Praktikum 12

Graph

POKOK BAHASAN:

- ✓ Konsep graph
- ✓ Struktur data graph
- ✓ Implementasi graph dengan matriks dan *linked list* dalam Bahasa C

TUJUAN BELAJAR:

Setelah melakukan praktikum dalam bab ini, mahasiswa diharapkan mampu:

- ✓ Memahami konsep dasar graph
- ✓ Mengimplementasikan graph dalam bentuk matriks dan *linked list*.
- ✓ Mengimplementasikan graph untuk pencarian jalur terpendek

DASAR TEORI:

Sebuath graph G=<V,E> terdiri dari sekumpulan titik (*nodes* atau *vertices*) V dan sekumpulan garis (*arcs* atau *edges*) E. Sebuah garis menghubungkan dua titik u dan v; v dikatakan *adjacent* to u. Pada graph berarah (*directed graph*), setiap garis mempunyai arah dari u ke v dan dituliskan sebagai pasangan <u,v> atau u->v. Pada graph tak berarah (*undirected graph*), garis tidak mempunyai arah dan dituliskan sebagai pasangan {u,v} atau u<->v. Graph tak berarah merupakan graph berarah jika setiap garis tak berarah {u,v} merupakan dua garis berarah <u,v> dan <v,u>.

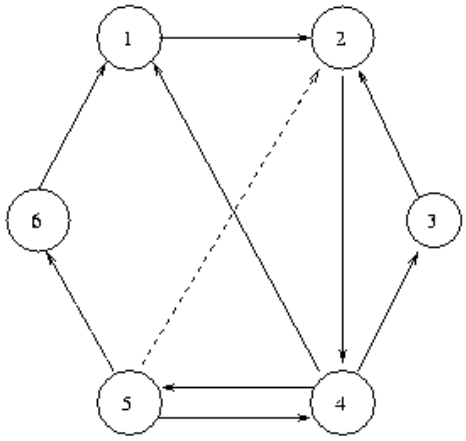
Sebuah jalur (*path*) pada G adalah sekumpulan titik $<v_0, v_1, v_2, \ldots, v_n>$ sehingga $<v_i, v_{i+1}>$ (atau $\{v_i, v_{i+1}\}$), untuk setiap I dari 0 ke n-1 adalah garis pada G. Jalur menjadi sederhana jika tidak ada dua titik yang identik. Jalur merupakan sebuah siklus

jika $v_0=v_n$. Jalur merupakan siklus yang sederhana jika $v_0=v_n$ dan tidak ada dua titik yang identik.

Graph banyak digunakan untuk menggambarkan jaringan dan peta jalan, jalan kereta api, lintasan pesawat, system perpipaan, saluran telepon, koneksi elektrik, ketergantungan diantara *task* pada sistem manufaktur dan lain-lain. Terdapat banyak hasil dan struktur penting yang didapatkan dari perhitungan dengan graph.

1. GRAPH DENGAN MATRIK *ADJACENCY*

Sebuah graph G= dapat direpresentasikan dengan matrik *adjacency* A sebagai |V|*|V|. Jika G berarah, A_{ij} = true jika dan hanya jika $<v_i, v_j>$ berada pada E. Terdapat paling banyak $|V|^2$ garis pada E.

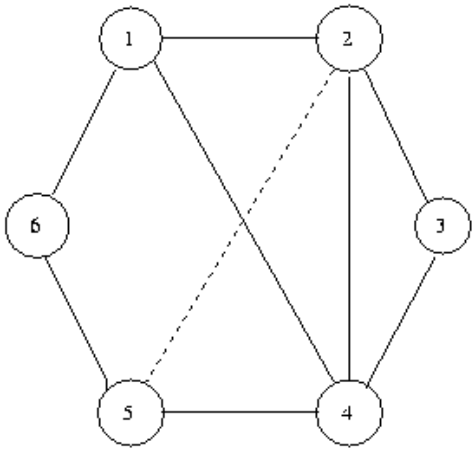


Gambar 12.1 Graph berarah

Matriks *adjacency* dari Graph berarah adalah sebagai berikut :

	1	2	3	4	5	6
1		T				
2				T		
3		T				
4	T		T		T	
5		T		T		T
6	T					

Jika G adalah graph tak berarah, $A_{ij}=A_{ji}$=true jika $\{v_i,v_j\}$ berada pada E dan $A_{ij}=A_{ji}$=false apabila G adalah graph berarah.



