



Implementasi Struktur Data Pada *Graph*

(TIB11 – Struktur Data)

Pertemuan 25, 26

Sub-CPMK

- Mahasiswa dapat menerapkan representasi *graph* dalam bentuk *linked list* dan *array*, serta melakukan penelusuran *graph* (C3, A3)

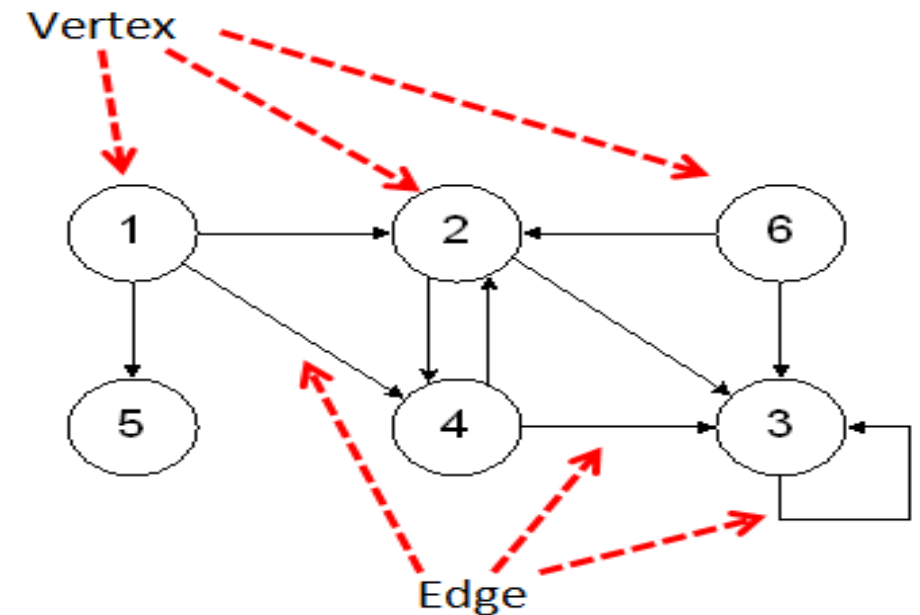
Materi:

1. Pengertian *Graph*
2. *Adjacency Matrix*
3. *Adjacency List*
4. Penelusuran *Graph*



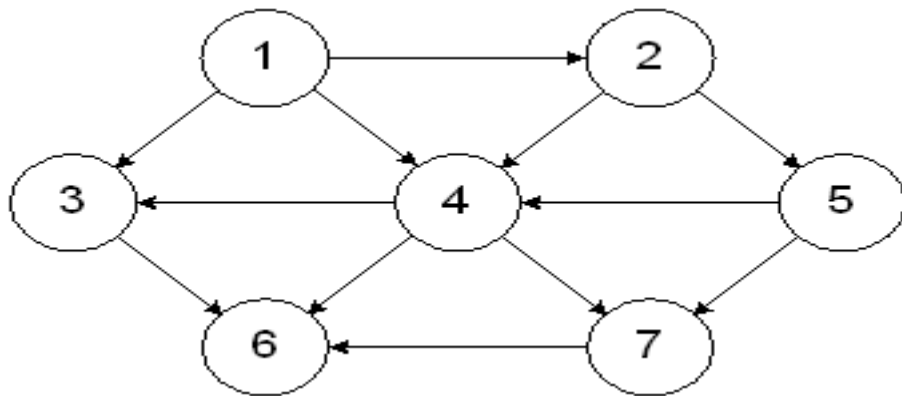
1. Pengertian *Graph*

- *Graph* terdiri dari satu *set* verteks dan satu *set* *edge*.
- Verteks adalah simpul yang membuat *graph*
- *Edge* adalah garis yang menghubungkan verteks-verteks

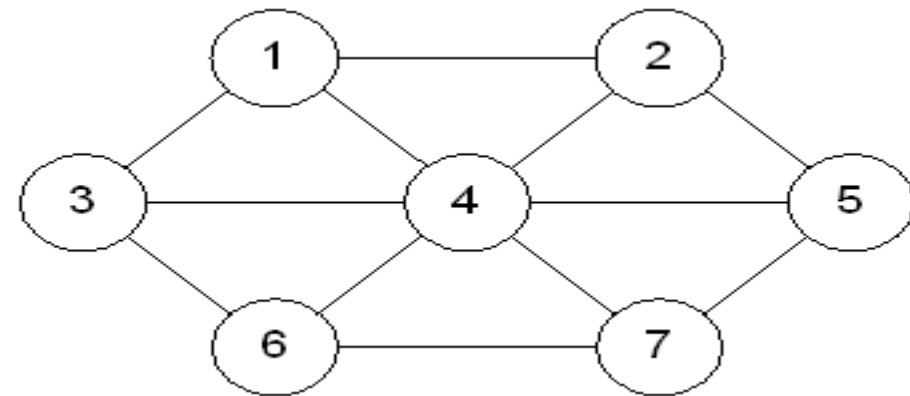


1.1. Dua Macam *Graph*

- *Graph Berarah (Directed Graph/digraph)*
- *Graph tak berarah (Undirected Graph)*



