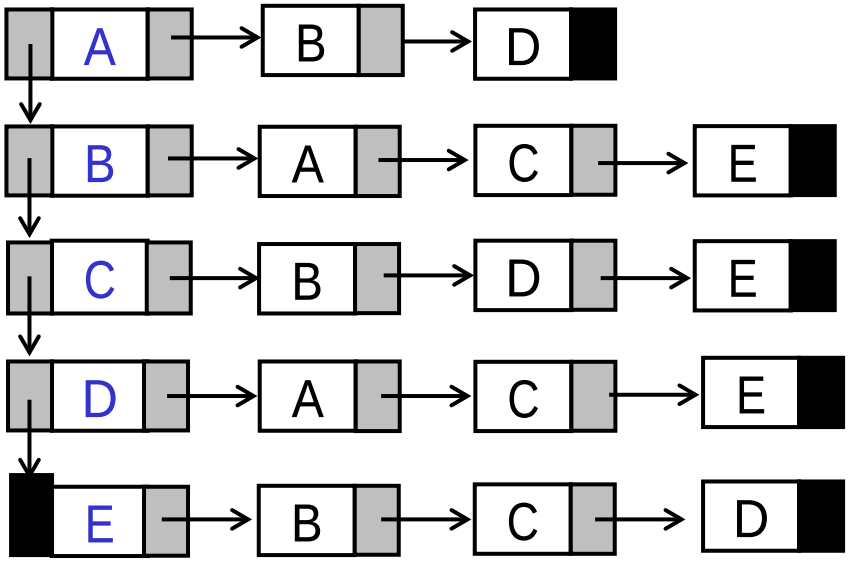
Adjacency List

- Direpresentasikan dengan linked list atau array.
  - Array list: array dua dimensi namun tidak ber-ordo $n \times n$.
  - Linked list: array of single linked list

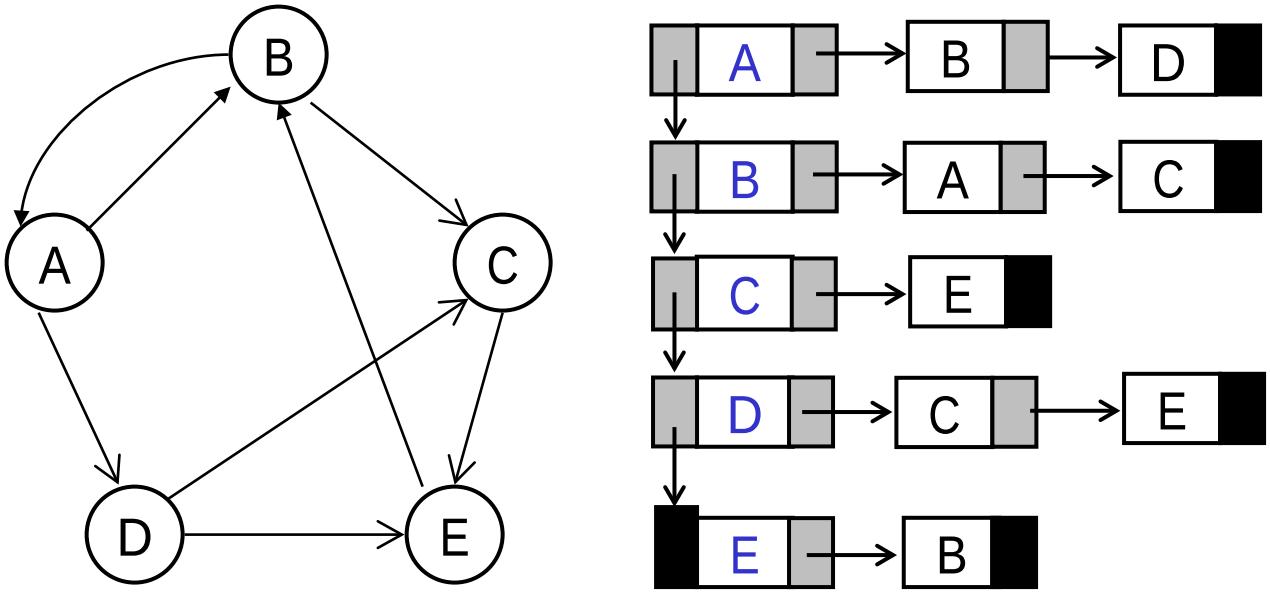
Undirected Graph



Graph



Directed Graph



Latihan

1. Gambarkan graph dari representasi Matrik berikut:

Direpresentasikan dalam matriks sbb.

