MODUL 5
GRAPH

5.1. TUJUAN

Tujuan modul ini, adalah:

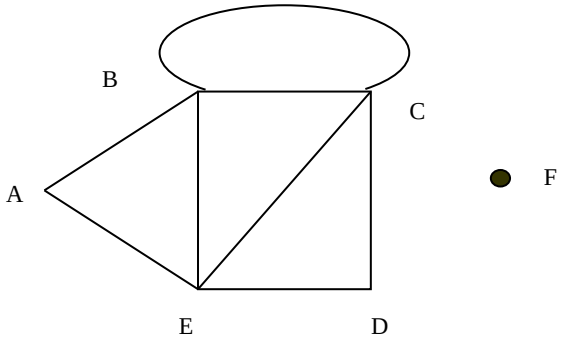
- Memperkenalkan penggunaan perangkat lunak bahasa pemrograman untuk mengimplementasikan struktur data (tipe data abstrak *graph*).
- Memperkenalkan cara pembuatan program yang berhubungan dengan konsep *graph*.
- Melatih mahasiswa agar mampu menyelesaikan masalah tentang *graph* dalam bentuk program aplikasi.

5.2. TEORI GRAPH (MATERI PERTEMUAN KESEBELAS)

Pada modul ini kita akan mempelajari struktur data yang disebut dengan *graph*, yang bisa dikatakan merupakan bentuk pohon yang lebih umum. *Graph* bisa dibayangkan sebagai kumpulan obyek atau aktifitas. Sebagai contoh, rute bis kota dari satu terminal ke terminal lain, rute perjalanan pak pos pada saat ia mengantar surat dari satu rumah ke rumah lain, dan contoh-contoh yang lain bisa disajikan sebagai suatu *graph*. Contoh kedua di atas merupakan contoh klasik dari teori *graph* yang lebih dikenal dengan persoalan *travelling salesman problem* atau *shortest path problem*, yang pada prinsipnya mencari jalur terpendek dari semua tempat yang harus dipenuhi, sehingga bisa menghemat waktu, tenaga, maupun biaya.

Terminologi

Graph secara umum bisa didefinisikan sebagai kumpulan titik (*nodes* atau *vertices*) dan garis (*arcs* atau *edges*). Karena garis selalu diawali pada suatu titik dan diakhiri pada titik yang lain, maka garis bisa dituliskan sebagai pasangan antara dua titik. Dalam notasi *graph*, garis dituliskan sebagai $e = [u,v]$, yang berarti bahwa garis *e* berawal pada titik *u* dan berakhir pada titik *v*. dua buah titik yang menghubungkan sebuah garis dikatakan sebagai titik yang bertetangga. Meskipun demikian, tidak harus semua titik yang ada pada sebuah *graph* dipasangkan dengan titik lain untuk membentuk garis. Titik-titik yang tidak dihubungkan dengan titik lain disebut dengan titik terisolir (*isolated node*). Gambar 5.1 menunjukan contoh sebuah *graph* yang mempunyai 6 buah titik.



Gambar 5.1. Contoh *graph*

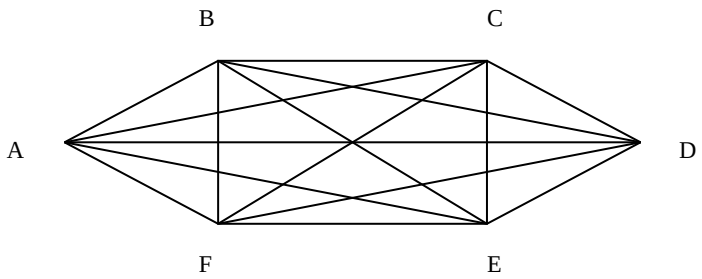
Dari contoh *graph* pada gambar 5.1 di atas bisa dilihat bahwa *graph* tersebut mempunyai 6 buah titik {A,B,C,D,E,F}, dan 9 buah garis {[A,B], [A,E], [B,C],[C,B],C,D],[D,E],C,E],D,D],E,B]}, dengan sebuah titiknya merupakan titik yang terisolir (titik F). Dari contoh *graph* di atas juga bisa dilihat bahwa dari titik B ke C (atau sebaliknya) terdapat dua buah garis. Kondisi ini disebut sebagai garis ganda (*multiple edges*). Selain itu, dalam *graph* di atas juga terdapat sebuah garis yang berujung dan berpangkal pada titik yang sama, yaitu pada titik D; garis ini disebut dengan kalang (*loop*). Sebuah jalur (*path*) dengan panjang *n* dari titik u ke titik *v* didefinisikan sebagai deretan *n*+1 buah titik, dan ditulis dengan notasi :

