

PERTEMUAN 13

GRAPH

Hermanto, S.Kom. M.Kom

Sub Topik

- Overview : Graph
- Jenis Graph
- Implementasi Graph
- Graph Property
- Representasi Graph
- Metode Penelusuran Graph

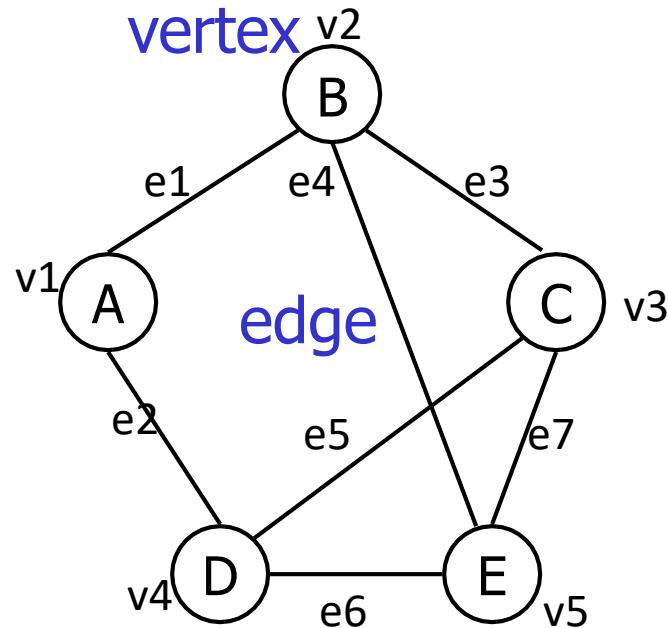
Graph

- Graph adalah struktur data yang memiliki relasi **many to many**, yaitu tiap element dapat memiliki 0 atau lebih dari 1 cabang.
- Graph merupakan kumpulan dari vertex dan edge yang secara matematis dinyatakan sebagai :

$$G = (V, E)$$

- Dimana :
 - G** adalah Graph
 - V** adalah Vertex, menyatakan entitas data.
 - E** adalah Edge, menyatakan garis penghubung antar node.
- Vertex (Vertices) disebut juga sebagai node atau point.
- Edge disebut juga sebagai arcs atau lines.
- **Setiap edge menghubungkan dua vertices.**

Contoh graph :

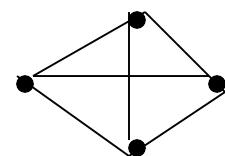
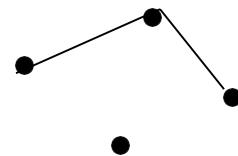
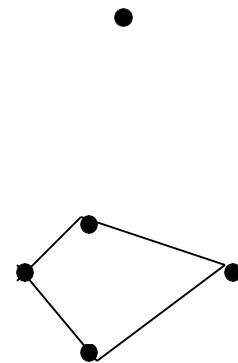


V terdiri dari v_1, v_2, \dots, v_5

E terdiri dari e_1, e_2, \dots, e_7

Undirected graph

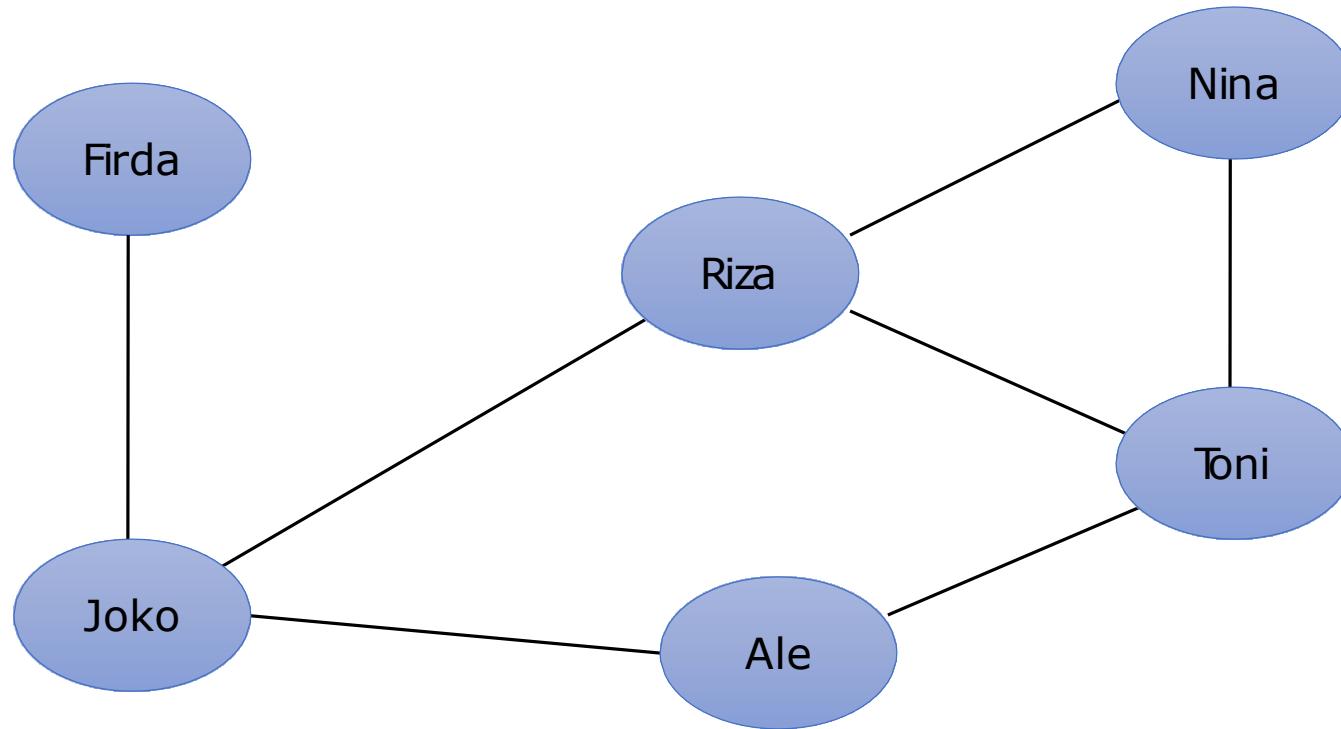
- Sebuah graph mungkin hanya terdiri dari satu simpul
- Sebuah graph belum tentu semua simpulnya terhubung dengan busur
- Sebuah graph mungkin mempunyai simpul yang tak terhubung dengan simpul yang lain
- Sebuah graph mungkin semua simpulnya saling berhubungan



Contoh Graph

- Contoh persoalan di dunia nyata yang dapat direpresentasikan dengan graph adalah : Jaringan pertemanan pada Facebook.

Jaringan pertemanan pada Facebook



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

Penjabaran

- Node(vertex): para pemakai Facebook
- Lines(edge) : Garis penghubung antara pemakai satu dengan yang lain.
- Sehingga dari contoh graph diatas dapat dijabarkan sebagai berikut :

$$V = \{Nina, Toni, Ale, Riza, Joko, Firda\}$$

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

Jenis Graph

- Directed Graph (Digraph)
- Undirected Graph
 - Complete Undirected Graph
 - Connected Undirected Graph
- Weight Graph

Directed Graph (Digraph)

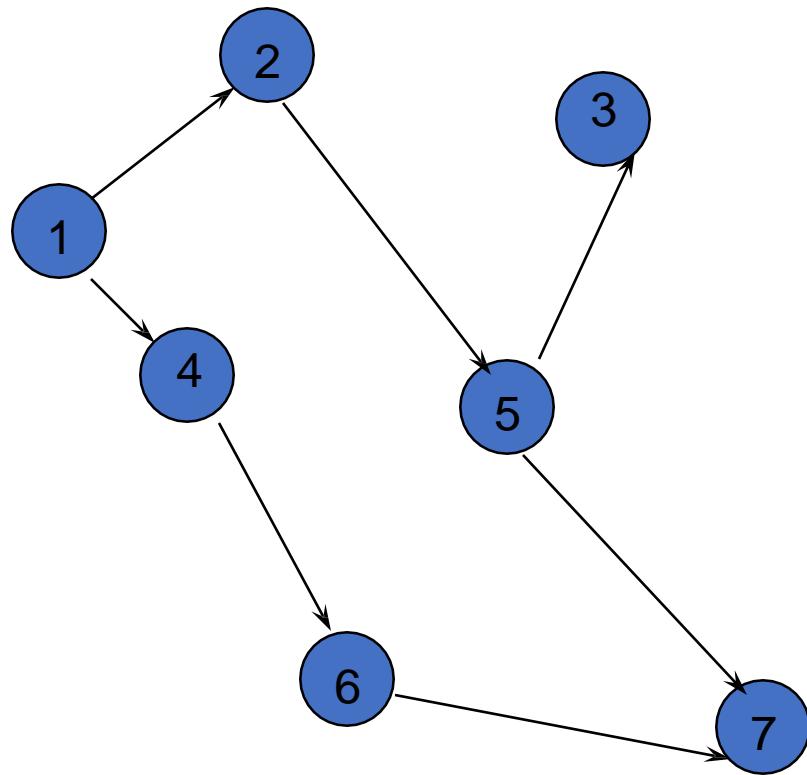
- Graph yang memiliki orientasi/arah.
- Setiap lines/edge *Digraph* memiliki anak panah yang mengarah ke node tertentu.
- **Secara notasi sisi digraph ditulis sebagai vektor (u, v) .**

u = origin (vertex asal)

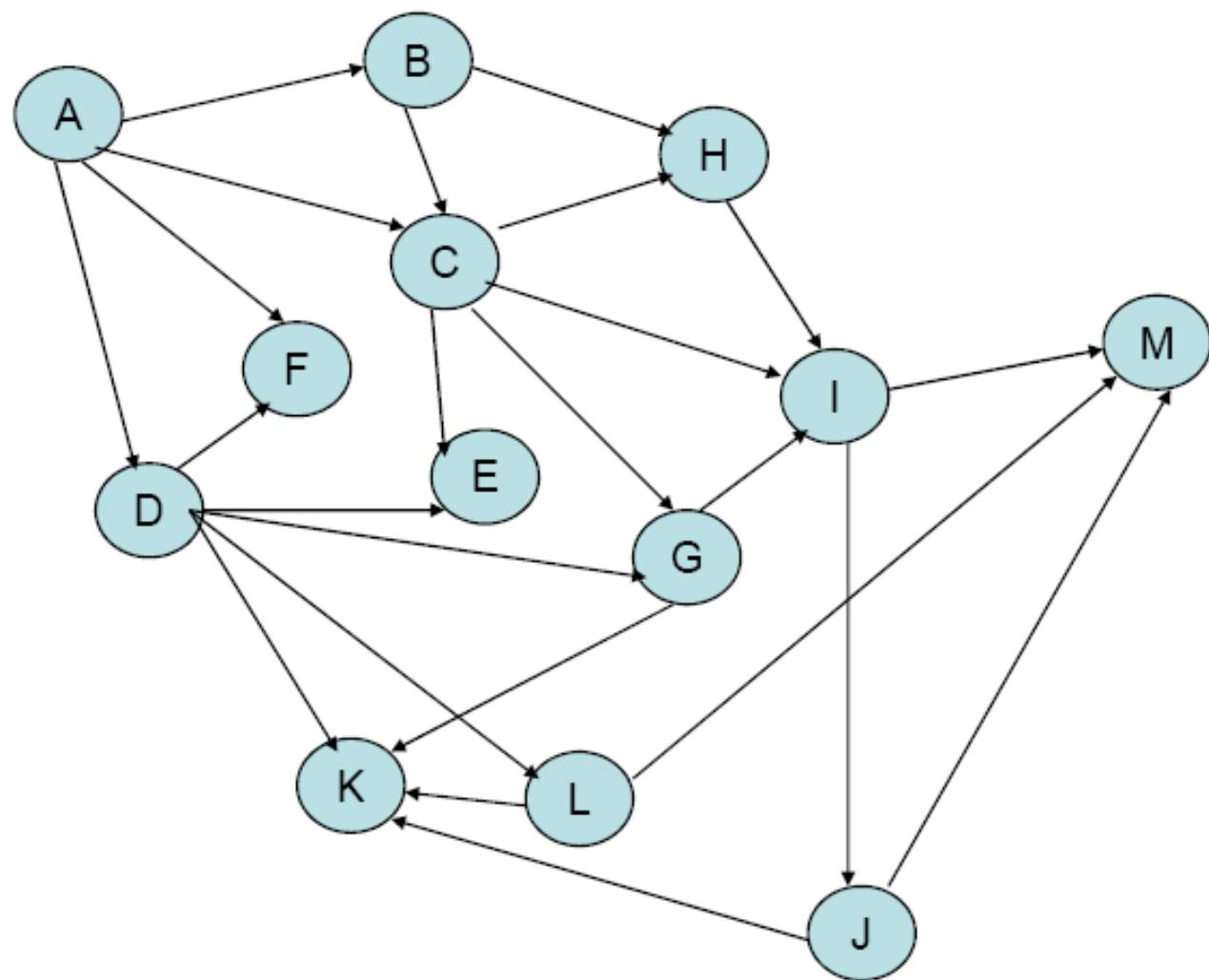
v = terminus (vertex tujuan)

$$u \longrightarrow v$$

Contoh Digraph (1)



Gambar Digraph (2)



Contoh Digraph (2)