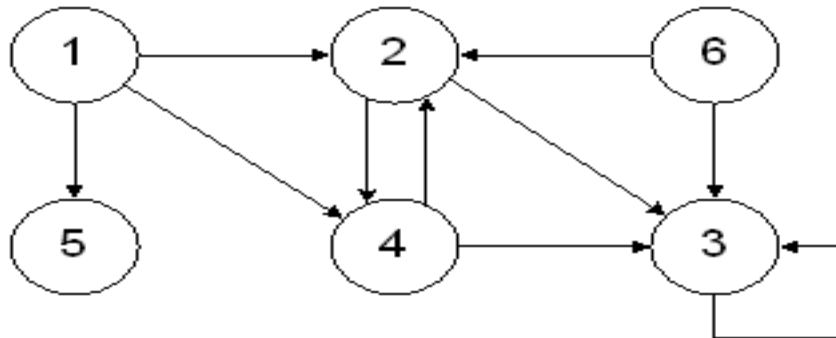
Graph Berarah



Graph Tak Berarah

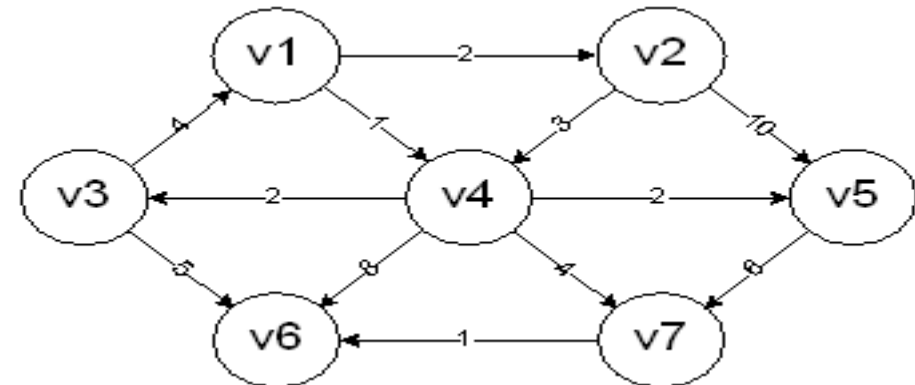
1.2. Graph Berarah

- Sebuah *graph* berarah atau **directed graph** atau **digraph**, adalah *graph* yang setiap verteks nya terhubung dengan arah yang spesifik
- Contoh disamping adalah sebuah *digraph* dengan enam verteks yang terdiri dari verteks {1, 2, 3, 4, 5, 6}
- Dari enam verteks tersebut memiliki sepuluh edges berarah yang terdiri dari {1 ke 2, 1 ke 4, 1 ke 5, 2 ke 3, 2 ke 4, 3 dirinya sendiri, 4 ke 2, 4 ke 3, 6 ke 2, 6 ke 3}



1.3. *Graph* Dengan Harga Pada *Edge*

- *Edge* yang menghubungkan verteks-verteks dapat mempunyai nilai
- Representasi pada *Adjacency* Matriks, sel verteks i ke verteks j dapat diisi dengan nilai pada *edge* tersebut



1.4. Tree Pada Digraph

Tree adalah kasus khusus dari sebuah *digraph* berdasarkan karakteristik:

- salah satu dari verteks yaitu *root* tidak memiliki *edge* yang masuk
- setiap verteks selain *root* dapat dicapai dengan kriteria hanya boleh dilalui satu kali secara berurutan

1.5. Perbedaan *Digraph* dengan *Tree*

- tidak perlu memiliki *root node*
- Mungkin ada beberapa jalur (atau tidak) dari satu titik ke titik yang lain
- diproses secara berbeda.

Penyisipan misalnya, untuk menambahkan simpul ke pohon, bidang tautan harus ditambahkan ke simpul induk. Pada *digraph* simpul dapat dimasukkan di mana saja, namun tidak perlu ada busur dari atau ke sana; atau *edge* bisa disisipkan di antara dua *node* yang ada.

1.6. Beberapa Definisi Yang Diasosiasikan Dengan *Graph*

- Dua verteks dikatakan berdampingan (*adjacent*) jika ada *edge* yang menghubungkan
- Lintasan dari sebuah *graph* adalah urutan dari verteks dan *edge*
- Panjang dari sebuah lintasan adalah jumlah *edge* pada lintasan, setara dengan jumlah verteks - 1
- *Loop* adalah lintasan yang dibuat dari *edge* yang mengarah ke verteks itu sendiri
- Lintasan sederhana adalah lintasan dimana verteks yang dilaluinya berbeda kecuali yang pertama dan terakhir

