Graph

IF1210 – Algoritma dan Pemrograman 1
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Tujuan

- Mahasiswa memahami definisi graph
- Berdasarkan pemahaman tersebut, mampu membuat fungsi sederhana yang memanipulasi graph

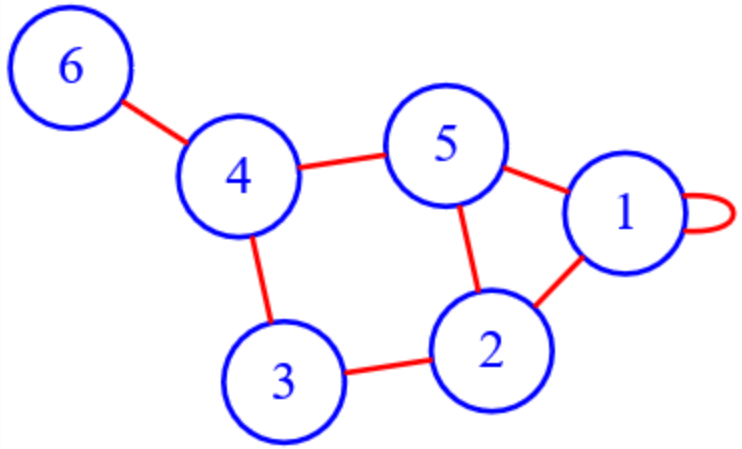
Definisi Graph

Graph (atau Graf) $G = (V, E)$ adalah suatu struktur yang terdiri dari:

- Sekumpulan simpul (*vertices* atau *nodes*) = V, tidak boleh kosong
- Sekumpulan busur (*edges*) yang menghubungkan simpul-simpul = E, boleh kosong

Graph biasanya digambarkan dalam bentuk: sekumpulan titik yang menyatakan simpul, yang dihubungkan oleh garis-garis yang menyatakan busur.

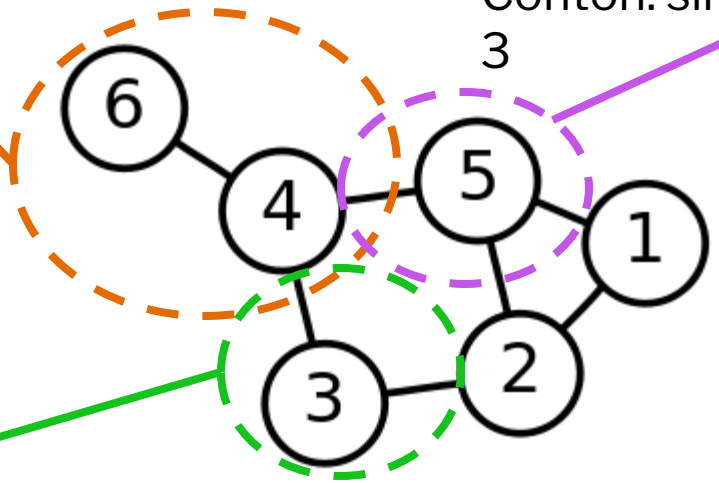
Contoh:



Terminologi

Bertetangga (*adjacent*): dua simpul yang dihubungkan busur disebut bertetangga.
Contoh: simpul ④ dan ⑥ bertetangga.

Derajat (*degree*) dari sebuah simpul adalah jumlah busur yang berhubungan dengan simpul tersebut.
Contoh: simpul ⑤ memiliki derajat 3



Berhubungan (*incident*): sebuah simpul dinyatakan *incident* dengan semua busur yang menghubungkan simpul tersebut dengan simpul lainnya.
Contoh: simpul ③ *incident* dengan busur (2,3) dan (3,4).