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

sedemikian rupa sehingga titik yang merupakan pangkal dari suatu garis menjadi ujung dari garis berikutnya kecuali titik pertama dan terakhir (titik v_0 dan v_n). Dalam notasi di atas, garis pertama akan terbentuk dari titik v_0 dan v_1, garis kedua terbentuk dari titik v_1 dan v_2 dan seterusnya. Jalur yang terbentuk dari titik-titik yang berbeda disebut jalur sederhana (*simple path*). Dalam gambar 5.1 (C,D,D,E,A,B) adalah sebuah jalur dengan panjang 5, dan (C,D,E,A,B) disebut sebagai jalur sederhana dengan panjang 4.

Jalur yang kedua titik ujungnya sama ($v_0 = v_n$) disebut dengan jalur tertutup (*cloased path*). Jalur tertutup yang terbentuk dari jalur sederhana disebut sebagai lingkar (*cycle*). Dalam gambar 5.1 (C,D,D,E,B,C) disebut sebagai jalur tertutup, dan (C,D,E,B,C) disebut sebagai lingkar.

Graph disebut terhubung (*connected graph*) apabila dari sembarang titik yang ada bisa dibuat jalur ke titik yang lain. Dengan demikian, *graph* pada gambar 5.1 bukan *graph* terhubung, karena terdapat sebuah titik yang terisolir (titik F). jika titik terisolir tersebut dihapus, maka *graph* akan menjadi *graph* terhubung. Apabila titik F dihubungkan dengan titik C, misalnya, maka *graph* menjadi *graph* terhubung.



Gambar 5.2. Contoh Graph lengkap.

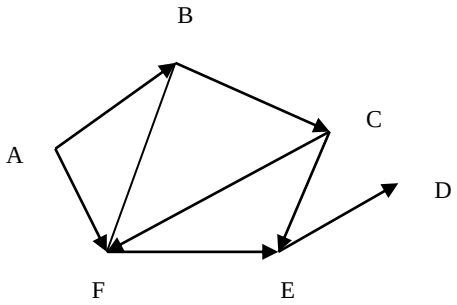
Suatu *graph* disebut *graph* lengkap (*complete graph*) apabila setiap titik yang ada dihubungkan ke titik-titik yang lain. *Graph* pada gambar 5.2 adalah *graph* lengkap, dan *graph* pada gambar 5.1 bukan *graph* lengkap. Sembarang *graph* lengkap dengan *n* buah titik akan mempunyai garis sebanyak $n(n-1)/2$ buah.

Graph terhubung yang tidak mempunyai lingkar disebut sebagai *graph* pohon (*tree graph*) atau pohon bebar (*free tree*) atau pohon. Dengan demikian, suatu pohon adalah salah satu bentuk khusus dari *graph*.

