11. Graph

ARNA FARIZA

YULIANA SETIOWATI

Capaian Pembelajaran

Mahasiswa mengerti konsep Graph.

Mahasiswa dapat mengimplementasikan graph dalam bahasa pemrograman.

Mahasiswa dapat mengimplementasikan algoritma pencarian rute terpendek menggunakan wharshall dan djikstra.

Materi

Definisi Graph

Jenis Graph

Representasi Graph

Algoritma Shortest Path

- Warshall

- Djikstra

Graph

Graph merupakan suatu cabang ilmu yang memiliki banyak terapan. Banyak sekali struktur yang bisa direpresentasikan dengan *graph*, dan banyak masalah yang bisa diselesaikan dengan bantuan *graph*. Seringkali *graph* digunakan untuk merepresentasikan suatu jaringan.

Misalkan jaringan jalan raya dimodelkan *graph* dengan kota sebagai simpul (*vertex/node*) dan jalan yang menghubungkan setiap kotanya sebagai sisi (*edge*) yang bobotnya (*weight*) adalah panjang dari jalan tersebut.

Graph

Graph merupakan kumpulan titik (**vertex**), beberapa dari vertex tsb terhubung dengan suatu garis (edge)

$G = \langle V,E \rangle$

V = {vertex} E = {edge}

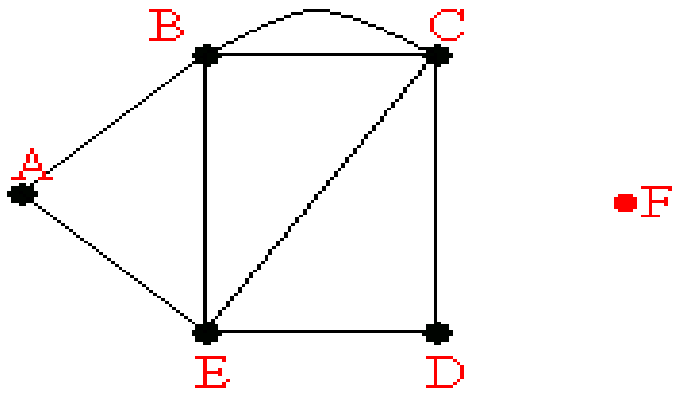
Karena garis selalu diawali pada suatu titik dan diakhiri dengan titik lain, maka garis bisa dituliskan sebagai pasangan antara dua titik.

Notasi e = [u,v]

Dua buah titik yang dihubungkan dengan sebuah garis disebut sebagai **titik yang bertetangga**

Titik-titik yang tidak dihubungkan dengan titik lain disebut dengan **titik terisolir (isolated node)**

Graph



- $G = \langle V, E \rangle$
- V = {A, B, C, D, E, F}
- E = {(A,B) (A,E) (B,C) (C,B) (C,D) (D,E) (C,E) (D,D) (E,B)}

- Dari B ke C terdapat dua garis disebut **multiple edge**
- Terdapat garis yang berujung dan berpangkal yang sama = **loop**

Graph

Pertidaksamaan untuk graph yang tidak memiliki loop

$0 \leq | E | \leq |V| (|V| - 1)/2$

Graph yang tiap verteknya terhubung dengan edge = complete

Notasi = $K_{|V|}$

Path

Sebuah jalur (path) dengan panjang n dari titik u ke v didefinisikan sebagai deretan n+1 titik ditulis

$P = (v_0, v_1, v_2, \ldots, v_n)$

Sedemikian rupa sehingga titik yang merupakan pangkal dari garis menjadi ujung dari garis berikutnya kecuali **titik pertama dan terakhir (titik v_0 dan v_n)**

Jalur yang terbentuk dari titik-titik yang berbeda disebut dengan **simple path**

Path

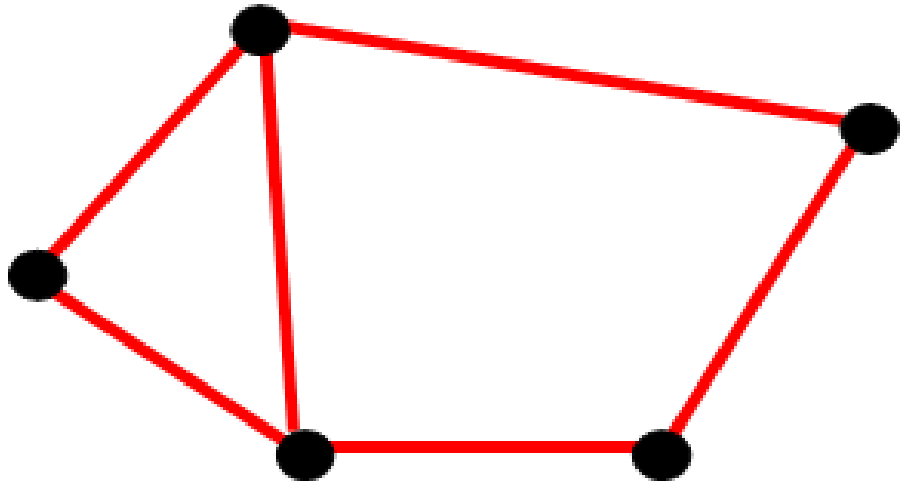
Jalur yang kedua titik ujungnya sama ($v_0 = v_n$) disebut **closed path**

Closed Path yang terbentuk dari simple path disebut **cycle**

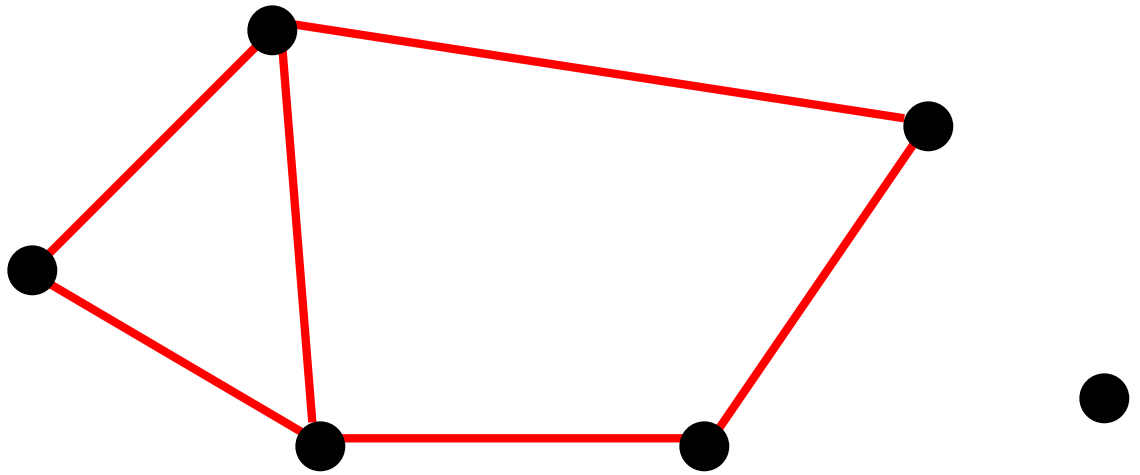
Graph **disebut terhubung (connected graph)** apabila dari sembarang titik yang ada bisa dibuat jalur ke titik yang lain