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

$$V = \{A, B, C, D, E, F, G, H, I, J, K, L, M\}$$

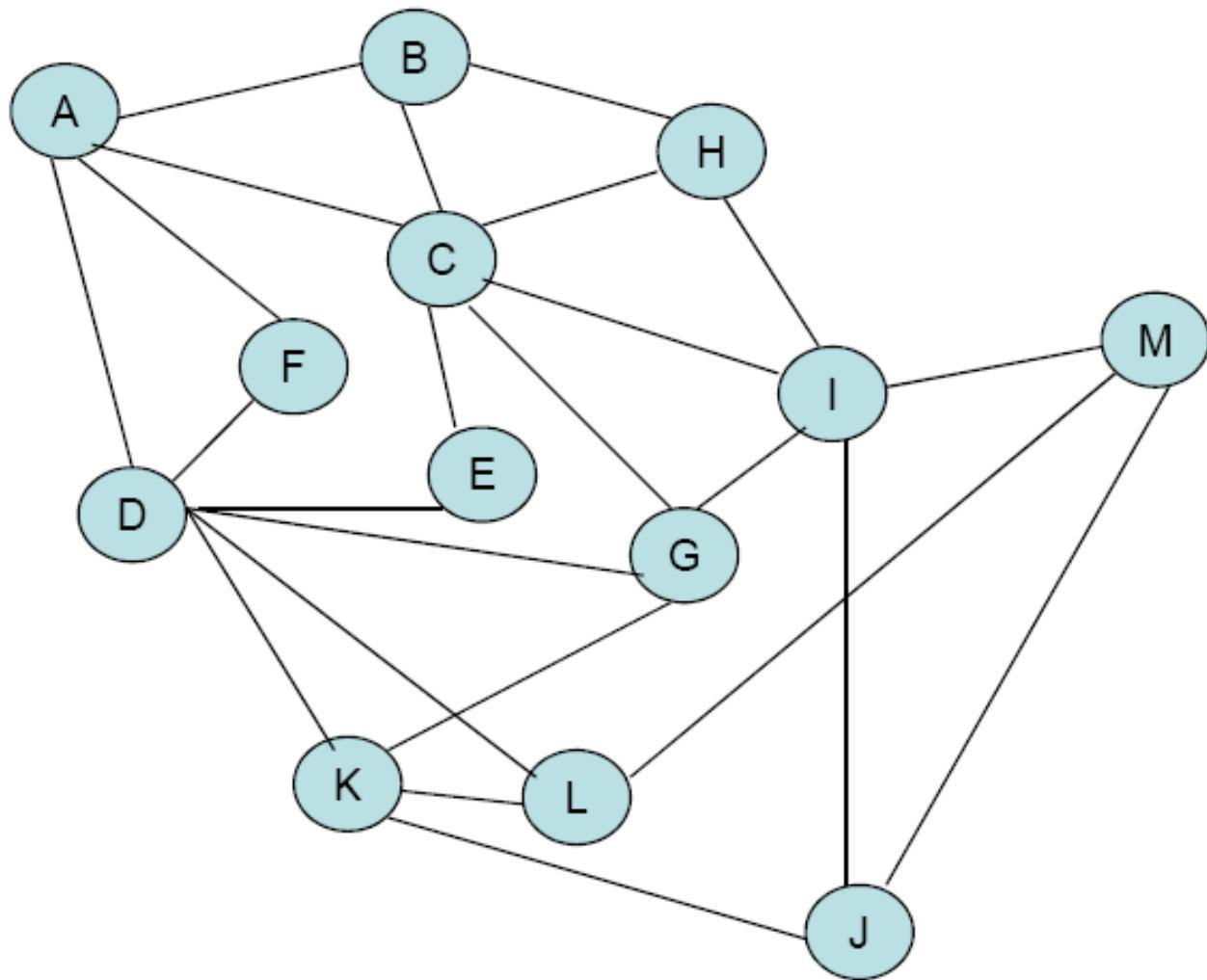
$$\begin{aligned}E = & \{(A,B), (A,C), (A,D), (A,F), (B,C), (B,H), & (C,E), (C,G), (C,H), (C,I), \\& (D,E), & (D,F), (D,G), (D,K), (D,L), (E,F), (G,I), & (G,K), (H,I), (I,J), (I,M), \\& (J,K), (J,M), & (L,K), (L,M)\}.\end{aligned}$$

Undirected Graph (Digraph)

- Graph yang tidak memiliki orientasi/arah.
- Setiap sisi $\{u, v\}$ berlaku pada kedua arah.
- Misalnya : $\{x, y\}$. 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.

$u — v \quad \{U, V\} \text{ atau } \{V, U\}$

Gambar Undi-Graph



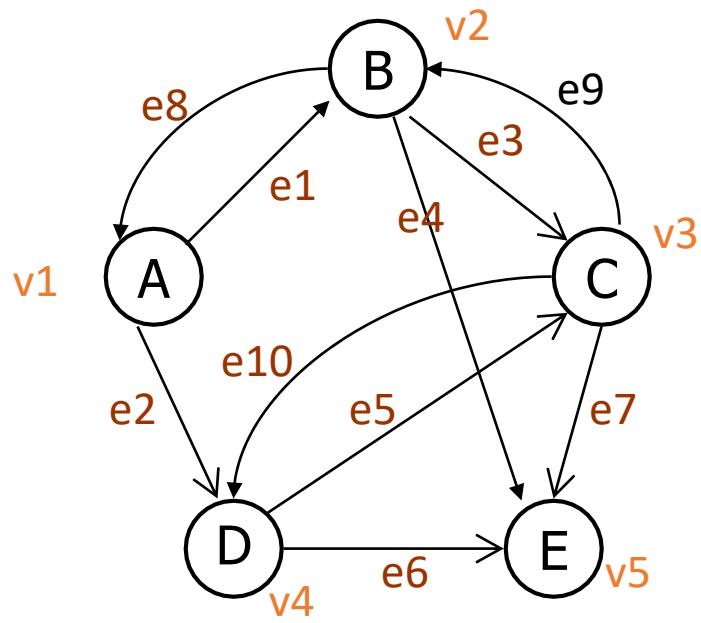
Contoh Undi-Graph

$G = \{V, E\}$

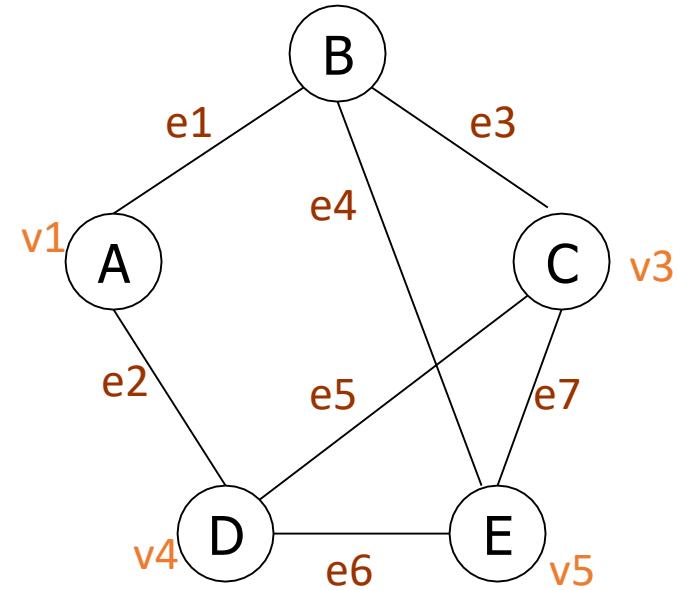
$V = \{A, B, C, D, E, F, G, H, I, J, K, L, M\}$

$E = \{ \{A,B\}, \{A,C\}, \{A,D\}, \{A,F\}, \{B,C\}, \{B,H\}, \{C,E\}, \{C,G\}, \{C,H\}, \{C,I\}, \{D,E\}, \{D,F\}, \{D,G\}, \{D,K\}, \{D,L\}, \{E,F\}, \{G,I\}, \{G,K\}, \{H,I\}, \{I,J\}, \{I,M\}, \{J,K\}, \{J,M\}, \{L,K\}, \{L,M\} \}.$

Graph Berarah dan Graph Tak Berarah :



Directed graph

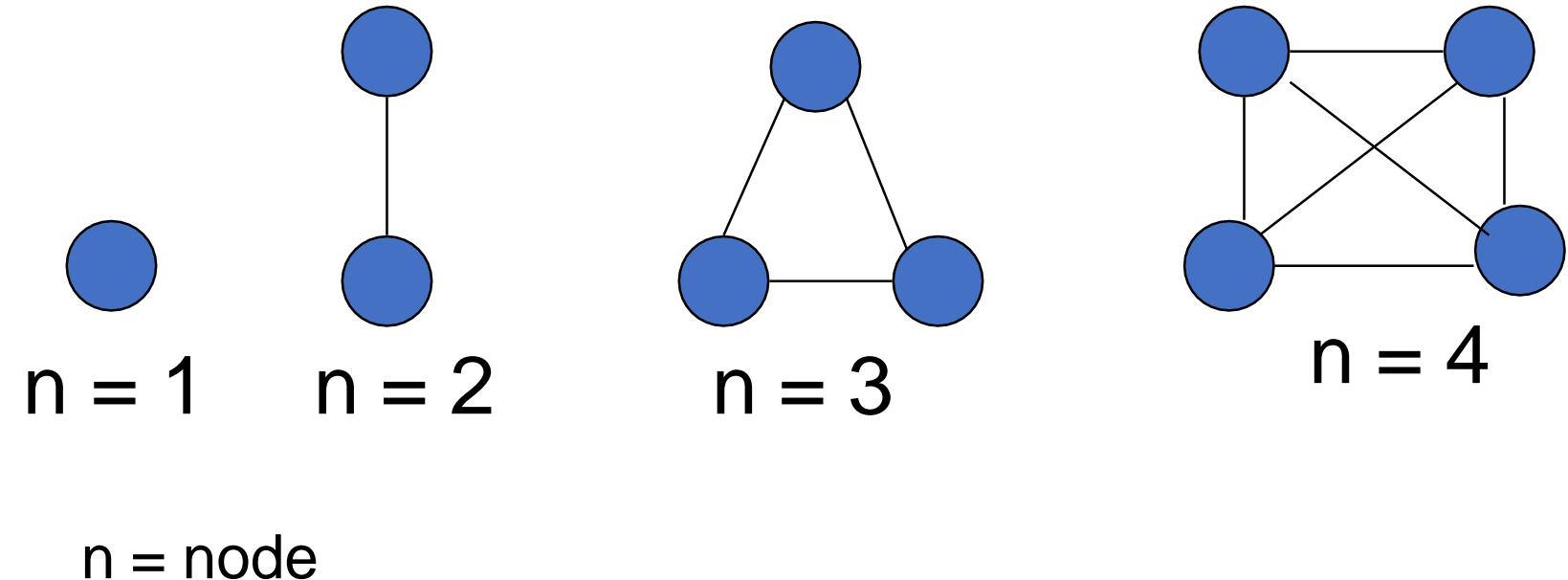


Undirected graph

Dapat dilihat dari bentuk edge yang artinya urutan penyebutan pasangan 2 simpul.

Complete Undirected Graph

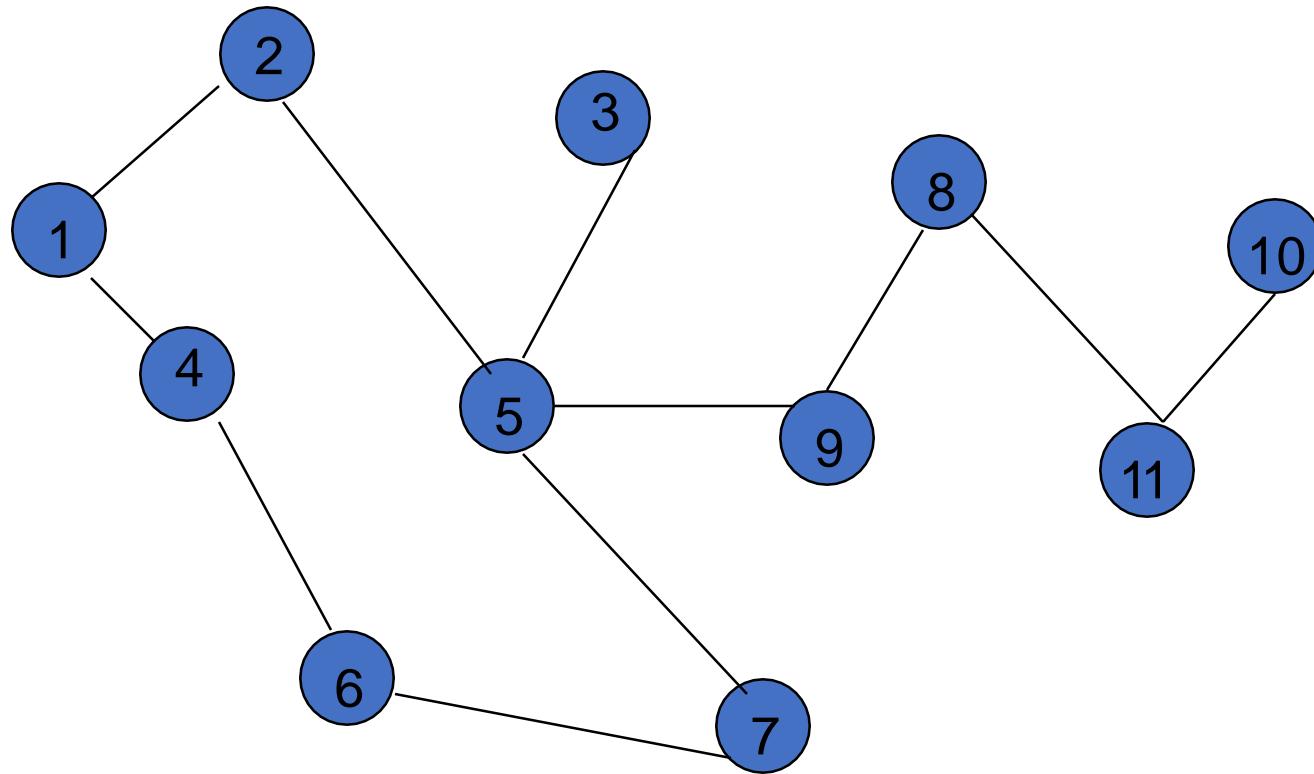
- Semua node memiliki edge/lines untuk setiap node pada graph.