1.6. Beberapa Definisi Yang Diasosiasikan Dengan *Graph* (Lanj.)

- Siklus adalah jalur yang paling sedikit panjang 1 dimana simpul pertama dan terakhir sama dengan tepi yang berbeda untuk grafik yang tidak berarah.
- Sebuah *digraph* adalah sebuah non siklus *graph*
- Sebuah *digraph* dikatakan terhubung secara kuat (*strongly connected*) jika ada sebuah *path* dari setiap verteks ke setiap verteks lainnya
- *Complete Graph* adalah *graph* yang memiliki sebuah *edge* diantara setiap pasangan pasangan verteks

1.7. Representasi *Graph* Pada Struktur Data

Dapat direpresentasikan dengan

- *Adjacency Matrix* (ada yang menyebutnya *adjacency Table*)
 - Matriks biasanya diterapkan menggunakan *Array* 2 Dimensi
 - Ruang *memory* yang diperlukan adalah V^2 , dimana V adalah jumlah verteks
- *Adjacency List*
 - Diterapkan dengan *linked list* atau paduan *Array* 1 dimensi dengan *linked list*
 - Ruang *memory* yang dibutuhkan adalah $E+V$, E adalah jumlah *edge* dan V jumlah verteks.



2. Adjacency Matrix

Cara membentuk *Adjacency matrix*

- Jumlah verteks dari digraph 1, 2, ..., n
- Bentuk sebuah matriks $n \times n$
- Untuk setiap *entry* baris i dan kolom j , sisipkan sebuah angka 1 jika ada *edge* dari verteks i ke verteks j ; jika tidak sisipkan 0

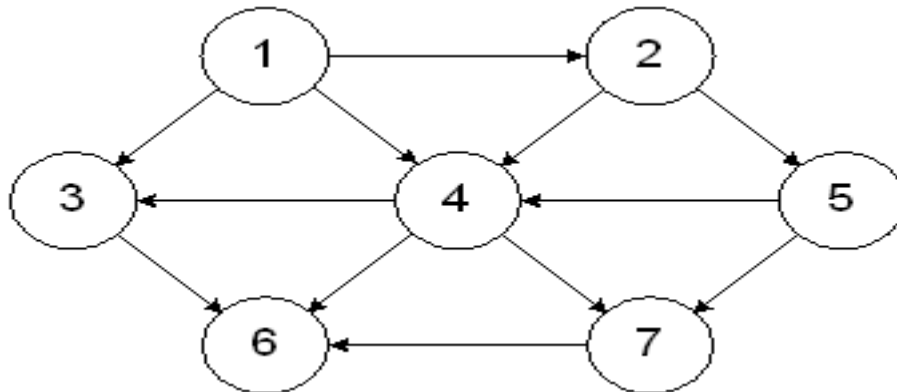
Catatan:

- *untuk edge dengan nilai masukkan nilai edge*
- *Pada beberapa penerapan, verteks yang tidak terhubung sering kali diisi dengan nilai \sim (tak berhingga) atau nilai sangat tinggi yang melebihi range nilai yang ada*

2.1. *Adjacency* Matriks Pada *Directed Graph*

- Hanya diisi nilai yang memiliki *edge* dari verteks i ke verteks j
- Antara baris dengan kolom tidak simetris, karena adanya arah pada *edge* nya

2.2. Contoh *Adjacency Matrix* Untuk *Directed Graph*

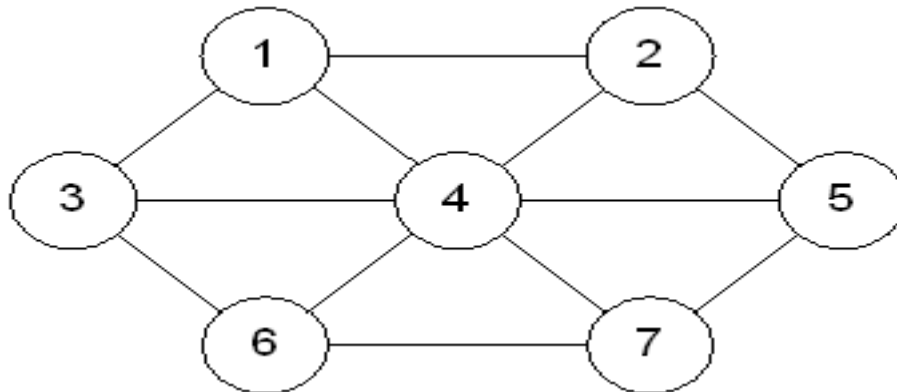


| | [1] | [2] | [3] | [4] | [5] | [6] | [7] |
|-----|-----|-----|-----|-----|-----|-----|-----|
| [1] | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| [2] | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| [3] | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| [4] | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| [5] | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| [6] | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| [7] | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

2.3. *Adjacency* Matriks Pada *Graph* Tak Berarah

- Antara baris dengan kolom simetris, karena tidak adanya arah pada edge nya
- Setiap verteks i dan verteks j yang terhubung akan mempunyai nilai

2.4. Contoh *Adjacency* Matriks Untuk *Undirected Graph*



| | [1] | [2] | [3] | [4] | [5] | [6] | [7] |
|-----|-----|-----|-----|-----|-----|-----|-----|
| [1] | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| [2] | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| [3] | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| [4] | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| [5] | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| [6] | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| [7] | 0 | 0 | 0 | 1 | 1 | 1 | 0 |

