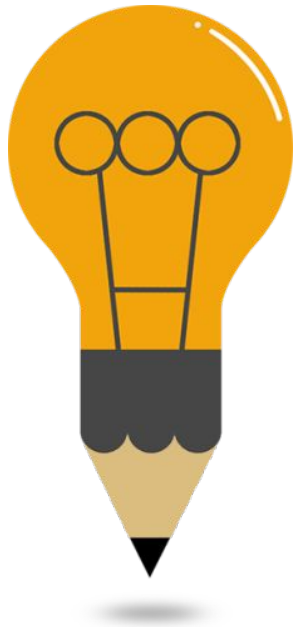
STRUKTUR DATA

Pertemuan 13

Agenda Pertemuan



1. Graph

Graph

- **Graph** adalah kumpulan node (simpul) di dalam bidang dua dimensi yang dihubungkan dengan sekumpulan edge (garis)
- Graph dapat digunakan untuk merepresentasikan objek-objek diskrit (direpresentasikan sebagai node) dan hubungan antara objek-objek tersebut (direpresentasikan sebagai edge)
- Secara matematis dinyatakan sebagai :

$$G = (V, E)$$

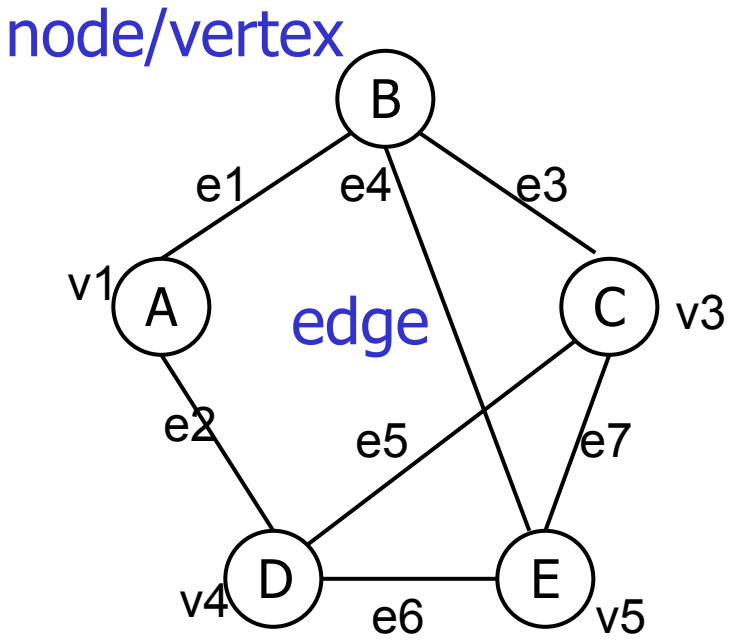
Dimana

G = Graph

V = Vertex, atau Node, atau Simpul, atau Titik

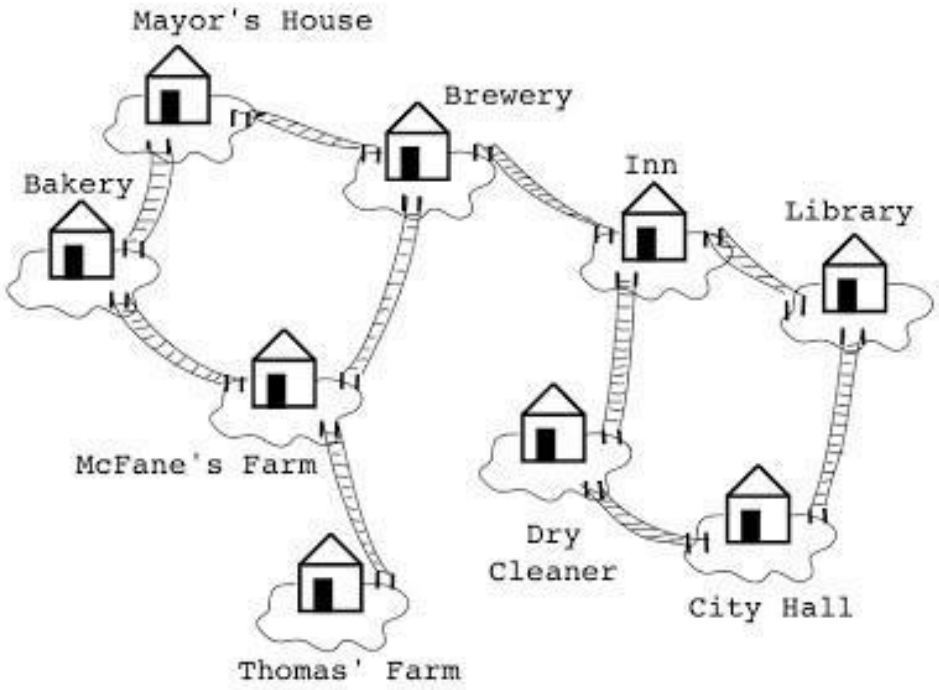
E = Edge, atau Busur, atau Garis

Graph



- V terdiri dari v1, v2, …, v5
- E terdiri dari e1, e2, … , e7

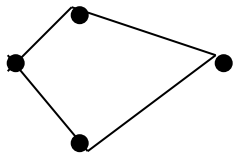
Contoh Implementasi Struktur Data Graph



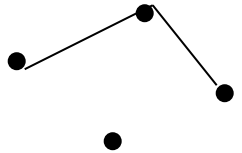
Graph

- Sebuah graph mungkin hanya terdiri dari satu node

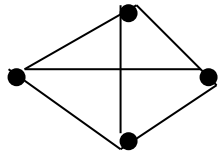
- Sebuah graph belum tentu semua nodenya terhubung dengan edge



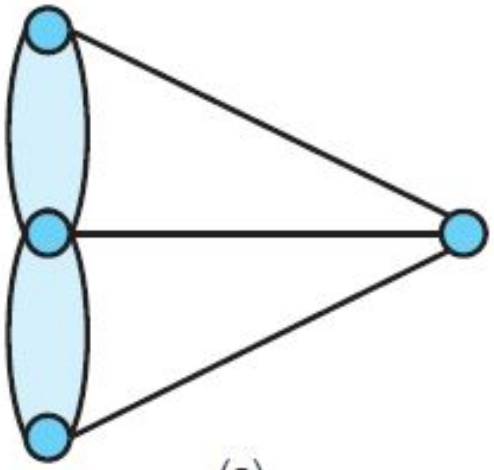
- Sebuah graph mungkin mempunyai node yang tak terhubung dengan node yang lain



- Sebuah graph mungkin semua nodenya saling berhubungan



Graph



(a)

(b)

Bagaimana dengan ini?

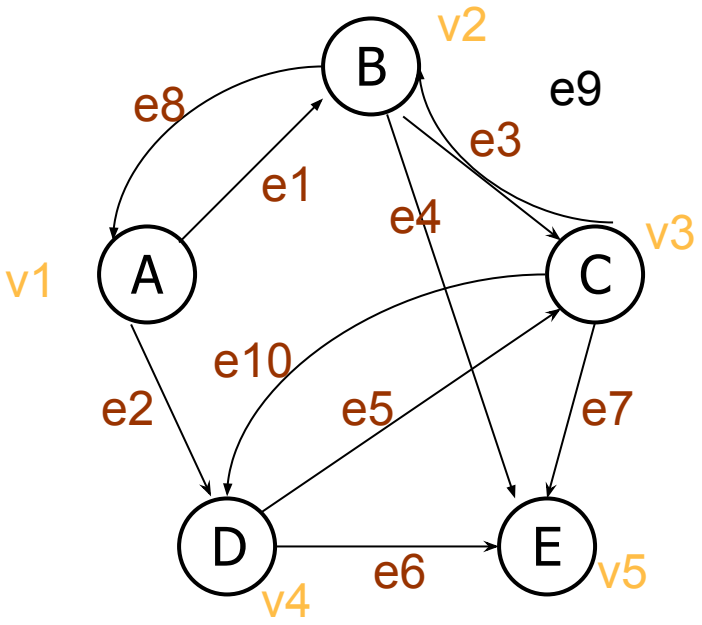
(a) Multigraph / Multiple edges

(b) Loop ke diri sendiri

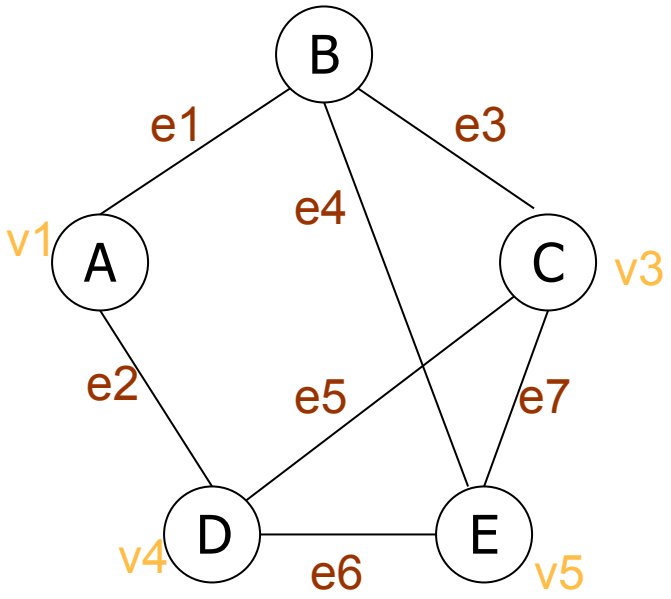
(a) dan (b) Bukan Graph

Jenis – Jenis Graph

- **Graph Berarah (Directed Graph)** dimana urutan node mempunyai arti. Misal node AB adalah e1 sedangkan busur BA adalah e8.
- **Graph Tak Berarah (Undirected Graph atau Non-directed Graph)** dimana urutan node dalam sebuah edge tidak dipentingkan. Misal edge e1 dapat disebut edge AB atau BA.



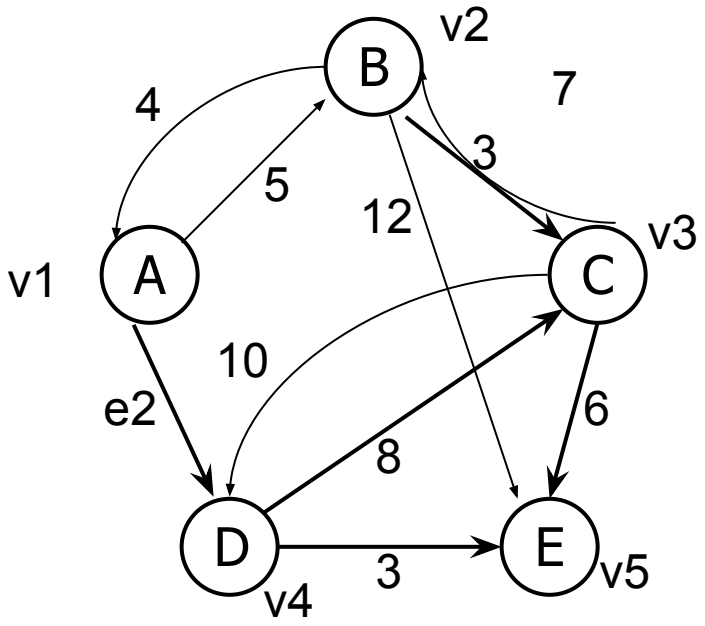
Directed graph



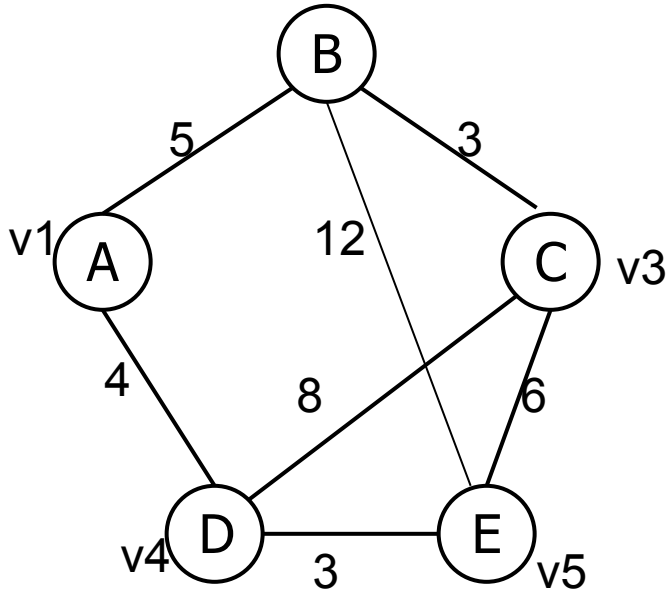
Undirected graph

Jenis – Jenis Graph

- **Graph Berbobot (Weighted Graph):**
  - Jika setiap edge mempunyai nilai yang menyatakan hubungan antara 2 buah node, maka edge tersebut dinyatakan memiliki bobot.
  - Bobot sebuah edge dapat menyatakan panjang sebuah jalan dari 2 buah titik, jumlah rata-rata kendaraan perhari yang melalui sebuah jalan, dll.



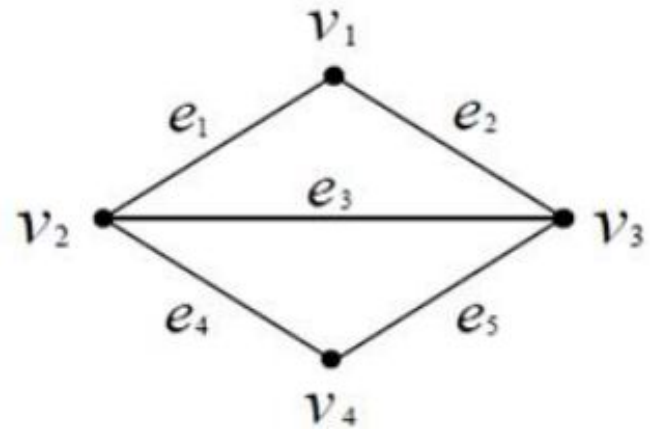
Directed graph



Undirected graph

Istilah dalam Graph

- **Incident**: Jika e merupakan edge dengan node-nodenya adalah v dan w yang ditulis e=(v,w), maka v dan w disebut “terletak” pada e, dan e disebut incident dengan v dan w
- **Adjacent**: Dua buah node dikatakan berdekatan (adjacent) jika dua buah node tersebut terhubung dengan sebuah edge

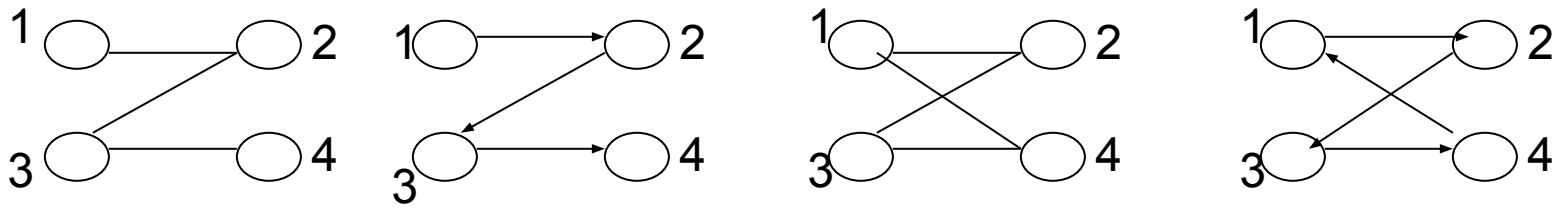


Edge **e3 = v2v3** ***incident*** dengan node **v2** dan node **v3**, tetapi edge **e3 = v2v3** tidak ***incident*** dengan node **v1** dan node **v4**. Node **v1** ***adjacent*** dengan node **v2** dan node **v3**, tetapi node **v1** tidak ***adjacent*** dengan node **v4**

- **Weight / Bobot**: Sebuah graph **G = (V, E)** disebut sebuah graph berbobot (weighted graph), apabila terdapat sebuah fungsi bobot bernilai real **W** pada himpunan **E**