Gambar 12.2 Graph tak berarah

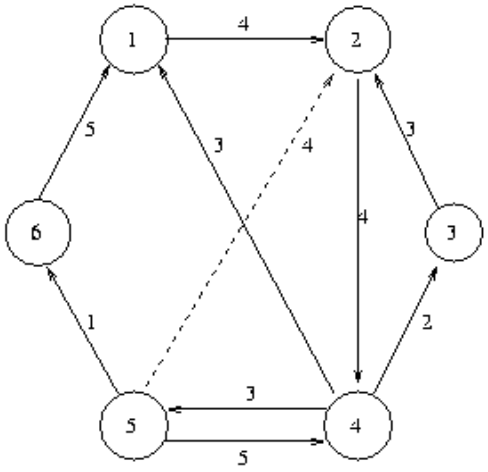
Matriks *adjacency* dari Graph berarah adalah sebagai berikut :

	1	2	3	4	5	6
1			T		T		T
2		T			T	T	T
3			T		T		
4		T	T	T			T
5			T		T		T
6		T					T

Pada graph tak berarah, matriks *adjacency* dapat berupa matriks segitiga atas sebagai berikut :

	1	2	3	4	5	6
1			T		T		T
2				T	T	T	
3					T		
4						T	
5							T
6							

Baik graph berarah maupun tak berarah dapat diberikan pembobot (*weight*). Pembobot diberikan untuk setiap garis. Hal ini biasanya digunakan untuk menggambarkan jarak antara dua kota, waktu penerbangan, harga tiket, kapasitas elektrik suatu kabel atau ukuran lain yang berhubungan dengan garis. Pembobot biasanya disebut panjang garis, khususnya jika graph menggambarkan peta. Pembobot atau panjang dari suatu jalur atau siklus merupakan penjumlahan pembobot atau panjang dari komponen garis.

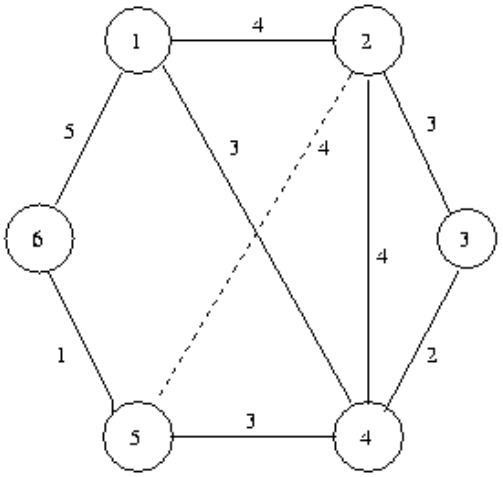


Gambar 12.3 Graph berarah dengan pembobot

Matriks *adjacency* dari graph dengan pembobot dapat digunakan untuk menyimpan pembobot dari setiap garis. Jika garis kehilangan nilai, kemungkinan nilai negative, nol atau bilangan besar menunjukkan nilai yang tak terbatas (*infinity*).

Matriks *adjacency* pada graph berarah dengan pembobot adalah sebagai berikut :

	1	2	3	4	5	6
1		4				
2				4		
3		3				
4	3		2		3	
5		4		5		1
6	5					



Gambar 12.4 Graph tak berarah dengan pembobot

Matriks *adjacency* pada graph tak berarah dengan pembobot adalah sebagai berikut :

```
     1  2  3  4  5  6
1       4     3     5
2    4     3  4  4
3       3     2
4    3  4  2     3
5       4     3     1
6    5              1
```

Atau berupa matriks segitiga atas sebagai berikut :

```
     1  2  3  4  5  6
1       4     3     5
2          3  4  4
3             2
4                3
5                   1
6
```

Sebuah graph dikatakan lengkap jika semua kemungkinan garis direpresentasikan. Sebuah graph dikatakan padat jika hamper semua garis digunakan. Dikatakan jarang jika sebagian besar garis tidak digunakan dimana $|E|<<|V|^2$. Matriks *adjacency* sangat efisien untuk graph yang padat dan tidak efisien untuk graph yang jarang karena sebagian besar tidak terdapat garis. Oleh karena itu matriks *adjacency* menggunakan *linked list* tidak digunakan untuk graph yang jarang.