2.5. Kerugian dan Keuntungan Adjacency Matriks

Kerugian

- Terpakai atau tidak terpakai, ruang *memory* harus dialokasikan. Sehingga akan banyak yang terisi dengan 0 atau *mirroring* dari diagonalnya pada *Undirected Graph*

Keuntungan

- Lebih mudah mengaksesnya, cukup menggunakan *nested loop* per baris untuk tiap kolomnya



3. Adjacency List

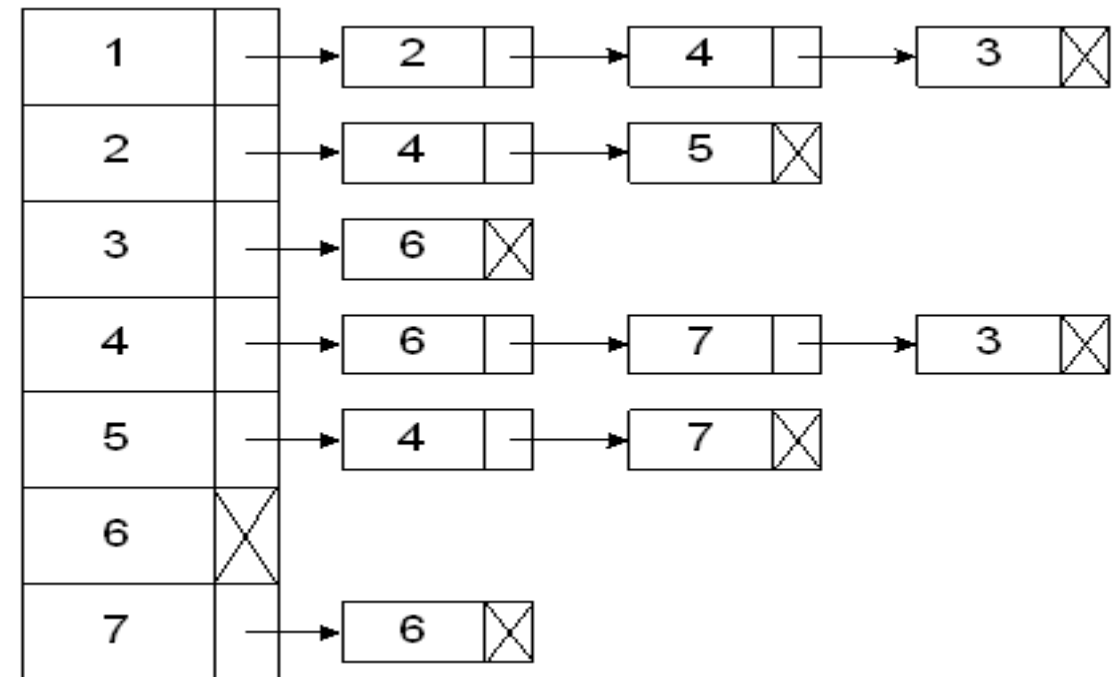
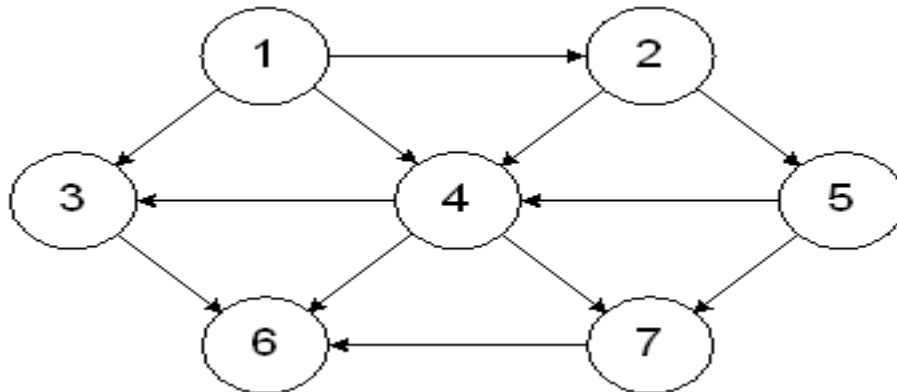
3.1. *Adjacency List* Dengan *Linked List*

- *List* Vertex direpresentasikan sebagai *node*/simpul secara berurutan
- Pada tiap simpul *list* verteks terdapat dua *link*, *link* ke node urutan berikutnya dan *link* ke salah satu verteks tujuan
- Verteks tujuan digambarkan sebagai *node*/simpul dengan satu *link* ke *node*/simpul tujuan lain dan disusun secara berurutan

3.2. *Adjacency List* Dengan Paduan *Array* dan *Linked List*

- *List Vertex* direpresentasikan dengan *Array* satu dimensi secara berurutan
- Pada tiap simpul *array* terdapat sebuah *link* ke salah satu verteks tujuan
- Verteks tujuan digambarkan sebagai *node*/simpul dengan satu *link* ke *node*/simpul tujuan lain dan disusun secara berurutan