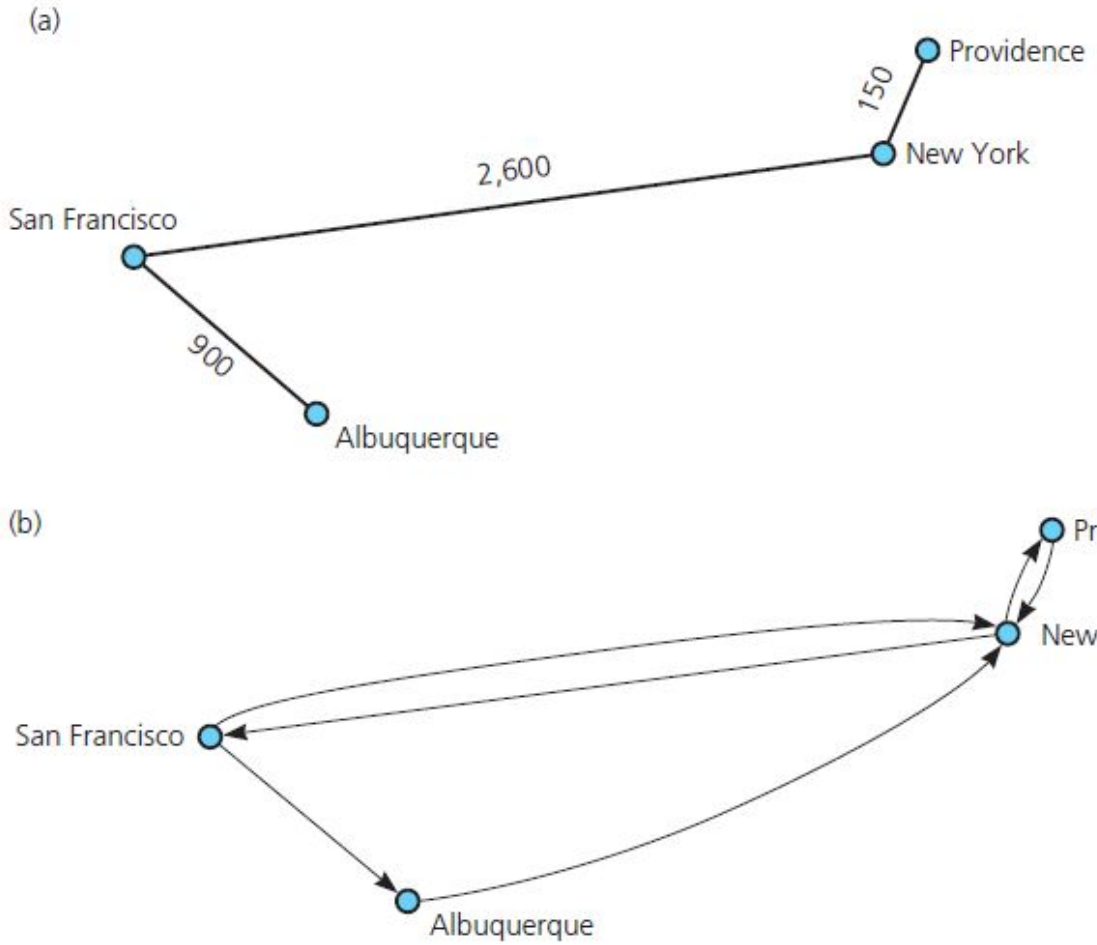
Istilah dalam Graph

- **Path / Jalur**: Serangkaian node-node yang berbeda, yang adjacent secara berturut-turut dari node satu ke node berikutnya



- **Degree** sebuah node: Jumlah edge yang incident dengan node tersebut
- **Indegree** sebuah node pada graph berarah: jumlah edge yang “masuk” atau menuju node tersebut
- **Outdegree** sebuah node pada graph berarah: jumlah busur yang “keluar” atau berasal dari node tersebut

Contoh



(a) Graph Berbobot (Weighted Graph)
(b) Graph Berarah (Directed Graph)

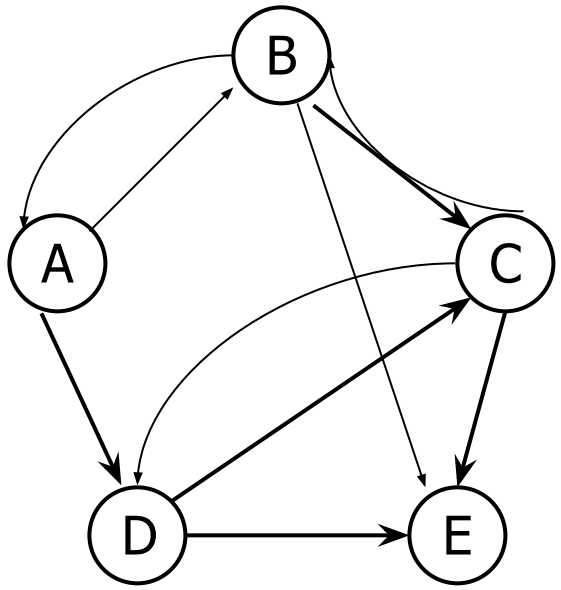
Representasi Graph

- Dalam pemrograman, agar data yang ada dalam graph dapat diolah, maka graph harus dinyatakan dalam suatu struktur data yang dapat mewakili graph tersebut
- Graph dapat direpresentasikan dalam:

1. **Adjacency Matrix**
   - dapat direpresentasikan dengan matriks (array 2 dimensi)
2. **Adjacency List**
   - dapat direpresentasikan dengan array dari linked list

Adjacency Matrix untuk Graph Berarah

Matriks ordo n x n, dimana n adalah jumlah node. Baris berisi node asal, Kolom berisi node tujuan



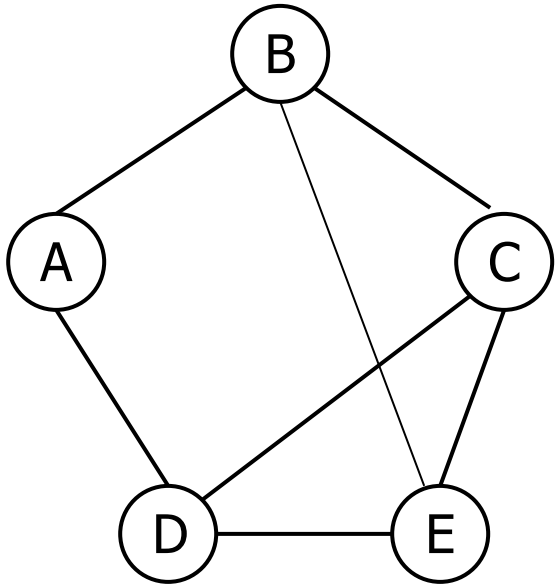
Graph

ke → / dari ↓		A	B	C	D	E
		0	1	2	3	4
A	0	0	1	0	1	0
B	1	1	0	1	0	1
C	2	0	1	0	1	1
D	3	0	0	1	0	1
E	4	0	0	0	0	0

Nilai matriks diisi dengan 1 atau 0. Nilai 1 jika ada edge, dan 0 jika tidak ada edge antar node.

Jika graph berbobot, maka nilai matriks diisi dengan bobot dari edge.

Adjacency Matrix untuk Graph Tak Berarah



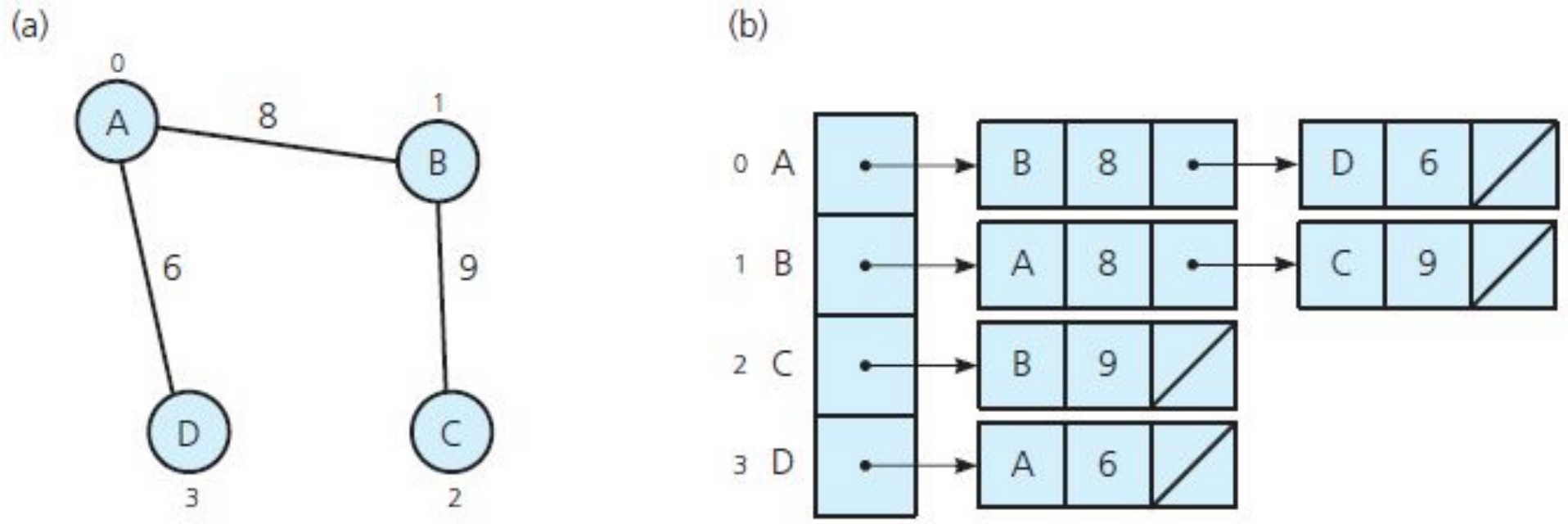
Graph

Urut abjad

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

Degree simpul D: 3

Adjacency List untuk Graph Tak Berarah

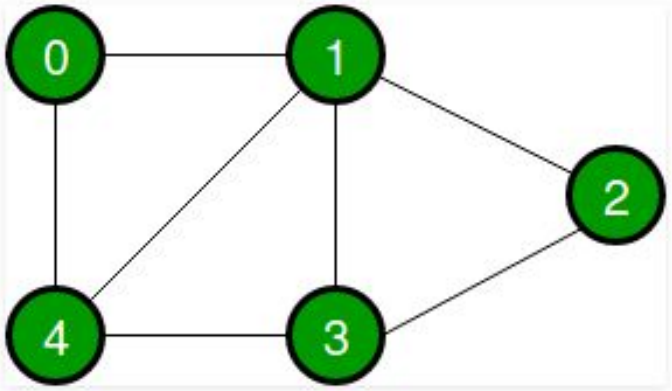


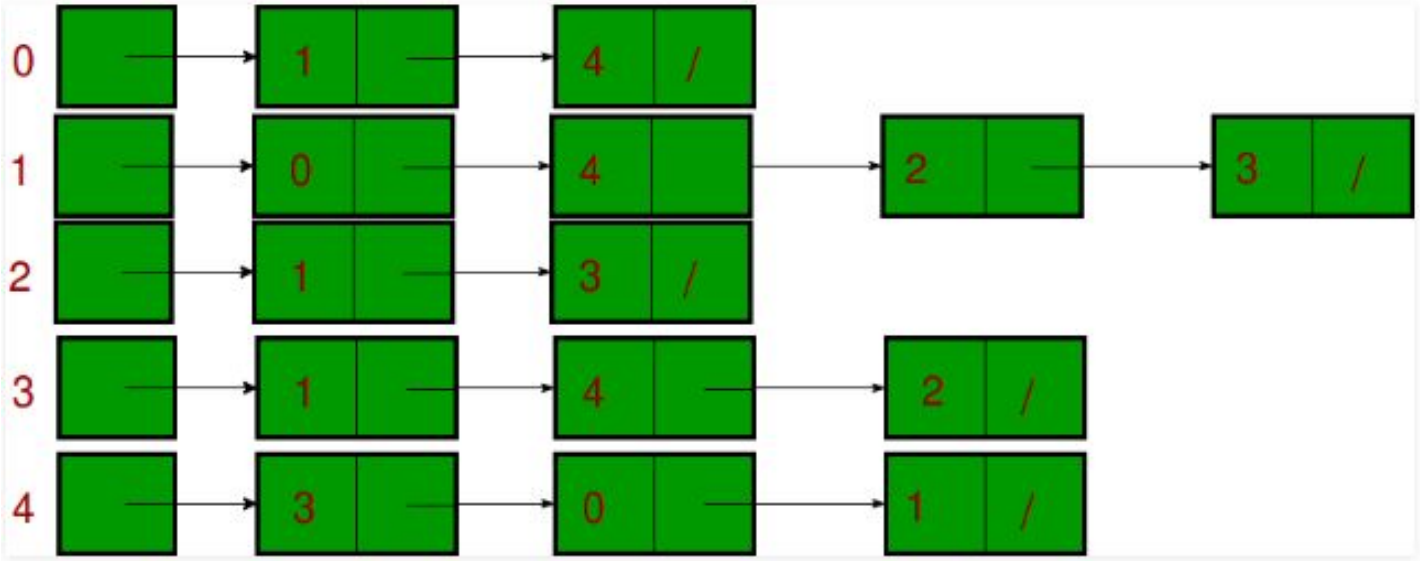
Direpresentasikan dengan array dari linked list

```
struct node
{
    int vertex;
    struct node* next;
};
```

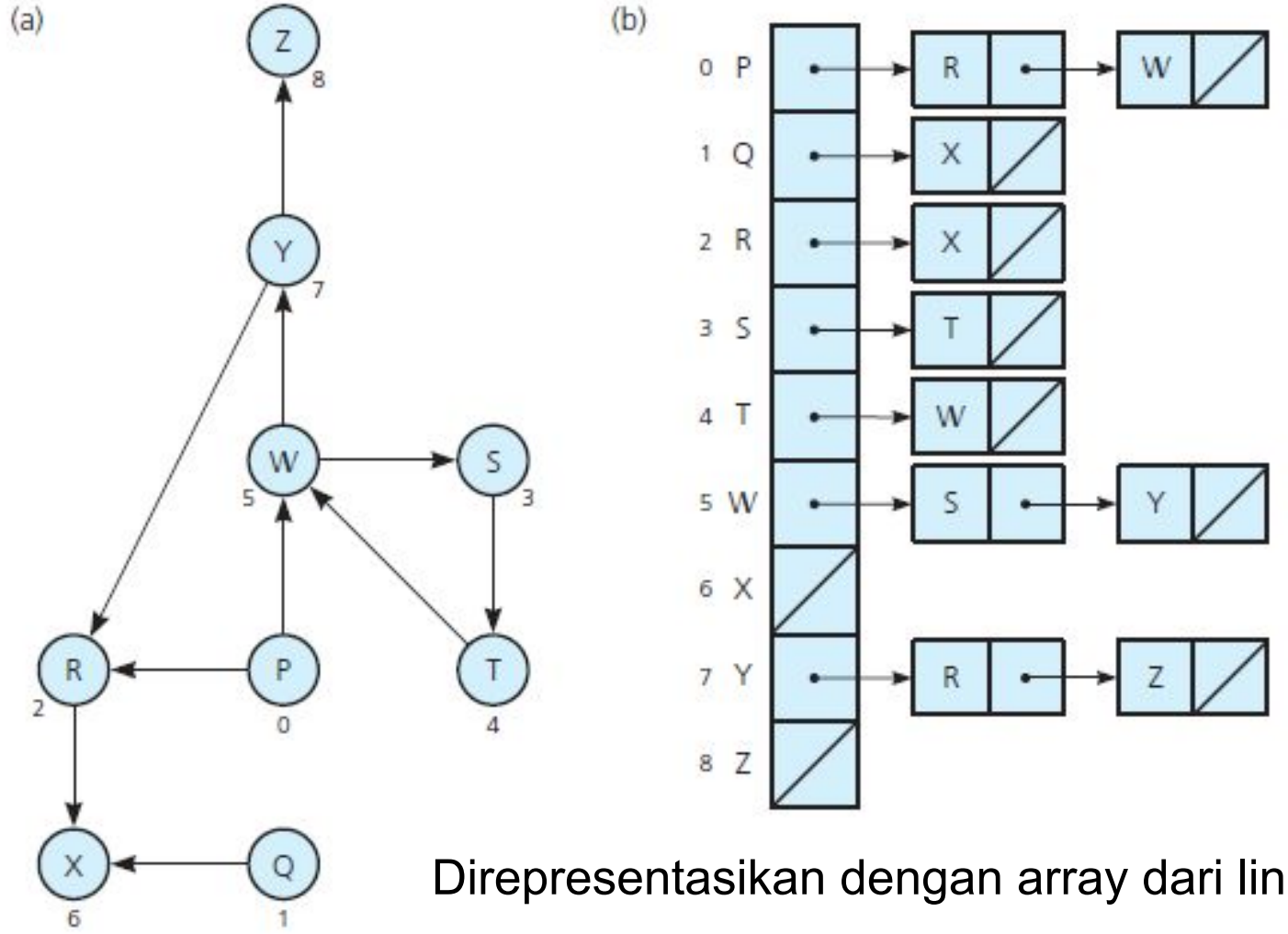
Adjacency List untuk Graph Tak Berarah

Contoh Lain:





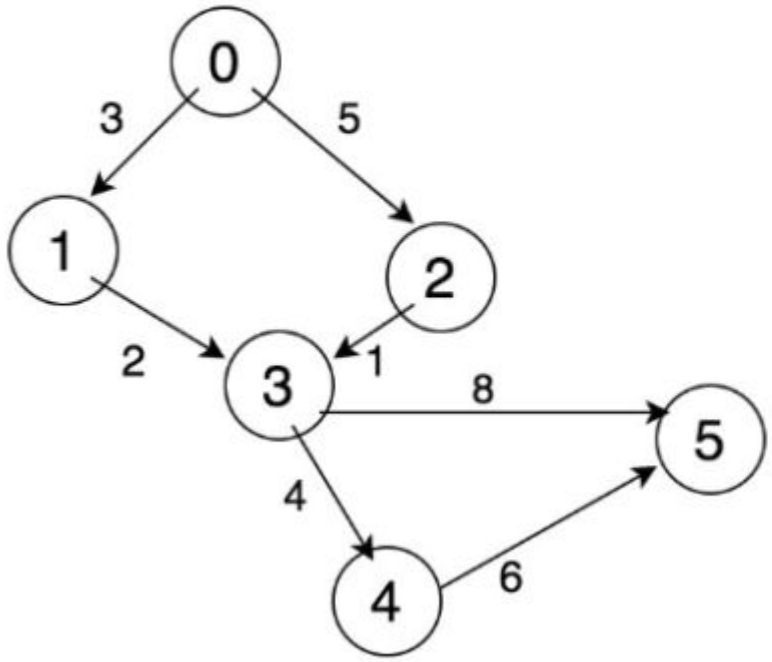
Adjacency List untuk Graph Berarah

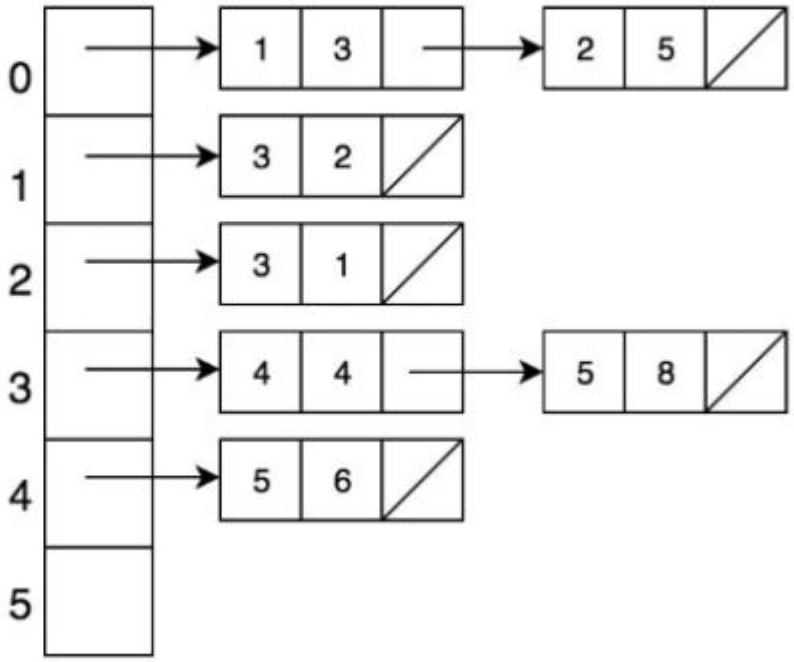


Direpresentasikan dengan array dari linked list (node-node yang keluar)

Adjacency List untuk Graph Berarah

Contoh Lain:

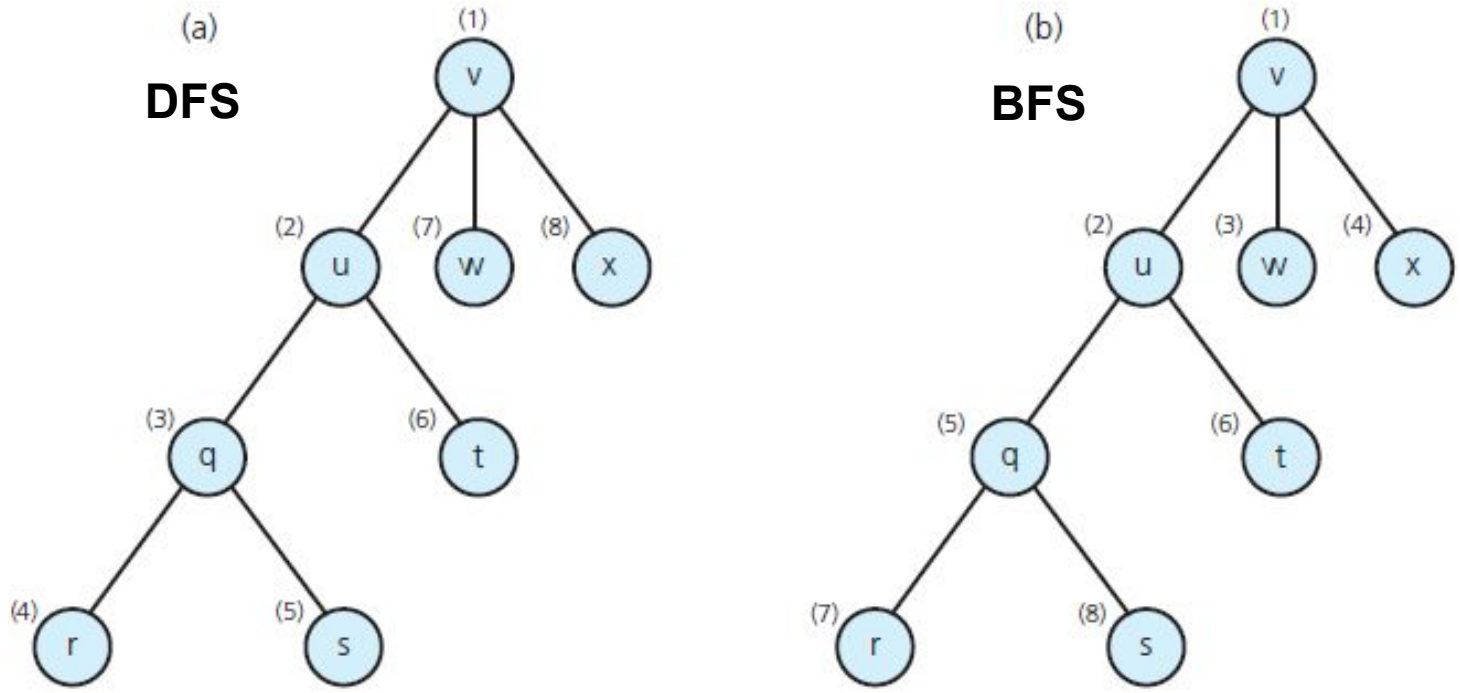




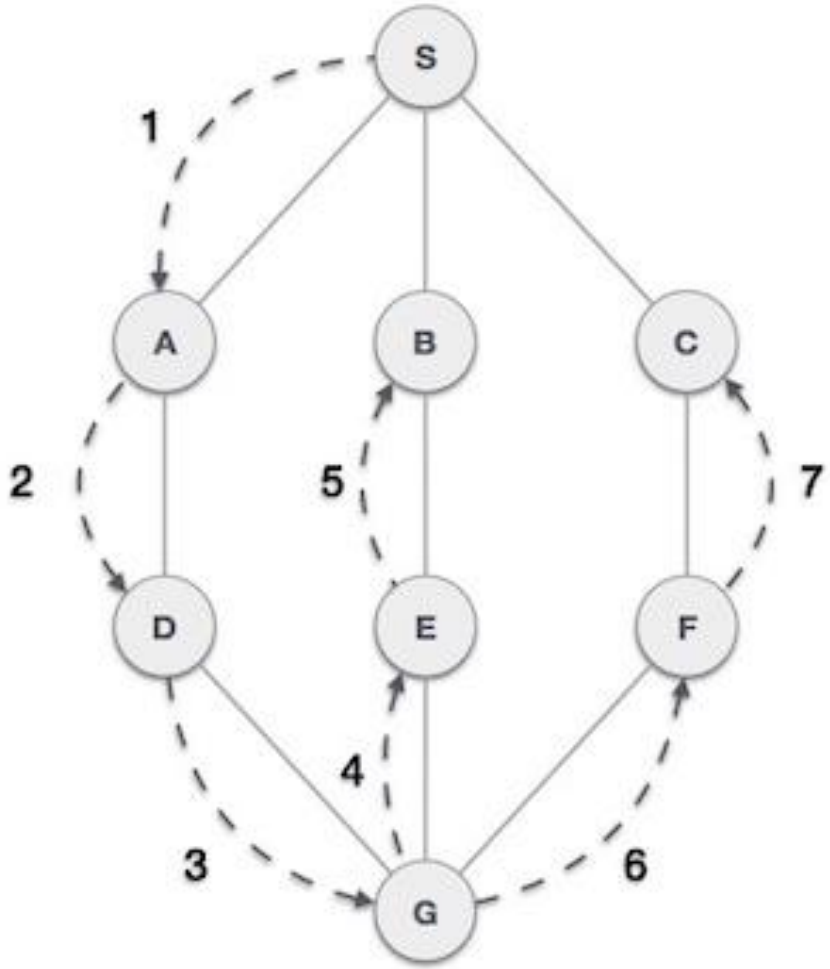
Graph Traversal

Mengunjungi tiap node dalam sebuah graph:

- **DFS (Depth First Search)**: Pencarian mendalam, algoritma akan memanfaatkan **stack**
- **BFS (Breadth First Search)**: Pencarian melebar, algoritma akan memanfaatkan **queue**



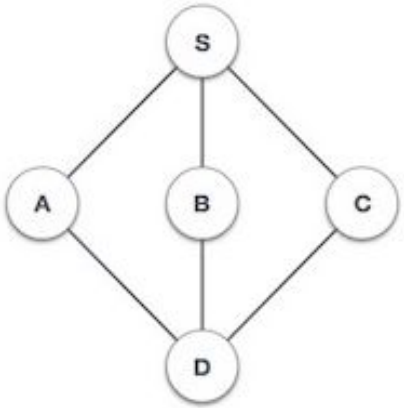
DFS (Depth First Search)



1. Kunjungi adjacent node yang belum dikunjungi. Tandai node tersebut dengan "**sudah dikunjungi**". Tampilkan (atau lakukan hal lain tergantung tujuan traversal). *Push* node tersebut ke stack.
2. Jika adjacent node tidak ditemukan, *pop* sebuah node dari stack. Kunjungi adjacent node dari TOP stack.
3. Ulangi langkah 1 dan 2 sampai stack habis.

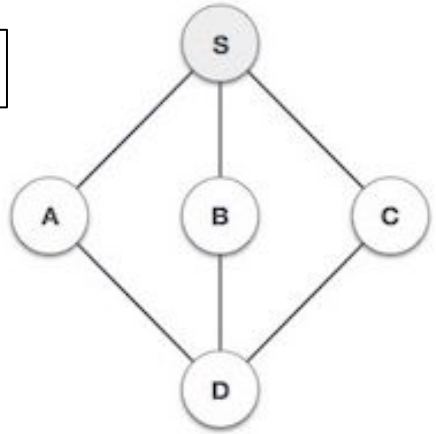
DFS (Depth First Search)

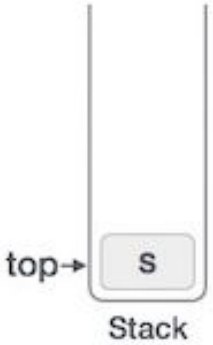
1



Inisialisasi

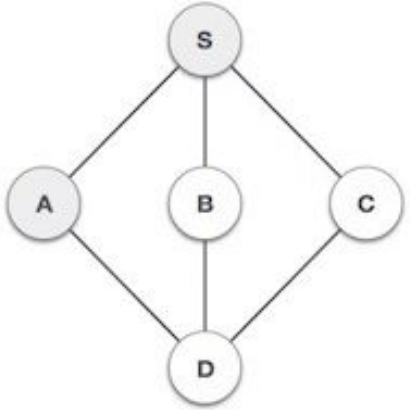
2

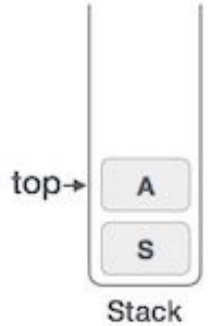




S

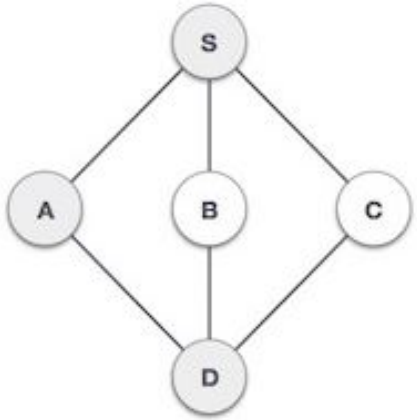
3

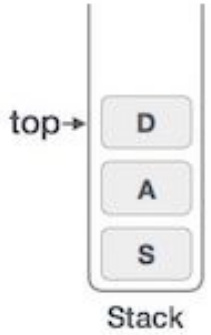




S A

4

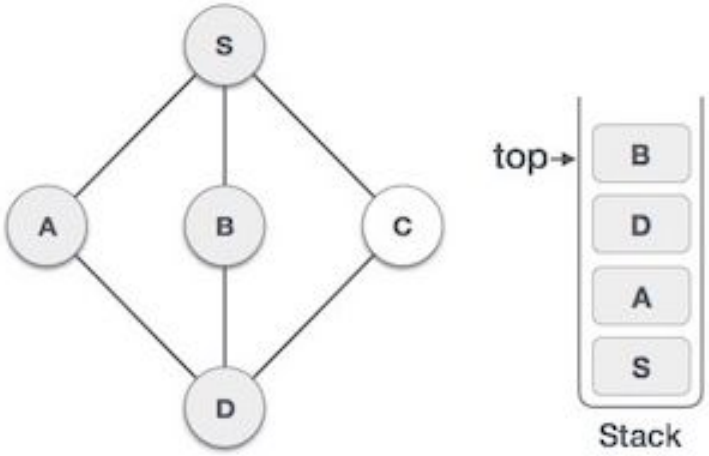




S A D

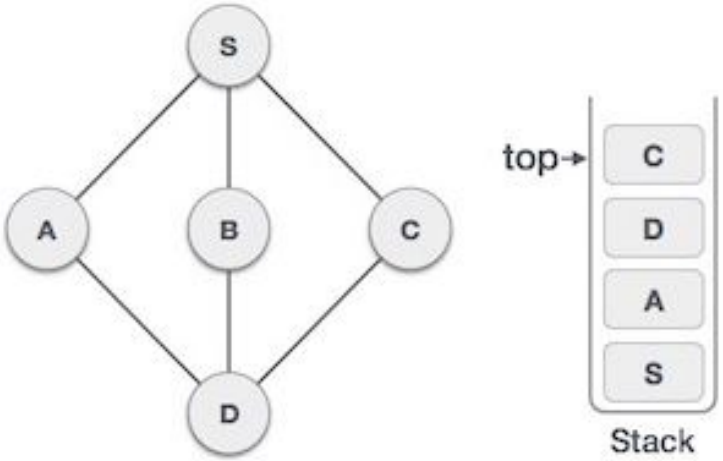
DFS (Depth First Search)