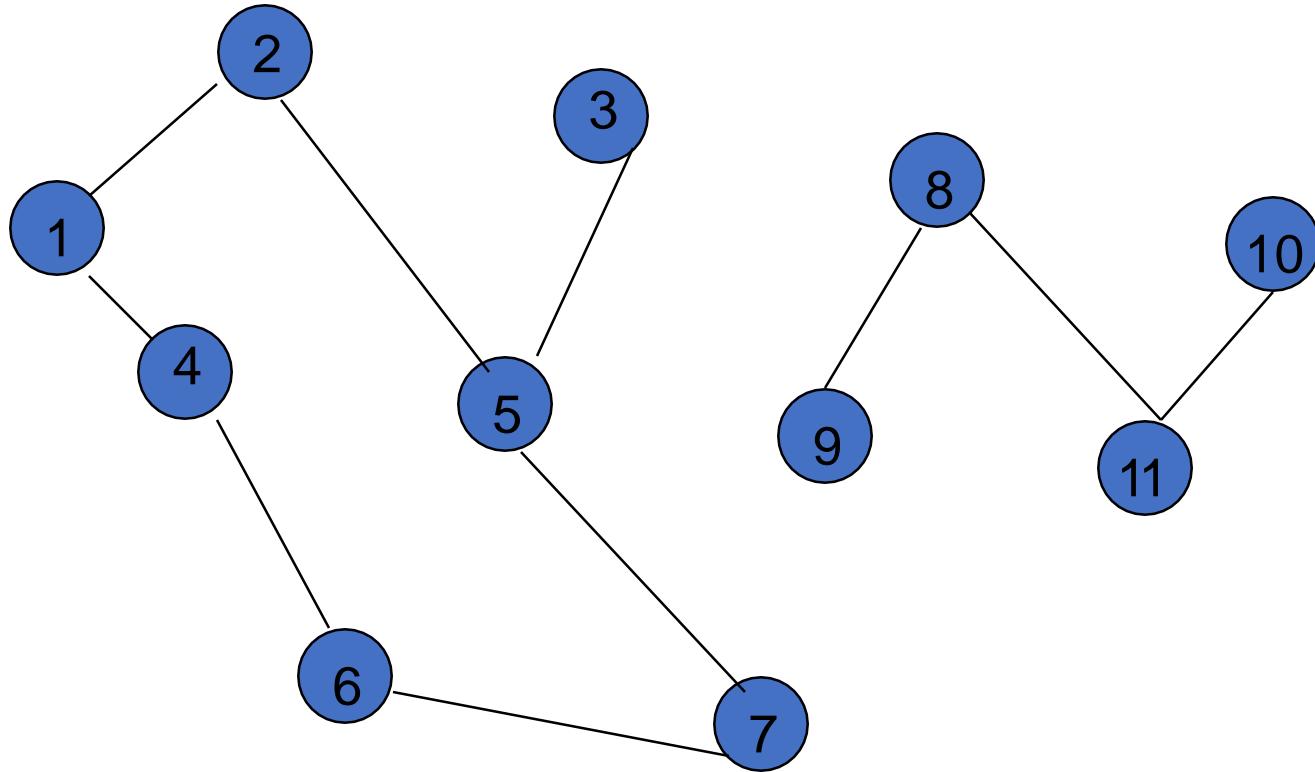
Connected Graph

- Termasuk Jenis Undirected graph.
- Setiap pasang vertex memiliki edge.

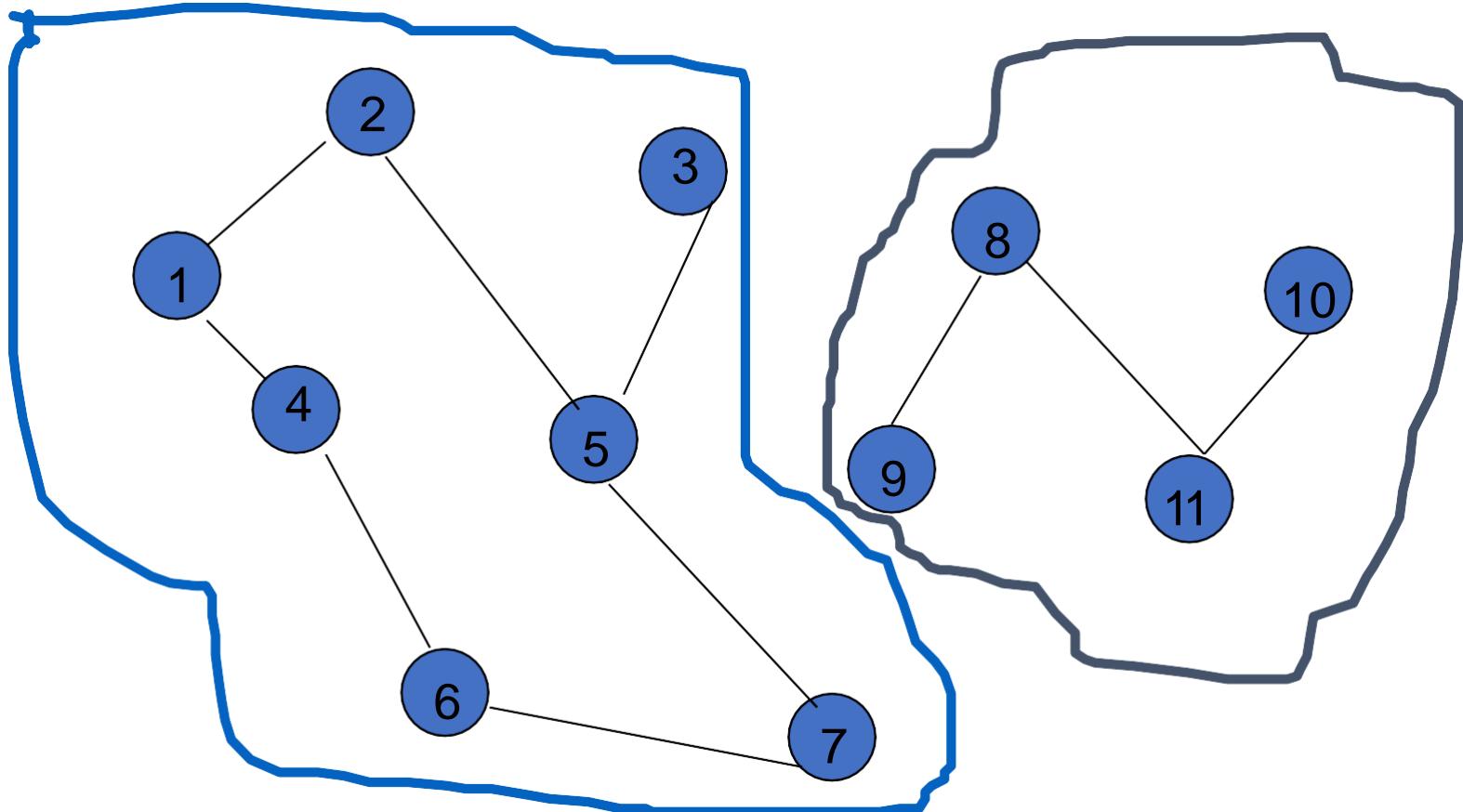
Contoh Connected Graph



Contoh Not Connected Graph



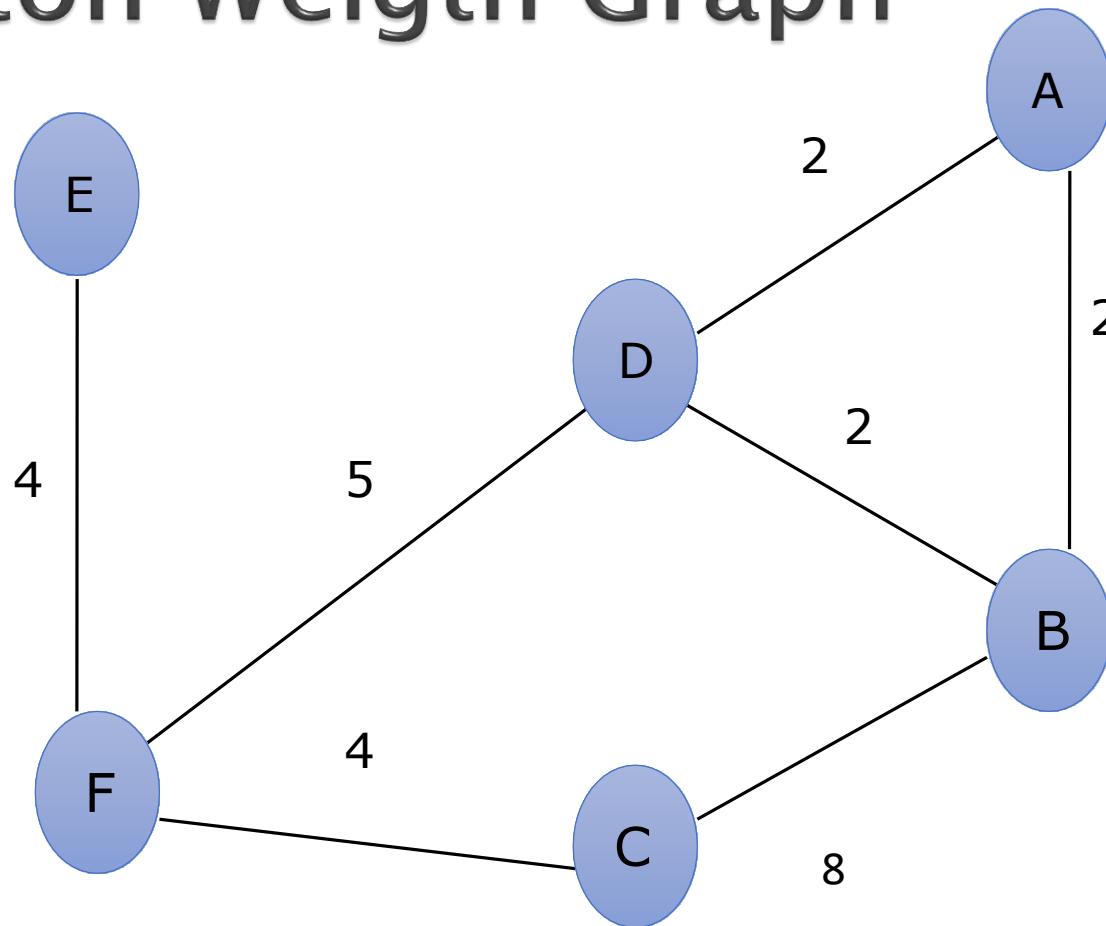
Connected Components



Weigth Graph

- Jika semua lines/edge dalam graph memiliki bobot/nilai (value) yang menyatakan hubungan antara 2 buah simpul, maka lines tersebut dinyatakan memiliki bobot.
- Bobot sebuah lines dapat menyatakan panjang sebuah jalan dari 2 buah titik, jumlah rata-rata kendaraan perhari yang melalui sebuah jalan, dll.
- Graph yang memiliki bobot, yaitu pada tiap edge-nya memiliki nilai.