3.3. Contoh *Adjacency List*





4. Penelusuran *Graph*

Dua macam penelusuran

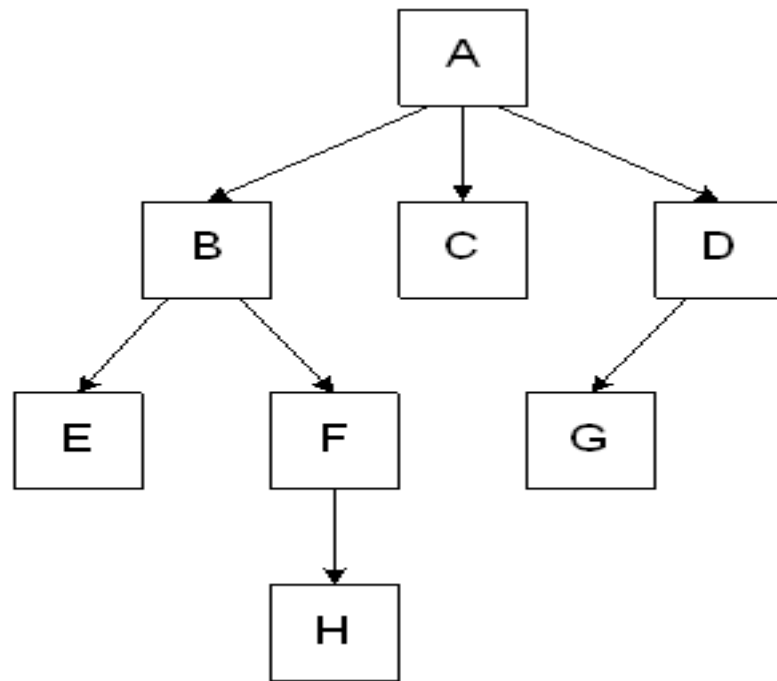
- *Depth First Search*
- *Breadth First Search*

4.1. *Depth-first Search*

Pencarian *graph* terarah yang dimulai dari simpul yang dipilih dan diarahkan oleh busur sebagai "terdalam" sebanyak mungkin untuk mengunjungi simpul yang dapat dijangkau simpul yang belum pernah dikunjungi. Setelah akhir rantai tercapai, lakukan *backtrack* / langkah mundur ke simpul sebelumnya dan lanjutkan pencarian.

4.1.1. Contoh Pada Non Siklus Graph

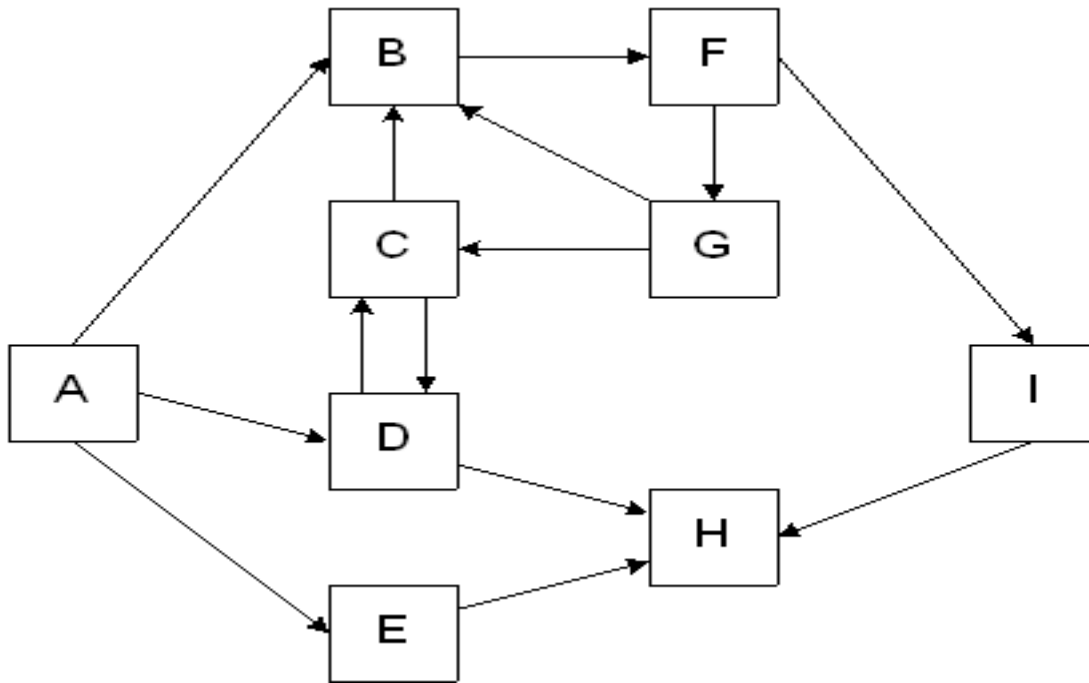
- Hasil Penelusuran:
A, B, E, F, H, C, D, G