Aplikasi

Graph sering digunakan untuk memodelkan keterhubungan antar objek

- Dalam *computer science*, graph digunakan untuk merepresentasikan a.l. jaringan komunikasi, organisasi data, alur komputasi
- Dalam bidang kimia, graph digunakan untuk mensimulasikan struktur atom dan molekul
- Dalam bidang sosiologi, graph digunakan untuk melakukan analisis jaringan sosial baik dalam bentuk *acquintanceship and friendship graphs*, *influence graphs*, atau *collaboration graphs*.
- Dalam bidang biologi, graph dapat digunakan untuk memodelkan jalur migrasi untuk mendapatkan pola perkembangbiakan atau menelusuri penyebaran penyakit.

Variasi Graph

- **Weighted graphs** adalah graph dengan busur yang memiliki bobot/nilai.
- **Directed graphs** adalah graph dengan busur yang memiliki arah. Artinya, jika pada graph terdefinisi busur (a,k) berarti ada busur dengan titik awal simpul a dan menuju simpul k [vs undirected graphs].
- **Simple graphs** adalah undirected graphs (tak berarah) yang tidak memiliki loop (busur dengan kedua ujung pada simpul yang sama) dengan maksimum satu busur antara dua simpul.
- **Regular graphs** adalah graph dengan seluruh simpul yang memiliki jumlah tetangga yang sama (berderajat sama).
- **Complete graphs** adalah graph dengan setiap pasang simpulnya dihubungkan oleh busur.
- **Empty graphs** adalah graph tanpa busur.

Definisi Fungsional

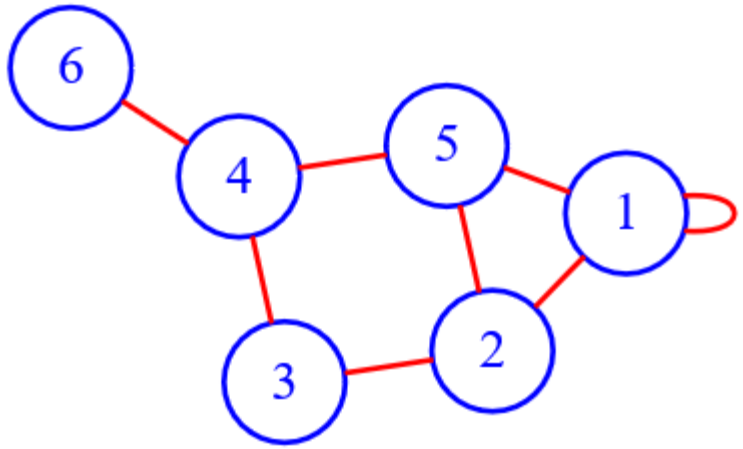
Jika diberikan **G** adalah Graph

```
CreateGraph: V, E → G              { menghasilkan sebuah graph G=(V,E),
                                     V tidak kosong, E bisa kosong }
    IsEmpty: G → boolean           { Tes apakah G adalah graph kosong }
   Adjacent: G, v1, v2 → boolean   { tes apakah v1 dan v2 bertetangga }
   Incident: G, v, e → boolean     { tes apakah v berhubungan dengan e }
  Neighbors: G, v → list of nodes  { daftar seluruh simpul yang bertetangga
                                     dengan v }
       AddV: G, v → G              { menambahkan simpul v pada G }
    DeleteV: G, v → G              { menghapus simpul v dari G,
                                     berikut semua busur incident v }
       AddE: G, v1, v2 → G         { menambahkan busur (v1,v2) pada G }
    DeleteE: G, v1, v2 → G         { menghapus busur (v1,v2) dari G }
```

Representasi Graph

1. Adjacency matrix

   Jika G terdiri dari n buah simpul, maka *adjacency matrix* adalah matriks $n \times n$ dengan $M_{i,j}$ menyatakan jumlah busur antara simpul i dengan simpul j.