Jenis-Jenis Graph
- *Graph* tak berarah (*undirected graph)*

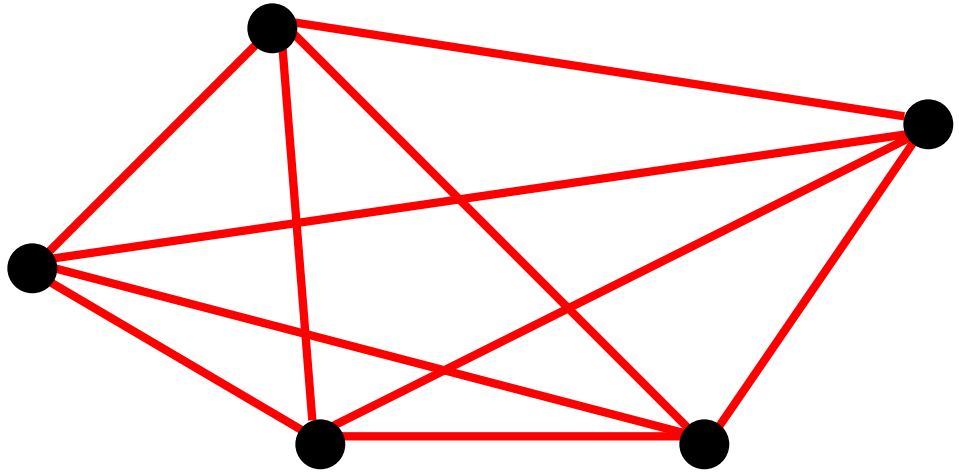
Urutan simpul dalam sebuah busur tidak dipentingkan. Edge (u,v) sama dengan edge(v,u)



Isolated graph

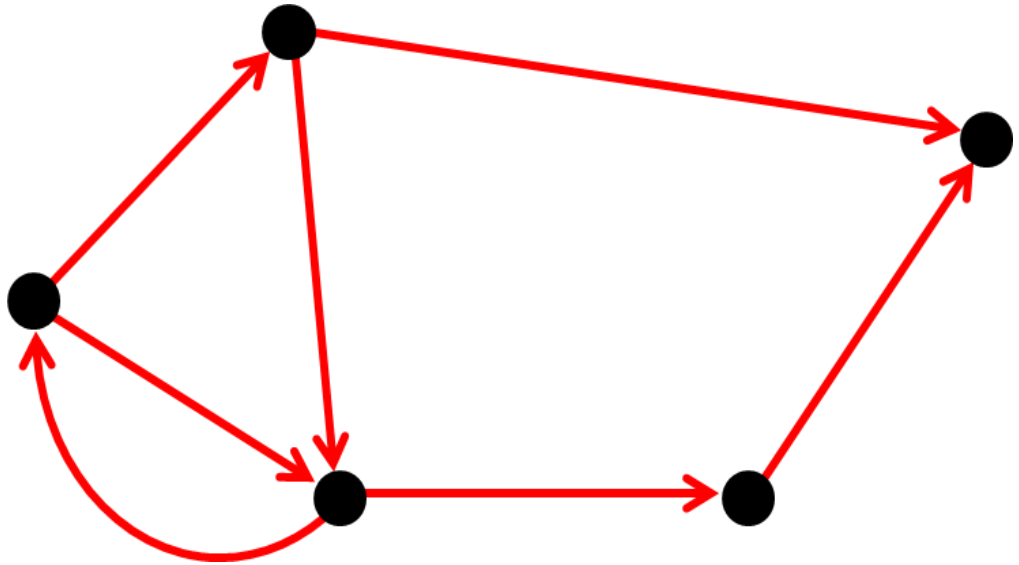


Completed graph



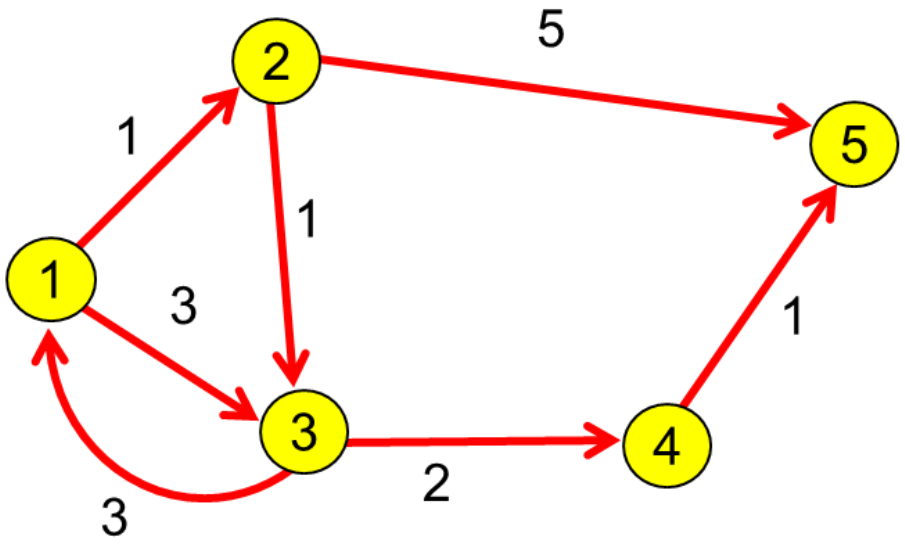
Jenis-Jenis Graph
- *Graph* berarah (*directed graph)*

urutan simpul mempunyai arti. Misal Edge (u,v) tidak sama dengan edge(v,u)

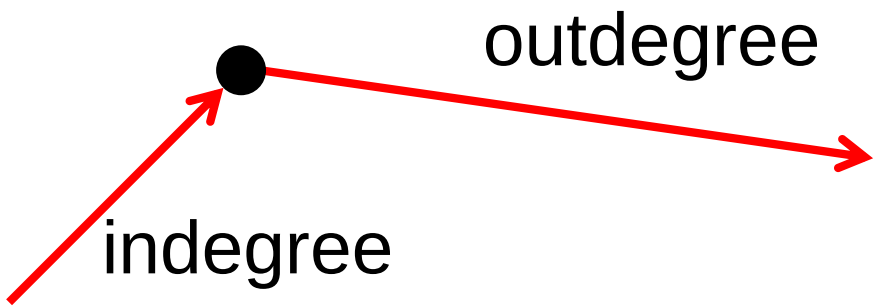


Jenis-Jenis Graph
Graph Berbobot (Weighted Graph)