5



S A D B

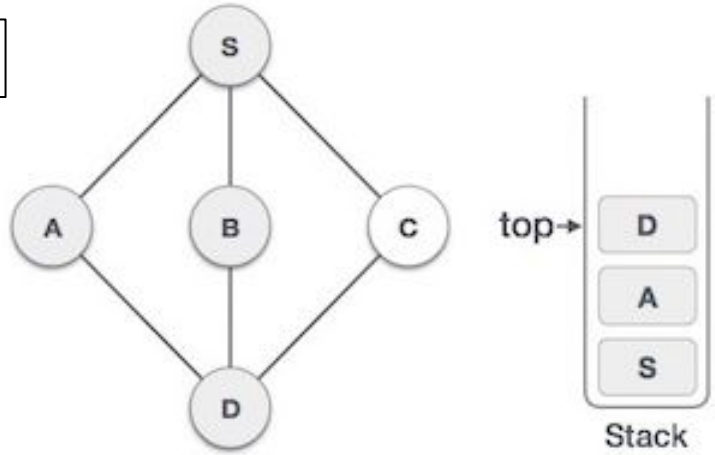
8



S A D B C

9. Lanjutkan sampai stack habis

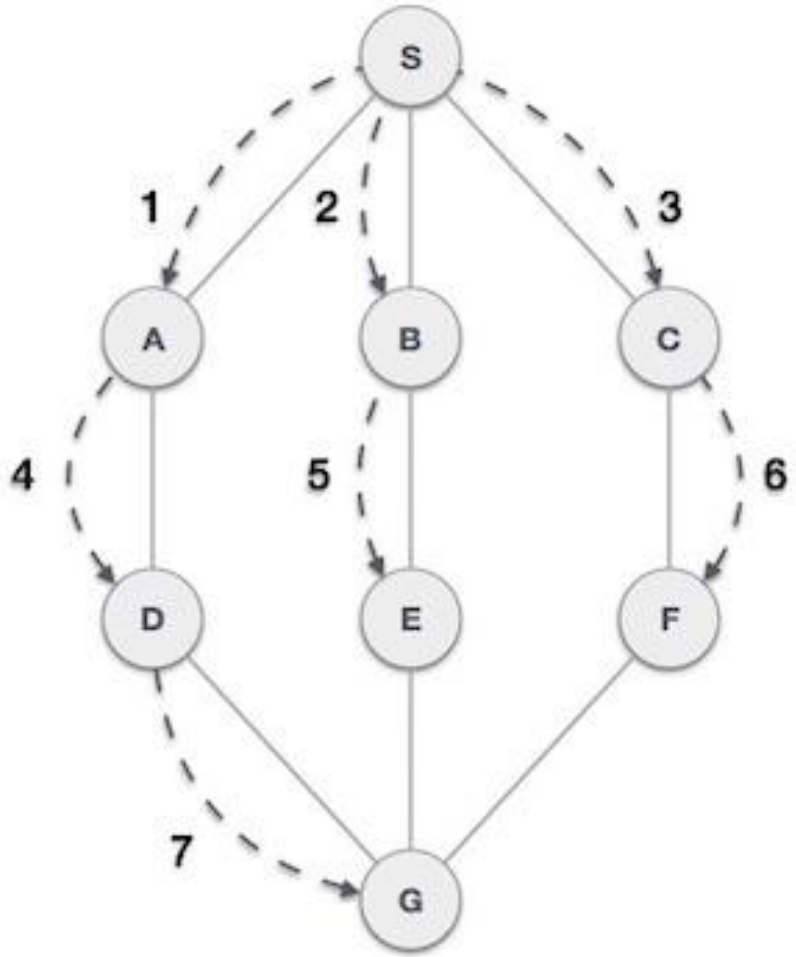
6



S A D B

Setelah ambil **B**, tidak ada adjacent node yang belum dikunjungi.
Jadi pop **B** dan cari adjacent node dari TOP stack (**D**) yaitu **C**

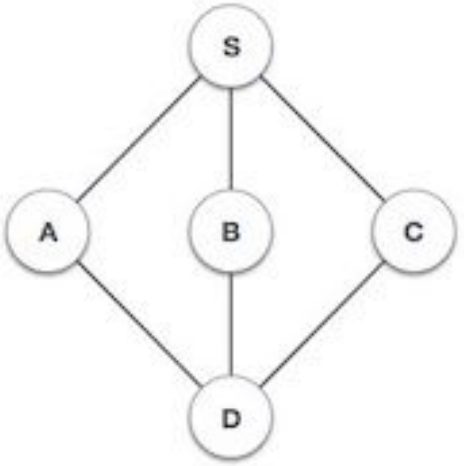
BFS (Breadth First Search)



1. Kunjungi adjacent node yang belum dikunjungi. Tandai node tersebut dengan "**sudah dikunjungi**". Tampilkan (atau lakukan hal lain tergantung tujuan traversal). Enqueue node tersebut ke queue.
2. Selama queue masih ada isinya:
   a. Dequeue node dari queue (misal node front ***w***)
   b. (*foreach*) Kunjungi setiap adjacent node dari node ***w***. Tandai node tersebut dengan "**sudah dikunjungi**". Tampilkan (atau lakukan hal lain tergantung tujuan traversal). Enqueue node tersebut ke queue.
3. Ulangi langkah 2 sampai queue habis.

BFS (Breadth First Search)

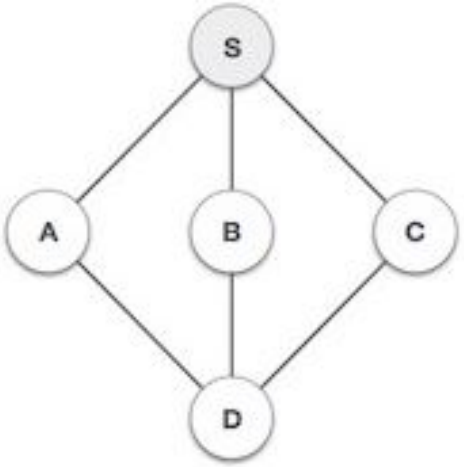
1



Inisialisasi

Queue

2

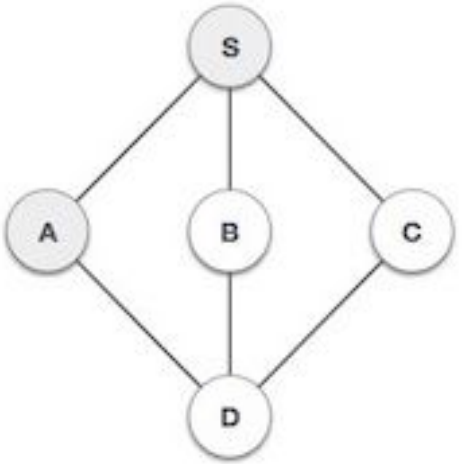


Lakukan **langkah 1** (tandai, tampilkan, enqueue **S**). Lakukan **langkah 2a** (dequeue S).

S

Queue

3



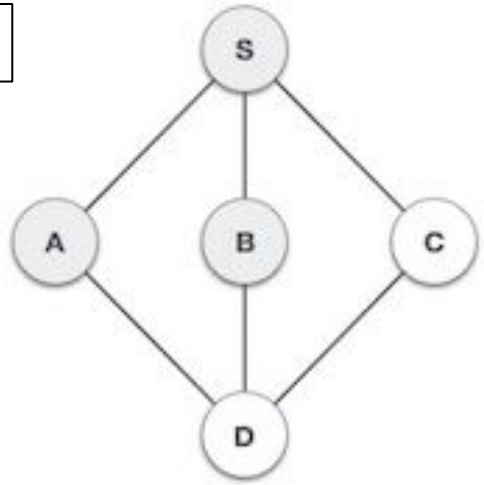
Langkah 2b

S A

A

Queue

4



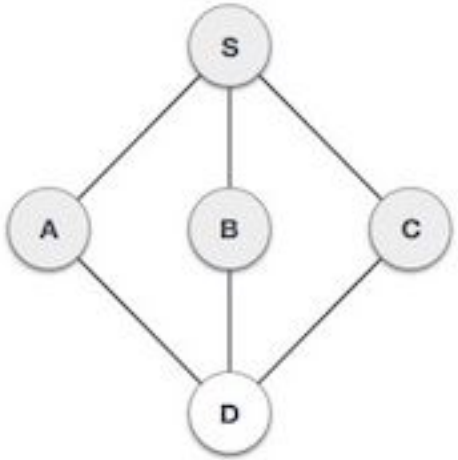
S A B

B A

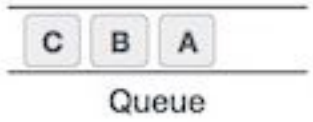
Queue

BFS (Breadth First Search)

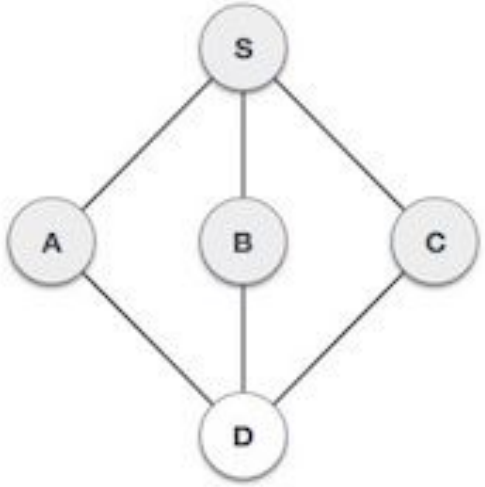
5



S A B C

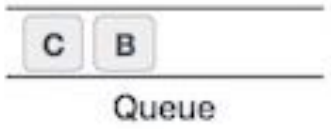


6

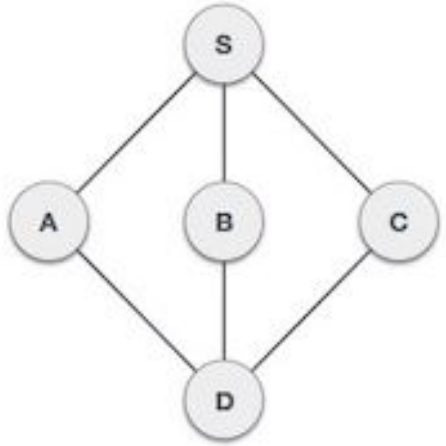


Langkah 2a
(dequeue **A**)

S A B C

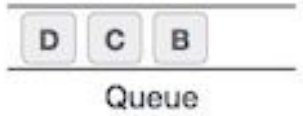


7



Langkah 2b

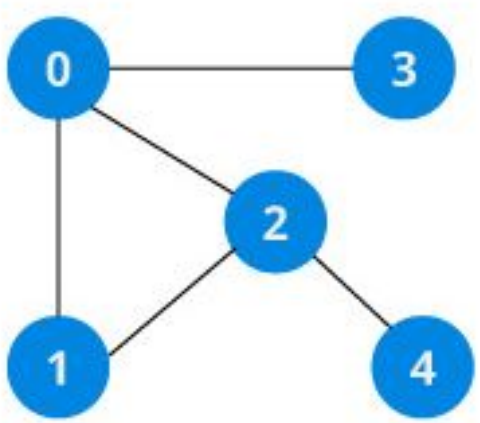
S A B C D



8. Lanjutkan sampai queue habis

Latihan

(a)

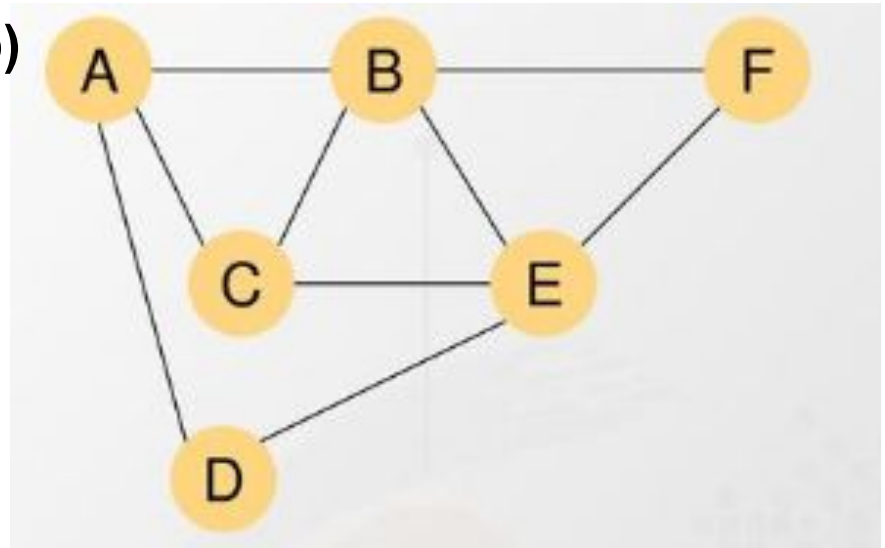


- Bagaimana traversal graph ini jika dimulai dari node 0/A dan node lebih kecil didahulukan?

(a) DFS : 0 1 2 4 3

BFS : 0 1 2 3 4

(b)

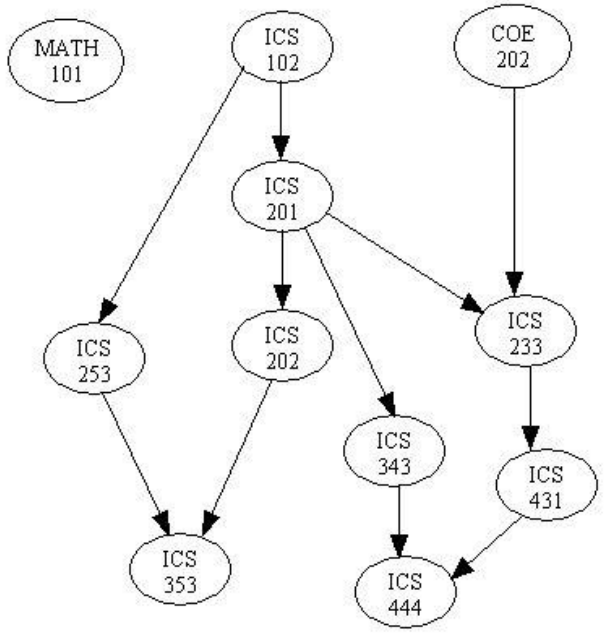


(b) DFS : A B C E D F

BFS : A B C D E F

Topological Order/Sort

- Untuk mengambil mata kuliah tertentu harus sudah lulus mata kuliah – mata kuliah lain. Bagaimana seharusnya urutan mata kuliah diambil?

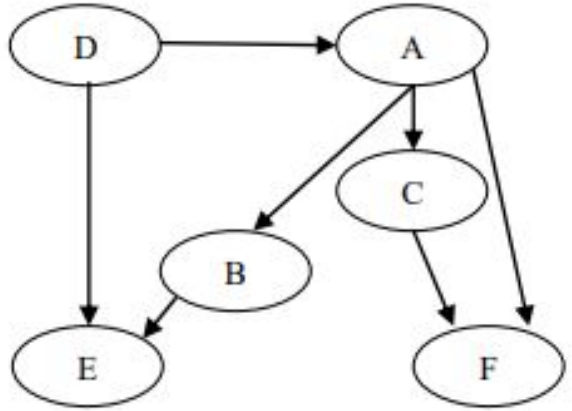


Yes! - Topological sort.

Harus **DAG (Directed Acyclic Graf)** = graph berarah dan tidak sirkuler (cycle)

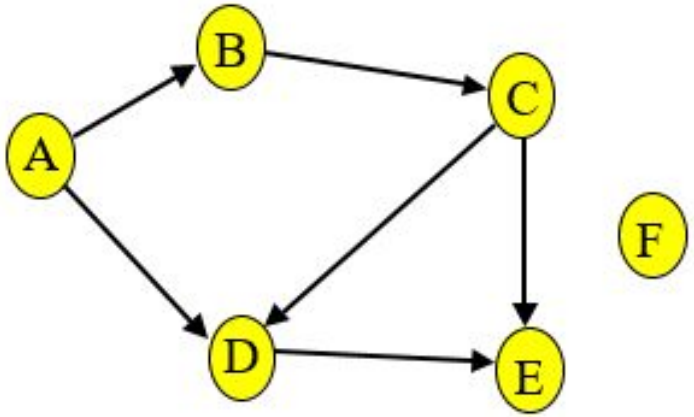
Bagaimana pengurutan pengerjaan jika:

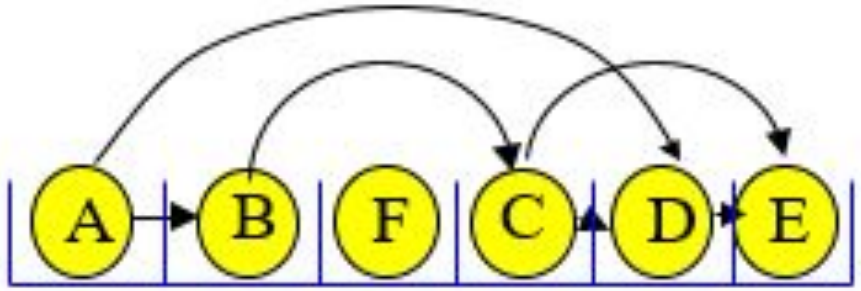
- A harus dikerjakan setelah D dikerjakan
- C baru dapat dikerjakan jika A dikerjakan
- B dikerjakan jika A dikerjakan
- E dikerjakan jika D dan B selesai dikerjakan
- F dikerjakan jika A dan C selesai dikerjakan