2. GRAPH DENGAN *LIST ADJACENCY*

Pada graph berarah maupun graph tak berarah kemungkinan kehilangan garis dapat disimpan dengan menggunakan *linked list* yang biasanya disebut *list adjacency*.

Misalnya pada graph berarah dengan pembobot pada Gambar 12.3, garis $<v_i,v_j>$ ditempatkan pada *linked list* yang berhubungan dengan v_i. Garis direpresentasikan dengan titik tujuan v_j dan pembobot.

List adjacency untuk graph berarah dengan pembobot adalah sebagai berikut :

```
1:--------->2,4:nil
2:--------->4,4:nil
3:--------->2,3:nil
4:--------->1,3:----->3,2:----->5,3:nil
5:--------->4,5:----->6,1:nil
6:--------->1,5:nil
```

Sedangkan pada graph tak berarah dengan pembobot seperti pada Gambar 12.4, *list adjacency* adalah sebagai berikut :

```
1:--------->2,4:----->4,3:----->6,5:nil
2:--------->1,4:-----:3,3:----->4,4:----->5,4:nil
3:--------->2,3:----->4,2:nil
4:--------->1,3:----->2,4:----->3,2:----->5,3:nil
5:--------->2,4:----->4,3:----->6,1:nil
6:--------->1,5:----->5,1:nil
```

Pada graph tak berarah, separuh bagian dapat disimpan hanya dengan menyimpan $\{v_i,v_j\}$ untuk i>=j sebagai berikut :

```
1:--------->2,4:----->4,3:----->6,5:nil
2:--------->3,3:----->4,4:----->5,4:nil
3:--------->4,2:nil
4:--------->5,3:nil
5:--------->6,1:nil
6:nil
```

List adjacency dapat didefinisikan menggunakan struktur dan pointer.

3. SINGLE SOURCE SHORTEST PATHS PADA GRAPH BERARAH

Pembobot garis pada graph berarah biasanya disebut panjang jalur. Panjang jalur $<v_0, v_1, ..., v_n>$ adalah penjumlahan panjang dari semua komponen garis $<v_i,v_{i+1}>$.

Sebagai contoh dari permasalahan jalur, dinas pemadam kebakaran menyimpan peta kota yang ditandai dengan lokasi rawan kebakaran seperti took kimia. Mereka ingin mengetahui rute terpendek dari kantor pemadam kebakaran ke masing-masing daerah tersebut. Sebagai catatan, panjang jalan berupa panjang fisik atau waktu estimasi perjalanan tetapi tidak keduanya.

Untuk menemukan jalur terpendek dari satu sumber ke semua titik dengan menemukan jalur terpendek antara beberapa dua titik. Biasanya sumber disebut v_1. Algoritma **Dijkstra** dapat digunakan untuk memecahkan permasalahan *single-source shortest paths problem*. Algoritma ini bekerja dengan menambahkan besarnya himpunan titik yang ‘sukses’ dimana jalur terpendek dari sumber diketahui.. Mula-mula ‘sukses’ hanya berisi sumber v_1. Titik yang sedang tidak berada pada himpunan ‘sukses’ ternyata terdekat dengan sumber ditemukan maka ditambahkan pada

himpunan tersebut. Modifikasi himpunan dilakukan terus-menerus sampai berisi semua titik.