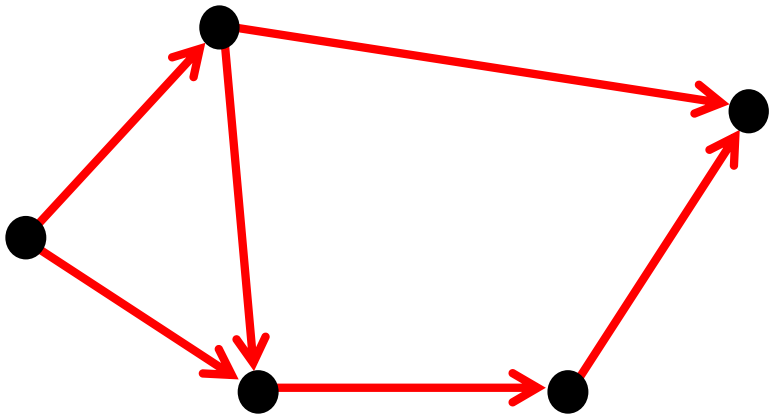
- Jika setiap busur mempunyai nilai yang menyatakan hubungan antara 2 buah simpul, maka busur tersebut dinyatakan memiliki bobot.
- Bobot sebuah busur dapat menyatakan panjang sebuah jalan dari 2 buah titik, jumlah rata-rata kendaraan perhari yang melalui sebuah jalan, dll.



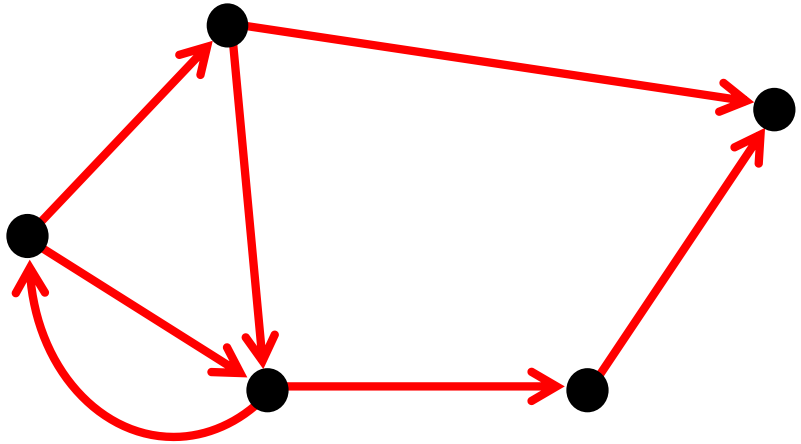
Directed graph



Directed graph



One-way traffic
(single path)

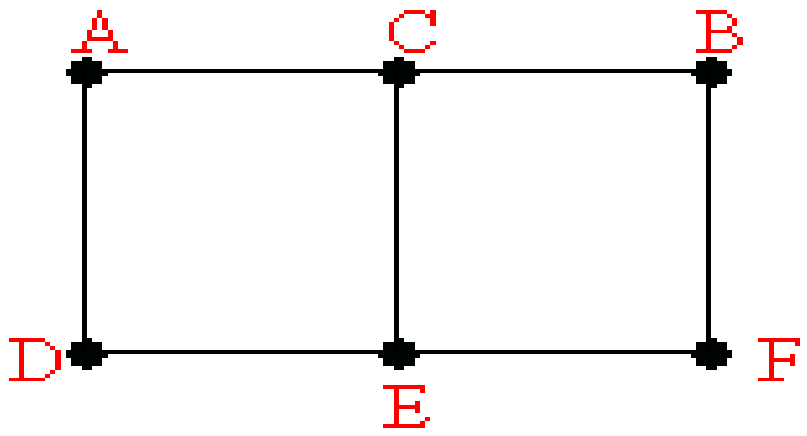


Two-way traffic
(multi path)

Representasi Graph

Adjacency Matrix

Adjacency Linked List



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

Adjacency Matrix

A	→	C	→	D		
B	→	C	→	F		
C	→	A	→	B	→	E
D	→	A	→	E		
E	→	C	→	D	→	F
F	→	B	→	E		

Adjacency Linked List

Algoritma *Shortest Path* : *Warshall*

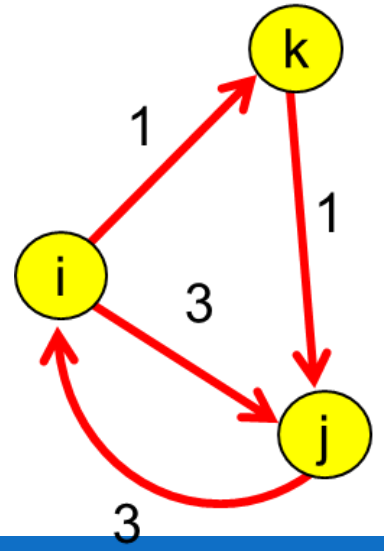
Algoritma Floyd-Warshall menghitung jarak terpendek (*shortest path*) untuk semua pasangan titik pada sebuah *graph*, dan melakukannya dalam waktu berorde kubik. Algoritma warshall digunakan untuk menyelesaikan permasalahan jalur terpendek *multi path*.

Algoritma melakukan pengecekan apakah beban langsung Q(i, j) memang lebih kecil daripada beban melalui titik perantara

Q(i,k)+Q(k,j)

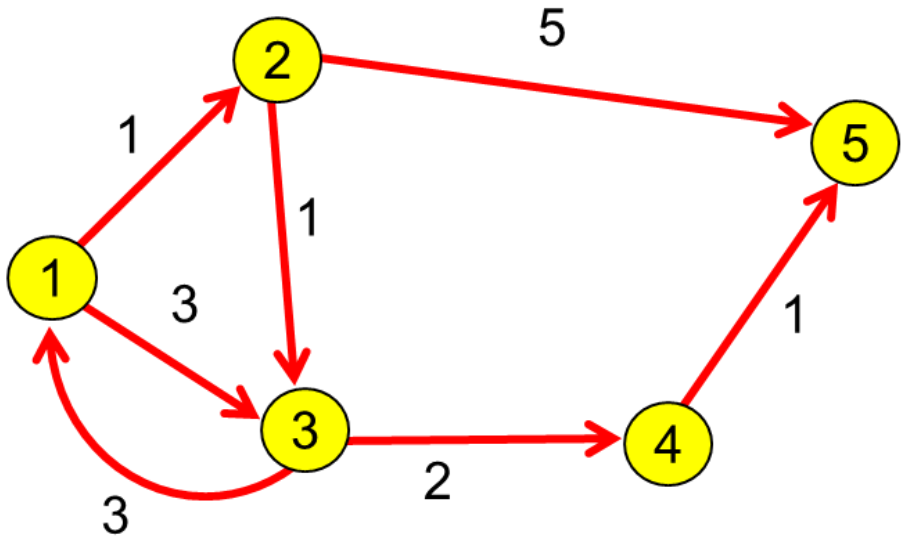
if ((Q(i,k)+Q(k,j))< Q(i, j))

$$Q(i, j) \Leftarrow Q(i,k)+Q(k,j)$$



Algoritma *Shortest Path* : *Warshall*

Representasi matriks beban di bawah ini menjadi array dua dimensi Q.