Graph berarah (*directed graph*, atau *digraph*) adalah merupakan bentuk yang lebih khusus dari *graph* seperti dijelaskan di atas, karena ke dalam *graph* berarah terkandung suatu alirah (*flow*), misalnya aliran beban, dari satu titik ke titik lain; dalam gambar biasanya disajikan menggunakan anak panah. Dengan demikian, pasangan titik yang menyatakan suatu garis harus disusun secara berurutan (*ordered pair*). Hal ini bisa dipahami dengan membayangkan suatu jalan yang lalu lintasnya hanya bisa dalam satu arah (*one-way traffic*). Dalam pasangan berurutan ini titik pertama menunjukkan titik asal aliran (*source*), dan titik kedua menunjukkan titik tujuan (*sink*). Dengan kenyataan di atas, maka *graph* dalam gambar 4.3 harus dituliskan sebagai :

N = {A,B,C,D,E,F}
E = {[A,B],[A,E],[B,C],[E,B],[C,E],[C,D],[E,D],[D,F]}

selain sebutan titik asal dan titik tujuan, titik pertama juga sering disebut sebagai *predesesor* dari titik kedua, dan titik kedua adalan *suksesor* dari titik pertama. Dengan demikian titik A adalah predesesor dari titik B dan E, dan titik B adalah suksesor dari titik A. dengan memperhatikan banyaknya anak panah yang keluar dan masuk pada suatu titik, dikenal dua istilah yang lain, yaitu *indegree*, yaitu banyaknya anak panah yang masuk ke suatu titik, dan *outdegree* adalah banyaknya anak panah yang meninggalkan suatu titik. Dengan mengambil contoh *graph* berarah pada gambar 5.3 maka *indegree* dari titik A adalah 0, dan *outdegree* dari titik A adalah 2. Dengan cara yang sama bisa diketahui banyaknya *indegree* maupun *outdegree* dari setiap titik yang ada pada suatu *graph* berarah.



Gambar 5.3. Contoh Graph berarah

Pengertian jalur, dan lingkar juga sama dengan pengertian jalur dan lingkar pada *graph* umum, tetapi dalam *graph* berarah harus memperhatikan arah aliran yang ditunjukkan dengan anak panah. Dengan mengambil contoh *graph* pada gambar 5.3, maka (A,E,D,F) adalah suatu jalur, dan (B,C,E,B) adalah suatu lingkar.

Contoh sederhana dari *graph* berarah adalah sebagai berikut. Diketahui sebuah berkas teks yang berisi sejumlah rekaman. Rekaman pertama berisi empat buah bilangan bulat positif, dan rekaman-rekaman berikutnya masing-masing berisi dua buah bilangan bulat positif. Rekaman pertama berisi data sebagai berikut. Bilangan pertama menunjukkan banyaknya kota, bilangan kedua dan ketiga menunjukkan dua buah kota yang diketahui, dan bilangan keempat menunjukkan banyaknya jalan yang harus dilalui (panjang jalur) untuk pergi dari kota pertama ke kota kedua. Persoalannya adalah untuk menentukan apakah ada jalur yang berisi ruas jalan dengan panjang tertentu dari suatu kota ke kota yang lain.

Penyajian *Graph*

Sub modul ini akan menyajikan cara penyajian *graph* berarah menggunakan larik. Penyajian *graph* berarah menggunakan larik cukup banyak variasinya, tetapi yang paling sering digunakan adalah matrix tetangga (*adjacency matrix*) matrix jalur (*path matrix*). Berikut akan dijelaskan masing-masing cara di atas.

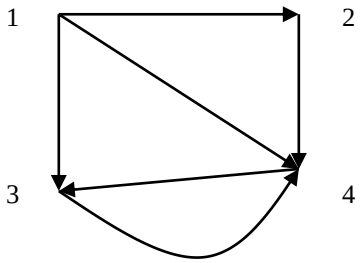
Matrix Tetangga

Matrix tetangga adalah matrix yang menunjukkan apakah suatu titik mempunyai tetangga atau tidak sesuai dengan pengertian tetangga dari *graph* berarah, yaitu tetangga suatu titik, misalnya v_1, adalah titik lain, misalnya v_j, yang dihubungkan dengan garis, dan arah mempunyai arah aliran adalah dari v_i ke v_j. Dalam gambar 5.4. titik 2 dikatakan bertetangga dengan titik 3, tetapi tidak untuk sebaliknya.