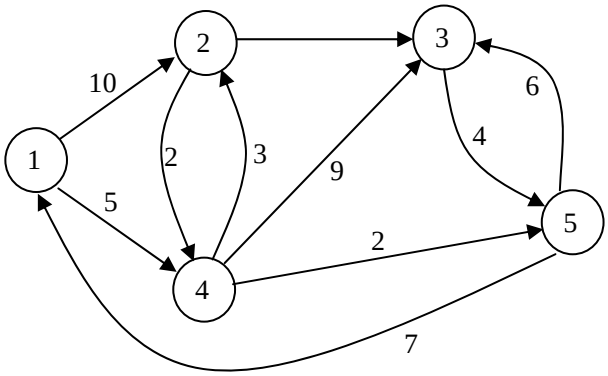
Algoritma Dijkstra untuk *Single-Source Shortest Paths* adalah sebagai berikut :

```
-- Graph G = <V, E>
-- P[i] adalah panjang jalur dari sumber ke titik i

done := {v1}

for vertex i in V-{1}
   P[i] := E[1,i]     --garis berarah
end for

loop |V|-1 times
  find closest vertex to v1 in V - done

  done +:= {closest}

  for vertex j in V - done
    P[j]:=min(P[j],  --update jalur terpendek,
              P[closest]+E[closest,j])
  end for
end loop
```

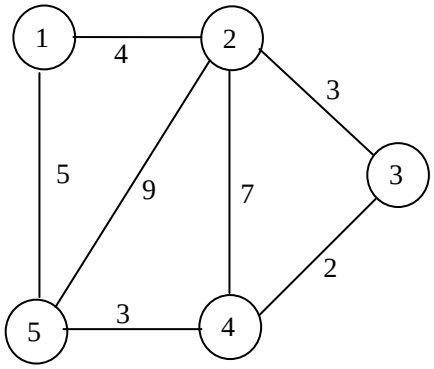
P merupakan garis langsung dari sumber. Pada iterasi pertama dari loop utama, titik dengan koneksi langsung yang terpendek dari sumber dipilih. Hal ini dapat dibenarkan karena sembarang jalur tak langsung ke titik ini merupakan garis yang terpanjang dari sumber. Bila suatu titik terdekat dipilih dan ditambahkan ke himpunan 'sukses', berarti merupakan jalur terpendek dari sumber, terdekat, daripada titik yang tak terpilih lain.

TUGAS PENDAHULUAN:

1. Perhatikan graph berarah pada Gambar 12.5.
   Buatlah matriks *adjacency* dan *list adjacency* –nya
2. Perhatikan graph tak berarah pada Gambar 12.6.
   Buatlah matriks *adjacency* dan *list adjacency* –nya.



Gambar 12.5 Graph berarah untuk Tugas 1



Gambar 126 Graph tak berarah untuk Tugas 2

3. Buatlah flowchart untuk pencarian jalur terpendek *Single-Source Shortest Path* pada graph berarah Gambar 12.3 dengan algoritma Dijkstra menggunakan matriks.

PERCOBAAN:

1. Buatlah workspace menggunakan Visual C++.
2. Buatlah project baru GRAPH.
3. Cobalah untuk masing-masing percobaan di bawah dengan nama source yang berbeda.

Percobaan 1 : Implementasi matriks *adjacency* pada graph berarah dengan pembobot

```
#include <stdio.h>
#include <stdlib.h>

#define N 6

int Vertex[N] = {1,2,3,4,5,6};

int Edge[N][N] =   {{0,4,0,0,0,0},
                    {0,0,0,4,0,0},
                    {0,3,0,0,0,0},
                    {3,0,2,0,3,0},
                    {0,4,0,5,0,1},
                    {5,0,0,0,0,0}};

typedef struct
{
      int V[N];
      int E[N][N];
}GraphB;

void Jalur(GraphB G)
{
      int i, j;
      printf("Daftar Garis (Edge) : \n");
      for(i=0; i<N; i++)
            for(j=0; j<N; j++)
                  if(G.E[i][j]>0)
                        printf("(%d, %d) = %d\n", G.V[i],
                        G.V[j], G.E[i][j]);
}

void DataGraph(GraphB *G)
{
      int i, j;
      for (i=0; i<N; i++)
            G->V[i] = Vertex[i];

      for (i=0; i<N; i++)
            for(j=0; j<N; j++)
                  G->E[i][j] = Edge[i][j];
}
```

```
void DataGraphRand(GraphB *G)
{
      int i,j;
      for (i=0; i<N; i++)
            G->V[i] = Vertex[i];

      srand(0);
      for (i=0; i<N; i++)
            for(j=0; j<N; j++)
                  G->E[i][j] = rand()/10000*3;

}

void main()
{
      GraphB G, GRand;

      DataGraph(&G);
      Jalur(G);

      DataGraphRand(&GRand);
      Jalur(GRand);
}
```