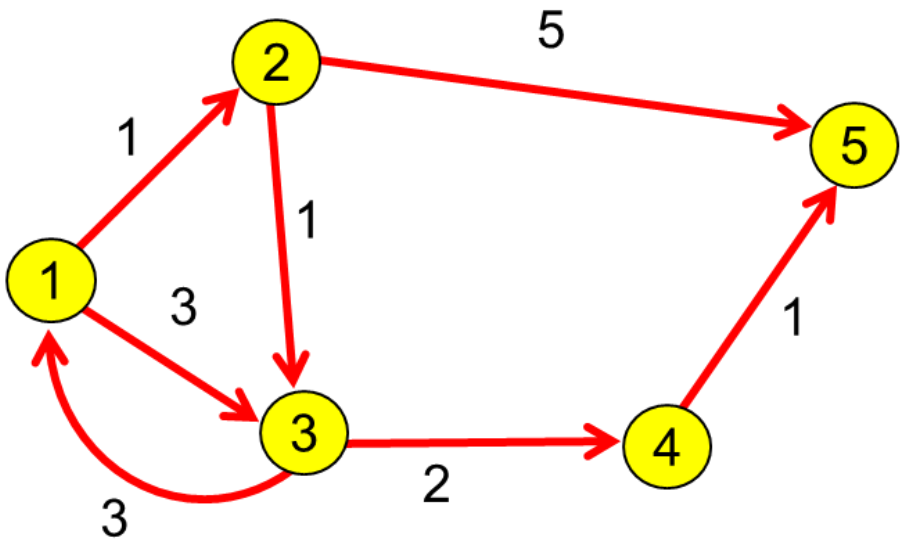
	1	2	3	4	5
1		1	3	-	-
2	-		1	-	5
3	3	-		2	-
4	-	-	-		1
5	-	-	-	-	

Q	1	2	3	4	5
1	M	1	3	M	M
2	M	M	1	M	5
3	3	M	M	2	M
4	M	M	M	M	1
5	M	M	M	M	M

Dimana M adalah big integer.

Algoritma *Shortest Path* : *Warshall*

Representasi matriks jalur di bawah ini menjadi array dua dimensi P



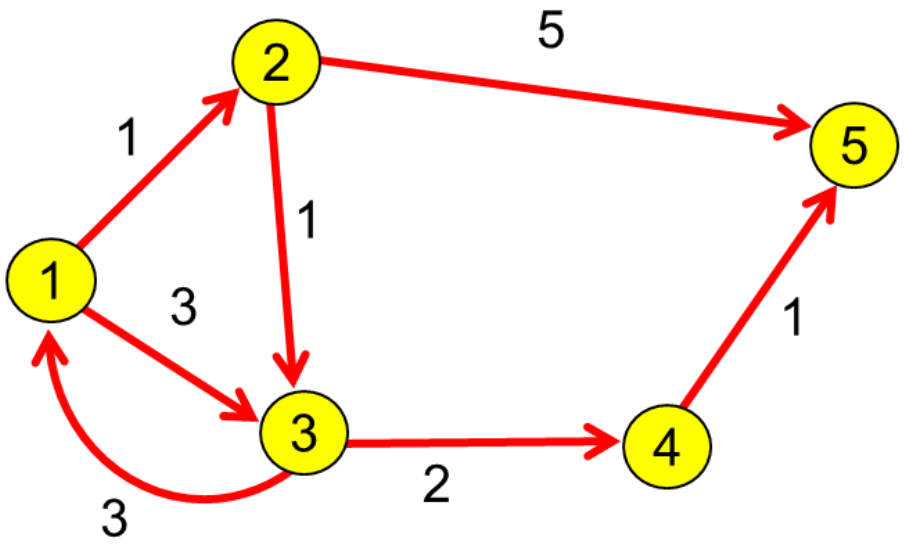
	1	2	3	4	5
1		√	√	-	-
2	-		√	-	√
3	√	-		√	-
4	-	-	-		√
5	-	-	-	-	

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

Algoritma *Shortest Path* : *Warshall*

Representasi matriks rute di bawah ini menjadi array dua dimensi

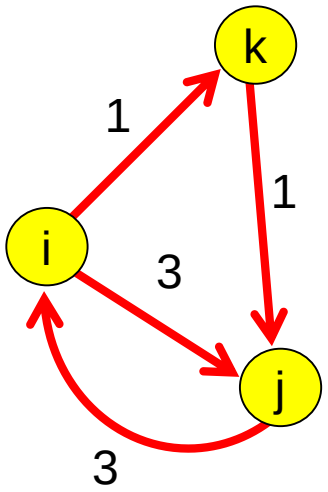
Dimana M adalah big integer



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

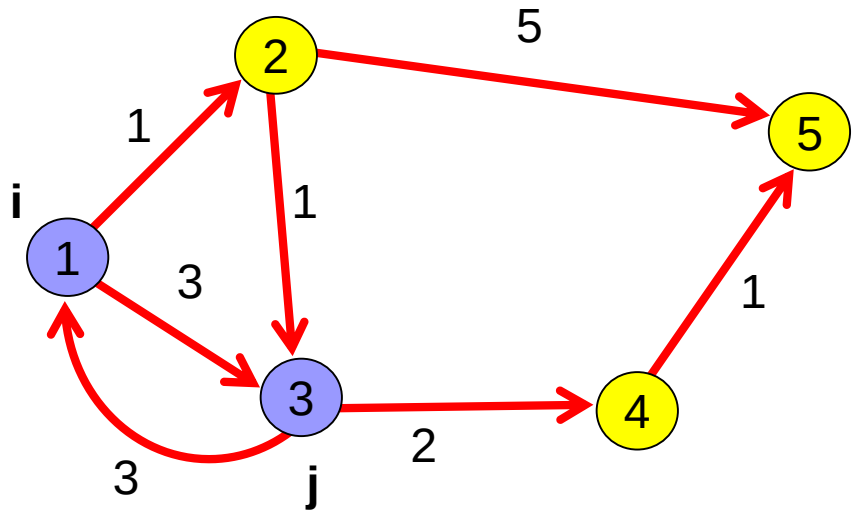
R	1	2	3	4	5
1	M	0	0	M	M
2	M	M	0	M	0
3	0	M	M	0	M
4	M	M	M	M	0
5	M	M	M	M	M