Dari\Ke	A	B	C	D	E	F	G	H	I	J	K	L	M
A	-	1	1	1	0	1	0	0	0	0	0	0	0
B	1	-	1	0	0	0	0	1	0	0	0	0	0
C	1	1	-	0	1	0	1	1	1	0	0	0	0
D	1	0	0	-	1	1	1	0	0	0	1	1	0
E	0	0	1	1	-	1	0	0	0	0	0	0	0
F	1	0	0	1	1	-	0	0	0	0	0	0	0
G	0	0	1	1	0	0	-	0	1	0	1	0	0
H	0	1	1	0	0	0	0	-	1	0	0	0	0
I	0	0	1	0	0	0	1	1	-	1	0	0	1
J	0	0	0	0	0	0	0	0	1	-	1	0	1
K	0	0	0	1	0	0	1	0	0	1	-	1	0
L	0	0	0	1	0	0	0	0	0	0	1	-	1
M	0	0	0	0	0	0	0	0	1	1	0	1	-

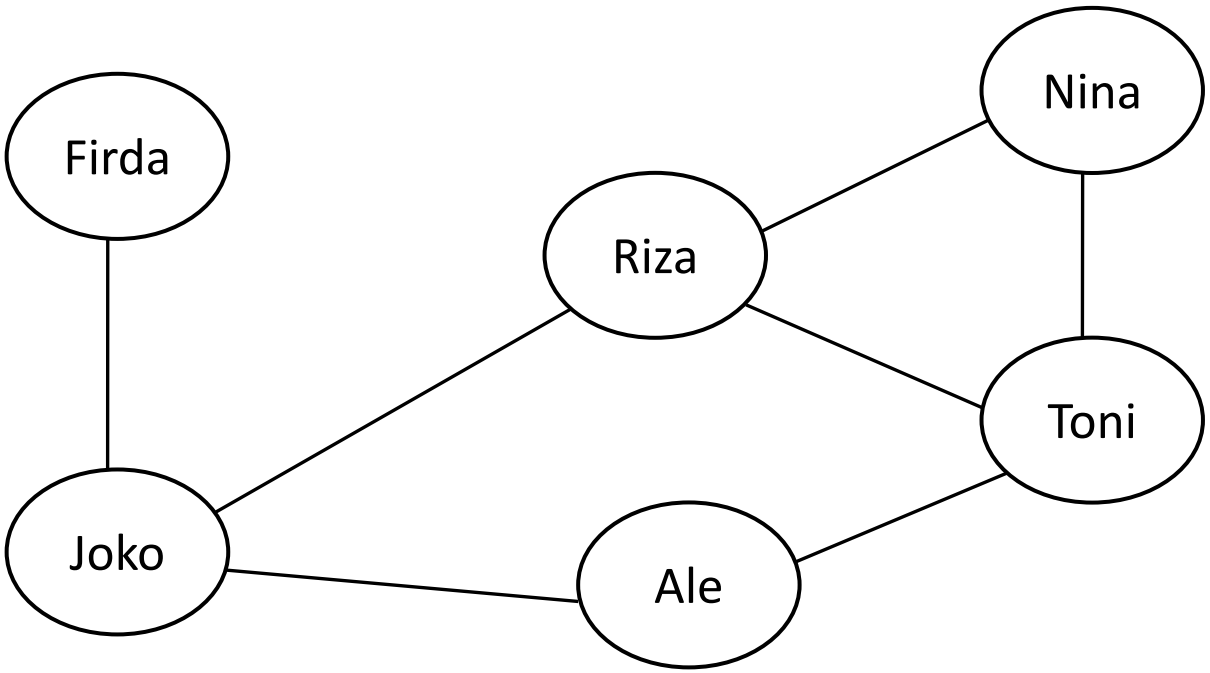
Latihan

2. Gambarkan graph dari representasi matriks berikut:

Dari\Ke	A	B	C	D	E	F	G	H	I	J	K	L	M
A	-	24	43	33	∞	31	∞	∞	∞	∞	∞	∞	∞
B	24	-	18	∞	∞	∞	∞	45	∞	∞	∞	∞	∞
C	43	18	-	∞	16	∞	22	35	15	∞	∞	∞	∞
D	33	∞	∞	-	19	22	39	∞	∞	∞	13	27	∞
E	∞	∞	16	19	-	15	∞	∞	∞	∞	∞	∞	∞
F	31	∞	∞	22	15	-	∞	∞	∞	∞	∞	∞	∞
G	∞	∞	22	39	∞	∞	-	∞	21	∞	13	∞	∞
H	∞	45	35	∞	∞	∞	∞	-	25	∞	∞	∞	∞
I	∞	∞	15	∞	∞	∞	21	25	-	19	∞	∞	35
J	∞	∞	∞	∞	∞	∞	∞	∞	19	-	10	∞	15
K	∞	∞	∞	13	∞	∞	13	∞	∞	10	-	19	∞
L	∞	∞	∞	27	∞	∞	∞	∞	∞	∞	19	-	25
