M.Bahrul Ulum, S.Kom, M.Kom



GRAPH

Struktur Data

**GRAPH**

Program Studi Teknik Informatika

Fakultas Ilmu Komputer

Universitas Esa Unggul

2018

**DASAR TEORI**

Graph adalah kumpulan node (simpul) di dalam bidang dua dimensi yang  
dihubungkan dengan sekumpulan garis (sisi). Graph dapat digunakan untuk  
merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut.  
Representasi visual dari graph adalah dengan menyatakan objek sebagai node, bulatan atau titik (Vertex), sedangkan hubungan antara objek dinyatakan dengan garis (Edge).

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 suaru jaringan. Misalkan jaringan jalan raya dimodelkan graphdengan kota sebagai simpul (vertex/node) dan jalan yang menghubungkan setiap  
kotanya sebagai sisi (edge) yang bobotnya (weight) adalah panjang dari jalan tersebut. Ada beberapa cara untuk menyimpan graph di dalam sistem komputer.  
Struktur data bergantung pada struktur graph dan algoritma yang digunakan untuk  
memanipulasi graph. Secara teori salah satu dari keduanya dapat dibedakan antara struktur linked list dan matriks (array dimensi 2), tetapi dalam penggunaannya struktur terbaik yang sering digunakan adalah kombinasi keduanya

**GRAPH** merupakan suatu koleksi dari himpunan VG dan EG.

Notasi : G = { VG, EG }

G = Graph

VG = Himpunan titik

EG = HImpunan garis

**Titik : Node / Vertex**

**Garis : Arc / Edge**

Contoh : Graph G terdiri dari : G = { VG, EG }

VG = { a,b,c,d }

EG = { 1,2,3,4,5,6,7,8 }



***Gambar 1. Graph***

Pada Gambar 1, edge 1 menghubungkan vertex a & b Edge 1 = (a,b)

edge 2 menghubungkan vertex b & c Edge 2 = (b,c)

edge 3 menghubungkan vertex b & c Edge 3 = (b,c), dst ...

Jumlah vertex dalam suatu graph disebut **ORDER** dari graph tersebut.

Contoh : Graph G dengan order = 4 (jumlah vertex = 4 ; a,b,c,d)

Suatu graph hanya ditentukan oleh vertex-vertex dan edge-edgenya. Posisi dari vertex-vertex dan edge-edge dalam penggambaran tidaklah penting.

**GRAPH EQUIVALEN** : penggambaran graph yang sama.



***Gambar 2***

***GRAPH EQUIVALEN***

Suatu graph G disebut **SIMPLE GRAPH,** jika :

1. tidak mempunyai edge yang **SELF LOOP** (tidak ada (V,V) dalam G)

2. tidak mempunyai **MULTIPLE EDGE** (hanya ada (Vi, Vj) dalam G)

(V1, V2, V3, ...... VG)



***Gambar 3***

***SIMPLE GRAPH***

**MULTIPLE EDGE** adalah 1 vertex dihubungkan oleh beberapa edge.

Contoh : Pada Gambar 2 ; vertex b dihubungkan oleh edge-edge 1,2,3,5,6,7

vertex c dihubungkan oleh edge-edge 2,3,4

**SELF LOOP** adalah vertex yang dihubungkan oleh edge-edge menuju edge itu sendiri.

Contoh : Vertex a dihubungkan oleh edge 8 menuju a kembali (Gambar 2)

Suatu Graph G disebut **CONNECTED** (terhubung) jika Graph G dapat dipartisi (dibagi) menjadi 2 graph dengan menghapus paling sedikit 1 edge.

Contoh yang tidak connected :

Suatu graph G terdiri dari : G = { VG, EG }

VG = { e,f,g,h }

EG = { 1,2,3 }



***Gambar 4***

***UNCONNECTED GRAPH***

**PATH** dalam Graph adalah barisan dari 1 buah edge-edge yang menghubungkan 2 vertex.

Notasi : P(Vi, Vj) = (Vi, X1) (X1, X2) (X2, Xn-1) (Xn-1, Xn) (Xn, Vj)

Dari Gambar 3: P(b,d) = (b,c) (c,d)

P(b,d) = (b,c) (c,b) (b,c) (c,d)

P(b,d) = (b,d)