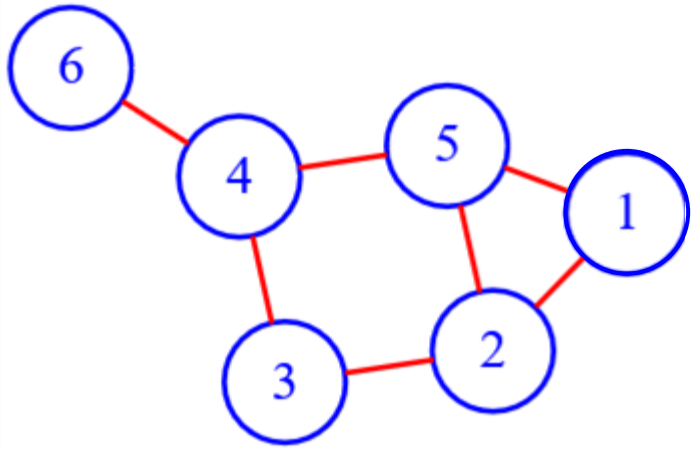


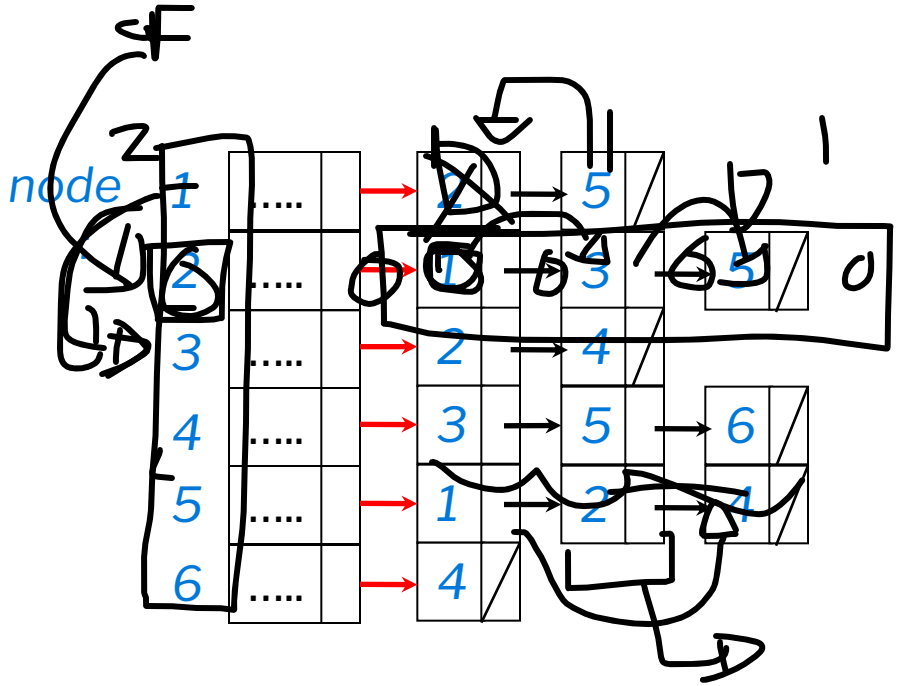
node:	1	2	3	4	5	6
1	1	1	0	0	1	0
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	0
6	0	0	0	1	0	0

Representasi Graph

2. Adjacency list

   Simpul disimpan sebagai objek/*record* yang selain memuat informasi tentang simpul tersebut juga mengandung list dari seluruh simpul yang terhubung dengan simpul tersebut.

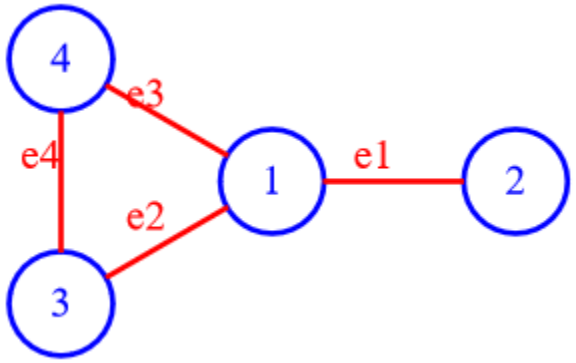




Representasi Graph

3. Incidence matrix

   matriks yang memperlihatkan hubungan antara dua kelompok objek pembangun graph, yaitu simpul sebagai elemen baris dan busur sebagai elemen kolom. $M_{i,j}$ akan bernilai *true* jika ada hubungan antara simpul i dengan busur j.