Shortest path problem Multi path (Algoritma Warshall)



Melakukan pengecekan apakah beban langsung Q(i, j) memang lebih kecil daripada beban melalui titik perantara Q(i,k)+Q(k,j)

if ((Q(i,k)+Q(k,j))< Q(i, j))
Q(i, j) ← Q(i,k)+Q(k,j)



Q(1,3) = 3 → Beban langsung

Q(1,1) + Q(1,3) = M+3
Q(1,2) + Q(2,3) = 2
Q(1,3) + Q(3,3) = 3+M
Q(1,4) + Q(4,3) = M+M
Q(1,5) + Q(5,3) = M+M

} Beban melalui perantara

Q(1,3) = 2

Algoritma warshall untuk beban

```
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            if ((Q(i,k) + Q(k,j)) < Q(i,j))
               Q(i,j) ⇐ (Q(i,k)+Q(k,j)
```

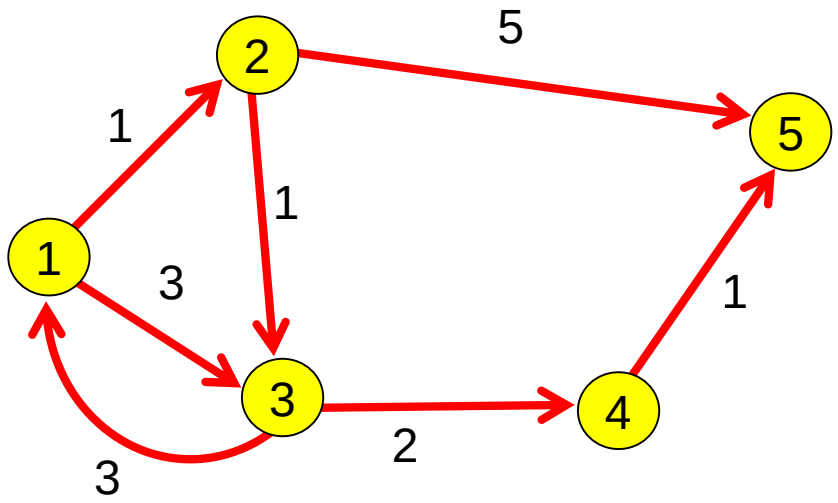
Algortitma warshall untuk jalur

```
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            P(i,j) ⇐ P(i,j) OR (P(i,k) AND P(k,j))
```