Percobaan 2 : Implementasi matriks *adjacency* pada graph tak berarah dengan pembobot

```
#include <stdio.h>
#include <stdlib.h>

#define N 6

int Vertex[N] = {1,2,3,4,5,6};

int Edge[N][N] =   {{0,4,0,3,0,5},
                    {0,0,3,4,4,0},
                    {0,0,0,2,0,0},
                    {0,0,0,0,3,0},
                    {0,0,0,0,0,1},
                    {0,0,0,0,0,0}};

typedef struct
{
      int V[N];
      int E[N][N];
}GraphTB;
```

```
void Jalur(GraphTB G)
{
      int i, j;
      printf("Daftar Garis (Edge) : \n");
      for(i=0; i<N; i++)
            for(j=i+1; j<N; j++)
                  if(G.E[i][j]>0)
                        printf("(%d, %d) = %d\n", G.V[i],
                        G.V[j], G.E[i][j]);
}

void DataGraph(GraphTB *G)
{
      int i, j;
      for (i=0; i<N; i++)
            G->V[i] = Vertex[i];

      for (i=0; i<N; i++)
            for(j=i+1; j<N; j++)
                  G->E[i][j] = Edge[i][j];
}

void DataGraphRand(GraphTB *G)
{
      int i,j;
      for (i=0; i<N; i++)
            G->V[i] = Vertex[i];

      srand(0);
      for (i=0; i<N; i++)
            for(j=i+1; j<N; j++)
                  G->E[i][j] = rand()/10000*3;

}

void main()
{
      GraphTB G, GRand;

      DataGraph(&G);
      Jalur(G);

      DataGraphRand(&GRand);
      Jalur(GRand);
}
```

Percobaan 3 : Implementasi *list adjacency* pada graph berarah dengan pembobot

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

#define N 6

int Vertex[N] = {1,2,3,4,5,6};

int Garis[N][N] =   {{0,4,0,0,0,0},
                     {0,0,0,4,0,0},
                     {0,3,0,0,0,0},
                     {3,0,2,0,3,0},
                     {0,4,0,5,0,1},
                     {5,0,0,0,0,0}};

typedef struct edge
{
      int bobot;
      int VTujuan;
      struct edge *next;
}Edge;

typedef struct
{
      int V[N];
      Edge *EFront, *ERear;
}GraphB;

void Jalur(GraphB *G, int i)
{
      printf("\n%d : ", G->V[i]);
      while(G->EFront != NULL)
      {
            printf("-----> %d, %d : ", G->EFront->bobot,
                                       G->EFront->VTujuan);
            G->EFront = G->EFront->next;
      }
}
```

```
void DataGraph(GraphB *G, int i)
{
      int j;
      Edge *ptr;
      for (j=0; j<N; j++)
      {

            G->V[j] = Vertex[j];
      }
      for(j=0; j<N; j++)
      {
            if (Garis[i][j]>0)
            {
                  ptr=(Edge *) malloc (sizeof(Edge));
                  ptr->bobot = Garis[i][j];
                  ptr->VTujuan = G->V[j];
                  ptr->next = NULL;
                  if(G->EFront == NULL)
                        G->EFront = G->ERear =ptr;
                  else
                  {
                        G->ERear->next = ptr;
                        G->ERear = ptr;
                  }
            }
      }
}

void DataGraphRand(GraphB *G, int i)
{
      int j;
      Edge *ptr;
      int BilRand;
      for (j=0; j<N; j++)
      {

            G->V[j] = Vertex[j];
      }

      for(j=0; j<N; j++)
      {
            BilRand = rand()/10000;
            if (BilRand>0)
            {
                  ptr=(Edge *) malloc (sizeof(Edge));
                  ptr->bobot = BilRand;
                  ptr->VTujuan = G->V[j];
                  ptr->next = NULL;
```