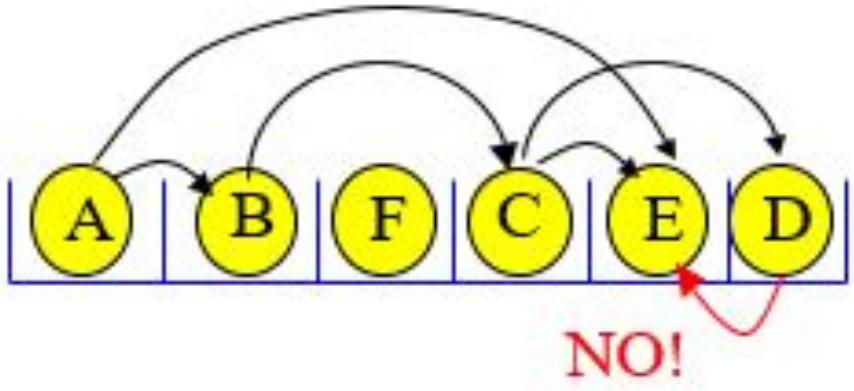
Topological Order/Sort

- Untuk graph berarah **G = (V, E),** topological order akan melakukan pengurutan linear node-nodenya sehingga: untuk setiap edge **(v, w)** di **E**, **v** mendahului **w** dalam urutan



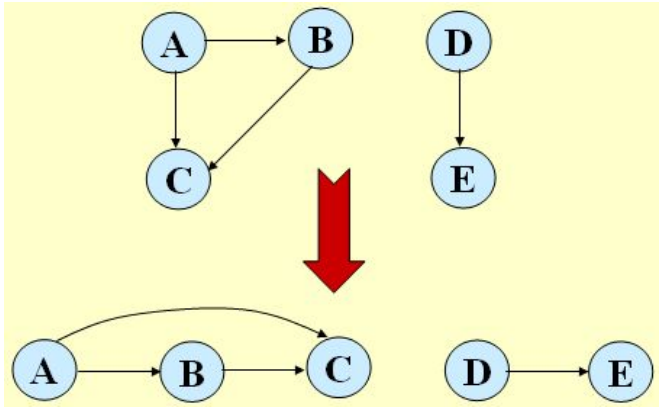


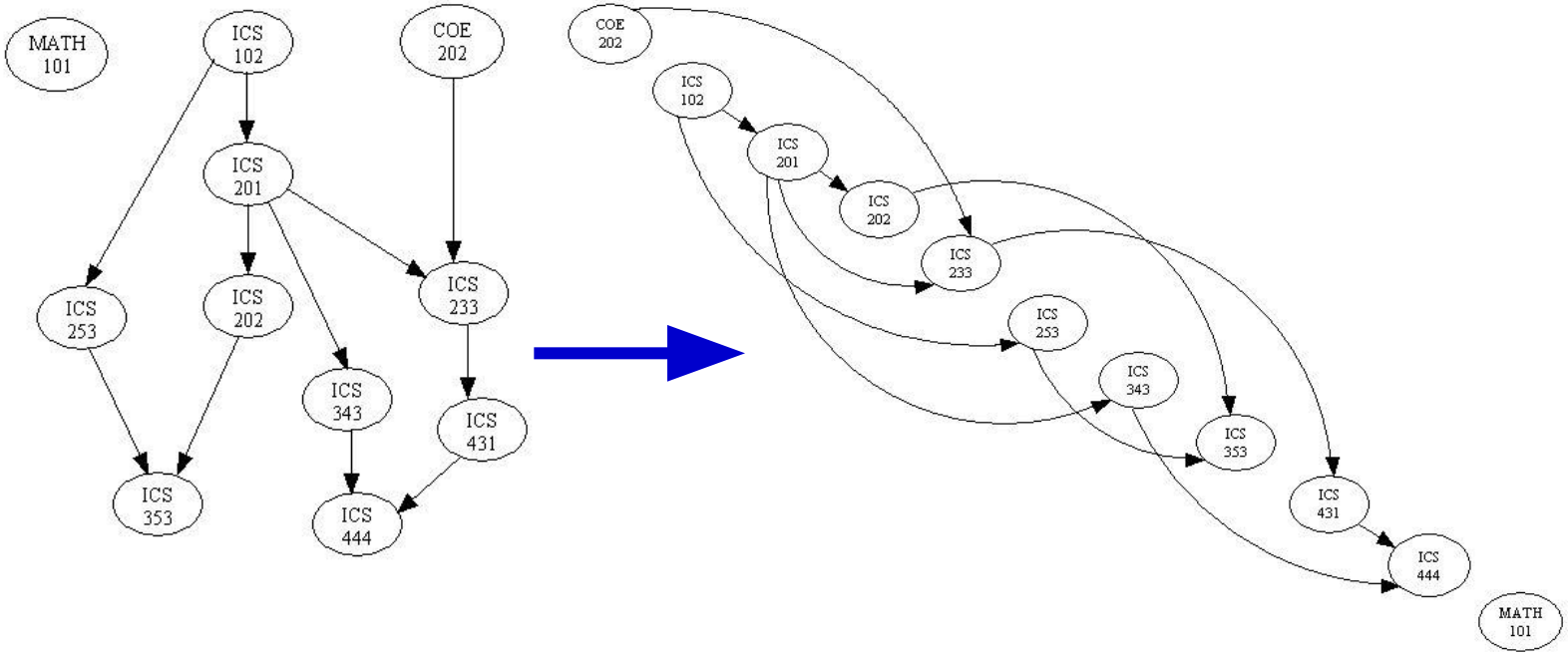
Tanda panah (arah) ke kanan!

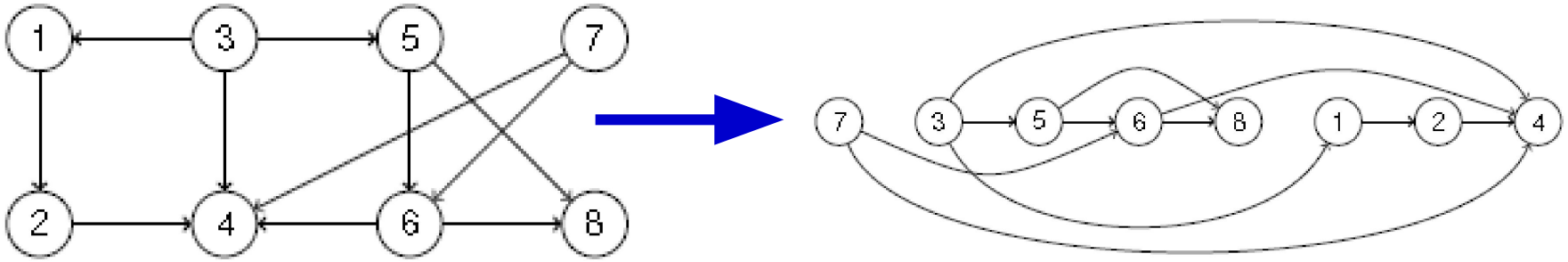


Topological Order/Sort

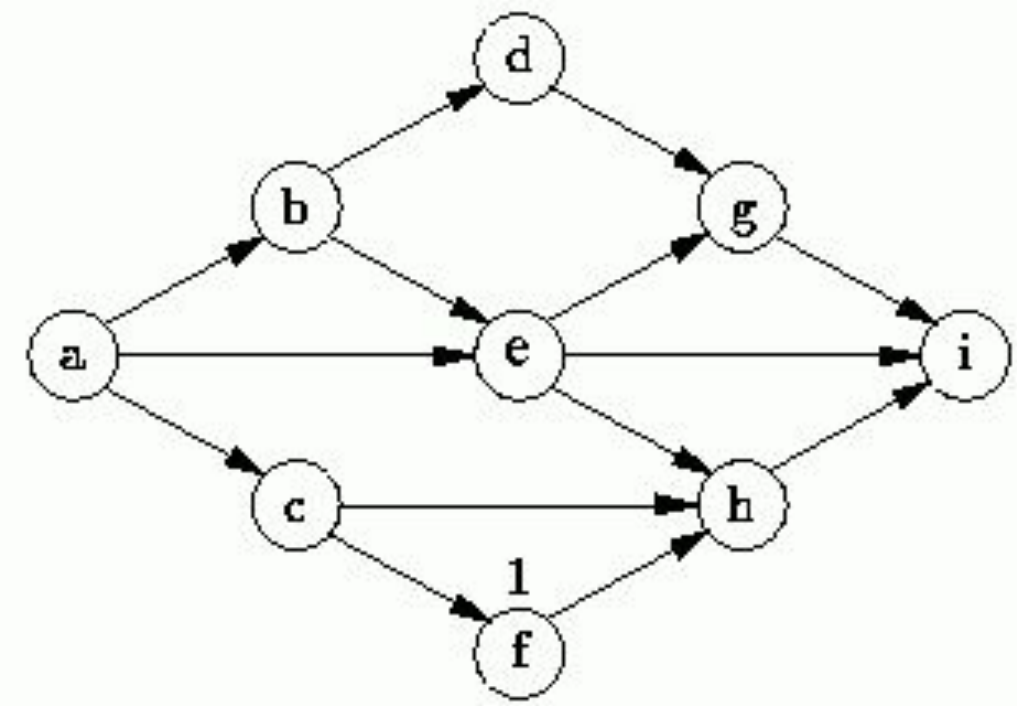
- Contoh Lain:







Topological Order/Sort = **Tidak Unique**



s1 = {a, b, c, d, e, f, g, h, i}

s2 = {a, c, b, f, e, d, h, g, i}

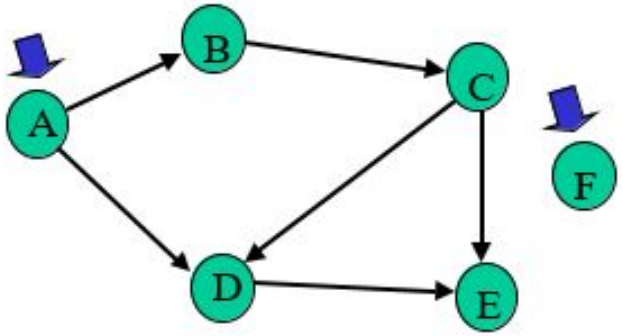
s3 = {a, b, d, c, e, g, f, h, i}

s4 = {a, c, f, b, e, h, d, g, i}

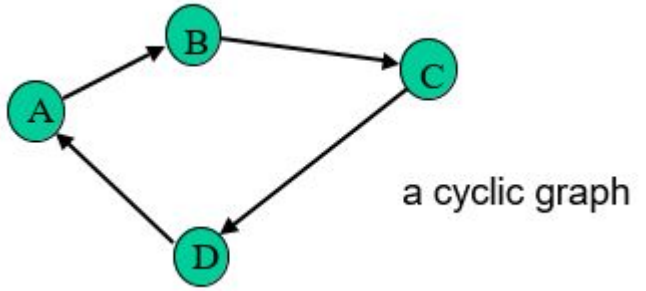
dan seterusnya.

Topological Order/Sort: Algoritma Topo Sort

- **Langkah 1**: identifikasi node-node yang tidak mempunyai edge yang masuk **indegree(node) = 0**

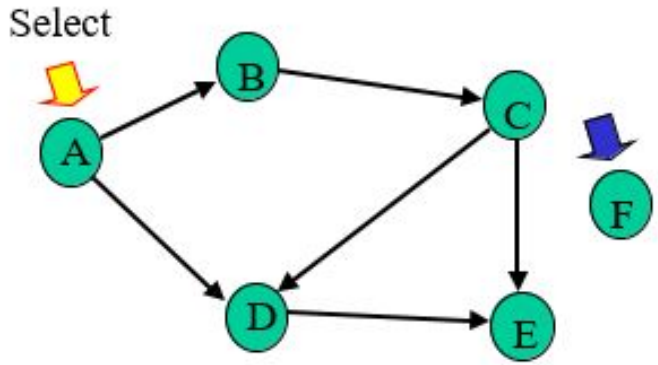


Jika tidak ada node tersebut berarti cycle STOP



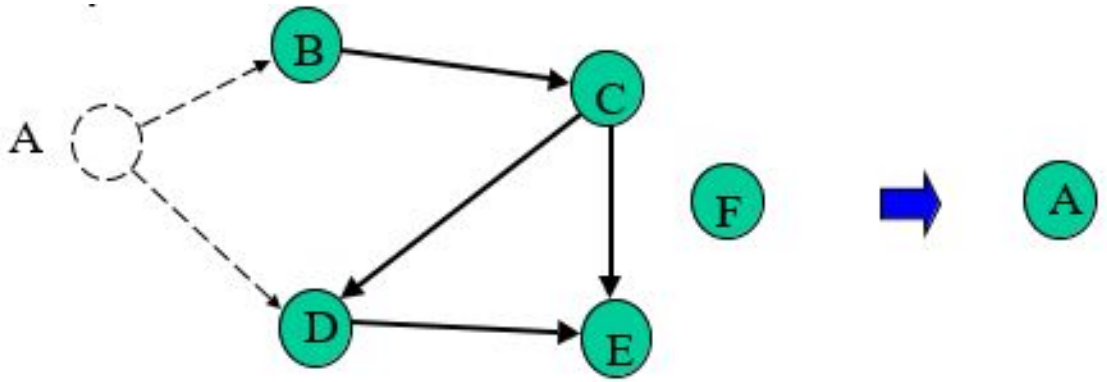
a cyclic graph

  Pilih salah satu node:

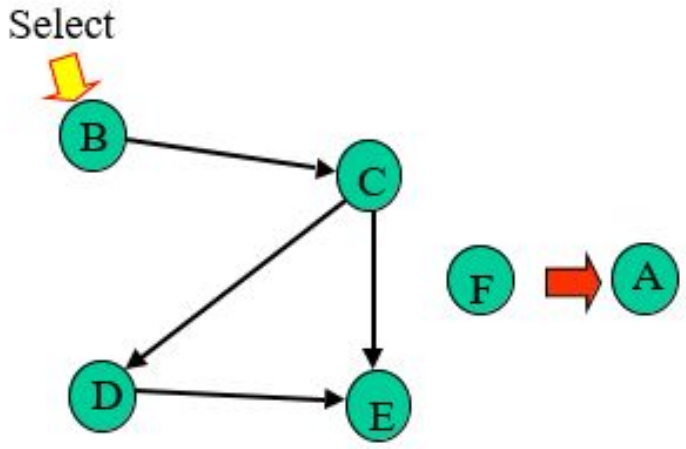


Topological Order/Sort: Algoritma Topo Sort

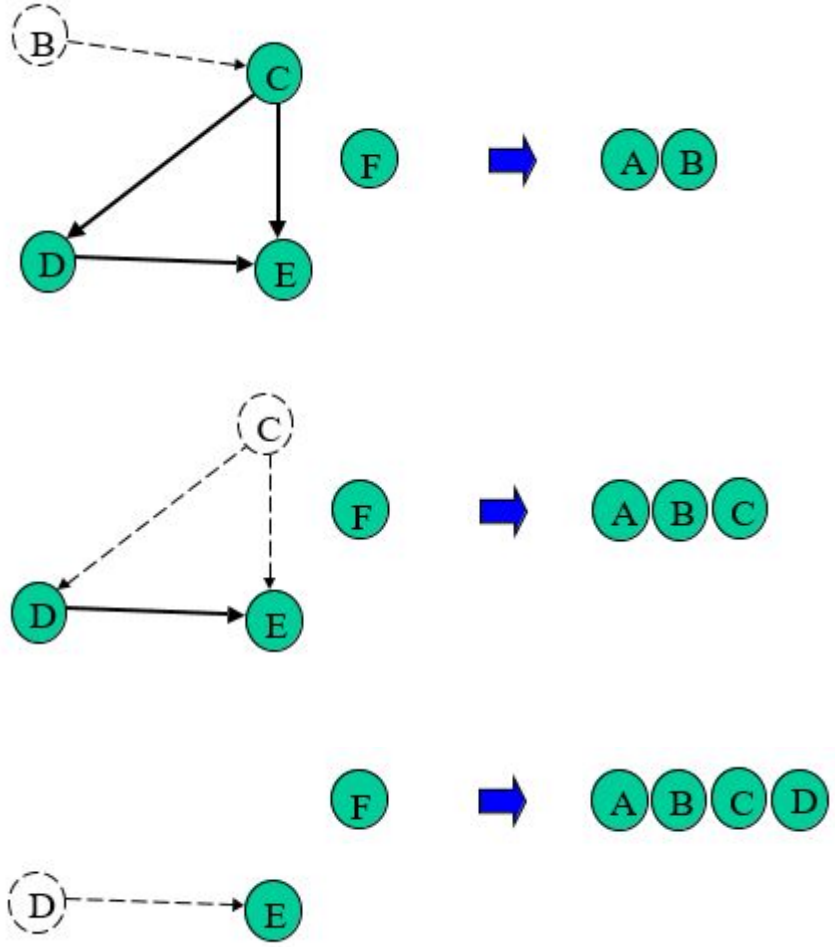
- **Langkah 2**: hapus node tersebut (node dengan indegree 0) dan semua edge yang keluar dari node tersebut. Tempatkan node tersebut sebagai output.