Untuk membentuk matrix tetangga, langkah pertama yang harus dilakukan adalah memberi nama untuk setiap titik yang ada, misalnya dari nomor 1 sampai dengan *n* dengan *n* adalah banyaknya titik yang ada. Setelah langkah di atas dilakukan, maka matrix tetangga bisa didefinisikan sebagai matrix berukuran *n* x *n* yang nilai elemennya bisa didefinisikan sebagai berikut :

$$t_{ij} = \begin{cases} 1 \text{ jika } v_i \text{ bertetangga dengan } v_j \\ 0 \text{ jika tidak } v_i \text{ bertetangga dengan } v_j \end{cases}$$

Dengan melihat definisi di atas bisa dikatakan bahwa nilai elemen dari matrix tetangga adalah 1 atau 0, yang sering disebut dengan matrix bit atau matrix boolean.



Gambar 5.4. Graph berarah

Dengan mengambil contoh *graph* berarah pada gambar 5.4, maka matrix tetangga untuk *graph* berarah tersebut adalah :

$T =$

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

Dari matrix tetangga di atas bisa dilihat bahwa banyaknya elemen bernilai 0 pada matrix *T* adalah sama dengan banyaknya garis pada *graph* berarah pada gambar 5.4. Untuk mengetahui bahwa di antara dua buah titik terdapat jalur dengan panjang tertentu dapat dilaksanakan dengan mempangkatkan matrix tetangga dengan bilangan tertentuk. Sebagai contoh, jika matrix *T* dikuadratkan (pangkat 2) akan kita lihat banyaknya jalur dengan panjang 2; jika maka matrix *T* dipangkatkan tiga, maka akan bisa dilihat banyaknya jalur dengan panjang 3, dan seterusnya. Dengan menggunakan contoh matrik tetangga *T* di atas, pangkat dua, pangkat tiga dan pangkat empat dari matrix *T* tersebut adalah :

$T^2 =$

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

$T^3 =$

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

$T^4 =$

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

Dengan melihat pada contoh di atas, maka terdapat jalur dengan panjang 2, panjang 3, dan panjang 4.

Matrix Jalur

Seperti halnya matrix tetangga, maka matrix jalur (*path matrix*) adalah matrix berukuran *n* x *n* yang nilai setiap elemennya ditentukan sebagai berikut :

$$P_{ij} = \begin{cases} 1 & \text{jika ada jalur antara } v_i \text{ dengan } v_j \\ 0 & \text{jika antara } v_i \text{ dengan } v_j \text{ tidak ada jalur} \end{cases}$$

Dengan memperhatikan bahwa antara titik v_i ke titik v_j ada kemungkinan terdapat lebih dari satu jalur dengan panjang berbeda, maka matrix jalur bisa ditentukan sebagai berikut. Dimisalkan bahwa terdapat jalur antara titik v_i ke titik v_j. Hal ini berarti bahwa terdapat jalur sederhana dari titik v_i ke titik v_j jika titik v_i tidak sama dengan titik v_j. Karena *graph* hanya mempunyai *n* titik, jalur sederhana tersebut pasti mempunyai panjang *n - 1* atau kurang, atau ada lingkar yang panjangnya *n* atau kurang.

Dari gambar 5.4 didapat : $B_4 = T + T^2 + T^3 + T^4$

Dengan memperhatikan matrix B tersebut di atas bisa diketahui bahwa nilai elemen b_{ij} menunjukkan panjang jalur dari titik v_i ke titik v_j.

Dengan memperhatikan penjelasan di atas bisa dilihat hubungan antara matrix tetangga *T* dengan matrix jalur *P*, yaitu bahwa jika *T* adalah matrix tetangga dan *P* adalah matrix jalur dari sembarang *graph*, maka $P_{ij} = 1$ bila dan hanya bila elemen b_{ij} tidak sama dengan nol dimana b_{ij} adalah elemen dari matrix

$$B_m = T + T^2 + T^3 + T^4 + \ldots + T^m$$

Dengan menggunakan contoh di atas maka matrix B_4 dan matrix bisa disajikan seperti terlihat berikut ini.