Algoritma warshall untuk rute

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
        if ((Q(i,k) + Q(k,j)) < Q(i,j))
            if (R(k,j) = 0)
                R(i,j) ⇐ k
                  else
                R(i,j)=R(k,j)
```

Cara membaca matriks rute



R	1	2	3	4	5
1	3	0	2	3	4
2	3	1	0	3	4
3	0	1	2	0	4
4	M	M	M	M	0
5	M	M	M	M	M

Rute 1-5 ?

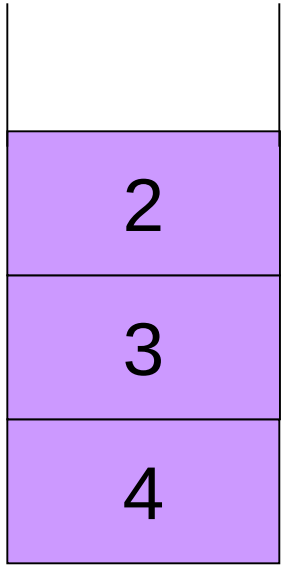
Rute 1-5?

Ambil nilai di baris 1, kolom 5 = 4 ➔ push

Ambil nilai di baris 1, kolom 4 = 3 ➔ push

Ambil nilai di baris 1, kolom 3 = 2 ➔ push

Ambil nilai di baris 1, kolom 2 = 0 (stop) ➔ pop



Rute =
1 – 2 – 3 – 4 - 5

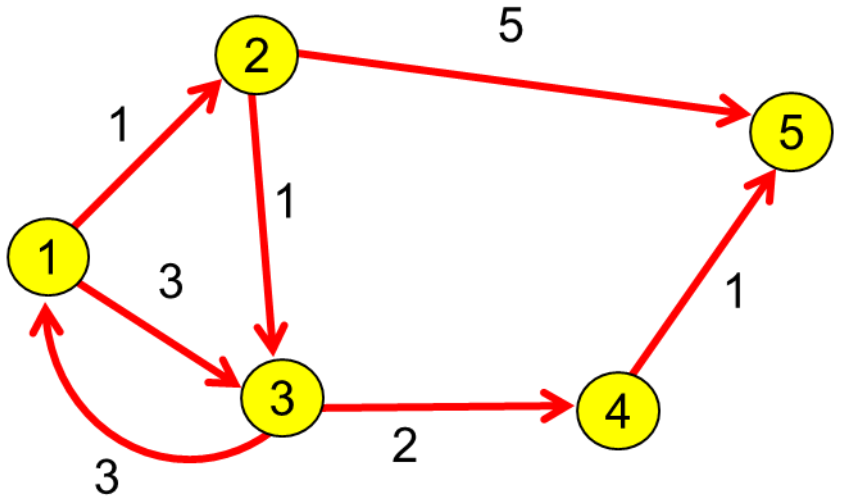
Algoritma Shortest Path : Djikstra

- Penemunya adalah Edsger Dijkstra
- Algoritma dengan prinsip greedy yang memecahkan masalah lintasan terpendek untuk sebuah graf berarah dengan bobot sisi yang tidak negatif.

Algoritma Shortest Path : Djikstra

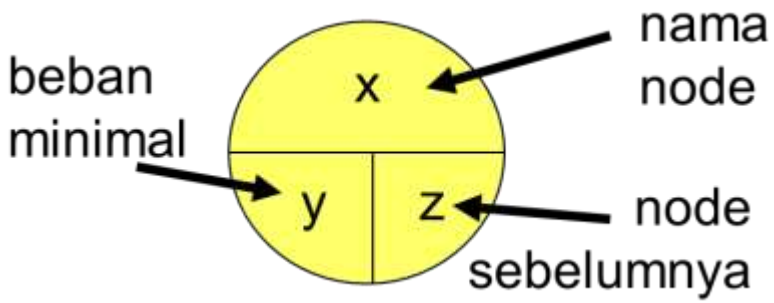
Langkah-langkah algoritma Djikstra adalah sebagai berikut :

- Tentukan titik asal dan titik tujuan sebelum proses
- Akumulasikan jarak minimal dan simpan ke titik berikutnya.
- Lakukan dari titik asal sampai titik tujuan



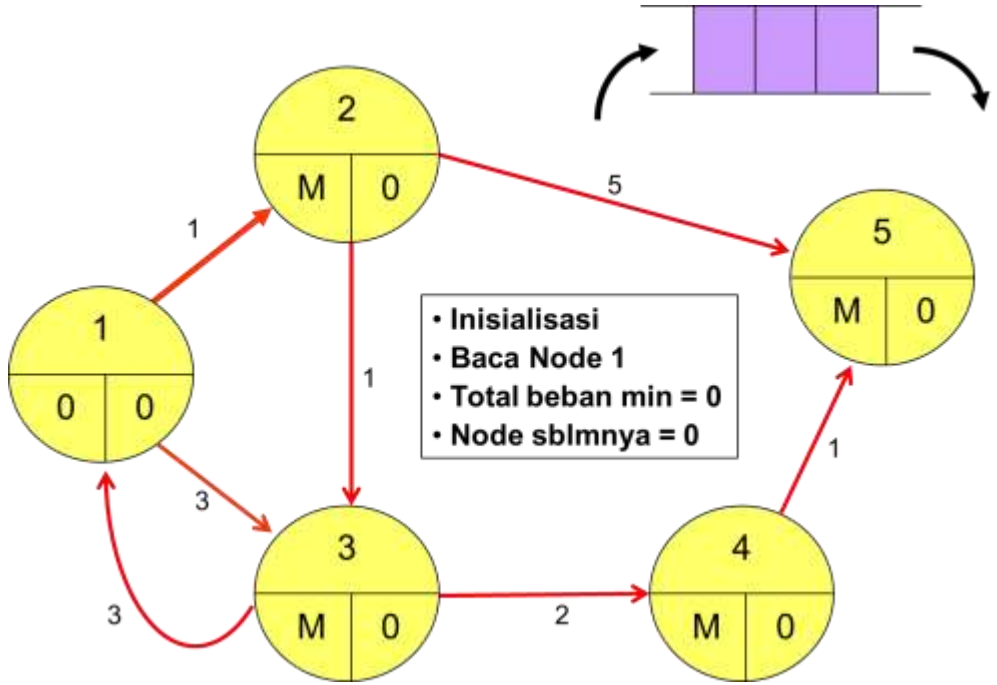
Langkah-Langkah (1)