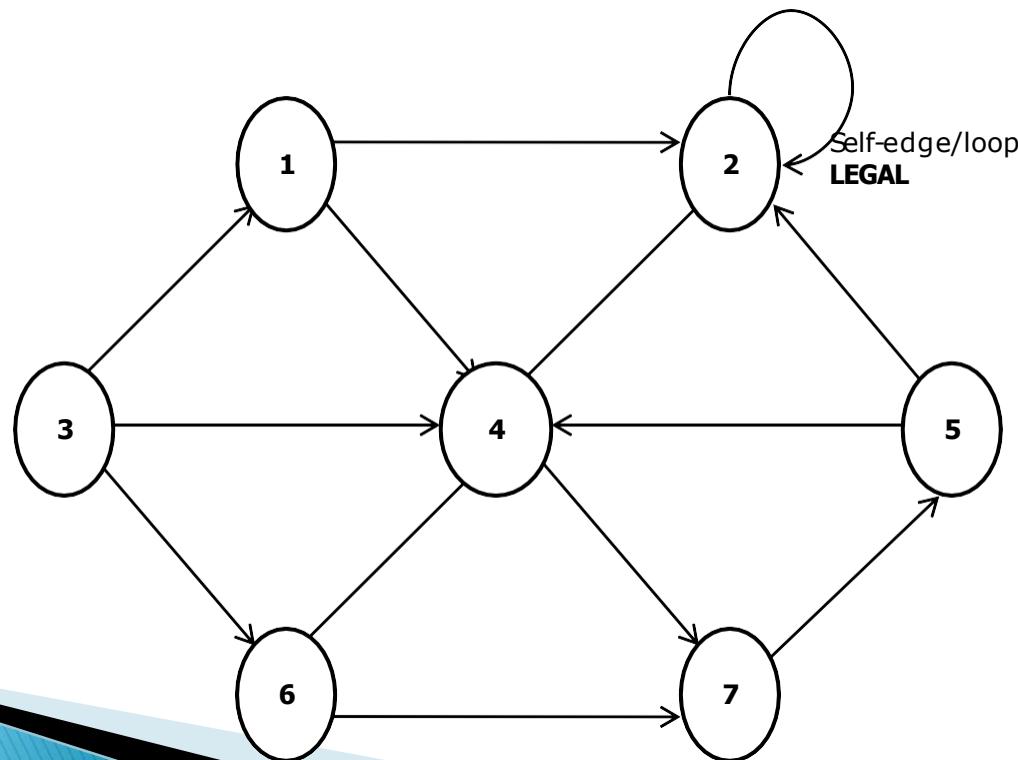
Contoh Weigth Graph



Panjang busur (atau bobot) mungkin tidak digambarkan secara panjang yang proposional dengan bobotnya. Misal bobot 5 digambarkan lebih panjang dari 8.

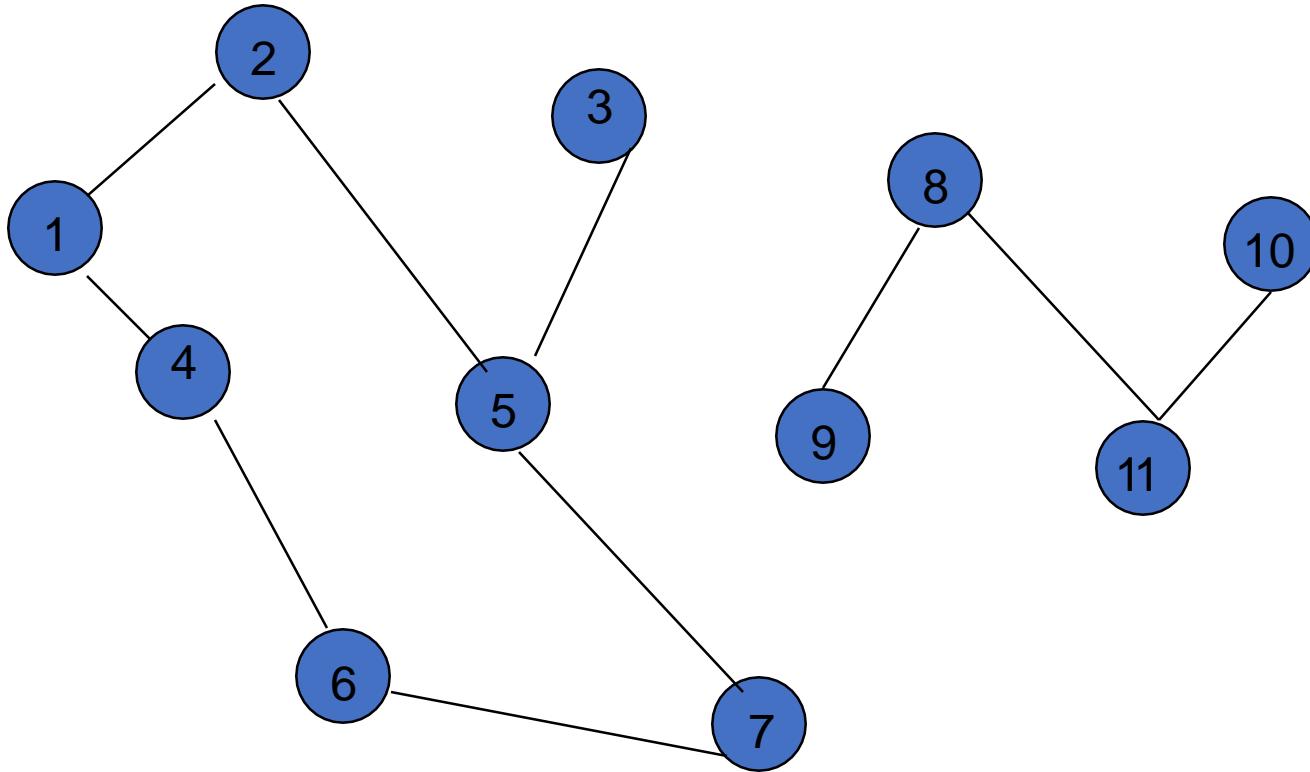
Loop

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



Implementasi Graph

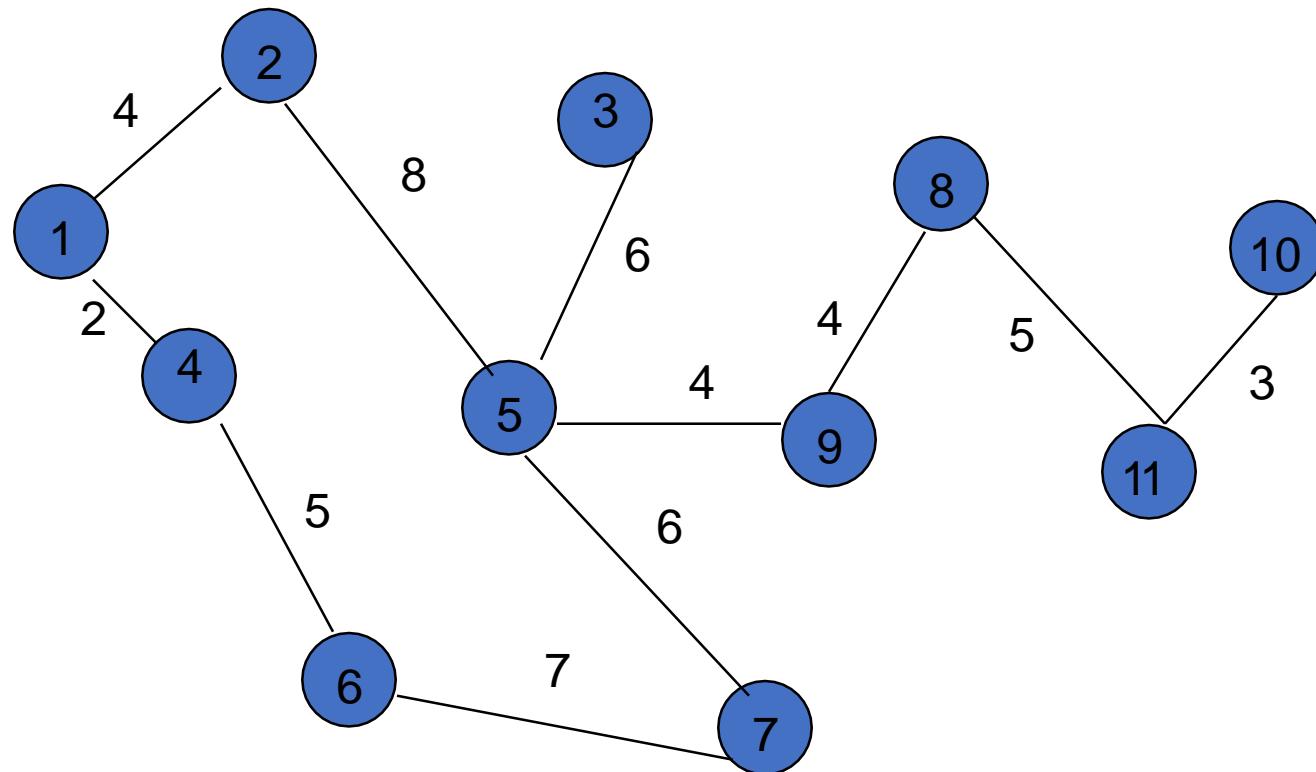
Aplikasi Jaringan Komunikasi



- Node/Vertex = kota
- Lines/Edge = communication link.

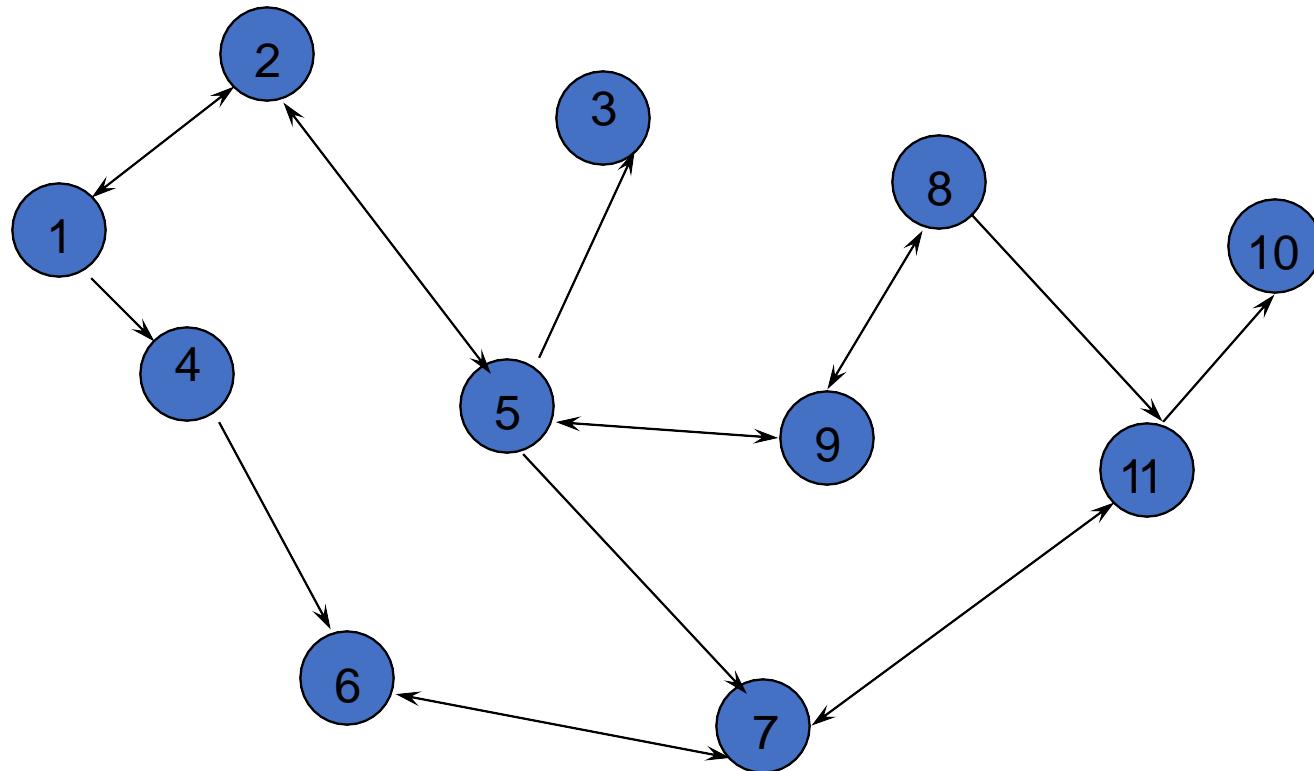
Peta Penelusuran Kota

(Berdasarkan Jarak/Waktu)



- Node/Vertex = Kota
- Lines/edge = jalur
- Weight = jarak/waktu

Peta kota



- Some streets are one way.

Graph Property

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

Jumlah Lines/Edge—Undirected Graph

- Setiap lines/edge memiliki bentuk (u,v) dimana $u \neq v$.
- Jumlah lines/edge undirected graph (lebih kecil sama dengan) $\leq n(n-1)/2$.
- Selama (u,v) **sama dengan** (v,u) Jumlah lines/edge pada complete undi-graph adalah $n(n-1)/2$.

