BAB 6

GRAPH (ALGORITMA MULTIPATH)

6.1. TUJUAN PEMBELAJARAN

Tujuan dari pembelajaran pada pertemuan ini adalah mahasiswa diharapkan mampu:

1. Memahami struktur data graph.
2. Mampu mengimplementasikan algoritma pencarian jalur terpendek.

6.2. DASAR TEORI

6.2.1. Pengertian Graph

Dalam pemrograman C++, *graph* dapat diimplementasikan sebagai sekumpulan *nodes* atau *vertices* yang terhubung melalui *edges* atau garis. Setiap node pada *graph* tersebut dapat mewakili objek atau entitas tertentu, sementara setiap *edge* mewakili relasi atau hubungan antar objek atau entitas tersebut. Representasi visual dari graph adalah dengan menyatakan objek sebagai node, bulatan atau titik (*Vertex*), sedangkan hubungan antara objek dinyatakan dengan garis (Edge).

$$G = (V, E) \tag{14.1}$$

dimana

G = Graph

V = Simpul atau Vertex, atau Node, atau Titik

E = Busur atau *Edge*, atau arc

Jalur pada *graph* dinotasikan sebagai

$$P = (V0, V1, \ldots, Vn) \tag{14.2}$$

Dimana

P : jalur

V_i : titik jalur

n : jumlah titik jalur

Graph merupakan suatu cabang ilmu yang memiliki banyak terapan. Implementasi *graph* biasanya berkaitan dengan matematika dan ilmu komputer. Banyak sekali struktur yang bisa direpresentasikan dengan *graph*, dan banyak masalah yang bisa diselesaikan dengan bantuan graph. Salah satu fungsi *graph* adalah untuk merepresentasikan suaru 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. Ada beberapa cara untuk menyimpan graph di dalam sistem komputer, diantaranya :

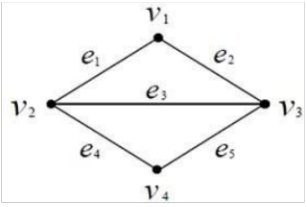
1. Adjacency List: Metode ini menyimpan setiap node pada graph sebagai sebuah list yang berisi node lain yang terhubung dengan node tersebut.
2. Adjacency Matrix: Metode ini menyimpan setiap node pada graph sebagai sebuah elemen pada matriks, dimana baris dan kolom dari matriks tersebut merepresentasikan node dan hubungan antar node tersebut dinyatakan melalui elemen pada matriks.
3. Incidence Matrix: Metode ini menyimpan setiap node pada graph sebagai baris pada matriks dan setiap edge sebagai kolom, dimana elemen pada matriks menyatakan hubungan antar node dan edge.

Struktur data bergantung pada struktur *graph* dan algoritma yang digunakan untuk memanipulasi *graph*. Sturktur data yang biasa digunakan dibedakan menjadi 2, yaitu struktur *linked list* dan matriks (array dimensi 2), meskipun demikian dalam penggunaannya struktur terbaik yang sering digunakan adalah dengan mengkombinasikan keduanya.

6.2.2. Istilah Pada Graph

Terdapat beberapa istilah yang berkaitan dengan graph, diantaranya:

1. Vertex : himpunan node / titik pada sebuah graph.
2. Edge : himpunan garis yang menghubungkan tiap node / vertex.
3. Adjacent : 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.



Gambar 6.1 Adjacent pada *graph*

4. *Weight* : Sebuah *graph G = (V, E). Graph* ini 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 dinyatakanpula sebagai *G = (V, E, W).*

   Graf berbobot *G = (V, E, W)* dapat menyatakan

   a. 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

   b. suatu sistem jaringan komputer, di mana

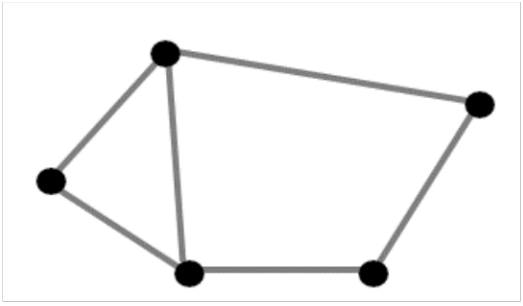
      - V = himpunan komputer
      - E = himpunan jalur komunikasi langsung antar dua komputer
      - W = fungsi bernilai real pada E yang menyatakan jarak atau ongkos atau waktu

5. *Path* : jalur dengan setiap vertex berbeda. Contoh, P = D5B4C2A Sebuah jalur (W) didefinisikan sebagai urutan (tidak nol) vertex dan edge. Diawali origin vertex (vertex awal) dan diakhiri terminus vertex (vertex akhir). Dan setiap 2 garis berurutan adalah series. Contoh, W = A1B3C4B1A2.
6. *Cycle* atau *Cirkuit* : lintasan yang berawal dan berakhir pada simpul yang sama.