Tentukan titik asal = 1 dan titik tujuan = 5. Kita tentukan setiap node mempunyai nilai x sebagai nama node, nilai y sebagai beban minimal dan nilai z sebagai node sebelumnya seperti Gambar dibawah. Nilai y dan z disimpan sebagai vektor.



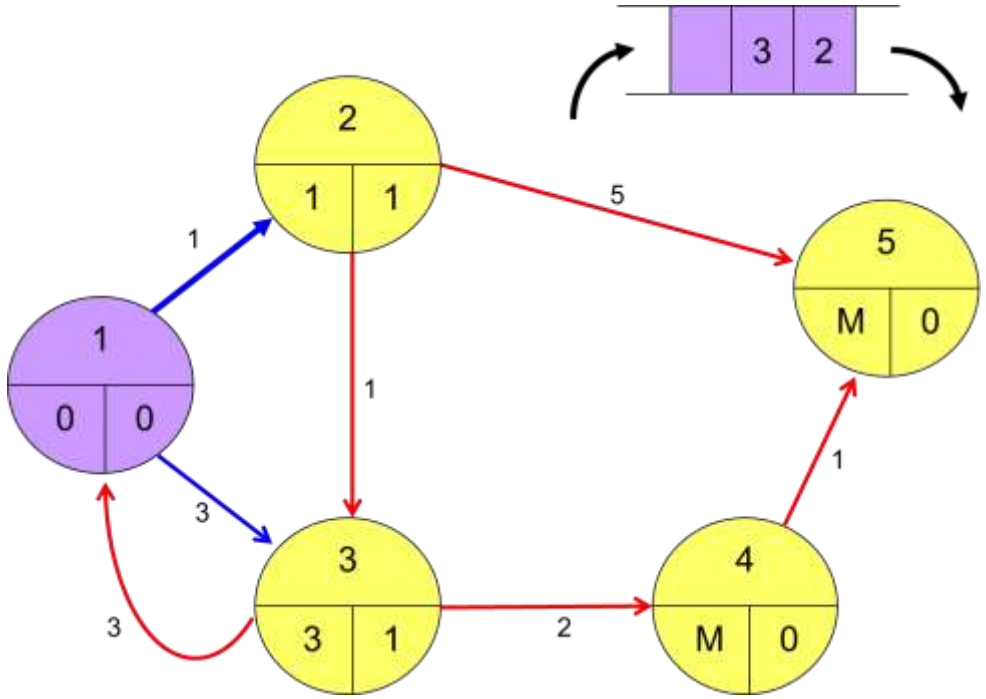
Langkah-Langkah (2)

Siapkan antrian untuk menyimpan node yang akan diproses. Sebagai inisialisasi awal, nilai y bernilai M (big integer) kecuali node awal bernilai 0, sedangkan nilai z bernilai 0. Node titik awal disimpan dalam antrian, dan baca nilai tersebut.



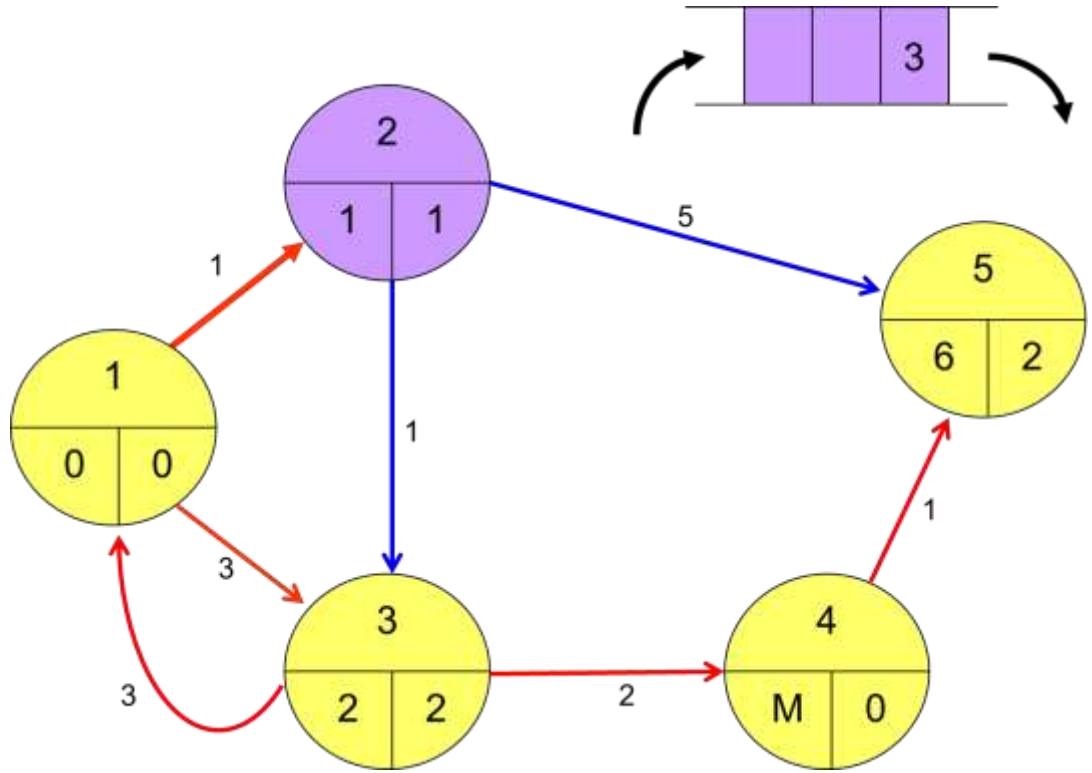
Langkah-Langkah (3)

Baca node 1 dan lakukan perubahan nilai y dan z menjadi nilai beban dan node sebelumnya yang terkecil. Masukkan node z pada antrian jika node tersebut tidak ada di antrian, bukan titik awal dan bukan titik akhir.



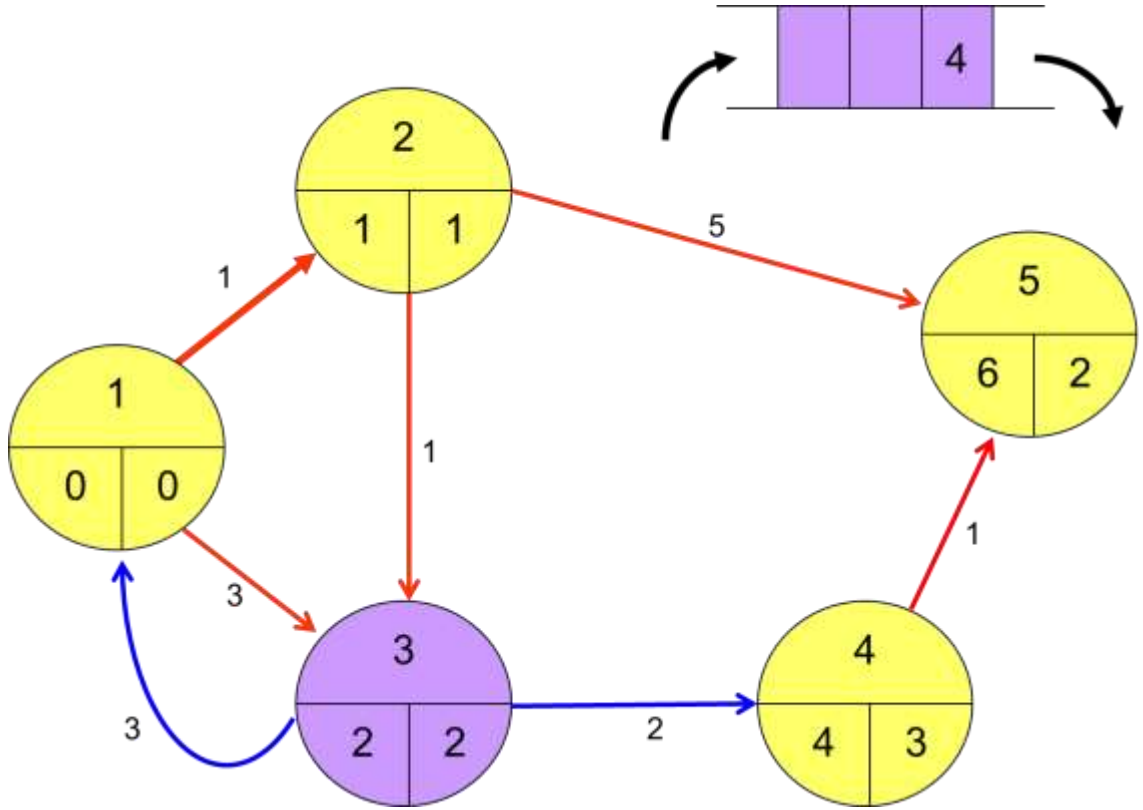
Langkah-Langkah (4)

Enqueue node pada antrian, baca node 2 dan lakukan hal yang sama dengan langkah 3.



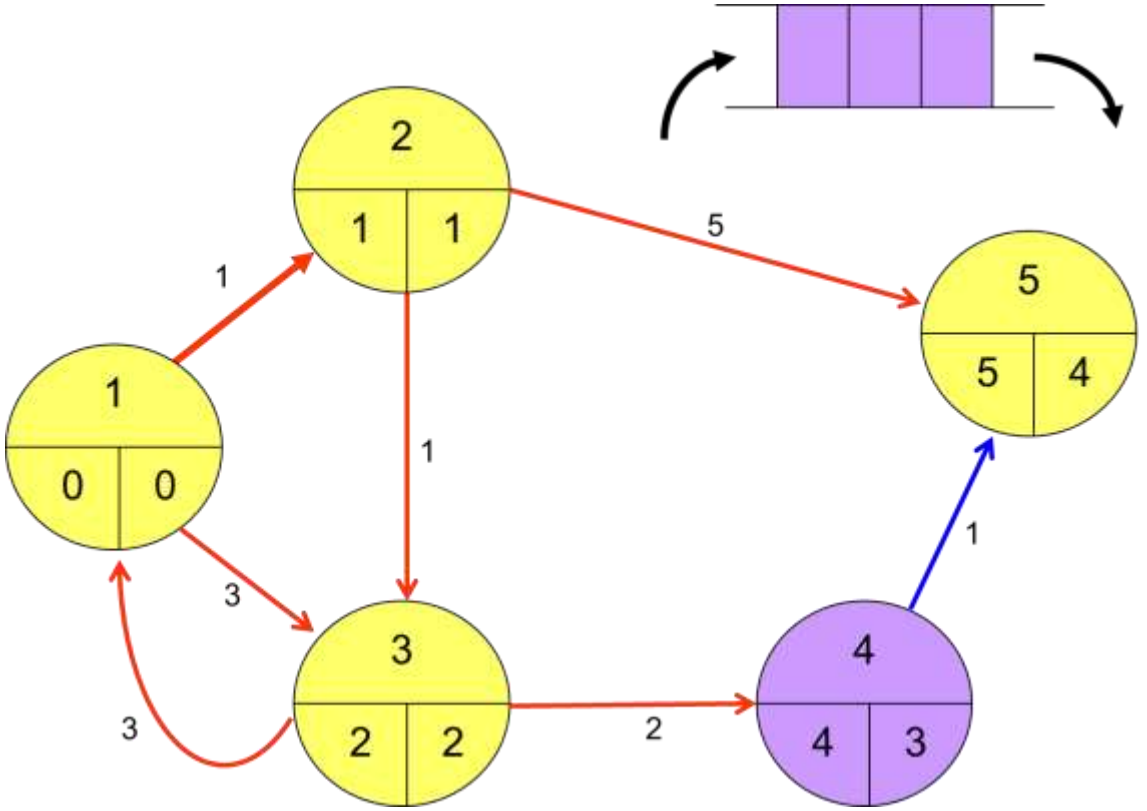
Langkah-Langkah (5)

Enqueue node pada antrian, baca node 3 dan lakukan hal yang sama dengan langkah 3.



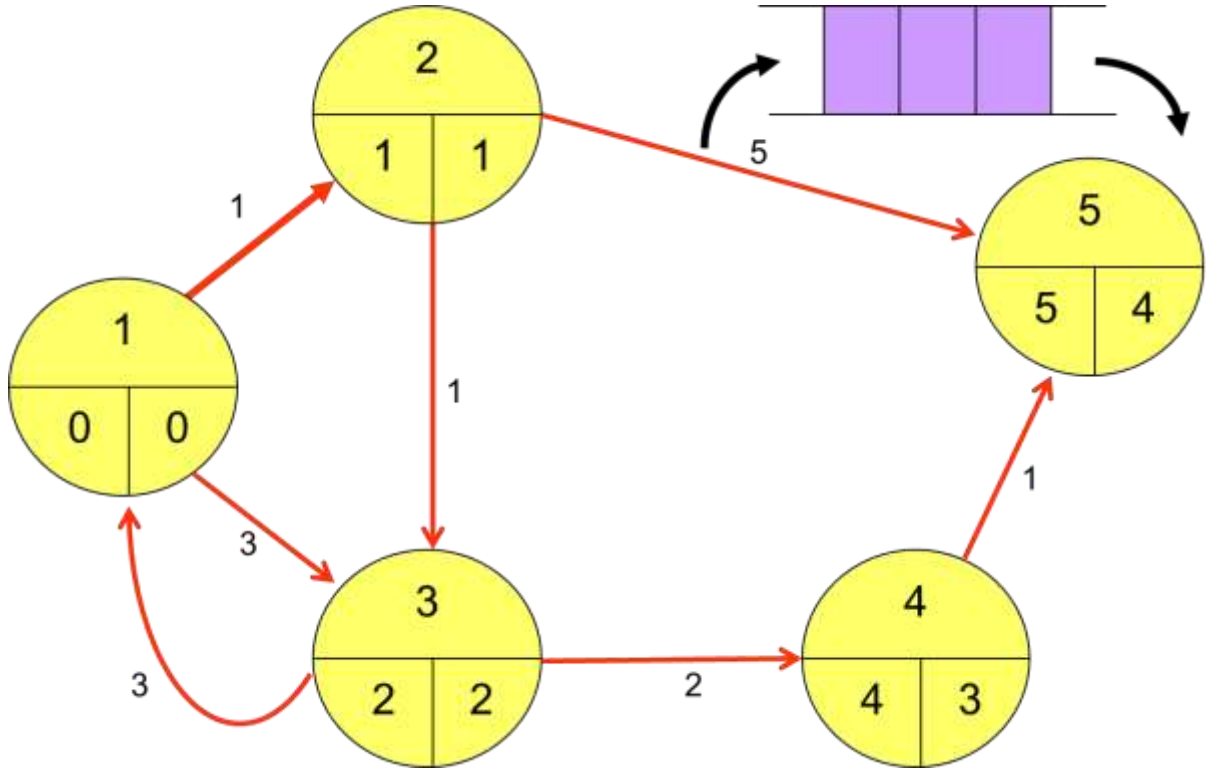
Langkah-Langkah (6)

Enqueue node pada antrian, baca node 4 dan lakukan hal yang sama dengan langkah 3.



Langkah-Langkah (7)

Bila antrian kosong, proses selesai sehingga menghasilkan vektor beban dan rute minimal.