M	∞	∞	∞	∞	∞	∞	∞	∞	35	15	∞	25	-

Latihan

3. Representasikan dengan adjacency list & adjacency matrix



Penelusuran Graph

Metode Penelusuran

- Graph Traversal: Mengunjungi tiap simpul/node secara sistematik.
- Metode:
  - DFS (Depth First Search): Pencarian Mendalam
  - BFS (Breadth First Search): Pencarian Melebar

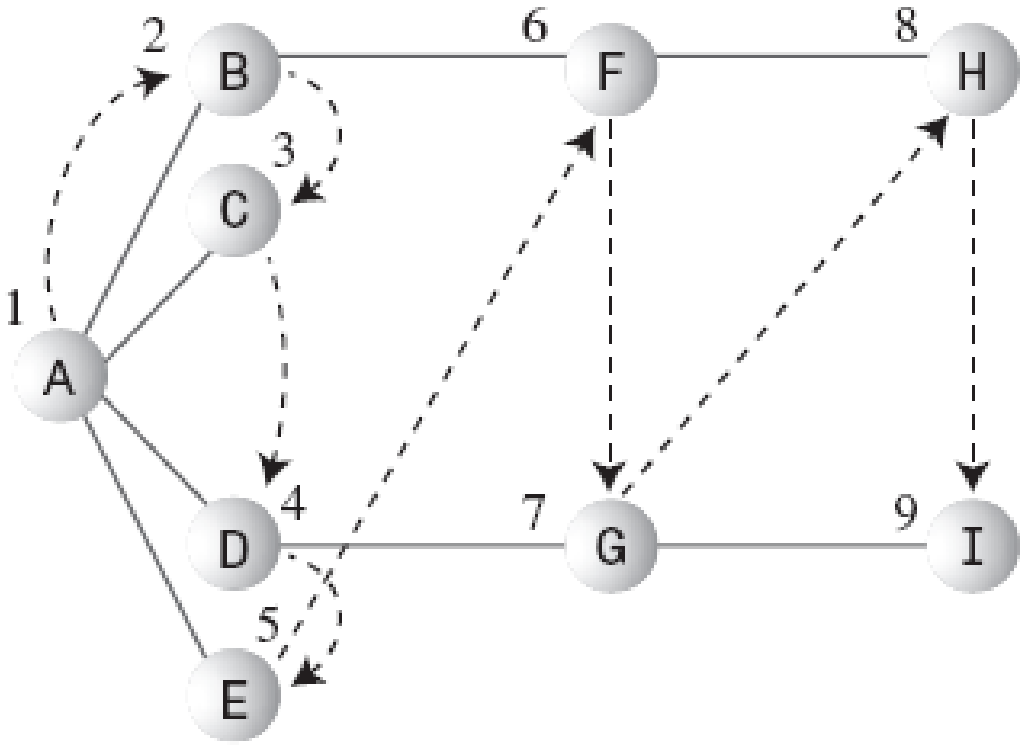
algoritme BFS

- BFS diawali dengan vertex yang diberikan, yang mana di level 0.
- Dalam stage pertama, kita kunjungi semua vertex di level 1.
- Stage kedua, kita kunjungi semua vertex di level 2. Disini vertex baru, yang mana adjacent ke vertex level 1, dan seterusnya.
- Penelusuran BFS berakhir ketika setiap vertex selesai ditemui.

algoritme BFS

- Traversal dimulai dari simpul *v.*
- algoritme:
  1. Kunjungi simpul *v,*
  2. Kunjungi semua simpul yang bertetangga dengan simpul *v* terlebih dahulu.
  3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.