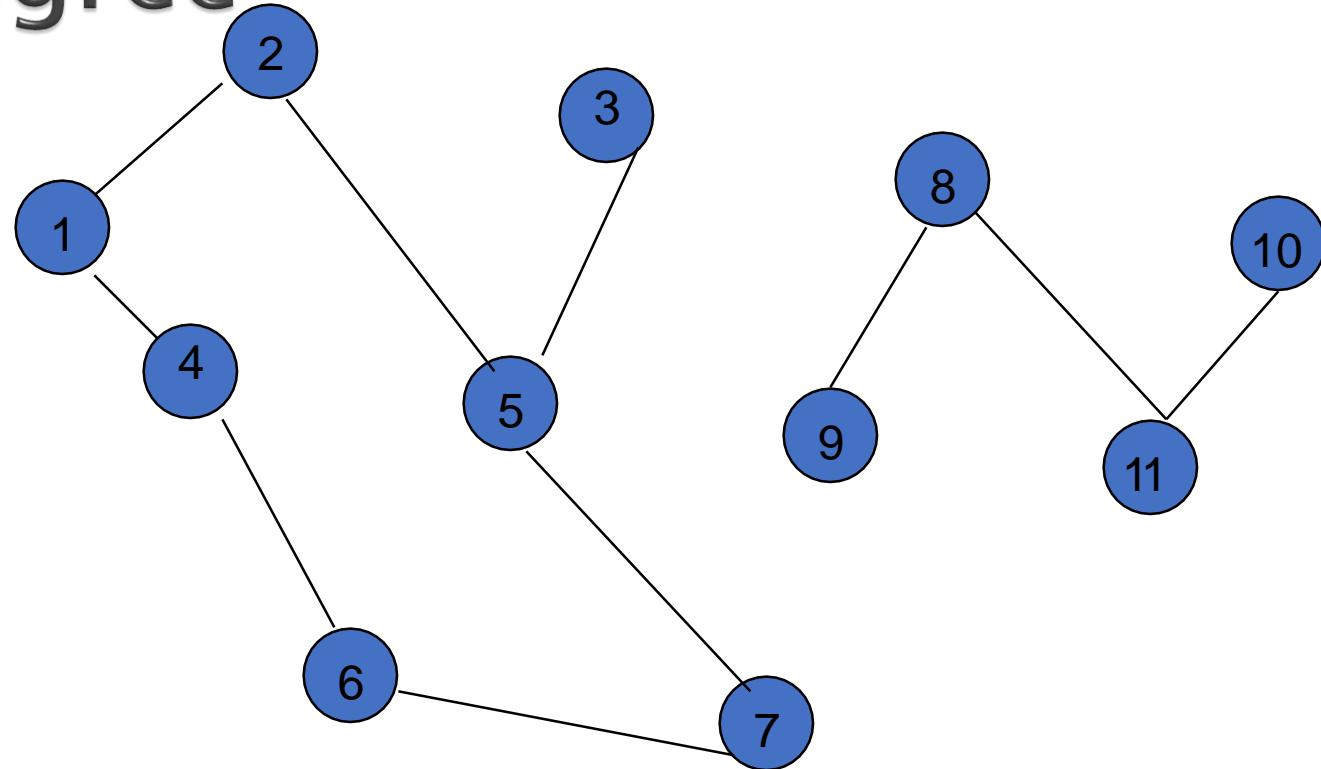
Jumlah Lines/Edges—Directed Graph

- Setiap lines/edge memiliki bentuk (u,v) dimana $u \neq v$.
- Selama $\text{edge}(u,v)$ **tidak sama** dengan (u,v) dan (v,u) maka jumlah edge untuk complete directed graph adalah $n(n-1)$.
- Jumlah edge untuk directed graph (lebih kecil sama dengan) $\leq n(n-1)$.

Vertex degree

- Degree : Jumlah edge/lines yang dimiliki node/vertex.

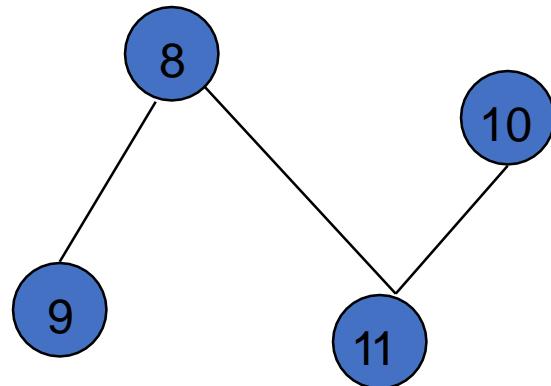
Contoh Vertex Degree



$\text{degree}(2) = 2$, $\text{degree}(5) = 3$, $\text{degree}(3) = 1$
 $\text{degree}(9) = ???$ $\text{degree}(11) = ???$

Jumlah Vertex Degrees

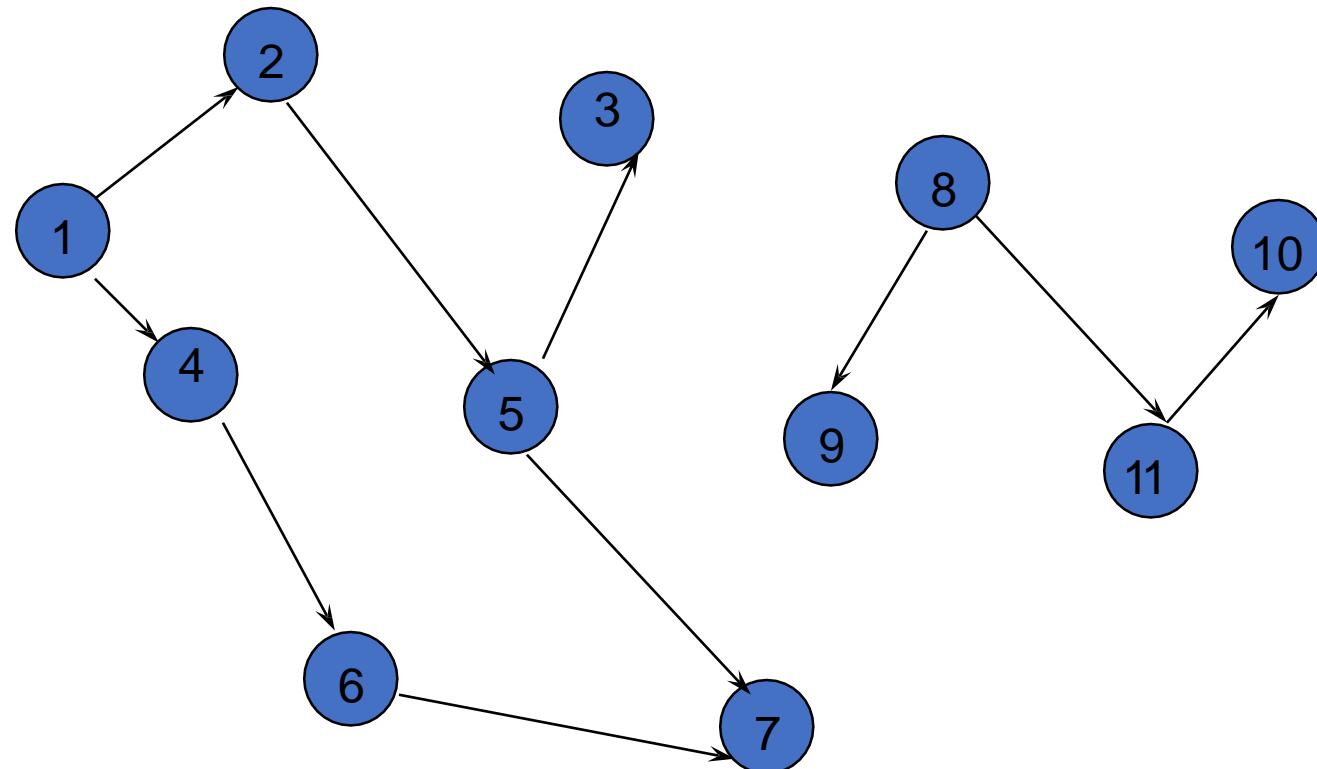
Jumlah Vertex degrees = **2e** (**e** adalah jumlah edge)



Jumlah Vertex Degree = 6

In-Degree

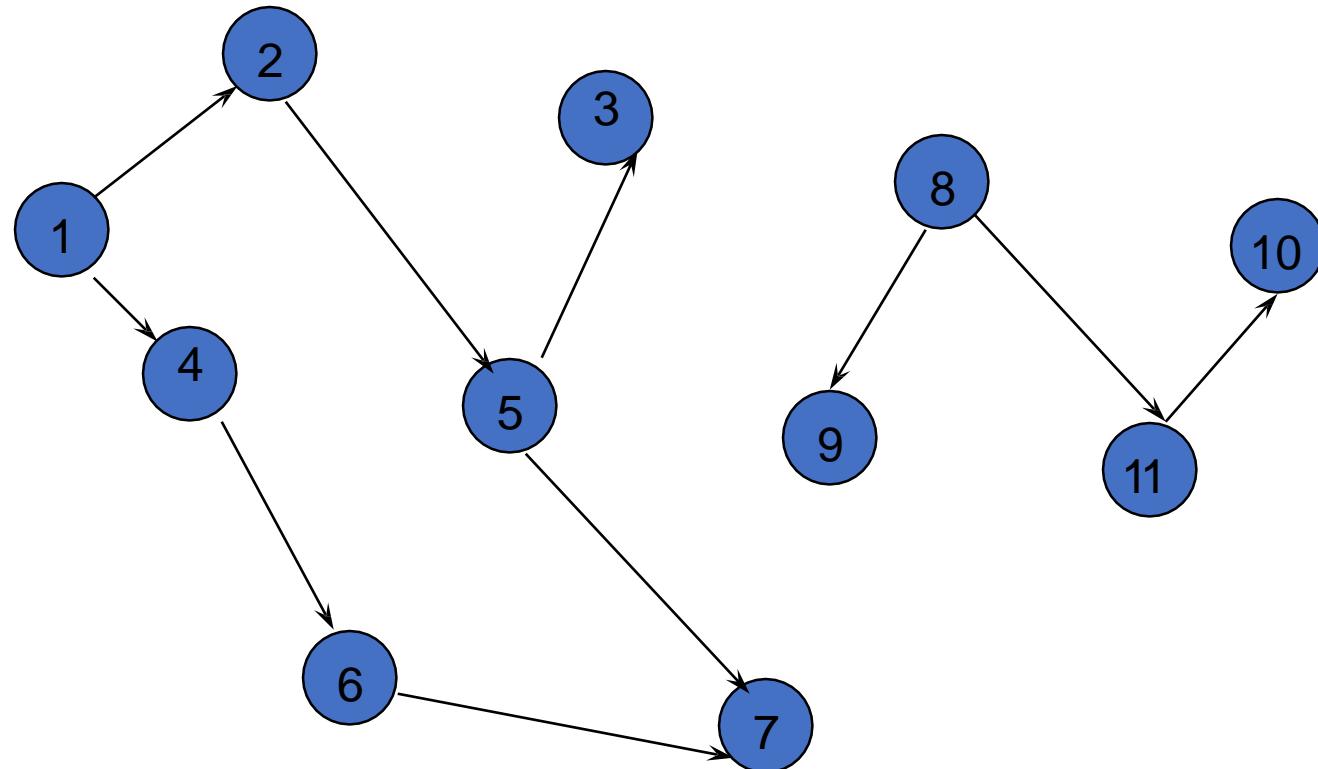
Jumlah edge yang mengarah ke Node/Vertex.



$$\text{indegree}(2) = 1, \text{indegree}(8) = 0, \quad \text{indegree}(7) = ???$$

Out-Degree

Jumlah edge yang keluar dari Node/Vertex.



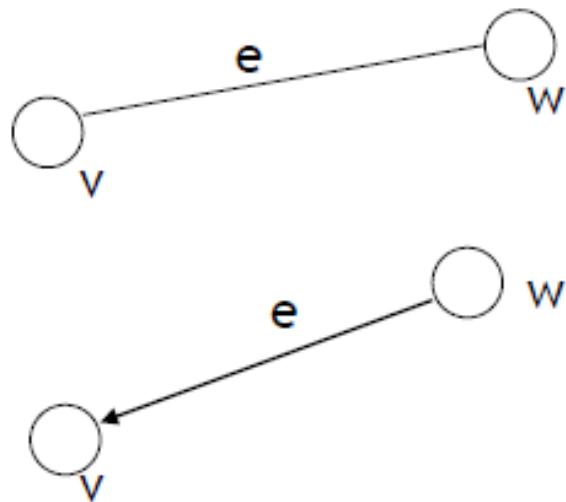
$\text{outdegree}(2) = 1$, $\text{outdegree}(8) = 2$, $\text{outdegree}(7) = ???$

Representasi Graph

- Adjacency Matrix
 - dapat direpresentasikan dengan matriks (array 2 Dimensi).
- Adjacency Lists
 - dapat direpresentasikan dengan array (bukan berupa matriks) maupun linked list.

Adjacency

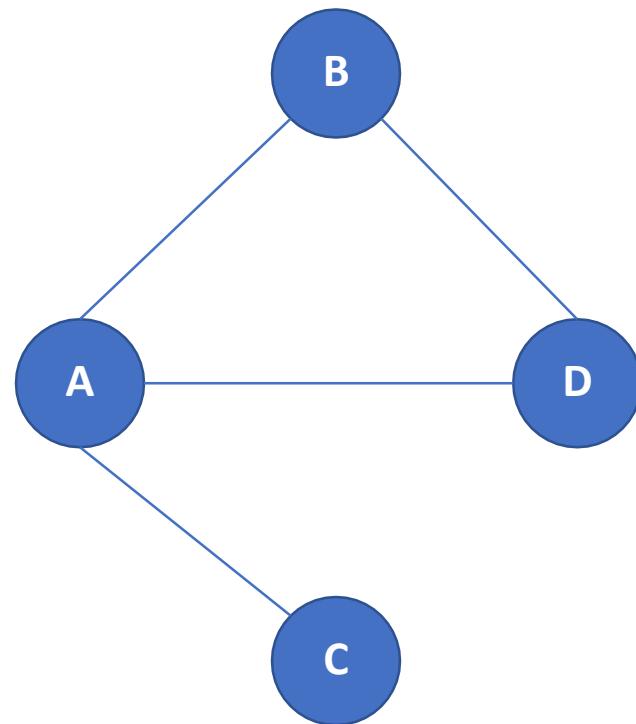
- Pada graph tidak berarah, 2 buah vertex disebut adjacent bila ada edge yang menghubungkan kedua vertex tersebut. Vertex v dan w disebut adjacent.
- Pada graph berarah, vertex v disebut adjacent dengan vertex w bila ada edge dari w ke v.