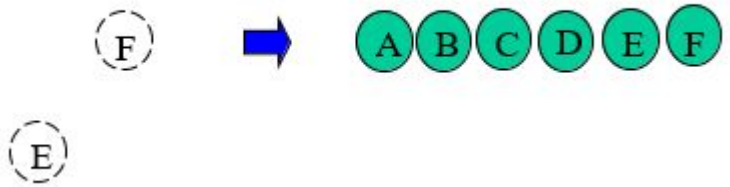


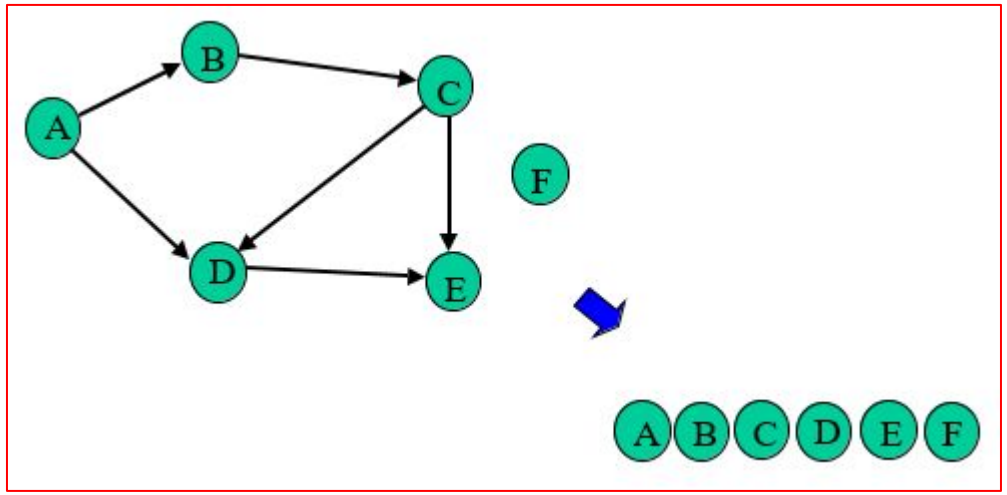
- Ulangi **langkah 1** dan **langkah 2** sampai graph kosong



Topological Order/Sort: Algoritma Topo Sort

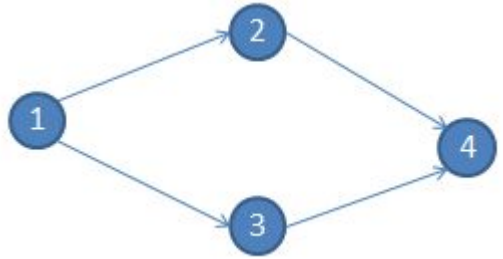






Implementasi dalam program

1. Simpan **indegree** dari masing-masing node dalam sebuah **array**
2. Inisialisasi sebuah **array/queue/stack** (atau struktur data lain) menyimpan berisi node-node dengan **indegree = 0**
3. Selama **array/queue/stack** masih ada isinya:
   a. **Dequeue/push** node dan masukan node sebagai output (misal node **w**)
   b. **Update array indegree** dari node-node yang adjacent dengan node **w** (**kurangi 1**)
   c. **Enqueue/pop** node-node dengan indegree yang berubaha menjadi 0
4. Jika semua node sudah di output **selesai**, jika tidak semua node di output cycle

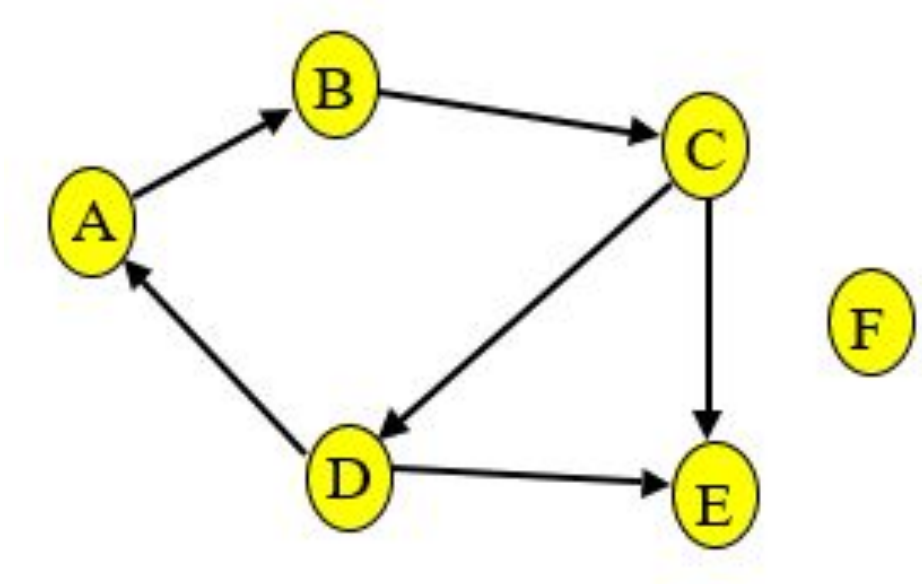


Cek file: topo_sort.c

```
Enter the no of vertices:
4
Enter the adjacency matrix:
Enter row 1
0 1 1 0
Enter row 2
0 0 0 1
Enter row 3
0 0 0 1
Enter row 4
0 0 0 0
```

Latihan

- Bagaimana topological order untuk graph di bawah ini?

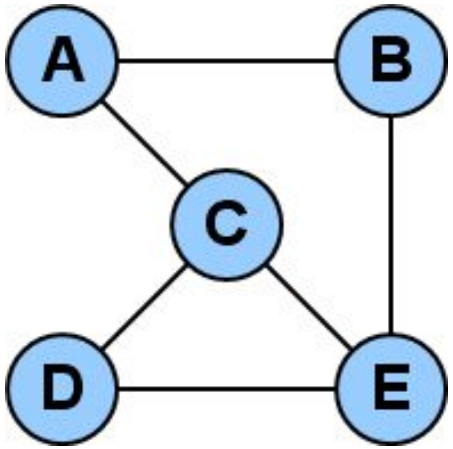


Tidak bisa, karena membentuk cycle bukan DAG

Graph VS Tree

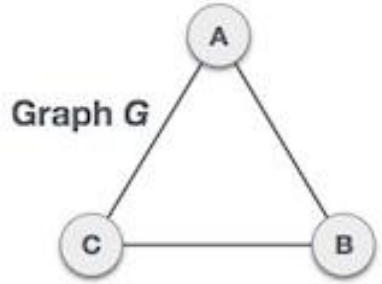
- Sebuah Graph memiliki ciri berbeda dengan Tree
  - Dalam Graph, edge bebas menghubungkan node-node mana pun.
  - Dalam Tree, satu node hanya boleh terhubung ke satu parent dan beberapa child, tidak boleh ke beberapa parent.

- Dalam sebuah Graph bisa dirunut jalur edge yang membentuk jalur putaran dari 1 node kembali ke node semula; **ini tidak boleh terjadi dalam Tree**

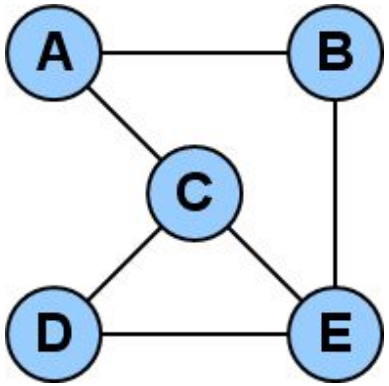


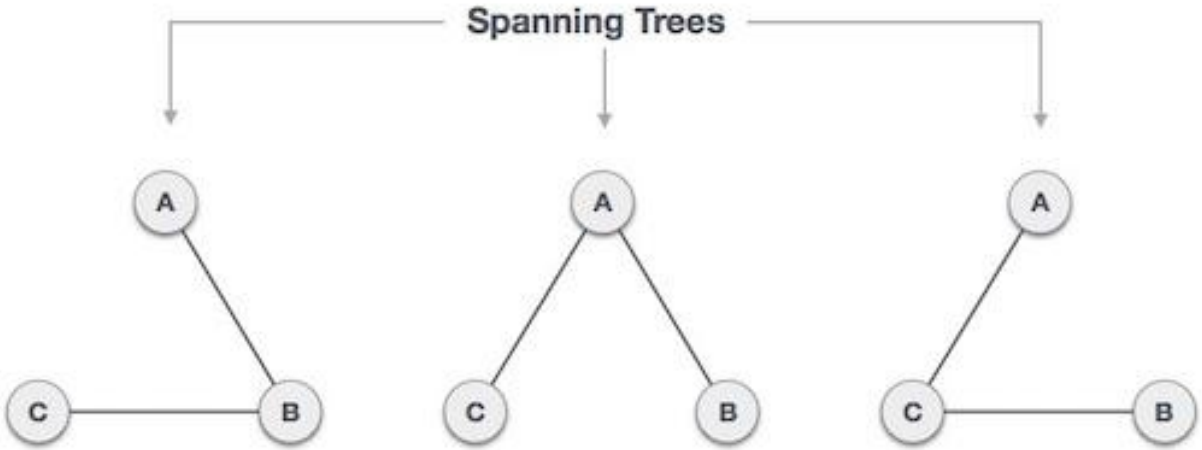
Spanning Tree

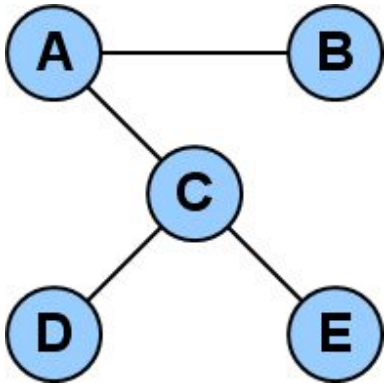
- **Spanning Tree** adalah sebuah Tree yang dibuat dari sebuah Graph dengan menghilangkan beberapa edge-nya. Tree ini harus mengandung semua node yang dimiliki Graph

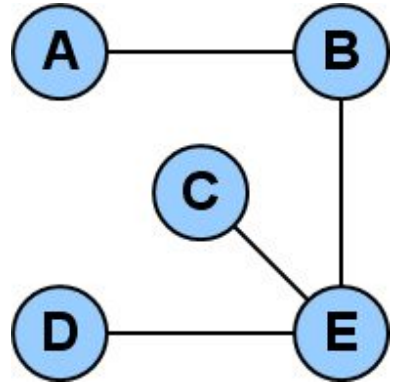


Menghilangkan jalur edge yang membentuk jalur putaran dari 1 node kembali ke node semula









Minimum Spanning Tree

- Jika Weighted Graph diubah menjadi Spanning Tree, tiap kombinasi Tree yang dapat dibuat memiliki total weight yang berbeda-beda
- **Problem Minimum Spanning Tree (MST)** berusaha mencari Tree seperti apa yang bisa dibuat dari sebuah Weighted Graph dengan **total weight seminimal mungkin**

Contoh Aplikasi:

- Membangun rel kereta api yang bisa menghubungkan beberapa kota
- Mendesain jaringan telekomunikasi (fiber-optic networks, computer networks, leased-line telephone networks, cable television networks, etc.)
- Mendesain jaringan pipa untuk menghubungkan sejumlah lokasi

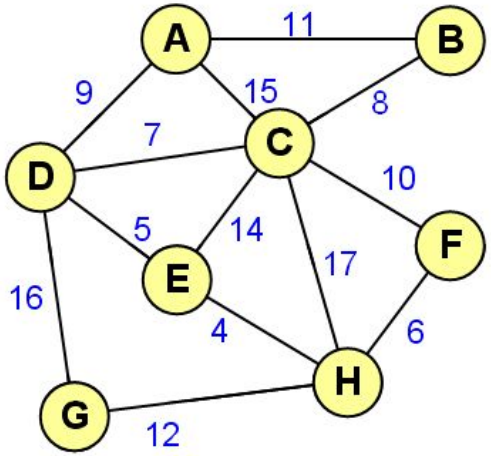
Minimum Spanning Tree dengan Algoritma Dijkstra

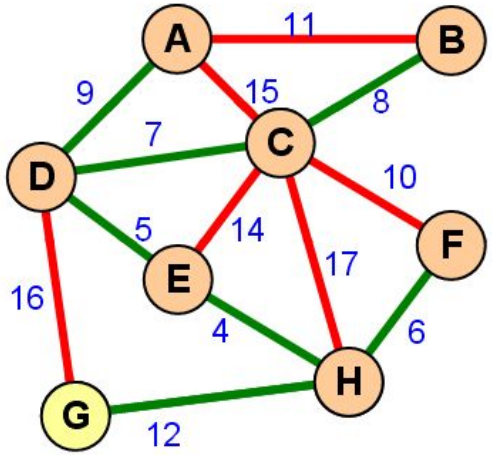
- Oleh Edsger Dijkstra di tahun 1959
- Pada algoritma ini, kita akan menandai node yang memiliki lintasan terpendek/minimum,dengan syarat setiap node yang terhubung tidak boleh membentuk siklus/circle. Algoritma akan berhenti jika semua node telah terhubung/terkunjungi oleh lintasan.

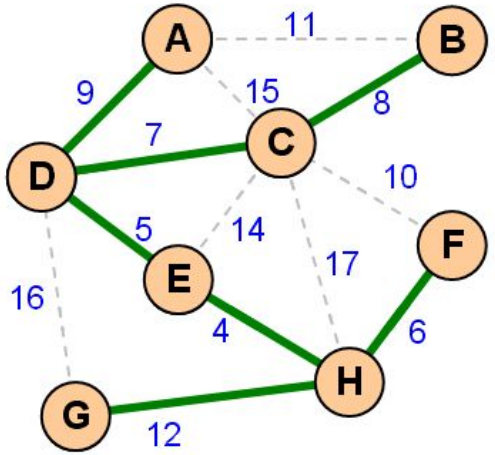
Langkah – langkah:

1. Pilih node awal. Dari semua edge yang terhubung ke node awal tersebut, pilih edge dengan bobot terkecil.
2. Tandai edge yang dipilih dengan warna hijau.
3. Apabila ada edge yang kedua node-nya sudah terkena jalur hijau, tandai edge tersebut dengan warna merah (karena jika dipilih akan membentuk jalur cycle melanggar syarat tree)
4. Bandingkan semua edge yang terhubung ke node-node hijau. Pilih bobot terkecil. Tandai edge yang dipilih dengan warna hijau.
5. Ulangi sampai semua node telah dilewati jalur hijau, maka jalur hijau yang terbentuk adalah MST yang dicari.