P(b,d) = (b,c) (c,b) (b,d)

**LENGTH** dari suatu path adalah jumlah edge-edge pada path tersebut.

Contoh : Perhatikan Gambar 3 : P(b,d) = (b,d) length = 1

= (b,c) (c,d) length = 2

= (b,c) (c,b) (b,d) length = 3

**CYCLE** adalah path yang memenuhi syarat sebagai berikut :

1. tidak ada edge yang tampil lebih dari satu kali dalam barisan edge dari path tersebut

contoh : Gambar 3 ; P(b,d) = (b,c) (c,b) (b,d)

⇒ tidak boleh

2. path harus berbentuk P(V,V)

3. tidak ada vertex yang dikunjungi lebih dari satu kali

contoh : P(a,a) = (a,b) (b,c) (c,d) (d,b) (b,a)

b dikunjungi lebih dari 1x

P(b,b) = (b,c) (c,b) (b,a) (a,c) (c,b)

c & b dikunjungi 3x

Contoh CYCLE : P(b,b) = (b,d) (d,c) (c,b)

**ACYCLIC** adalah graph yang tidak mempunyai cycle.

Contoh : Graph G terdiri dari : G = { VG, EG }

VG = { e,f,g,h }

EG = { 1,2,3 }



***Gambar 5***

***ACYCLIC GRAPH***

Graph yang Simple belum tentu graph yang Acyclic.

Graph yang Acyclic adalah graph yang Simple.

Graph yang berarah disebut **DI-GRAPH / DIRECTED GRAPH**, adalah merupakan graph dimana edge-edgenya mempunyai suatu arah.



***Gambar 6***

Pada Gambar 6 ; (a,b) 1 arah

(b,a) 0 arah

Graph yang tidak mempunyai arah boleh bolak-balik.



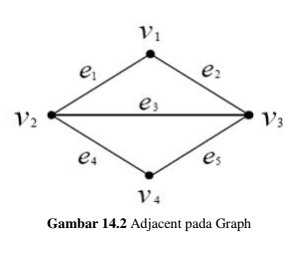
***Gambar 7***

Pada Gambar 7 ; (a,b) 1 arah

(b,a) 1 arah

**Istilah Pada *Graph***

Terdapat beberapa istilah yang berkaitan dengan *graph* yaitu:  
*a. Vertex*Adalah himpunan node / titik pada sebuah *graph*.  
*b. Edge*Adalah himpunan garis yang menghubungkan tiap *node* / *vertex*.  
*c. Adjacent*Adalah dua buah titik dikatakan berdekatan (*adjacent*) jika dua buah titik tersebut  
terhubung dengan sebuah sisi. Adalah Sisi *e3 = v2v3* insident dengan titik *v2* dan  
titik *v3*, tetapi sisi *e3 = v2v3* tidak insident dengan titik *v1* dan titik *v4*. Titik *v1*adjacent dengan titik *v2* dan titik *v3*, tetapi titik *v1* tidak adjacent dengan titik *v4*.



d. WeightAdalah Sebuah graph G = (V, E) disebut sebuah graph berbobot (weight graph),  
apabila terdapat sebuah fungsi bobot bernilai real W pada himpunan E,  
W : E → R (14.3)  
nilai W(e) disebut bobot untuk sisi e, ∀ e ∈ E. Graf berbobot tersebut dinyatakan  
pula sebagai G = (V, E, W).Graf berbobot G = (V, E, W) dapat menyatakan  
• suatu sistem perhubungan udara, di mana  
 V = himpunan kota-kota  
 E = himpunan penerbangan langsung dari satu kota ke kota lain  
 W = fungsi bernilai real pada E yang menyatakan jarak atau ongkos atau  
waktu  
• suatu sistem jaringan komputer, di mana  
 V = himpunan komputer  
 E = himpunan jalur komunikasi langsung antar dua komputer  
 W = fungsi bernilai real pada E yang menyatakan jarak atau ongkos atau  
waktu  
e. PathAdalah jalur dengan setiap vertex berbeda. Contoh, P = D5B4C2A Sebuah jalur  
(W) didefinisikan sebagai urutan (tidak nol) vertex dan edge. Diawali originvertex (vertex awal) dan diakhiri terminus vertex (vertex akhir). Dan setiap 2  
garis berurutan adalah series. Contoh, W = A1B3C4B1A2.

f. CycleAdalah Siklus (Cycle) atau Sirkuit (Circuit) yaitu lintasan yang berawal dan  
berakhir pada simpul yang sama .

|  |
| --- |
| **OUT DEGREE, IN DEGREE, DEGREE dari suatu vertex a** |

Vertex a mempunyai :

1. **OUT DEGREE** (derajat luar) = N

Jika vertex a mempunyai N edge mengarah keluar.