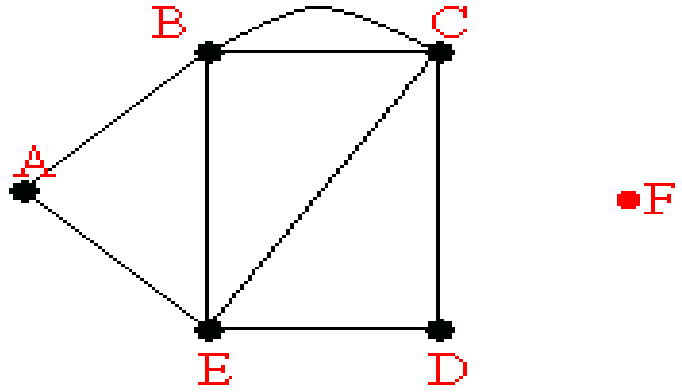
Algoritma

1. Tentukan input[n][n] -> beban langsung
2. Tentukan titik_asal dan titik_tujuan
3. Inisialisasi Q[n] (beban minimal) dan R[n] (node sebelumnya)
4. Enqueue (titik_asal)
5. Selama(queue tidak kosong) lakukan 6-15
6. CN=Dequeue
7. i=1;
8. Selama i<=n lakukan 9-15
9. Jika input[CN][i] != M lakukan 10-14
10. Jika Q[CN] + input[CN][i] < Q[i] lakukan 11-12
11. Q[i] = Q[CN] + input[CN][i]
12. R[i]=CN
13. Jika i != titik_asal dan i != titik_tujuan dan i tidak ada di queue lakukan 14
14. Enqueue (i)
15. i++

Kesimpulan

Banyak sekali struktur yang bisa direpresentasikan dengan *graph*, dan banyak masalah yang bisa diselesaikan dengan bantuan *graph*. Seringkali *graph* digunakan untuk merepresentasikan suatu jaringan.

Misalkan jaringan jalan raya dimodelkan *graph* dengan kota sebagai simpul (*vertex/node*) dan jalan yang menghubungkan setiap kotanya sebagai sisi (*edge*) yang bobotnya (*weight*) adalah panjang dari jalan tersebut.



Latihan Soal

Berdasarkan *graph* di bawah ini, representasikan matriks, gunakan algoritma warshall untuk mencari rute terpendek dan rute seperti pada Latihan 1.

Pada graph di bawah ini, selesaikan dengan algoritma djikstra. Jalankan program dan analisa hasilnya.