4.1.2. Contoh Pada *Graph* Dengan Siklus

Hasil Penelusuran

- Jika dimulai dari A
A, B, F, I, H, G, C, D, E
- Jika dimulai dari B
B, F, I, H, G, C, D



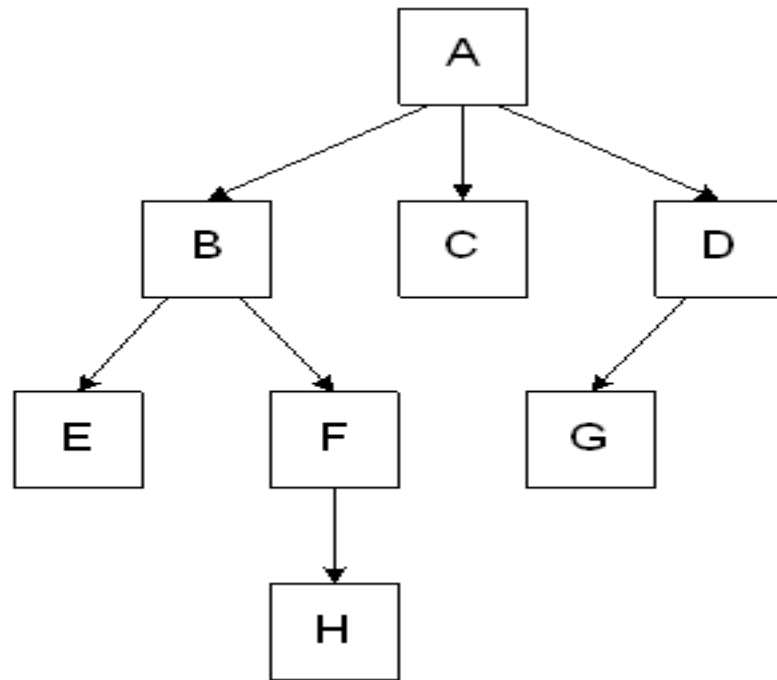
4.2. *Breadth-first Search*

- Pencarian *graph* berarah secara “melebar”, yang dimulai dari simpul yang dipilih dan kemudian semua simpul yang berdekatan dikunjungi. Setelah selesai, simpul pertama yang berdekatan menjadi titik awal yang baru, dan semua simpulnya yang berdekatan dikunjungi selama belum pernah dikunjungi sebelumnya.

4.2.1. Contoh Pada Non Siklus Graph

Hasil:

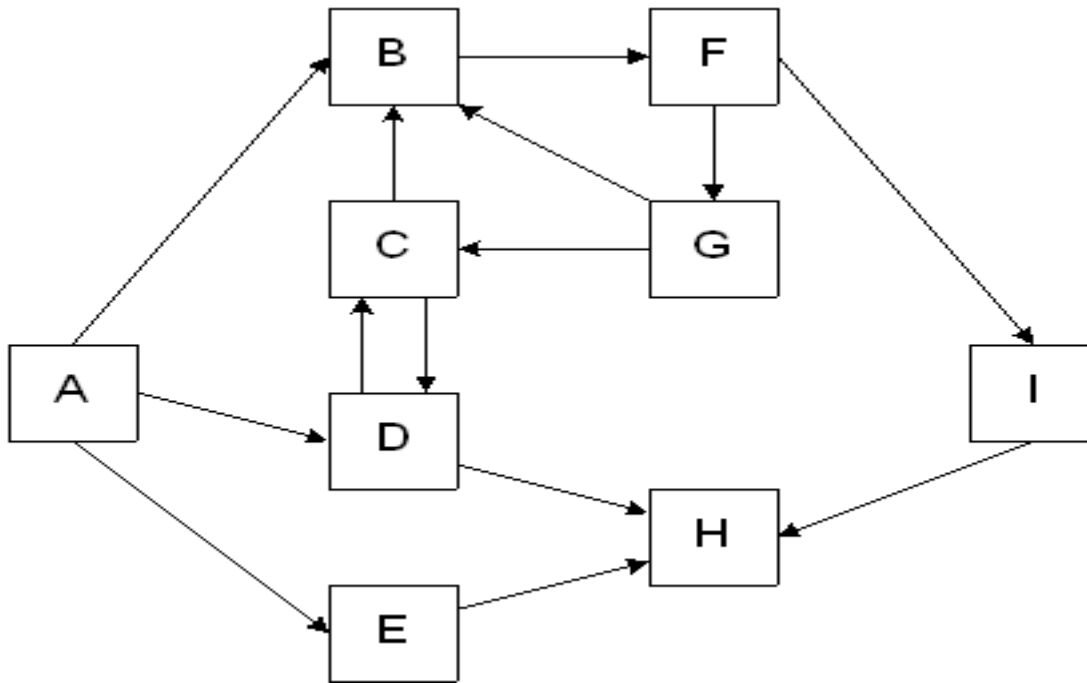
- A, B, C, D, E, F, G, H



4.2.2. Contoh Pada Siklus Graph

Hasil:

- Jika dimulai dari A
A, B, D, E, F, C, H, G, I
- Jika dimulai dari B
B, F, G, I, H, C, D



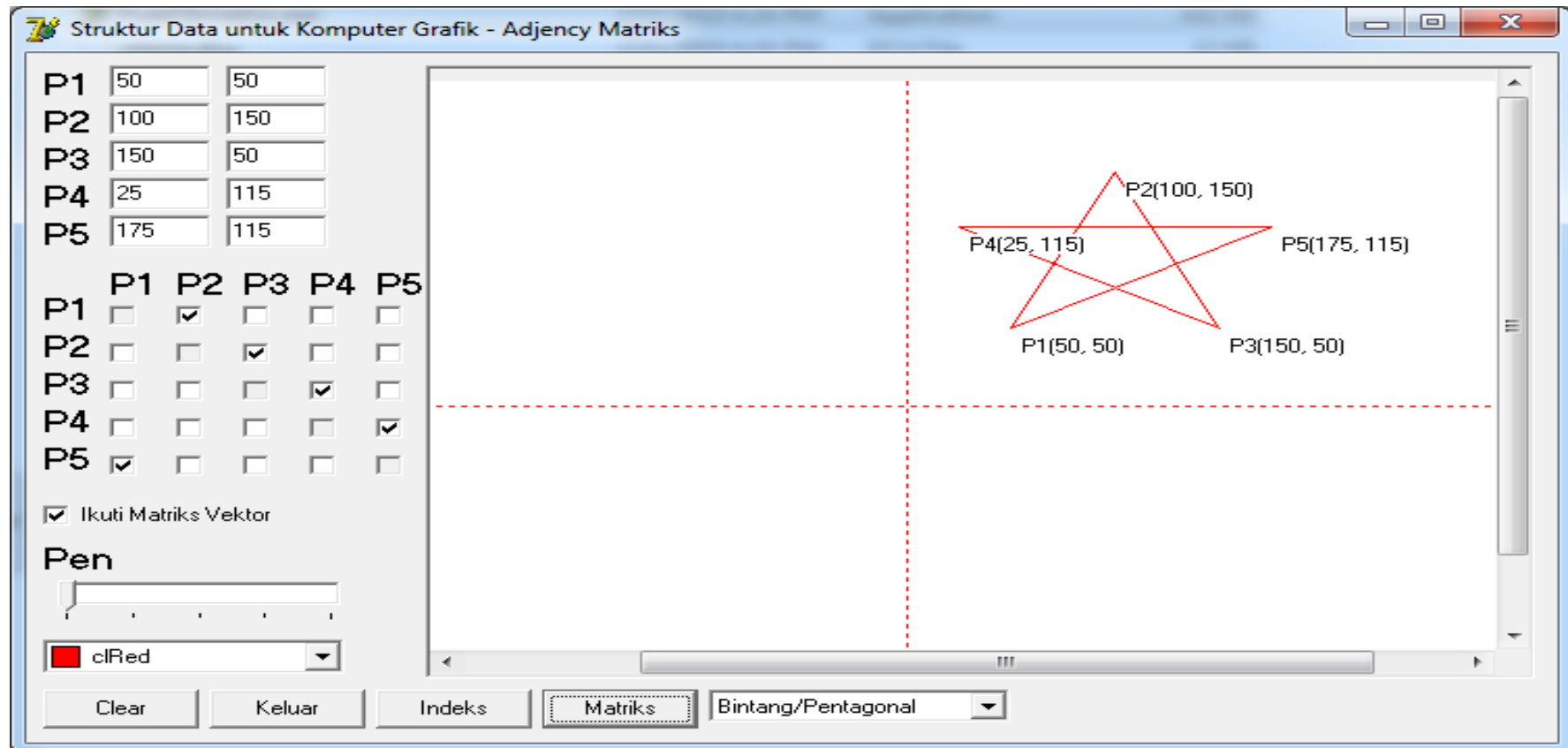
4.3. Pencarian Lintasan Terpendek

- Kasus yang sering muncul pada *Graph* adalah kasus pencarian lintasan terpendek berdasarkan harga pada *edge-edge* nya
- Salah satu algoritma pencarian lintasan terpendek adalah algoritma Dijkstra, berikut ini algoritma pencarian lintasan terpendek yang cukup terkenal, semua algoritma ini sudah dibahas pada mata kuliah matematika diskrit dan teori graph
 1. *Unweighted shortest path*
 2. *Dijkstra's algorithm*
 3. *Graphs with negative edge costs*
 4. *Acyclic graphs*