Misal : Vertex a mempunyai 2 edge mengarah ke luar (Gambar 6.)

2. **IN DEGREE** (derajat masuk) = N

Jika vertex a mempunyai N edge mengarah masuk (Gambar 6.)

3. **DEGREE** (derajat) = N

Jika Out Degree a ditambah In Degree a = N

misal : vertex b : In Degree = 2

Out Degree = 3

-----------------------

Degree = 5

Contoh : Pada Gambar 6 ; degree(a) = 3

degree(b) = 5

degree(c) = 3

degree(d) = 5

------------------------

16

Graph G dengan himpunan vertex Vo dan edge Eo diasumsikan graph berorder N untuk N 1.

Salah satu pendekatan untuk graph ini menggunakan matriks **ADJACENCY** dengan suatu array A ukuran N x N.

A(i, j)

⎨

1 jika edge (Vi, Vj) Eij

0 jika edge (Vi, Vj) Eij

Contoh : Graph **UNDIRECT** / Matriks Simetris



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 1 | 1 | 1 |
| 4 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 0 | 0 | 0 |
| 6 | 0 | 0 | 1 | 0 | 0 | 0 |

Contoh : Graph **DIRECT**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 1 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |
| --- |
| **Penggambaran NODE DIRECTORY** |

Penggambaran node dalam directory dibagi dalam 2 bagian :

1. Directory

2. Himpunan Link List (LL)

Setiap record dari Link List mempunyai 2 field ;

1. Node Identifier

2. Suatu link yang menghubungkan elemen lain dari list (next)

|  |  |
| --- | --- |
| **NODE** | **NEXT** |

Directory menggambarkan banyak node. Link list menunjukkan edge-edgenya.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Directory** |  | **Link List** |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. |  |  | 2 | null |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2. |  |  | 1 |  |  | 2 |  |  | 3 | null |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3. |  |  | 2 |  |  | 4 |  |  | 5 |  |  | 6 | null |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4. |  |  | 3 | null |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5. |  |  | 3 | null |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6. |  |  | 3 | null |  |  |  |  |  |  |  |  |  |



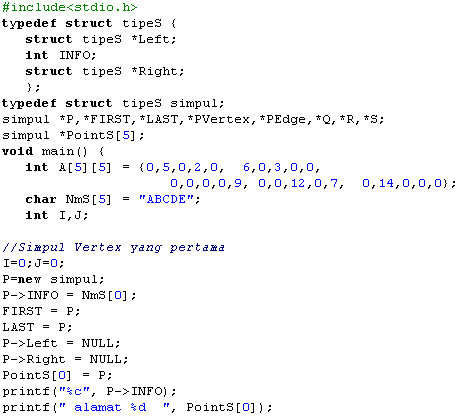
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Directory** |  | **Link List** |  |  |

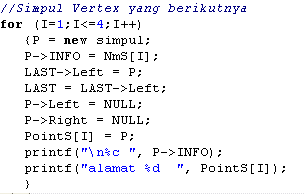
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. |  |  | 2 | null |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| 2. |  |  | 2 |  |  | 3 | null |  |
|  |  |  |  |  |  |  |  |  |
| 3. |  |  | 4 |  |  | 5 | null |  |
|  |  |  |  |  |  |  |  |  |
| 4. | null |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| 5. | null |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| 6. |  |  | 3 | null |  |  |  |  |