$B_4 =$

	1	2	3	4
1	0	5	6	8
2	0	1	2	2
3	0	3	3	5
4	0	2	3	5

$P =$

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

Dari matrix jalur di atas jika semua elemen bernilai tak nol, maka *graph* dikatakan sebagai *strongly connecnted graph*, yang berarti bahwa setiap titik bisa dicapai dari sembarang titik yang lain.

Matrix Beban

Penyajian *graph* seringkali tidak cukup hanya dengan menggunakan matrix tetangga atau matrix jalur. Pada persoalan *travelling salesman problem* sering harus dihitung besarnya biaya yang harus dikeluarkan untuk berjalan dari satu kota ke kota lain, dengan pikiran bahwa biayanya harus minimal.

Untuk menghitung biaya minimal, *graph* perlu disajikan dengan cara lain, yaitu menggunakan matrix beban (*weight matrix*) yang nilai elemennya menunjukkan biaya (atau beban) yang harus ditanggung atau dikeluarkan untuk berpindah dari satu kota ke kota lain.

Matrix beban pada dasarnya hampir sama dengan matrix tetangga, yaitu sebuah matrix berukuran *n* x *n* (dengan n adalah banyaknya titik) dan didefinisikan sebagai berikut :

$$W_{ij} = \begin{cases} beban & \text{jika ada jalur antara } v_i \text{ dengan } v_j \\ 0 & \text{jika tidak ada jalur antara } v_i \text{ dengan } v_j \end{cases}$$

dengan *beban* adalah suatu besaran yang menunjukkan biaya yang harus dikeluarkan (besarnya beban yang harus ditanggung) jika seseorang ingin bepergian dari kota v_i ke kota v_j. Gambar 5.5 berikut ini menyajikan contoh sebuah *graph* dan matrix bebannya.

W =	1	2	3	4
1	0	5	9	4
2	0	0	0	5
3	0	3	0	5
4	0	0	6	0

akan berarti, bahwa dari kota 1 ke kota 2 memiliki beban 5, dari kota 1 ke kota 3 memiliki beban 9 dan seterusnya, sehingga bias diketahuhi biaya yang harus dikeluarkan untuk berjalan ; misalnya, dari kota 2 ke kota 3 (yang ternyata tidak ada hubungan secara langsung).

5.3 SOAL TUGAS & LATIHAN (PERTEMUAN KESEBELAS)

5.3.1 TUGAS PENDAHULUAN

Tugas pendahuluan dikumpulkan pada pertemuan ke sepuluh :

Agar saudara tidak kesulitan dalam pembuatan program untuk menerapkan konsep matriks tetangga maka buatlah :

1. Contoh *graph* berarah dengan 4 buah *vertex* dengan 7 *edge* dan berikan beban untuk setiap *edge*-nya!
2. Matrik tetangga dari contoh *graph* yang telah saudara buat pada nomor 1!
3. Algoritma untuk melakukan pencarian dan penyajian informasi untuk masalah-masalah berikut :
    a. Adakah vertex v_1 terhubung langsung dengan v_2? bila ya, edge apa yang menghubungkannya?
    b. Vertex v_1 terhubung dengan vertek mana saja?

5.3.2. LATIHAN PRAKTIKUM (Pertemuan Kesebelas)

Latihan praktikum pada pertemuan kesebelas :

1. Buatlah program dengan menerapkan contoh *graph* yang telah saudara buat, serta buktikan kebenaran hasil dari matriks tetangga yang dihasilkan program! Program harus dapat:
    a. Menerima input jumlah simpul/vertex

b. Menerima input simpul apa saja yang terhubung ke setiap simpul
c. Menampilkan isi representasi graph