6.2.3. Jenis – Jenis Graph

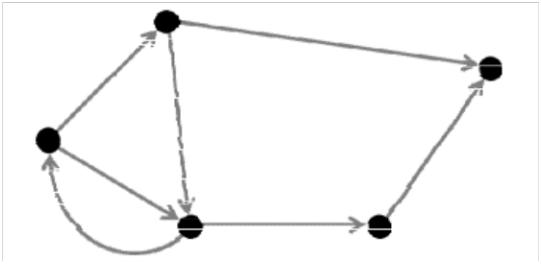
Berikut adalah jenis – jenis graph, diantaranya :

1. *Graph* tak berarah (*undirected graph* atau *non-directed graph*) : *graph* yang simpulnya hanya menghubungkan 2 vertex tanpa menunjukkan arah. Pada *graph* ini urutan simpul dalam sebuah busur tidak dipentingkan. Misal busur e_1 dapat disebut busur AB atau BA



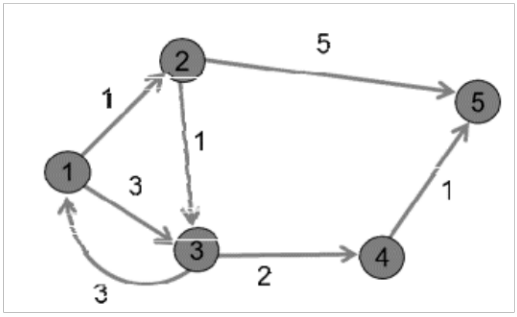
Gambar 6.2 *Graph* tidak berarah

2. *Graph* berarah (*directed graph*) : *graph* yang simpulnya tidak hanya menghubungkan 2 vertex tetapi juga menunjukkan arah. Pada *graph* ini urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.



Gambar 6.3 *Graph* berarah

3. *Graph* Berbobot (*Weighted Graph*) : *graf* yang setiap sisinya diberi sebuah harga bobot.
   a. Jika setiap busur mempunyai nilai yang menyatakan hubungan antara 2buah simpul, maka busur tersebut dinyatakan memiliki bobot.
   b. Bobot sebuah busur dapat menyatakan panjang sebuah jalan dari 2 buahtitik, jumlah rata-rata kendaraan perhari yang melalui sebuah jalan, dll.



Gambar 6.4 *Graph* berbobot

6.2.4. Representasi Graph dengan Matriks (Array 2 dimensi)

Dalam pemrograman, untuk mengolah data dari suatu *graph*, graf tersebut harus dinyatakan dalam struktur data yang mampu merepresentasikan *graph* tersebut. Dalam hal ini, *graph* harus disajikan dalam bentuk array dan dimensi yang sering disebut matriks.

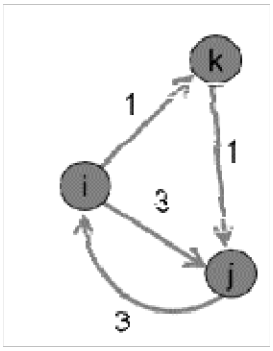
graph biasanya digunakan untuk menyelesaikan permasalahan lintasan terpendek. Jika diberikan sebuah *graph* berbobot, masalah lintasan terpendek adalah bagaimana kita mencari sebuah jalur pada *graph* yang meminimalkan jumlah bobot sisi pembentuk jalur tersebut. Terdapat beberapa macam persoalan lintasan terpendek antara lain:

1. Lintasan terpendek antara dua buah simpul tertentu (*a pair shortets path*).
2. Lintasan terpendek antara semua pasangan simpul (*all pairs shortest path*).
3. Lintasan terpendek dari simpul tertentu ke semua simpul yang lain (*single-source shoertest path*).
4. Lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*).