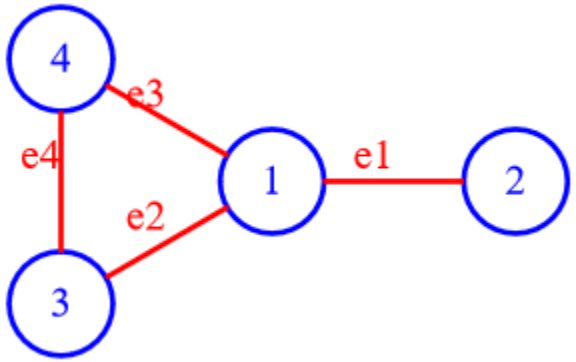
node :	edg e1	e:e2	e3	e4
1	1	1	1	0
2	1	0	0	0
3	0	1	0	1
4	0	0	1	1

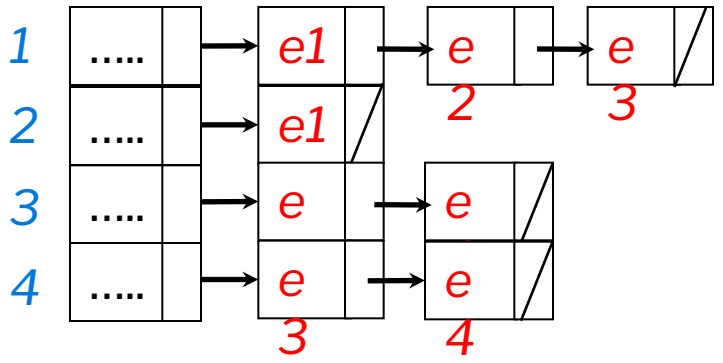
Representasi Graph

4. Incidence list

   Simpul dan busur disimpan sebagai objek/*record* yang memuat informasi tentang simpul/busur tersebut

   Setiap simpul menyimpan list dari busur yang terhubung dengannya

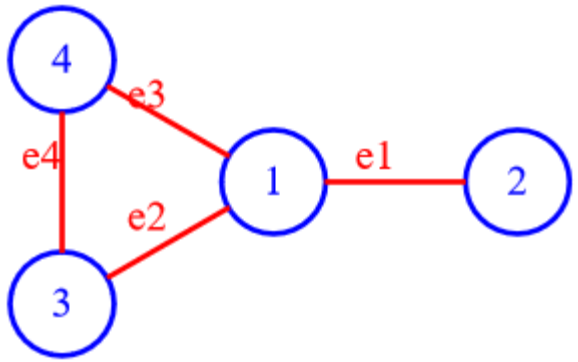




Representasi Graph

5. Edge List

   List sisi adalah suatu tabel pasangan simpul yang membentuk sisi.