Breadth First Search (BFS)



Urutan verteks hasil penelusuran: ABCDEFGHI

Breadth First Search (BFS)

Event	Queue (Front to Rear)
Visit A	
Visit B	B
Visit C	BC
Visit D	BCD
Visit E	BCDE
Remove B	CDE
Visit F	CDEF
Remove C	DEF
Remove D	EF
Visit G	EFG
Remove E	FG
Remove F	G
Visit H	GH
Remove G	H
Visit I	HI
Remove H	I
Remove I	
Done	

→ .... → ....

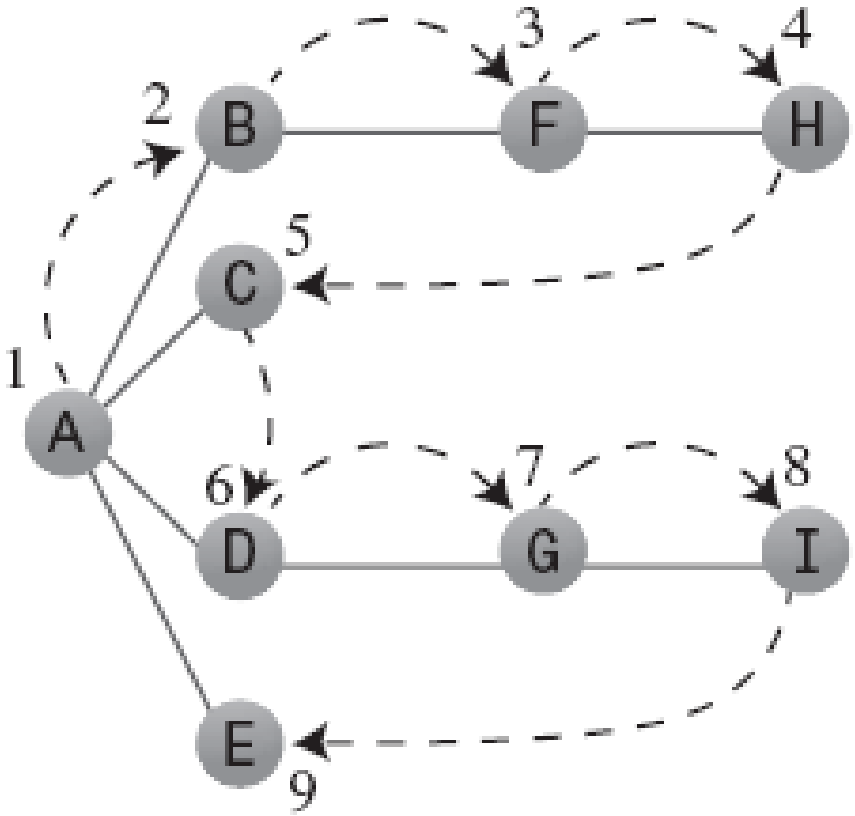
Depth First Search (DFS)

- Pada setiap pencabangan, penelusuran verteks-verteks yang belum dikunjungi dilakukan secara lengkap pada pencabangan pertama, kemudian selengkapnya pada pencabangan kedua, dan seterusnya secara rekursif.

algoritme DFS

- Traversal dimulai dari simpul v.
- Algoritme:
  1. Kunjungi simpul v,
  2. Kunjungi simpul w yang bertetangga dengan simpul v.
  3. Ulangi DFS mulai dari simpul w.
  4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
  5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Depth First Search (DFS)



Urutan verteks hasil penelusuran: ABFHCDGIE

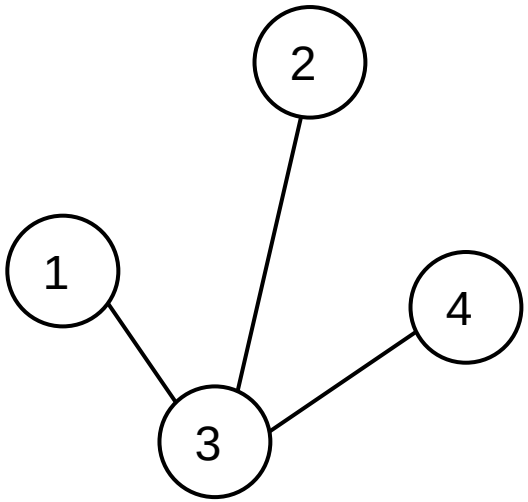
Depth First Search (DFS)

Event	Stack
Visit A	A
Visit B	AB
Visit F	ABF
Visit H	ABFH
Pop H	ABF
Pop F	AB
Pop B	A
Visit C	AC
Pop C	A
Visit D	AD
→	
.... →	
Visit G	ADG
Visit I	ADGI
Pop I	ADG
Pop G	AD
Pop D	A
Visit E	AE
Pop E	A
Pop A	
Done	