Adjacency Matrix

- Matriks digunakan untuk himpunan adjacency setiap verteks.
- Baris berisi vertex asal adjacency, sedangkan kolom berisi vertex tujuan adjacency.
- Bila sisi graph tidak mempunyai bobot, maka $[x, y]$ adjacency disimbolkan dengan 1 dan 0.
- Bila sisi graph mempunyai bobot, maka $[x, y]$ adjacency disimbolkan dengan bobot sisi tersebut.

Representasi Graph



Connected Graph

Penerapan Adjacency Matriks

A

B

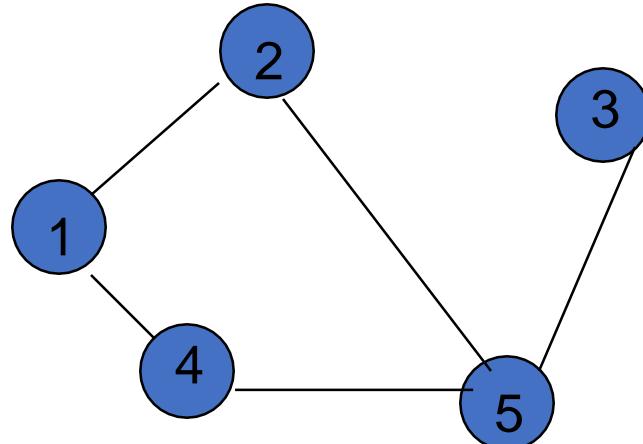
C

D

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

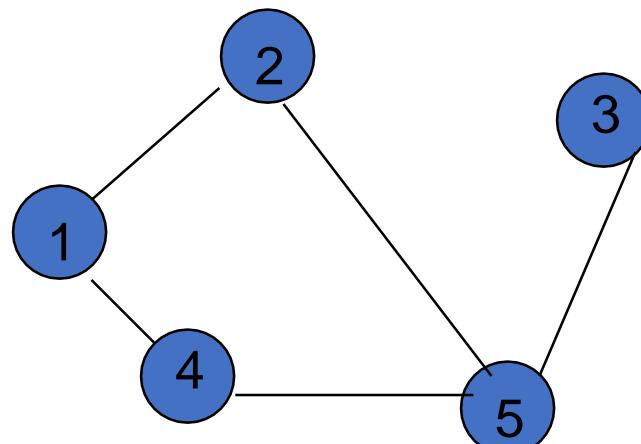
Adjacency Matrix

- Representasi Matrik ordo $n \times n$, dimana $n = \text{vertex/node}$
- $A(i,j) = 1$, jika antara node i dan node j terdapat edge/terhubung.



		1	2	3	4	5
		j				
i	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	

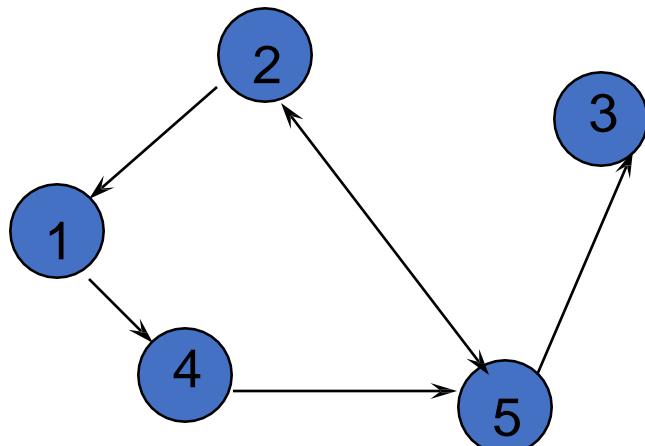
Adjacency Matrix (Undi-graph)



	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

- Diagonal entries are zero.
- Adjacency matrix of an undirected graph is symmetric.
 - $A(i,j) = A(j,i)$ for all i and j .

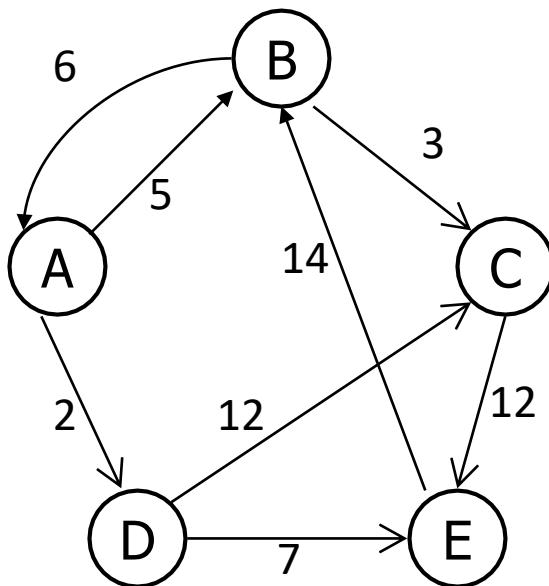
Adjacency Matrix (Digraph)



	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

- Adjacency matrix of a digraph need not be symmetric.

Graph berarah dan berbobot



	A	B	C	D	E
A	0	5	0	2	0
B	6	0	3	0	0
C	0	0	0	0	9
D	0	0	12	0	7
E	0	14	0	0	0

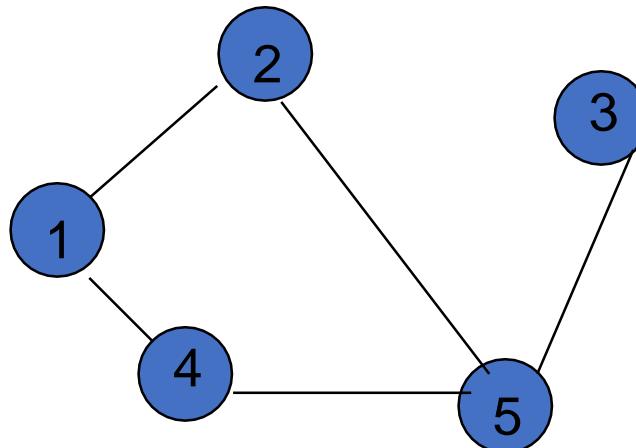
Perhatikan pemilihan nilai 0.

Adjacency List

- Direpresentasikan dengan linked list atau array.
 - Linked Adjacency Lists
 - Array Adjacency Lists

Array Adjacency Lists

- Adjacency list for vertex i is a linear list of vertices adjacent from vertex i .
- An array of n adjacency lists.



$aList[1] = (2,4)$

$aList[2] = (1,5)$

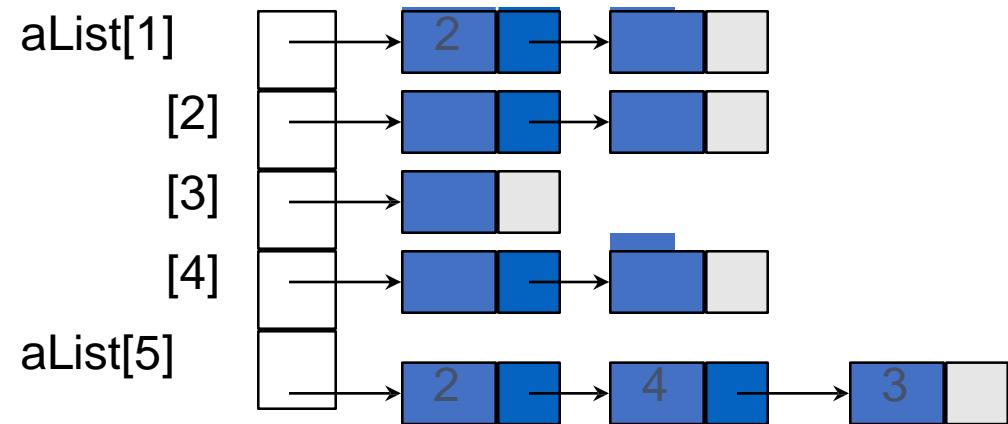
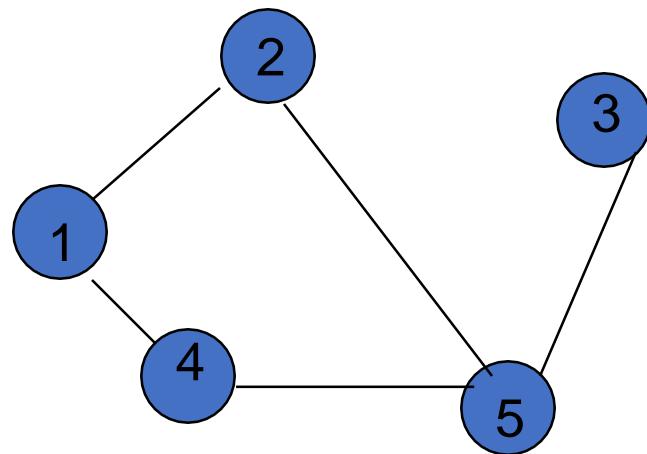
$aList[3] = (5)$

$aList[4] = (5,1)$

$aList[5] = (2,4,3)$

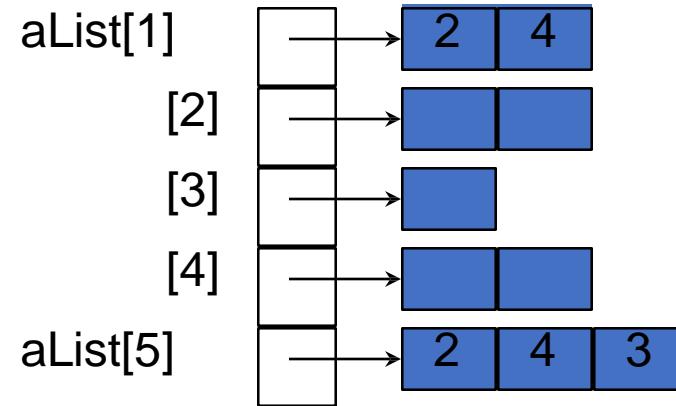
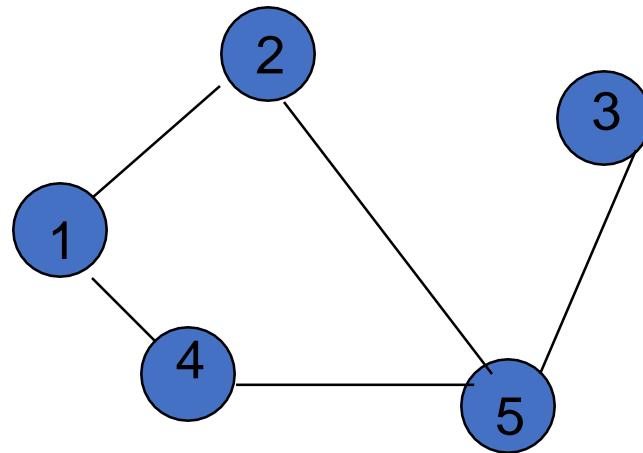
Adjacency Lists (Linked List)

- Each adjacency list is a chain.



Adjacency Lists (Array)

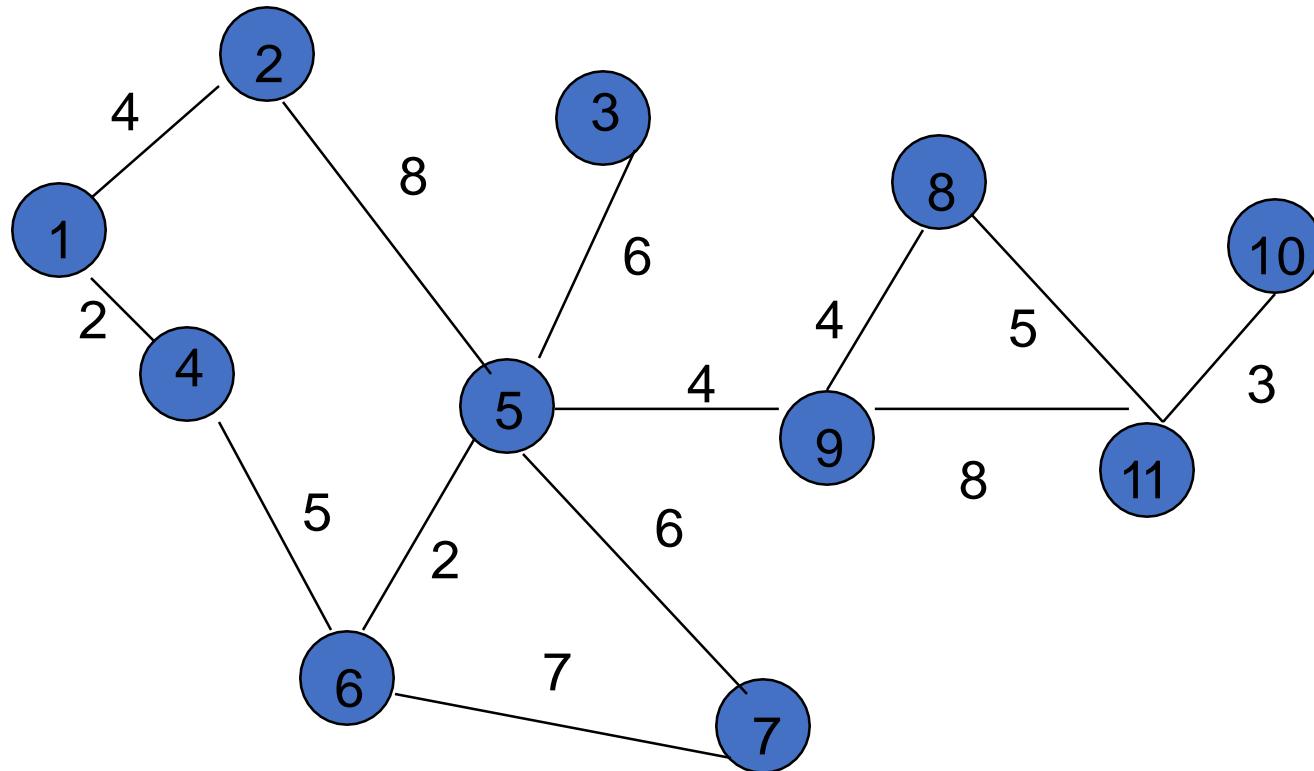
- Each adjacency list is an array list.