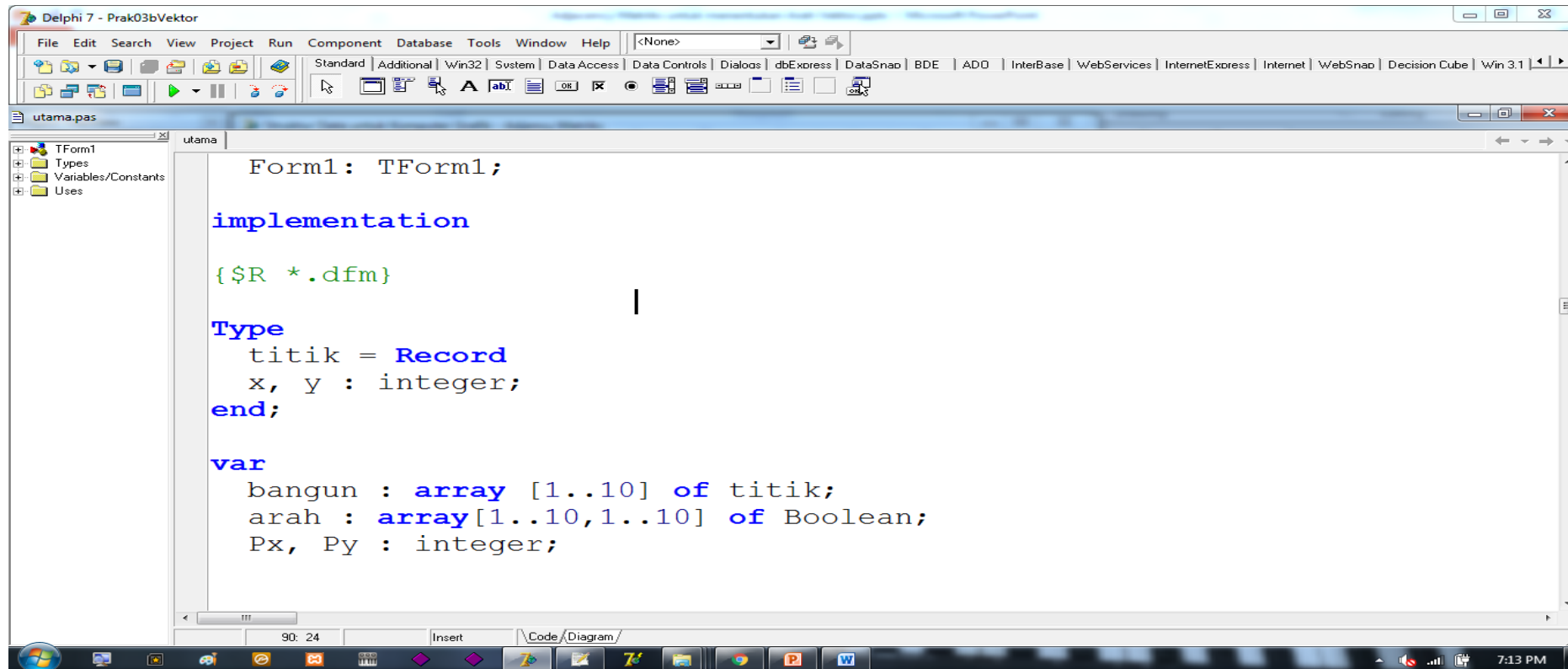
Ringkasan

- *Graph* terdiri dari satu set verteks dan satu set edge.
- Verteks adalah simpul yang membuat graph
- *Edge* adalah garis yang menghubungkan verteks-verteks
- *Graph* berarah atau *directed graph* atau *digraph*, adalah *graph* yang setiap verteks nya terhubung dengan arah yang spesifik
- *Graph* Pada Struktur Data terdiri dari ***Adjacency Matrix*** (ada yang menyebutnya *adjacency Table*) biasanya diterapkan menggunakan *Array* 2 Dimensi dan ***Adjacency List*** Diterapkan dengan *linked list* atau paduan *Array* 1 dimensi dengan *linked list*

Contoh Penentuan Arah Menggunakan Adjacency Matriks



Contoh Deklarasi Record Dan Matriks Arah (dalam Pemrograman Delphi)



```
Delphi 7 - Prak03bVektor
File Edit Search View Project Run Component Database Tools Window Help
Standard Additional Win32 System Data Access Data Controls Dialogs dbExpress DataSnap BDE ADO InterBase WebServices InternetExpress Internet WebSnap Decision Cube Win 3.1
utama.pas
TForm1
Types
Variables/Constants
Uses

Form1: TForm1;

implementation

{$R *.dfm}

Type
    titik = Record
        x, y : integer;
    end;

var
    bangun : array [1..10] of titik;
    arah : array[1..10,1..10] of Boolean;
    Px, Py : integer;
```

Contoh penerapan Arah dengan Adjacency Matriks

- Matriks arah dapat dinyatakan dengan array 2D.

```
for n:=1 to 5 do
begin
  for m:=1 to 5 do
  begin
    if arah[m,n] then
    begin
      Image1.Canvas.MoveTo (Px+bangun[m].x, Py-bangun[m].y) ;
      Image1.Canvas.LineTo (Px+bangun[n].x, Py-bangun[n].y) ;
    end;
  end;
end;
```



Terimakasih

TUHAN Memberkati Anda

Teady Matius Surya Mulyana (tmulyana@bundamulia.ac.id)