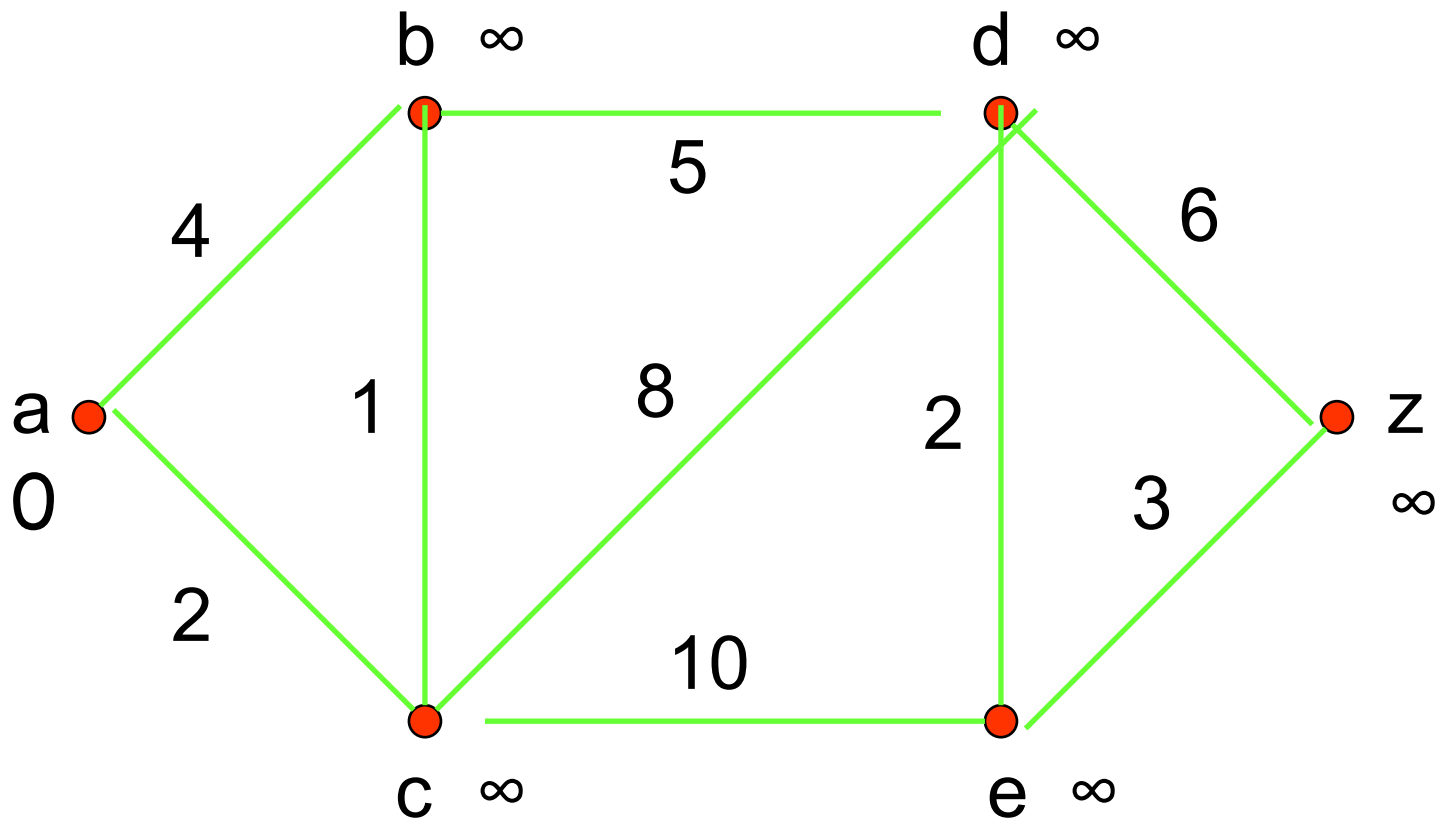
Minimum Spanning Tree dengan Algoritma Dijkstra