Minimum Spanning Tree

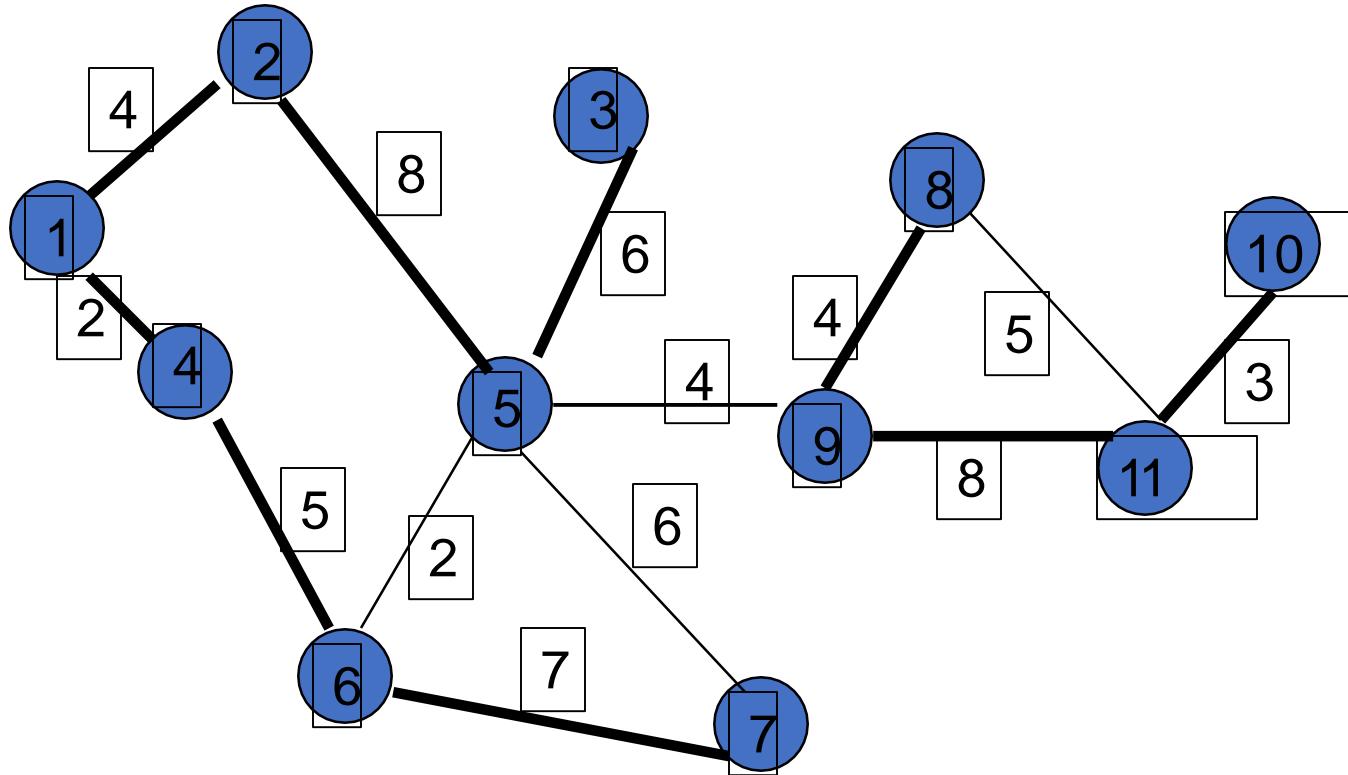
- Subgraf yang mencakup semua simpul dari graph asli.
- Subgraf adalah sebuah tree.
 - Jika graph asli memiliki n vertex, spanning tree memiliki n vertex dan edge $n-1$.
- Pencarian biaya yang minimum dari suatu graph sehingga membentuk pohon .
- Syarat Graph yang dapat dicari minimum spanning tree :
 - Graph harus terhubung
 - Ruasnya punya bobot
 - Graph tidak berarah
- Algoritma yang dipakai untuk menentukan minimum spanning tree :
 - Algoritma Kruskal
 - Algoritma Solin

Minimum Cost Spanning Tree



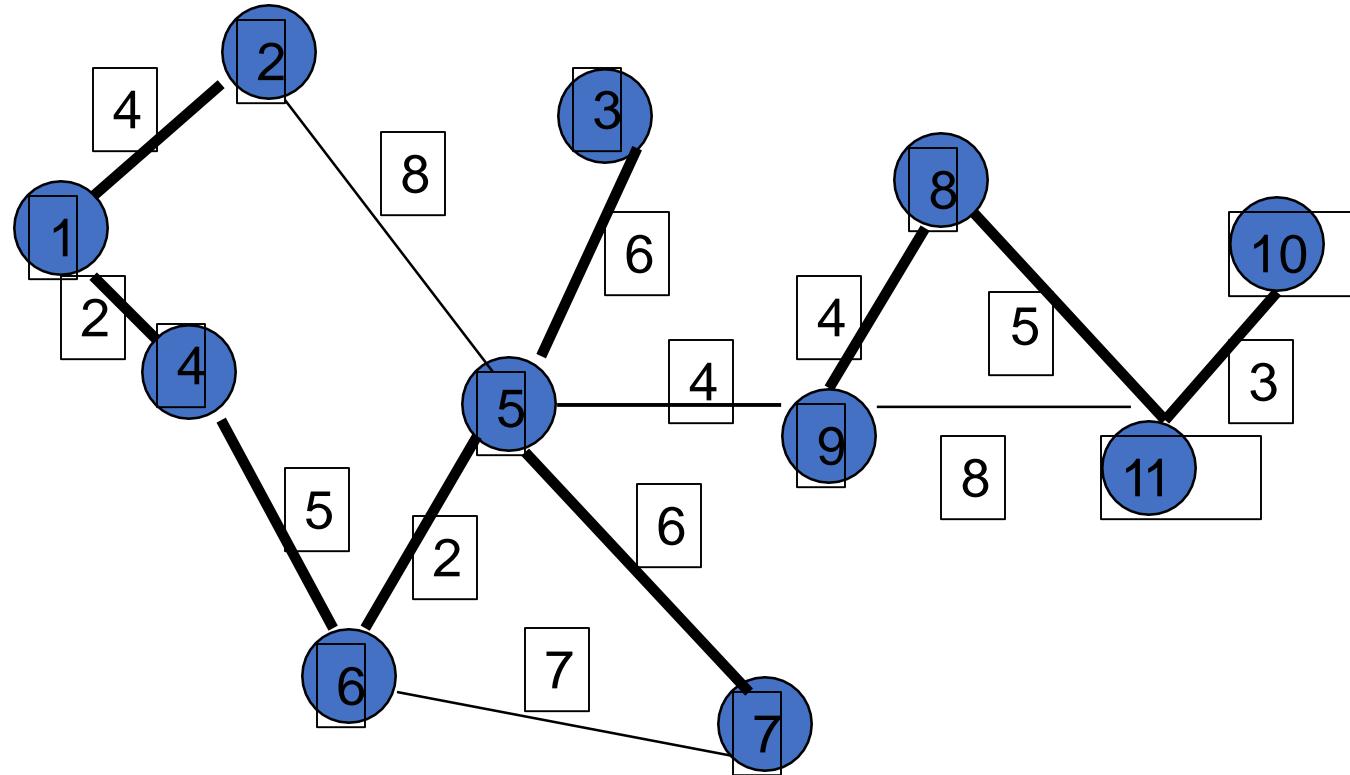
- Tree cost is sum of edge weights/costs.

A Spanning Tree



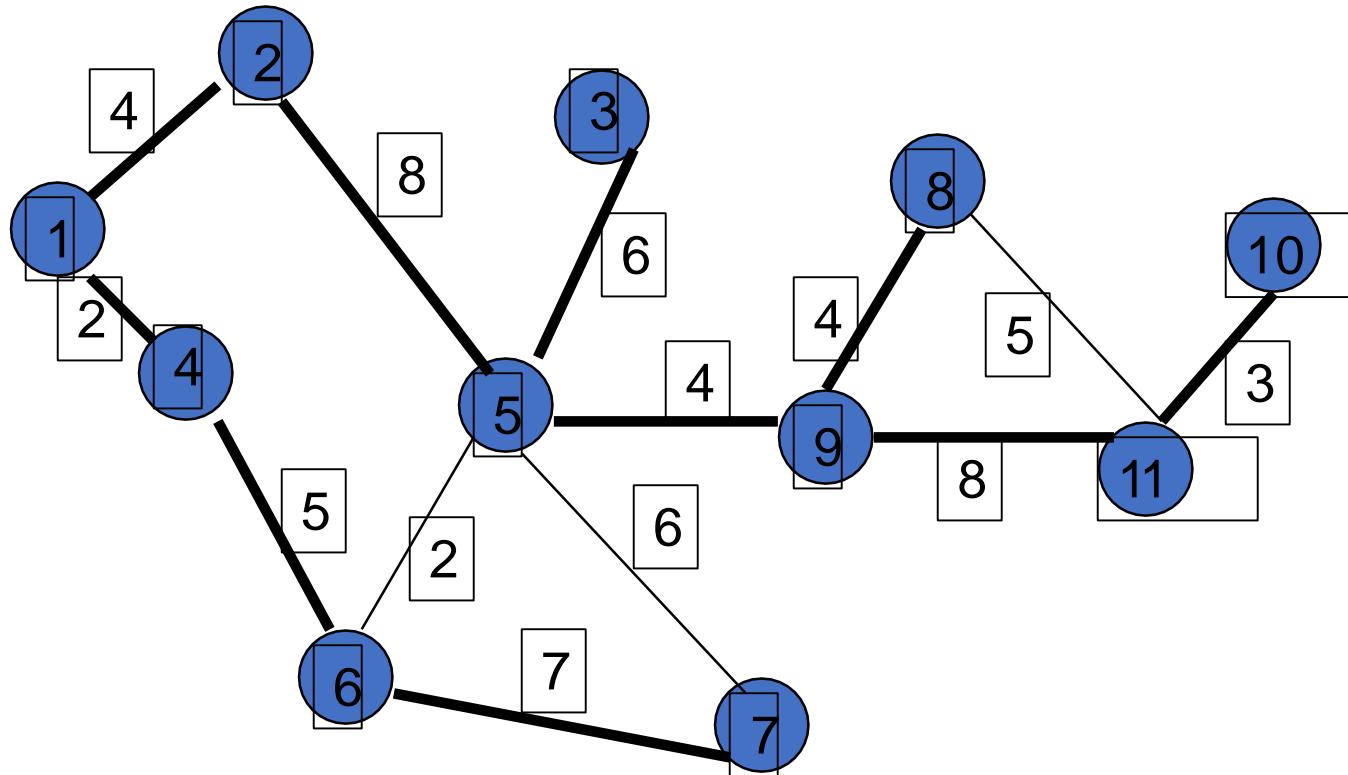
Spanning tree cost = 51.

Minimum Cost Spanning Tree



Spanning tree cost = 41.

A Wireless Broadcast Tree



Weights = needed power.