```
                if(G->EFront == NULL)
                      G->EFront = G->ERear =ptr;
                else
                      {
                            G->ERear->next = ptr;
                            G->ERear = ptr;
                      }
                }
      }
}

void main()
{
      GraphB G[N], GRand[N];

      int i;
      srand(0);

      printf("\nDaftar Garis (Edge) : \n");
      for(i=0; i<N; i++)
      {
          G[i].EFront=NULL;
            DataGraph(&G[i], i);
            Jalur(&G[i], i);
      }

      for(i=0; i<N; i++)
      {
          GRand[i].EFront=NULL;
            DataGraphRand(&GRand[i], i);
            Jalur(&GRand[i], i);
      }
}
```

Percobaan 4 : Implementasi *list adjacency* pada graph tak berarah dengan pembobot

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

#define N 6

int Vertex[N] = {1,2,3,4,5,6};
```

```
int Garis[N][N] =   {{0,4,0,3,0,5},
                     {0,0,3,4,4,0},
                     {0,0,0,2,0,0},
                     {0,0,0,0,3,0},
                     {0,0,0,0,0,1},
                     {0,0,0,0,0,0}};

typedef struct edge
{
      int bobot;
      int VTujuan;
      struct edge *next;
}Edge;

typedef struct
{
      int V[N];
      Edge *EFront, *ERear;
}GraphTB;

void Jalur(GraphTB *G, int i)
{
      printf("\n%d : ", G->V[i]);
      while(G->EFront != NULL)
      {
            printf("-----> %d, %d : ", G->EFront->bobot,
                                       G->EFront->VTujuan);
            G->EFront = G->EFront->next;
      }
}

void DataGraph(GraphTB *G, int i)
{
      int j;
      Edge *ptr;
      for (j=0; j<N; j++)
      {

            G->V[j] = Vertex[j];
      }
      for(j=i+1; j<N; j++)
      {
            if (Garis[i][j]>0)
            {
                  ptr=(Edge *) malloc (sizeof(Edge));
                  ptr->bobot = Garis[i][j];
                  ptr->VTujuan = G->V[j];
                  ptr->next = NULL;
```

```
                if(G->EFront == NULL)
                      G->EFront = G->ERear =ptr;
                else
                {
                      G->ERear->next = ptr;
                      G->ERear = ptr;
                }
          }
     }
}

void DataGraphRand(GraphTB *G, int i)
{
     int j;
     Edge *ptr;
     int BilRand;
     for (j=0; j<N; j++)
     {

          G->V[j] = Vertex[j];
     }

     for(j=i+1; j<N; j++)
     {
          BilRand = rand()/10000;
          if (BilRand>0)
          {
                ptr=(Edge *) malloc (sizeof(Edge));
                ptr->bobot = BilRand;
                ptr->VTujuan = G->V[j];
                ptr->next = NULL;
                if(G->EFront == NULL)
                      G->EFront = G->ERear =ptr;
                else
                {
                      G->ERear->next = ptr;
                      G->ERear = ptr;
                }
          }
     }
}
```

```
void main()
{
      GraphTB G[N], GRand[N];

      int i;
      srand(0);

      printf("\nDaftar Garis (Edge) : \n");
      for(i=0; i<N; i++)
      {
          G[i].EFront=NULL;
            DataGraph(&G[i], i);
            Jalur(&G[i], i);
      }

      for(i=0; i<N; i++)
      {
          GRand[i].EFront=NULL;
            DataGraphRand(&GRand[i], i);
            Jalur(&GRand[i], i);
      }
}
```

LATIHAN:

1. Implementasi Tugas Pendahuluan no 1 dengan menampilkan jalur yang ada.
2. Implementasi Tugas Pendahuluan no 2 dengan menampilkan jalur yang ada.
3. Tambahkan kode program untuk memasukkan input titik dan garis dari graph.
4. Implementasikan pencarian jalur terpendek *Single-Source Shortest Path* dengan algoritma Dijkstra menggunakan matriks pada Tugas Pendahuluan no. 3.
5. Berikan kesimpulan dari percobaan dan latihan yang telah Anda lakukan.