Latihan

1. Telusuri graph disamping dengan mengunakan BFS dan DFS. Secara berturut-urut root dimulai dari 1,2,3 dan 4.

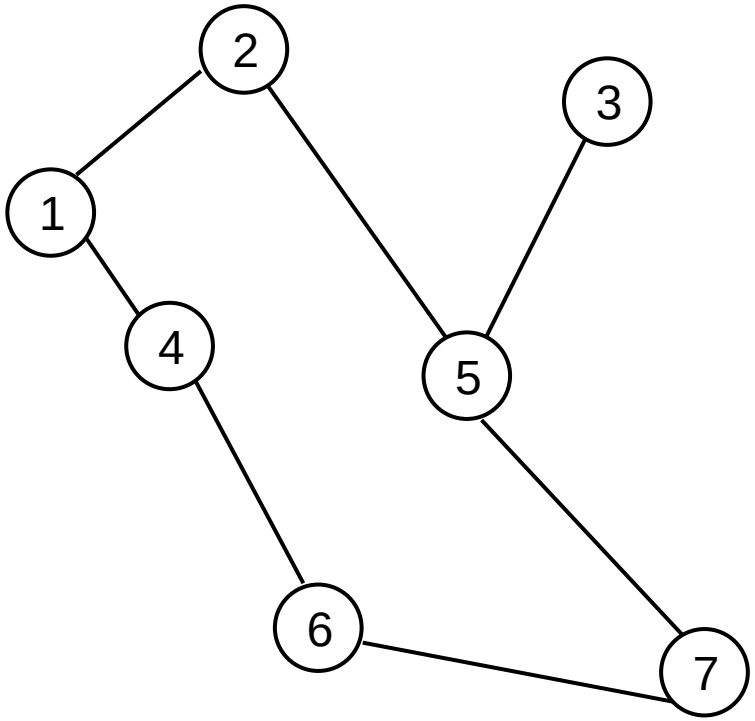
   Bandingkan hasilnya!