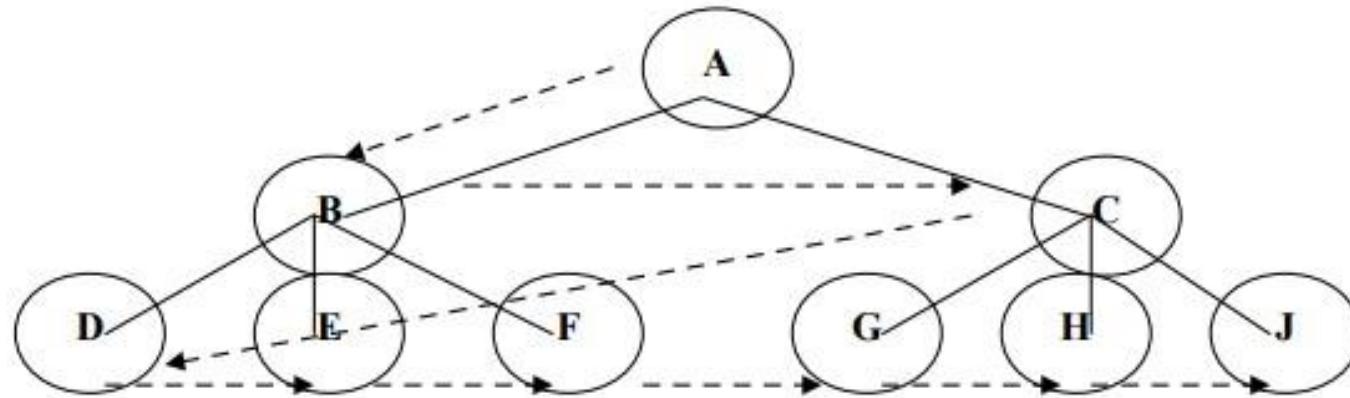
$$\text{Cost} = 4 + 8 + 5 + 6 + 7 + 8 + 3 = 41.$$

Metode Penelusuran Graph

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

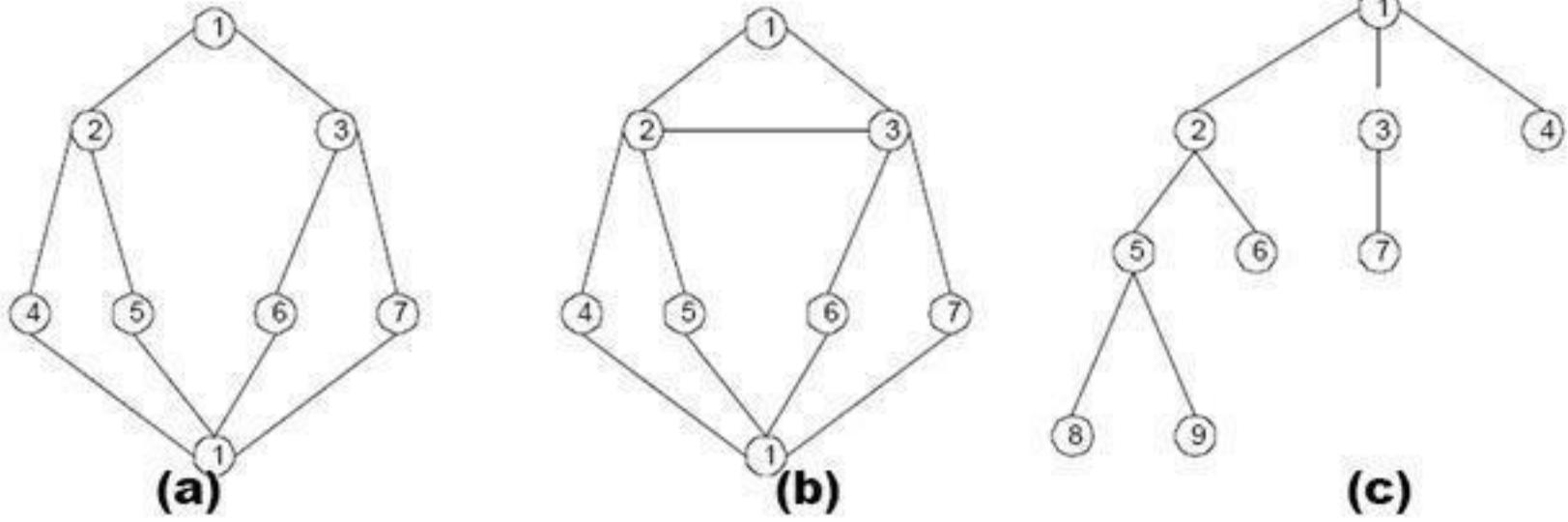
Algoritma BFS

- 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 kevertex level 1, dan seterusnya.
- Penelusuran BFS berakhir ketika setiap vertex selesai ditemui.



Langkah-langkah algoritma BFS

1. Masukkan simpul ujung (akar) kedalam antrian
2. Ambil simpul dari awal antrian, lalu cek apakah simpul merupakan solusi
3. Jika simpul merupakan solusi, pencarian selesai dan hasil dikembalikan.
4. Jika simpul bukan solusi, masukkan seluruh simpul yang bertetangga dengan simpul tersebut (simpul anak) kedalam antrian
5. Jika antrian kosong dan setiap simpul sudah dicek, pencarian selesai
‣ dan mengembalikan hasil solusi tidak ditemukan
6. Ulangi pencarian dari langkah kedua

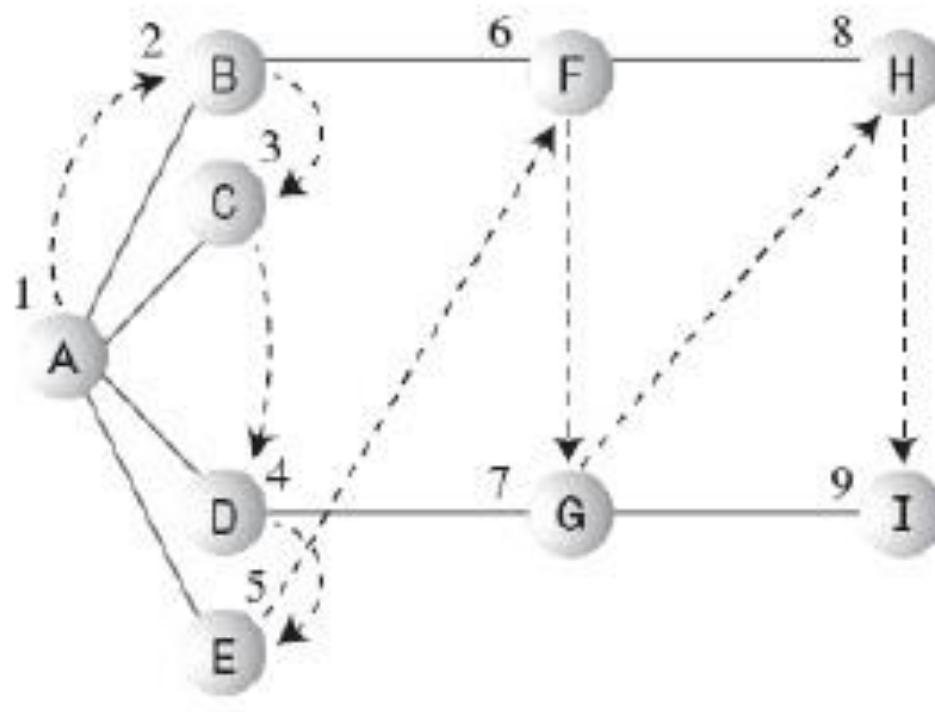


Gambar (a) $\text{BFS}(1): 1, 2, 3, 4, 5, 6, 7, 1$.

Gambar (b) $\text{BFS}(1): 1, 2, 3, 4, 5, 6, 7, 1$

Gambar (c) $\text{BFS}(1): 1, 2, 3, 4, 5, 6, 7, 8, 9$

Breadth First Search (BFS)



Urutan verteks hasil penelusuran : ABCDEFGHI

Penerapan BFS

```
procedure BFS(input v:integer)
{ Traversai graf dengan algoritma pencarian BFS.

    Masukan: v adalah simpul awal kunjungan
    Keluaran: semua simpul yang dikunjungi dicetak ke layar
}
Deklarasi
    w : integer
    q : antrian;

procedure BuatAntrian(input/output q : antrian)
{ membuat antrian kosong; kepala(q) diisi 0 }

procedure MasukAntrian(input/output q:antrian, input v:integer)
{ memasukkan v ke dalam antrian q pada posisi belakang }

procedure HapusAntrian(input/output q:antrian, output v:integer)
{ menghapus v dari kepala antrian q }

function AntrianKosong(input q:antrian) → boolean
{ true jika antrian q kosong, false jika sebaliknya }

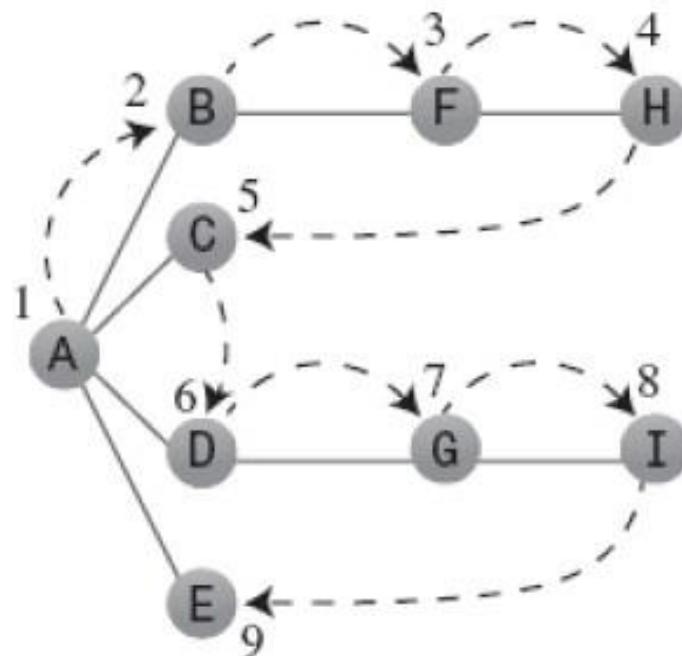
Algoritma:
    BuatAntrian(q)          { buat antrian kosong }

    write(v)                { cetak simpul awal yang dikunjungi }
    dikunjungi[v]←true   { simpul v telah dikunjungi, tandai dengan
                           true}
    MasukAntrian(q,v)       { masukkan simpul awal kunjungan ke dalam
                           antrian}

{ kunjungi semua simpul graf selama antrian belum kosong }
while not AntrianKosong(q) do
    HapusAntrian(q,v)     { simpul v telah dikunjungi, hapus dari
                           antrian }
    for tiap simpul w yang bertetangga dengan simpul v do
        if not dikunjungi[w] then
            write(w)         {cetak simpul yang dikunjungi}
            MasukAntrian(q,w)
            dikunjungi[w]←true
        endif
    endfor
endwhile
{ AntrianKosong(q) }
```

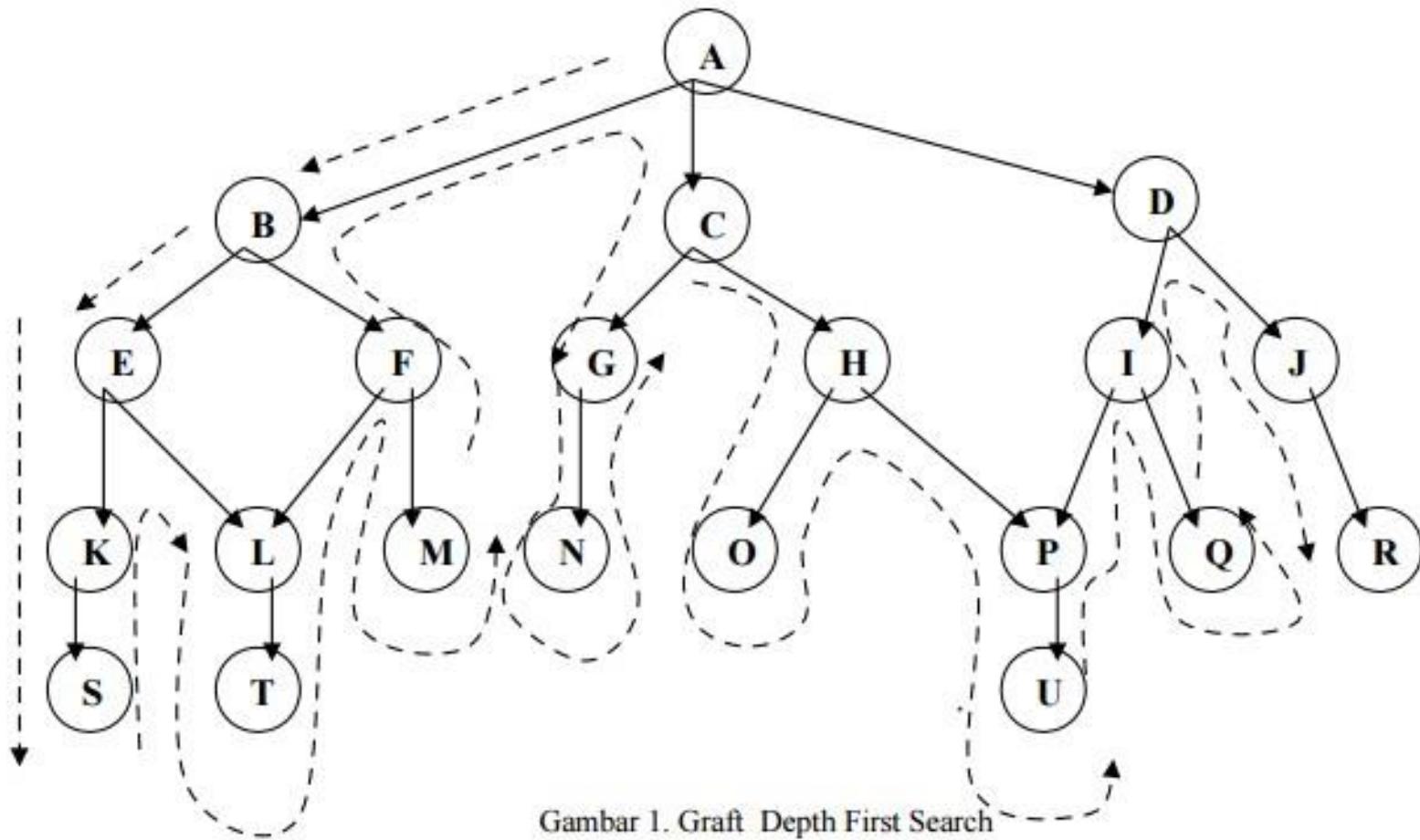
Depth First Search (DFS)

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



Urutan verteks hasil penelusuran : ABFHCDGIE

Depth First Search (DFS)



Algoritma DFS

- Traversal dimulai dari simpul v.
- Algoritma :
 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.



Terima
kasih