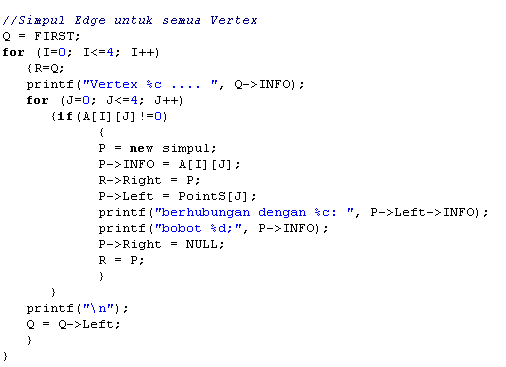
Kalau punya harga (untuk manajemen proyek) penggambaran node-nya di bagi3

|  |  |  |
| --- | --- | --- |
| NODE | WEIGHT | NEXT |

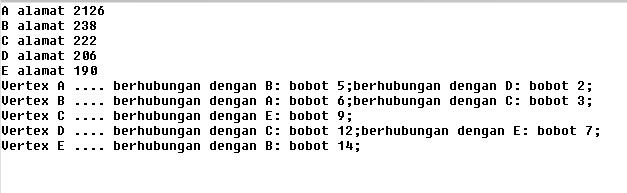






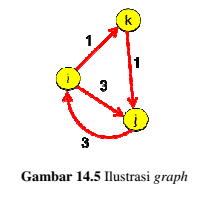


Hasil



**Algoritma Warshall**

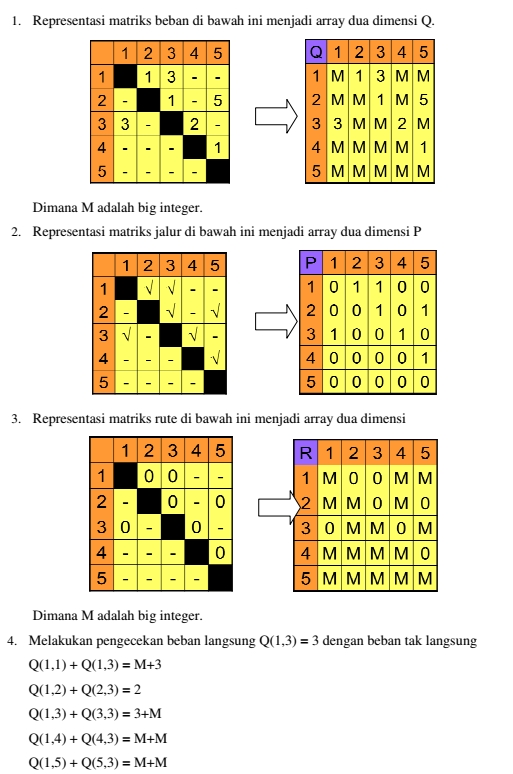
Algoritma Floyd-Warshall menghitung jarak terpendek (*shortest path*) untuk  
semua pasangan titik pada sebuah *graph*, dan melakukannya dalam waktu berorde kubik. Algoritma warshall digunakan untuk menyelesaikan permasalahan jalur terpendek *multi path*. Algoritma Floyd-Warshall memiliki input *graph* berarah dan berbobot (V,E), yang berupa daftar titik (*node/vertex* V) dan daftar sisi (e*dge* E). Jumlah bobot sisi-sisi pada sebuah jalur adalah bobot jalur tersebut. Sisi pada E diperbolehkan memiliki bobot negatif, akan tetapi tidak diperbolehkan bagi *graph* ini untuk memiliki siklus dengan bobot negatif. Algoritma ini menghitung bobot terkecil dari semua jalur yang menghubungkan sebuah pasangan titik, dan melakukannya sekaligus untuk semua pasangan titik.



Sebagai ilustrasi pada Gambar 14.5, 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) ⇐ Q(i,k)+Q(k,j)

Misalkan pada contoh kasus Gambar 14.4 di atas, langkah-langkah penyelesaian  
dengan algoritma Warshall sebagai berikut:

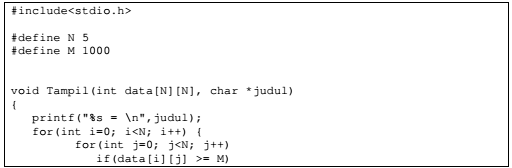


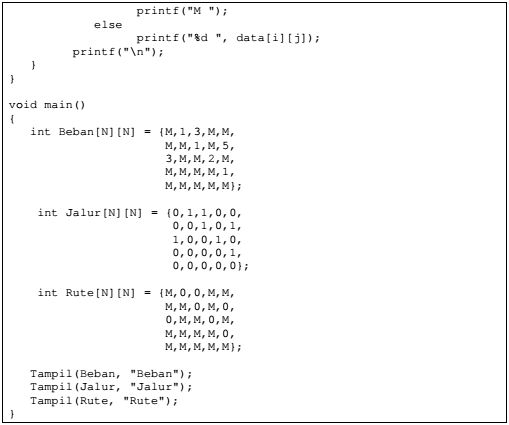
Sehingga Beban terkecil adalah Q(1,3) = 2  
Algoritma warshall untuk beban adalah sebagai berikut :  
for k = 1 to n  
for i = 1 to n  
for j = 1 to n  
if ((Q(i,k) + Q(k,j)) < Q(i,j))  
Q(i,j) ⇐ (Q(i,k)+Q(k,j)  
Algortitma warshall untuk jalur adalah sebagai berikut :  
for k = 1 to n  
for i = 1 to n  
for j = 1 to n  
P(i,j) ⇐ P(i,j) OR (P(i,k) AND P(k,j))  
Algoritma warshall untuk rute adalah sebagai berikut :  
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)

**TUGAS PENDAHULUAN**Jawablah pertanyaan berikut ini :  
1. Jelaskan representasi graph dengan matriks untuk Beban, Jalur dan Rute  
2. Tuliskan algoritma Warshall

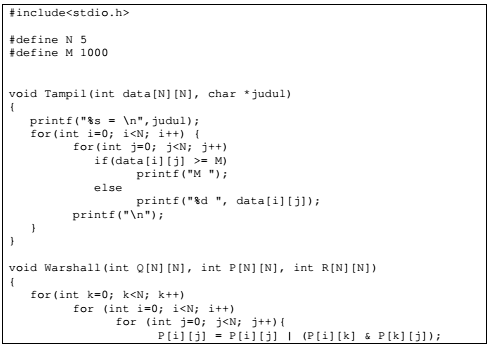
**PERCOBAAN**

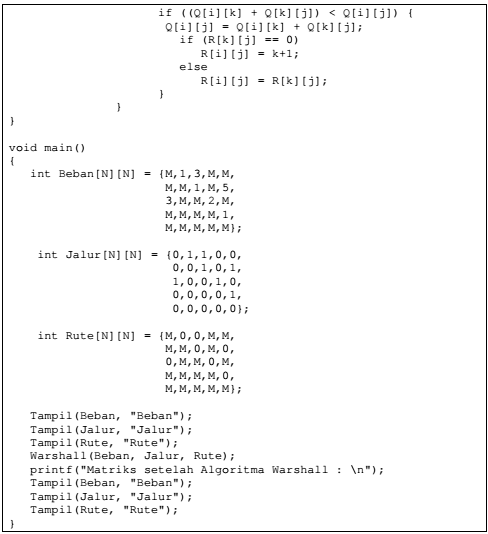
Percobaan 1 : Mendeklarasikan matriks Beban, Jalur dan Rute





Percobaan 2 : Algoritma Warshall untuk pencarian jalur terpendek multipath





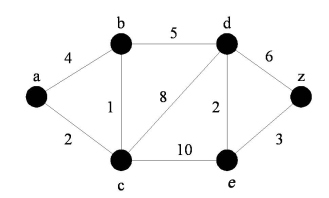
**LATIHAN**

1. Dari percobaan 2 diatas, tambahkan fungsi untuk menampilkan rute dari satu titik ke titik berikutnya berdasarkan matriks rute. Input berupa titik awal dan titik akhir. Gunakan struktur data Stack untuk mencari rute seperti langkah-langkah di bawah ini.

1. Rute 1-5?  
2. Ambil nilai di baris 1, kolom 5 = 4 → push  
3. Ambil nilai di baris 1, kolom 4 = 3 → push  
4. Ambil nilai di baris 1, kolom 3 = 2 → push  
5. Ambil nilai di baris 1, kolom 2 = 0 (stop) → pop sampai stack kosong  
6. Sehingga rutenya menjadi 1-2-3-4-5 dengan beban minimal 5.

Beban minimal diperoleh dari matriks beban dengan nilai baris, kolom sama dengan 1,5. Lakukan hal yang sama untuk input titik awal dan titik akhir yang lain.

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



1. Rancanglah sendiri sebuah graph dan lakukan hal yang sama dengan Latihan 2