Sisi	Simpul-1	Simpul-2
e1	*1*	*2*
e2	*1*	*3*
e3	*1*	*4*
e4	*3*	*4*

Latihan Graph

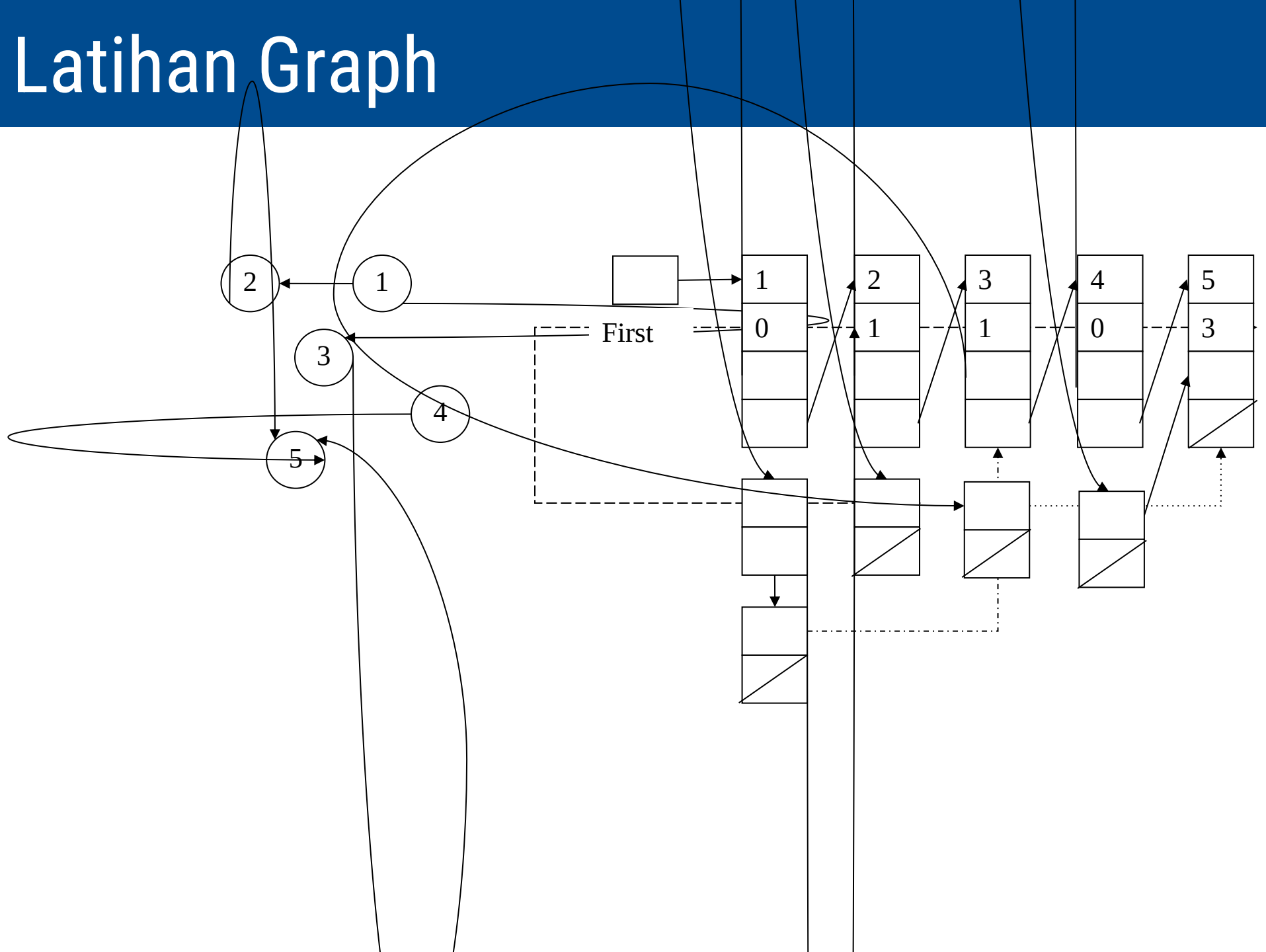
Graph Berarah adalah graph dengan busur yang memiliki arah. Artinya, jika pada graph terdefinisi busur (a,k) berarti ada busur dengan titik awal simpul a (*predecessor*) menuju simpul k (*successor*).

Graph Berarah akan diimplementasikan dengan menggunakan variasi *adjacency list*, dalam bentuk multilist berikut:

List simpul (*leader list*) adalah list dengan elemen seluruh simpul yang terdapat di dalam graph. Setiap elemen list berisi identitas simpul (id), jumlah busur yang masuk ke simpul tersebut (npred), pointer ke list simpul *successor*nya (trail), dan pointer ke elemen simpul berikutnya (next). Implementasi dalam bentuk list dengan pencatatan First, dengan elemen terurut membesar berdasarkan atribut id.

List successor (*trailer list*) adalah list yang berisi pointer ke simpul-simpul yang menjadi successor dari simpul utamanya.