Latihan

2. Telusuri dengan BFS dan DFS!

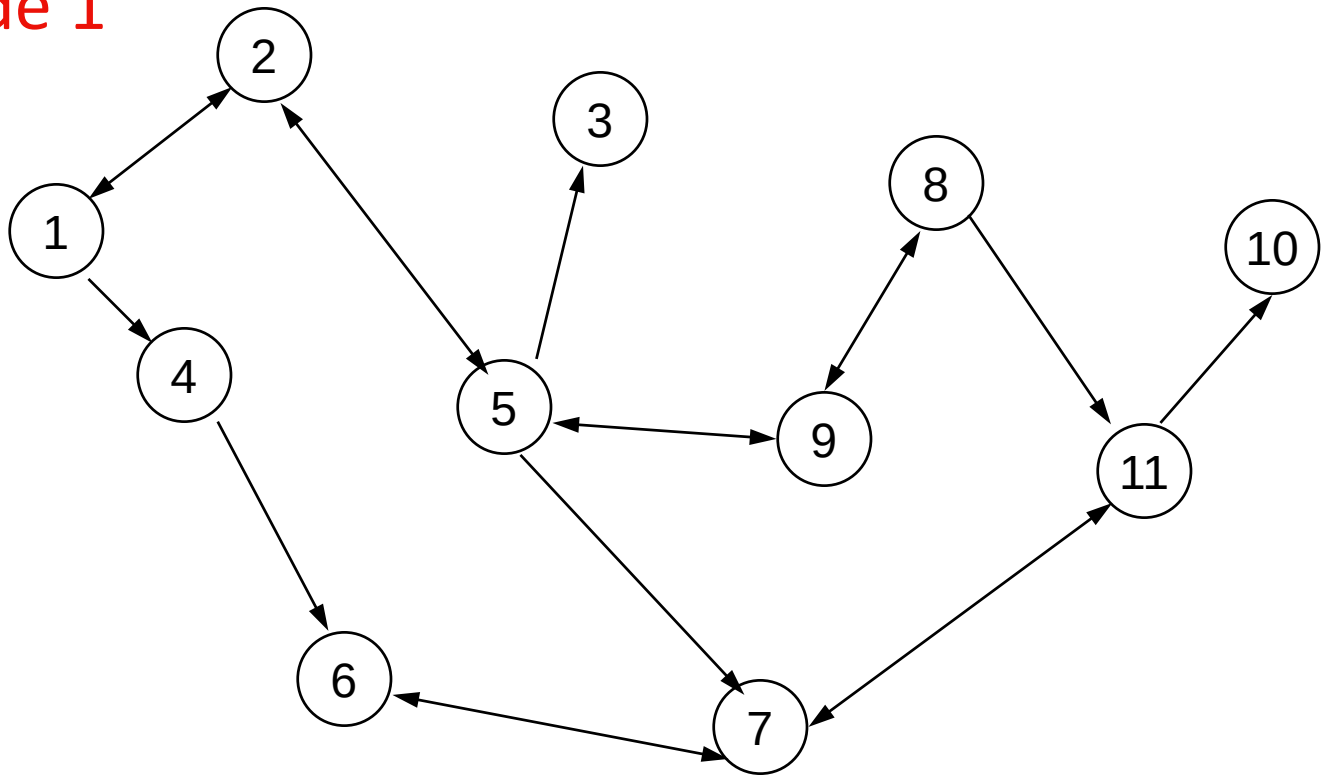
   root: node 1



Latihan

3. Telusuri dengan BFS dan DFS!

   root: node 1



Implementasi Program

Operasi-operasi

Menggunakan adjacency matrix, operasi-operasinya sebagai berikut:

1. Deklarasi
2. Inisialisasi
3. Penambahan node
4. Penambahan edge
5. Menandai Node
6. Traversal
7. Display node

Contoh Program

- Deklarasi

```
public class AdjacencyMatriksGraph {
    private final int MAX_VERTS = 20;
    private Vertex vertexList[];
    private int adjMat[][];
    private int nVerts;
    private StackX theStack;
    private Queue theQueue;
```

Contoh Program

- Inisialisasi

```
public AdjacencyMatriksGraph() // constructor
{
    vertexList = new Vertex[MAX_VERTS];
    // adjacency matrix
    adjMat = new int[MAX_VERTS][MAX_VERTS];
    nVerts = 0;
    for(int j=0; j<MAX_VERTS; j++) // set adjacency
        for(int k=0; k<MAX_VERTS; k++) // matrix to 0
            adjMat[j][k] = 0;
    theStack = new StackX();
    theQueue = new Queue();
}
```

Contoh Program

- Tambah Node

```
public void addVertex(char lab)
{
        vertexList[nVerts++] = new Vertex(lab);
}
```

Contoh Program

- Tambah Edge

```
public void addEdge(int start, int end)
{
        adjMat[start][end] = 1;
        adjMat[end][start] = 1;
}
```

Contoh Program

- Menandai Node

```
public int getAdjUnvisitedVertex(int v)
{
    for(int j=0; j<nVerts; j++)
        if(adjMat[v][j]==1 && vertexList[j].wasVisited==false)
            return j;
    return -1;
}
```

- DFS

```
public void dfs() // depth-first search
{ // begin at vertex 0
    vertexList[0].wasVisited = true; // mark it
    displayVertex(0); // display it
    theStack.push(0); // push it
    while( !theStack.isEmpty() ) // until stack empty,
    {

        int v = getAdjUnvisitedVertex( theStack.peek() );
        if(v == -1) // if no such vertex,
        theStack.pop();
        else // if it exists,
        {
            vertexList[v].wasVisited = true;
            displayVertex(v); // display it
            theStack.push(v); // push it
        }
    } // end while
    for(int j=0; j<nVerts; j++) // reset flags
        vertexList[j].wasVisited = false;
}
```

- BFS

```
public void bfs() // breadth-first search
{ // begin at vertex 0
    vertexList[0].wasVisited = true; // mark it
    displayVertex(0); // display it
    theQueue.insert(0); // insert at tail
    int v2;
    while( !theQueue.isEmpty() )
    {
        int v1 = theQueue.remove();
        // until it has no unvisited neighbors
        while( (v2=getAdjUnvisitedVertex(v1)) != -1 )
        { // get one,
            vertexList[v2].wasVisited = true; // mark it
            displayVertex(v2); // display it
            theQueue.insert(v2); // insert it
        } // end while
    } // end while(queue not empty)
    // queue is empty, so we're done
    for(int j=0; j<nVerts; j++) // reset flags
        vertexList[j].wasVisited = false;
}
```

Contoh Program

- Display Node

```
public void displayVertex(int v)
{
    System.out.print(vertexList[v].label);
}
```