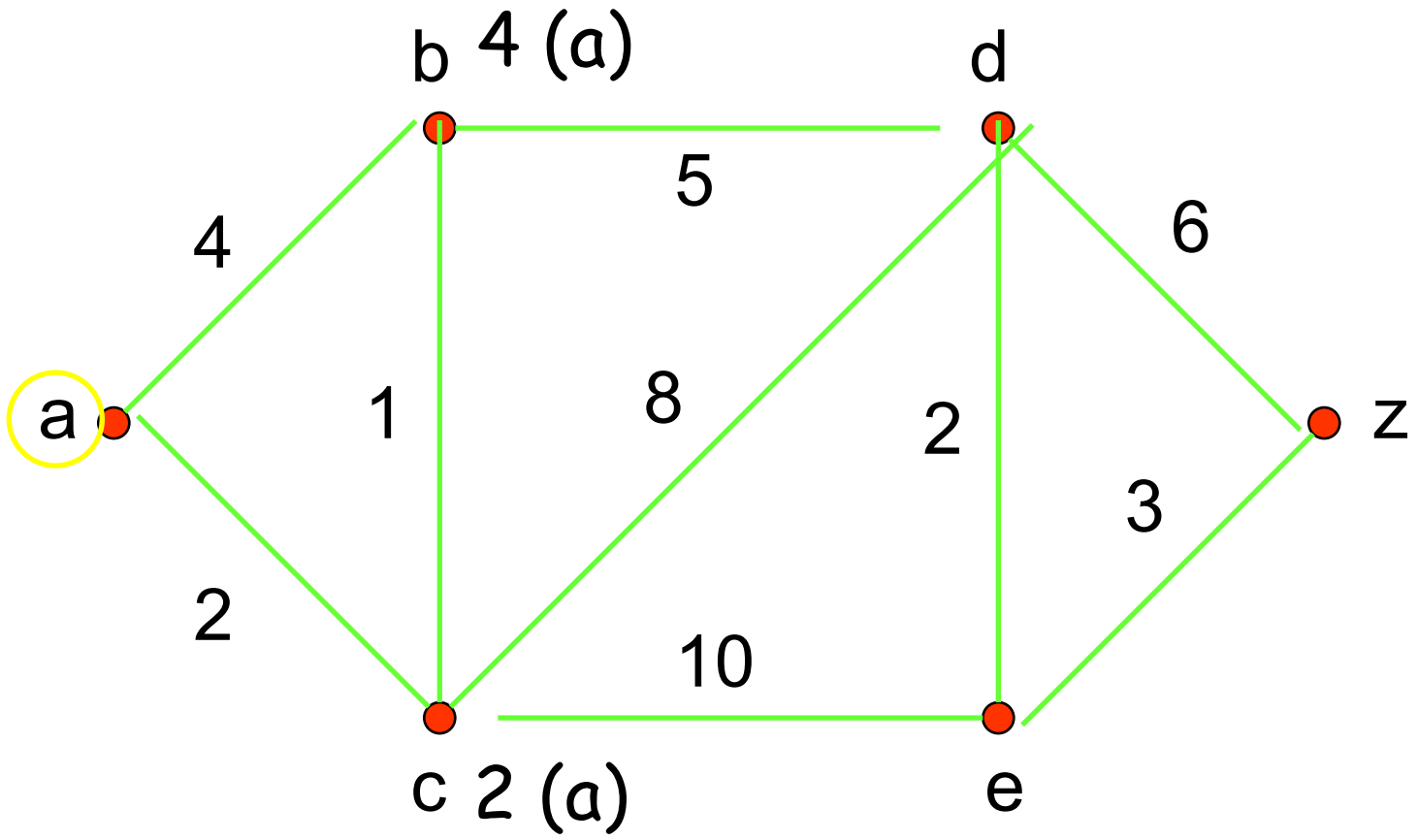




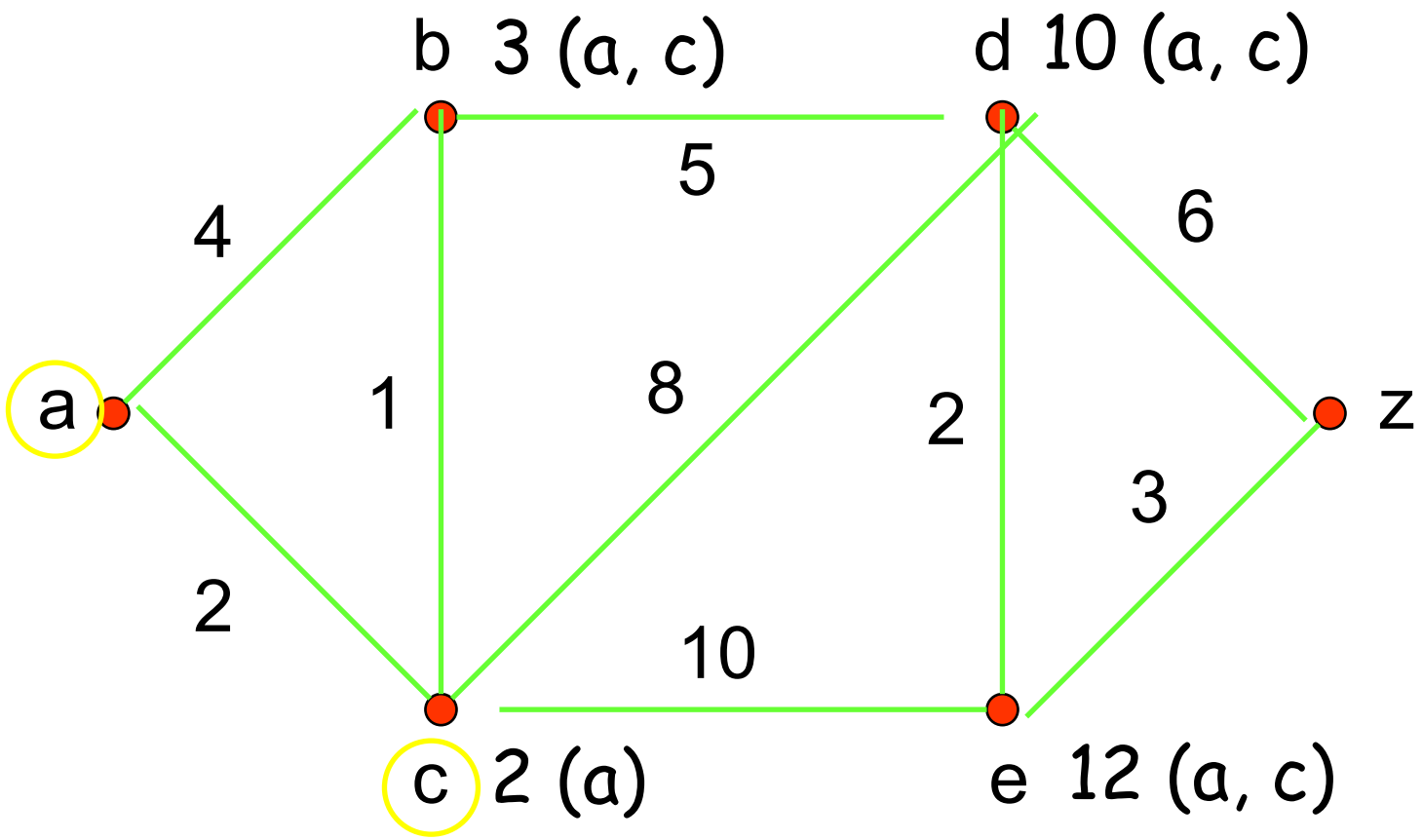
Contoh Algoritma Dijkstra



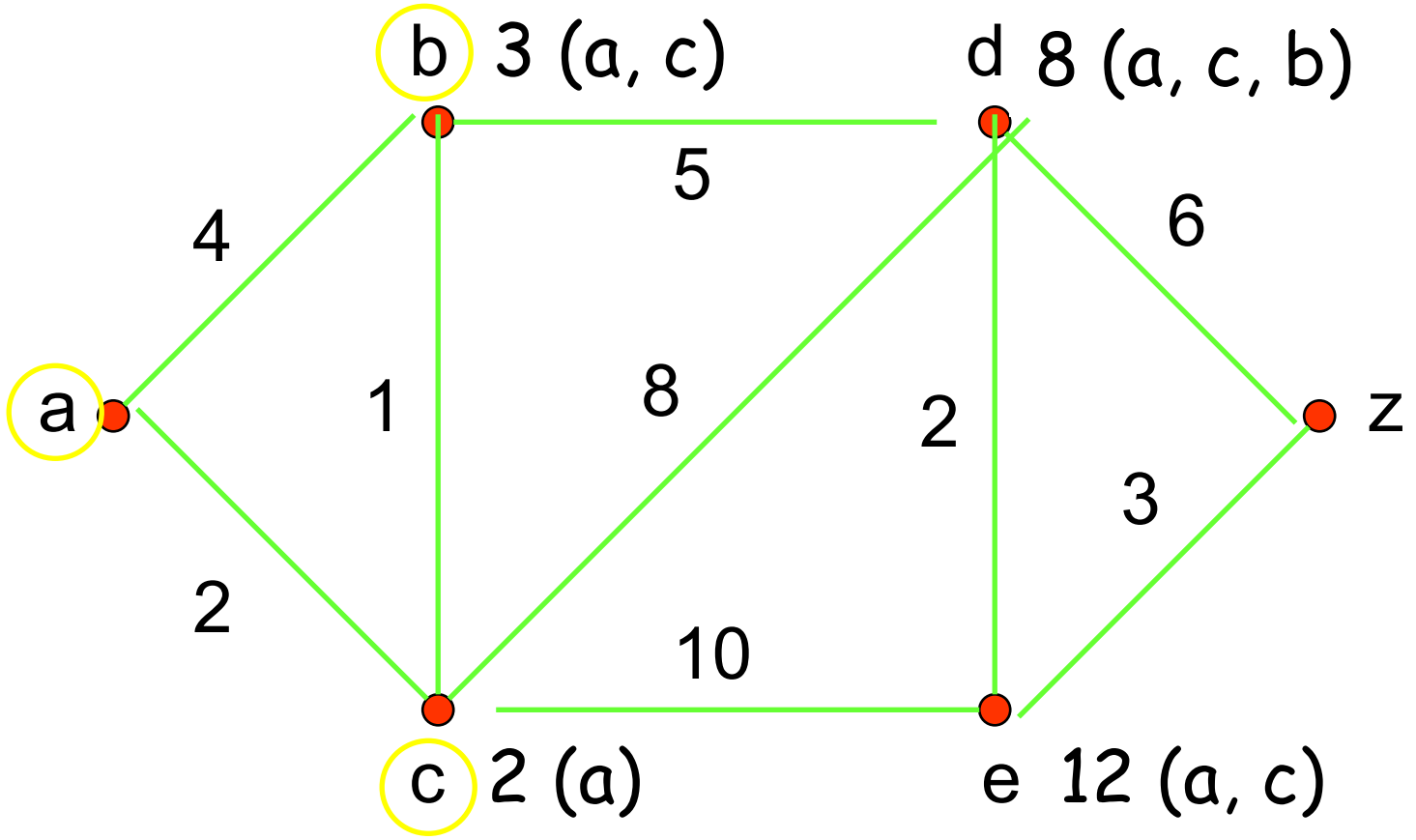
Step 0



Step 1

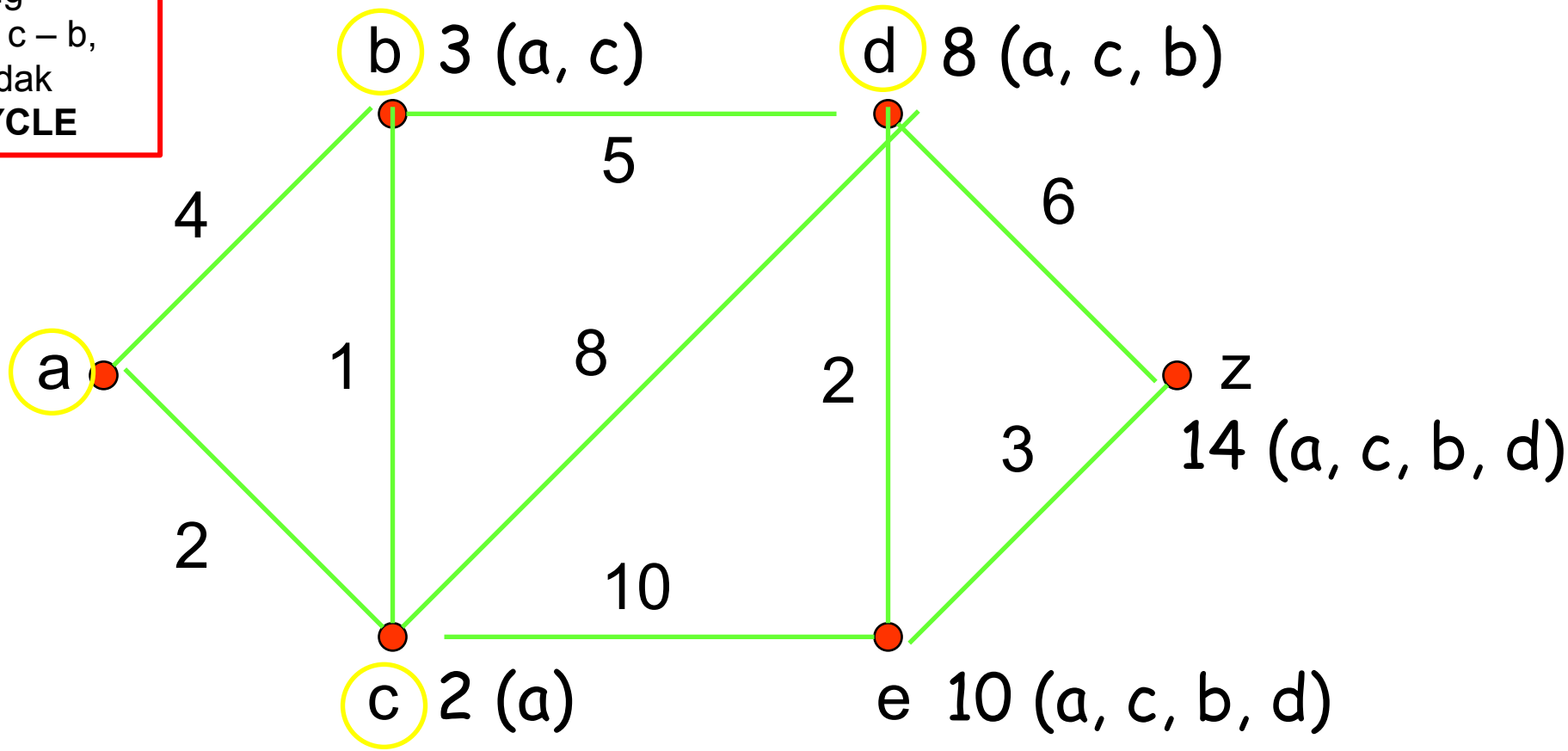


Step 2

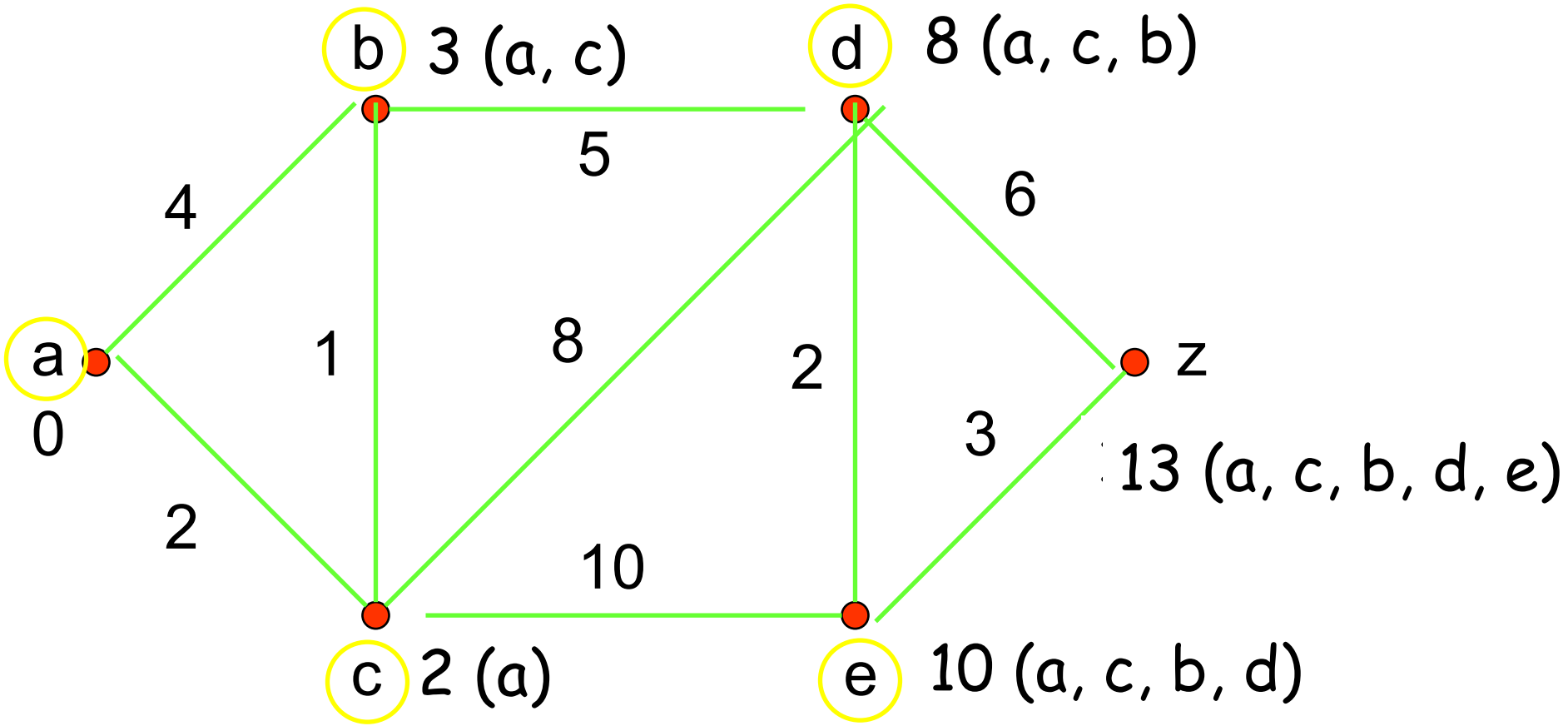


Step 3

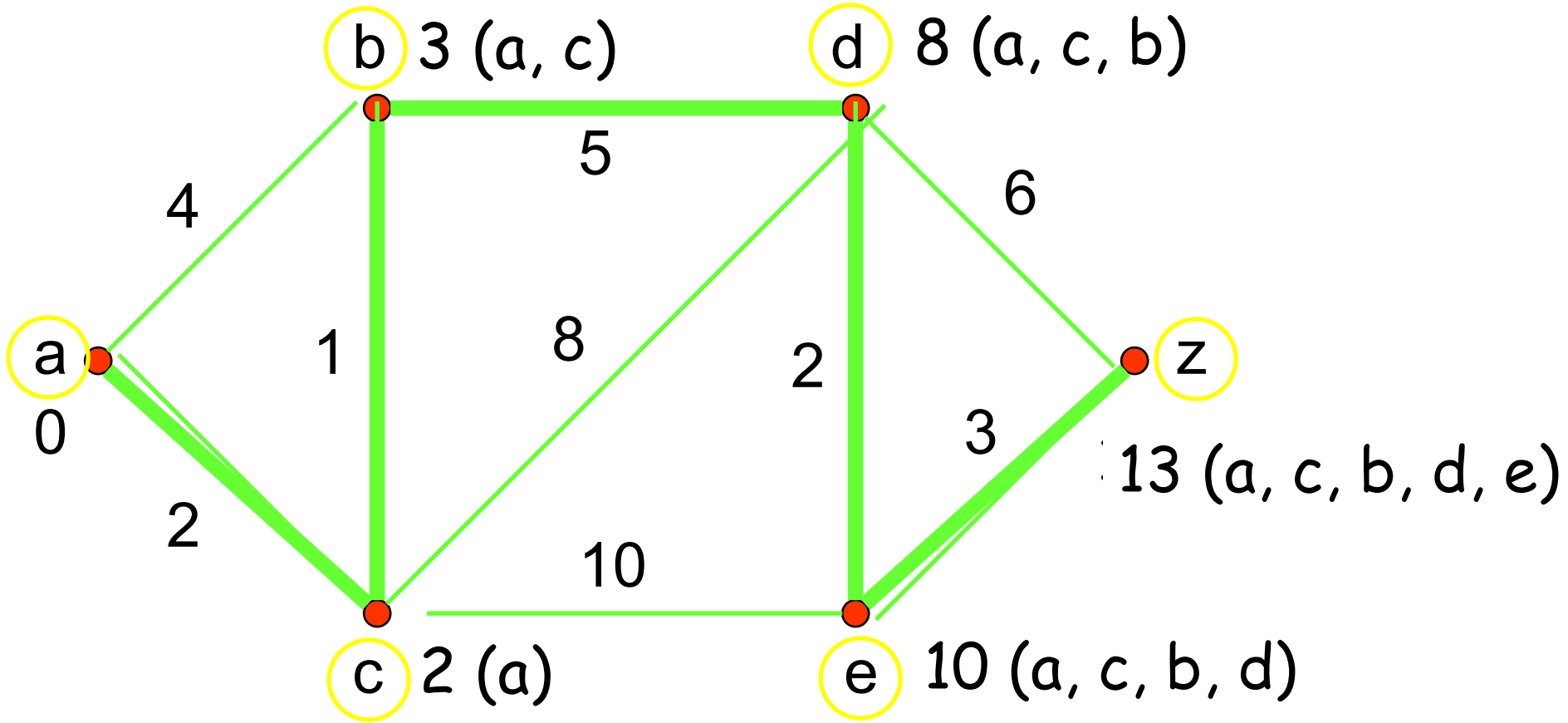
Edge (b,d) terhubung dengan lintasan a – c – b, kenapa edge(b,a) tidak dihitung juga? □ **CYCLE**



Step 4

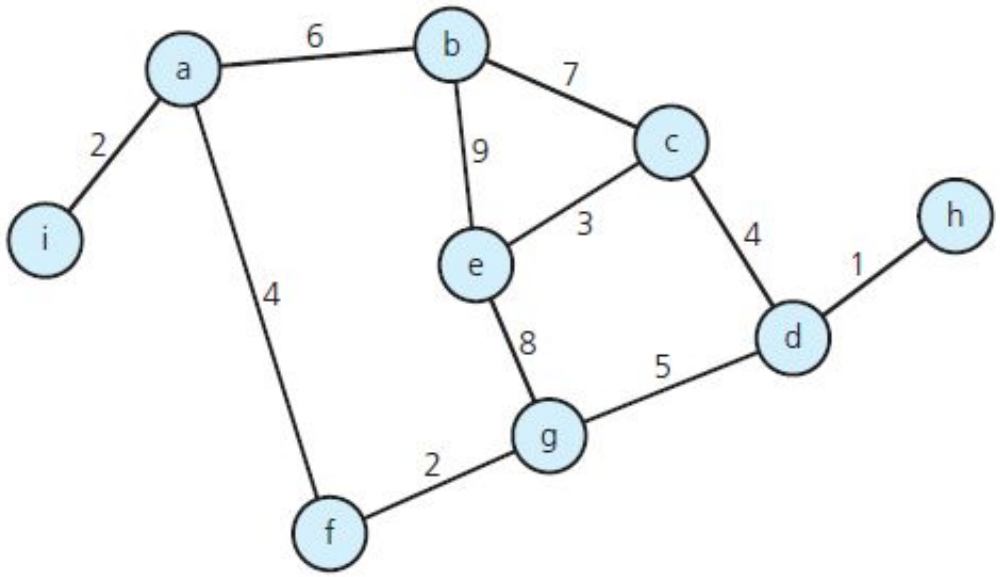


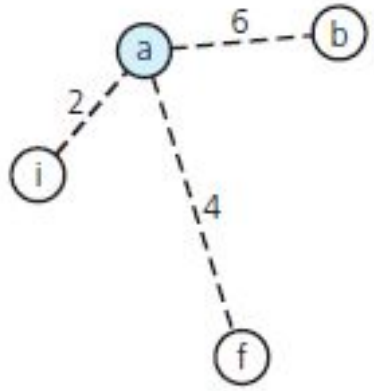
Step 5



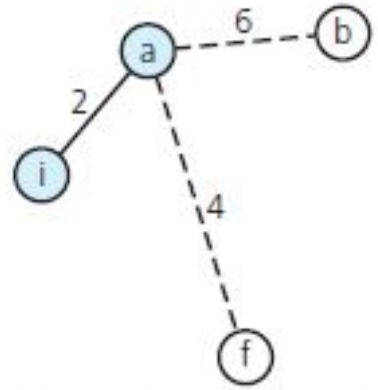
Step 6

Latihan

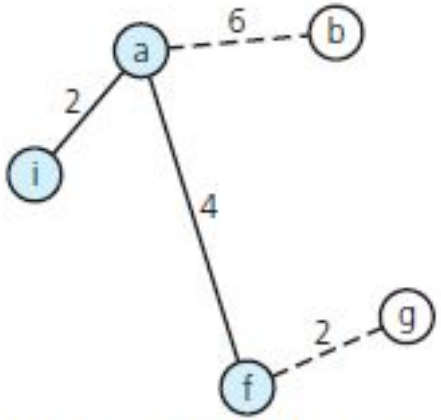




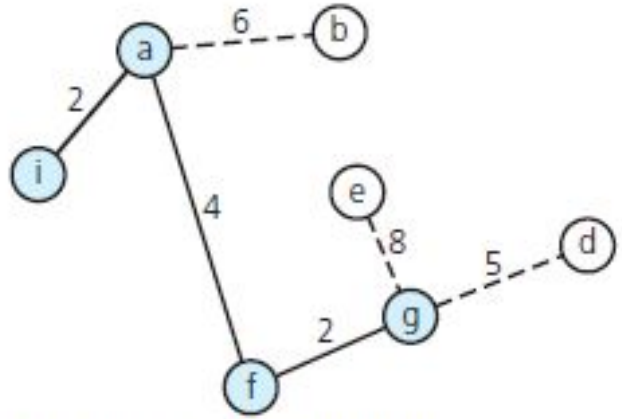
(a) Mark a, consider edges from a



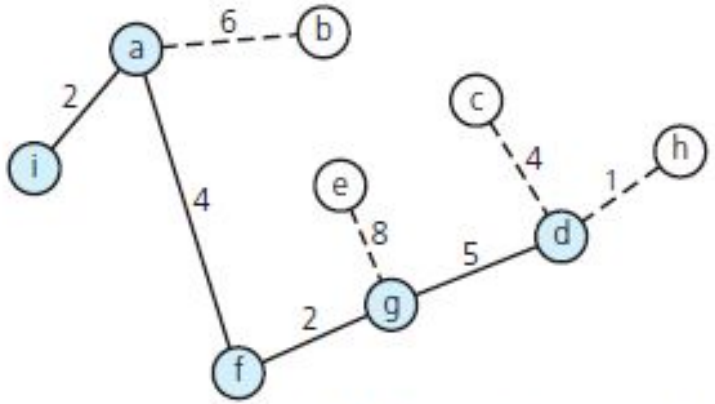
(b) Mark i, include edge (a, i)



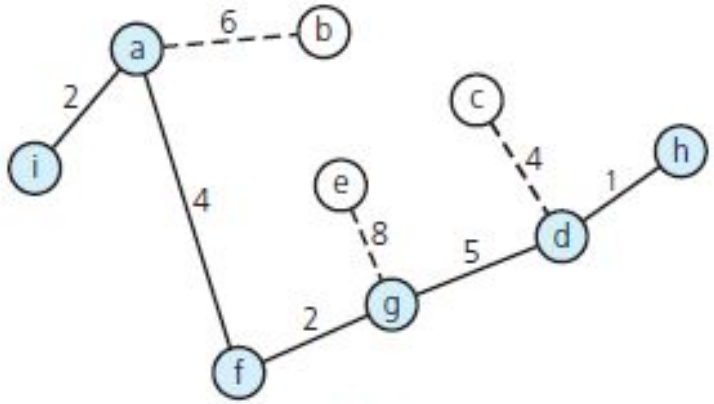
(c) Mark f, include edge (a, f)



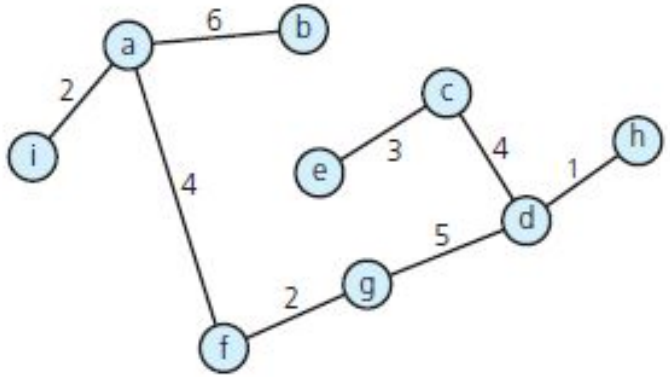
(d) Mark g, include edge (f, g)



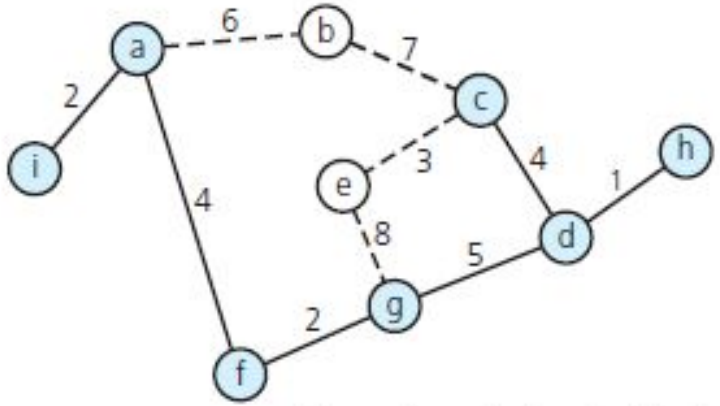
(e) Mark d, include edge (g, d)



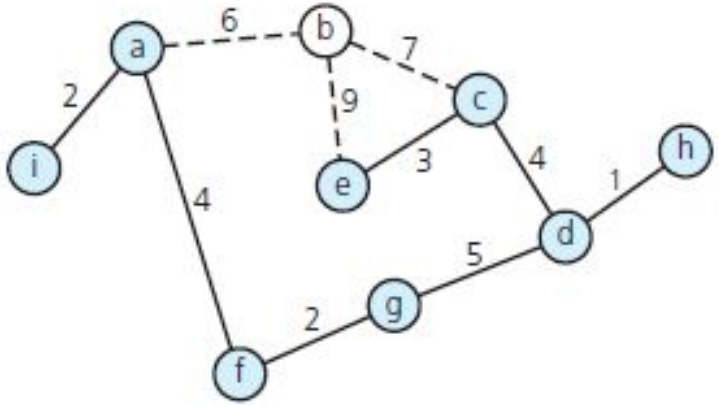
(f) Mark h, include edge (d, h)



(i) Mark b, include edge (a, b)



(g) Mark c, include edge (d, c)



(h) Mark e, include edge (c, e)

TERIMA KASIH