6.2.5. Algoritma Warshall

Algoritma Floyd-Warshall adalah algoritma untuk menghitung jarak terpendek untuk semua pasangan titik pada sebuah lokasi yang dapat diubah menjadi sebuah graf berarah dan berbobot, yang berupa titik-titik (node/vertex V) dan sisi-sisi (edge E) serta paling memiliki minimal satu sisi pada setiap titik. Algoritma warshall digunakan untuk menyelesaikan permasalahan jalur terpendek *multi path*.

Pada Algoritma Floyd-Warshall 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.



Gambar 6.5 Illustrasi *graph*

Sebagai ilustrasi pada Gambar 6.5, algoritma melakukan pengecekan apakah beban langsung Q(i, j) memang lebih kecil daripada beban melalui titik perantara Q(i,k)+Q(k,j)

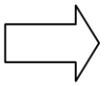
if ((Q(i,k)+Q(k,j))< Q(i, j))

Q(i, j) ⇐ Q(i,k)+Q(k,j)

Seperti pada contoh kasus Gambar 6.4 di atas, langkah-langkah penyelesaian dengan algoritma Warshall sebagai berikut:

1. 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	-	-	-	-	

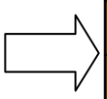


Q	1	2	3	4	5
1	M	1	3	M	M
2	M	M	1	M	5
3	3	M	M	2	M
4	M	M	M	M	1
5	M	M	M	M	M

Dimana M adalah big integer.

2. 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

3. Representasi matriks rute di bawah ini menjadi array dua dimensi

	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

Dimana M adalah big integer.

4. Melakukan pengecekan beban langsung Q(1,3) = 3 dengan beban tak 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

   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)
```

6.3. PERCOBAAN

1. Buatlah workspace menggunakan Replit.
2. Buatlah project baru Graph1 yang berisi file C++ source untuk pembuatan program sederhana graph dan algoritma warshall

3. Cobalah untuk masing-masing percobaan di bawah

Percobaan 1 : Graph dengan matrix

```
#include <iostream>
#include <vector>

using namespace std;

const int MAX = 100;

int graph[MAX][MAX];
int n;

void addEdge(int x, int y) {
  graph[x][y] = 1;
  graph[y][x] = 1;
}

void displayGraph() {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      cout << graph[i][j] << " ";
    }
    cout << endl;
  }
}

int main() {
  cout << "Masukan jumlah dimensi array ";
  cin >> n;

  addEdge(0, 1);
  addEdge(0, 2);
  addEdge(1, 2);
```

```
  addEdge(2, 3);

  displayGraph();

  return 0;
}
```

OUTPUT

Masukan jumlah dimensi array 2
0 1
1 0

```
g++ -o grafarray grafarray.cpp
./grafarray
Masukan jumlah dimensi array 2
0 1
1 0
```

Percobaan 2 : Algoritma Warshall untuk pencarian jalur terpendek multipath

```
#include <iostream>
#include <cstring>

using namespace std;

const int N = 100;
int dist[N][N];

void floydWarshall(int n)
{
    for (int k = 0; k < n; k++)
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                {
```

```
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
}

int main()
{
    int n;
    cout << "Masukkan jumlah node pada graph: ";
    cin >> n;
    memset(dist, 0x3f, sizeof dist);
    for (int i = 0; i < n; i++)
    {
        dist[i][i] = 0;
    }
    cout << "Masukkan jarak antar node: ";
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            int w;
            cin >> w;
            dist[i][j] = w;
        }
    }
    floydWarshall(n);
    cout << "Hasil jalur terpendek antar node: " << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
```

```
            cout << dist[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

OUTPUT

Masukkan jumlah node pada graph: 3

Masukkan jarak antar node: 2

1

2

3

4

Hasil jalur terpendek antar node:

1 2

3 4

```
Masukkan jumlah node pada graph: 3
Masukkan jarak antar node: 2
1
2
3
4
Hasil jalur terpendek antar node:
1 2
3 4
```

6.4. TUGAS DAN LATIHAN

Buatlah program C++ yang menghasilkan output seperti dibawah ini!

```
Masukkan Jumlah Kota : 4
Nilai Cost Matrix
Cost Element Baris ke-: 1
0
4
1
3

Cost Element Baris ke-: 2
4
0
2
1

Cost Element Baris ke-: 3
1
2
0
5

Cost Element Baris ke-: 4
3
1
5
0

Cost List :
       0       4       1       3
       4       0       2       1
       1       2       0       5
       3       1       5       0

Jalur Terpendek :
1--->3--->2--->4--->1

Minimum Cost : 7
```

Gambar 6.6 Soal nomor 1