Latihan Graph



```
{ Graph berarah diimplementasi sebagai Multilist }
constant NIL: ...
type AdrNode: pointer to Node
type AdrSuccNode: pointer to SuccNode
type Node: < id: integer,   {identitas simpul}
          nPred: integer,   {jumlah busur yang masuk ke simpul =
                                  jumlah simpul pred}
          trail: AdrSuccNode, {pointer ke list trailer (simpul successor)}
           next: AdrNode >
type SuccNode: < succ: AdrNode,   {address simpul successor}
                 next: AdrSuccNode >
type Graph: < first: AdrNode >
```

```
{ *** Konstruktor *** }
procedure CreateGraph (input x: integer, output g: Graph)
{ I.S. Sembarang;  F.S. Terbentuk Graph dengan satu simpul dengan Id=X }

{ *** Manajemen Memory List Simpul (Leader) *** }
function newGraphNode(x: integer) → AdrNode
{ Mengembalikan address hasil alokasi Simpul x. }
{ Jika alokasi berhasil, maka address tidak NIL, misalnya
  menghasilkan p, maka p↑.id=x, p↑.nPred=0, p↑.trail=NIL,
  dan p↑.next=NIL. Jika alokasi gagal, mengembalikan NIL. }

procedure deallocGraphNode (input p: AdrNode)
{ I.S. P terdefinisi; F.S. P dikembalikan ke sistem }

{ *** Manajemen Memory List Successor (Trailer) *** }
function newSuccNode (pn: AdrNode) → AdrSuccNode
{ Mengembalikan address hasil alokasi. }
{ Jika alokasi berhasil, maka address tidak NIL, misalnya
  menghasilkan pt, maka pt↑.succ=pn dan pt↑.next=NIL. Jika
  alokasi gagal, mengembalikan NIL. }

procedure deallocSuccNode (input p: AdrSuccNode)
{ I.S. p terdefinisi; F.S. p dikembalikan ke sistem }
```

```
{ *** Fungsi/Prosedur Lain *** }
function searchNode (g: Graph, x: integer) → AdrNode
{ mengembalikan address simpul dengan id=x jika sudah ada pada graph g,
  NIL jika belum }

function searchEdge (g: Graph, prec: integer, succ: integer) → adrSuccNode
{ mengembalikan address trailer yang menyimpan info busur <prec,succ>
  jika sudah ada pada graph g, NIL jika belum }

procedure insertNode (input/output g: Graph, input x: integer, output pn: adrNode)
{ Menambahkan simpul x ke dalam graph g, jika alokasi x berhasil. }
{ I.S. g terdefinisi, x terdefinisi dan belum ada pada g. }
{ F.S. Jika alokasi berhasil, x menjadi elemen terakhir g, pn berisi
       address simpul x. Jika alokasi gagal, g tetap, pn berisi NIL }

procedure insertEdge (input/output g: Graph, input prec, succ: integer)
{ Menambahkan busur dari prec menuju succ ke dalam G }
{ I.S. g, prec, succ terdefinisi. }
{ F.S. Jika belum ada busur <prec,succ> di g, maka tambahkan busur
       <prec,succ> ke g. Jika simpul prec/succ belum ada pada g,
       tambahkan simpul tersebut dahulu. Jika sudah ada busur <prec,succ>
       di g, maka g tetap. }
```

```
procedure deleteNode (input/output g: Graph, input x: integer)
{ Menghapus simpul x dari g }
{ I.S. g terdefinisi, x terdefinisi dan ada pada g, jumlah simpul
       pada g lebih dari 1. }
{ F.S. simpul x dan semua busur yang terhubung ke x dihapus
       dari g. }
```

Implementasikan primitif **searchNode**, **searchEdge**, **insertNode**, **insertEdge**, dan **deleteNode** yang ada sesuai definisi dan spesifikasi di atas, tanpa menulis ulang spesifikasinya. Tidak boleh membuat fungsi/prosedur baru.