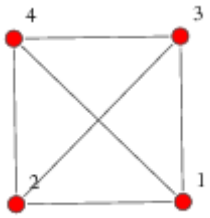
MODUL 4 ASD 2014 : Graph

Secara sederhana graf didefinisikan sebagai kumpulan titik yang dihubungkan oleh garis. Titik tersebut disebut dengan vertex dan garis disebut dengan edge. Terdapat dua jenis graph, yaitu graph tak-berarah dan graph berarah. Graph tak-berarah (*undirected graph*) adalah graf yang tidak memiliki orientasi arah pada setiap edge yang dimiliki. Edge $e = (u,v)$, dimana e adalah edge yang menghubungkan vertex u dan vertex v sama saja dengan edge $e = (v,u)$. Graf berarah (*directed graph*/ *digraph*) adalah graf yang memiliki orientasi arah pada setiap edge yang dimiliki. Sehingga, edge $e = (u,v)$ untuk e yang menghubungkan vertex u dan vertex v berbeda maknanya dengan edge $e = (v,u)$ yang menghubungkan vertex v dan vertex u.

Adjacency Matrix



$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Contoh 1.1 Adjacency-Matrix

Gambar diatas adalah contoh pengaplikasian graph ke dalam bentuk matrix. Pada gambar diatas graph yang dibentuk adalah undirected graph. Edge yang terbentuk secara bolak-balik. Pada contoh 1.1 kita bisa melihat vertex 1 terhubung dengan vertex 3 sehingga pada matrix baris 1 kolom 3 bernilai 1, begitu juga sebaliknya pada baris 3 kolom 1 bernilai 1. Melalui ini kita dapat menyebut bahwa baris sebagai vertex asal dan kolom sebagai vertex tujuan.

Menyiapkan matrix yang akan dibuat graph:

```
int edges,i,j,tail,head,adj[10][10],n;

printf("Enter number of nodes : ");
scanf("%d",&n);

for(i=1;i<=n;i++)
{
    for(j=1;j<=n;j++)
    {adj[i][j]=0;}
}

edges=n*(n-1)/2; //maksimum jumlah edges
```

Memasukkan edge yang menghubungkan matrix:

```
printf("\nEnter -1 -1 to exit\n\n");
for(i=1;i<=edges;i++)
{
    printf("Edge %d : ",i);
    scanf("%d %d",&tail,&head);
    if(tail==-1||head==-1) break;
    if(tail>n||head>n||tail<=0||head<=0)
    {
        printf("Invalid edge!\n");
        i--;
    }
    else
    {
        adj[tail][head]=1;
        adj[head][tail]=1;
    }
}
```

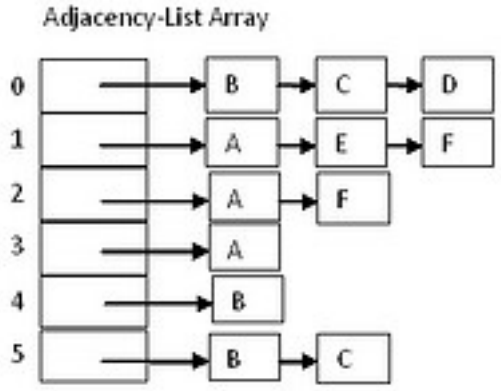
Mencetak graph yang telah dibuat dalam bentuk matrix:

```
printf("\nAdjacency Matrix \n\n");
for(i=1;i<=n;i++)
{
    for(j=1;j<=n;j++)
    {
        printf("\t%d",adj[i][j]);
    }
    printf("\n");
}
```

Jika program dijalankan:

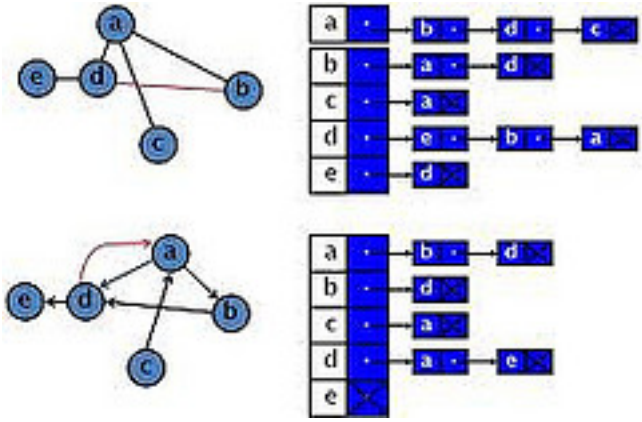
```
Enter number of nodes : 5

Enter -1 -1 to exit

Edge 1 : 1 2
Edge 2 : 1 3
Edge 3 : 2 4
Edge 4 : 2 5
Edge 5 : -1 -1

Adjacency Matrix

        0       1       1       0       0
        1       0       0       1       1
        1       0       0       0       0
        0       1       0       0       0
        0       1       0       0       0

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Adjacency List



Contoh 2.1 Adjacency-List

Seluruh vertex disimpan dalam bentuk array yang berisi data dan link. Link disini berfungsi sebagai edge dari vertex menuju vertex berikutnya. Berikut ini contoh lebih jelas dari adjacency list. Pada gambar berikut bisa kita lihat kedua graph memiliki 5 vertex. Kedua graph ini disebut "undirected graph", dimana edge yang menghubungkan vertex tidak memiliki orientasi (bisa bolak-balik) .



Contoh 2.2 Adjacency-List

Pendefinisian vertex pada graph:

```
struct vertex
{
    int vertexKey;
    struct edge *edgePtr;
}vertex;
```

Note:

- vertexKey adalah variabel berisi nama dari vertex (disini berupa integer)
- edgePtr adalah pointer dari edge (sebagai pointer next)

Pendefinisian edge pada graph:

```
struct edge
{
    int vertexIndex;
    struct edge *edgePtr;
}edge;
```

Note:

- vertexIndex adalah variabel berisi vertex tujuan

Deklarasi graph:

```
struct vertex graph[10];
int vertexCount=0;
```

Note: Disini kita membuat maksimum dari vertex berjumlah 10

Membuat vertex baru:

```
void InsertVertex(int vertexKey)
{
    graph[vertexCount].vertexKey=vertexKey;
    graph[vertexCount].edgePtr=NULL;
    vertexCount++;
}
```

Membuat edge baru:

```
void insertEdge(int vertex1, int vertex2)
{
    struct edge *e,*e1,*e2;
    e=graph[vertex1].edgePtr;
    while(e&& e->edgePtr)
    {
        e=e->edgePtr;
    }
    e1=(struct edge *)malloc(sizeof(*e1));
    e1->vertexIndex=vertex2;
    e1->edgePtr=NULL;
    if(e)
    e->edgePtr=e1;
    else
    graph[vertex1].edgePtr=e1;

    e=graph[vertex2].edgePtr;
    while(e&& e->edgePtr)
    {
        e=e->edgePtr;
    }
    e2=(struct edge *)malloc(sizeof(*e2));
    e2->vertexIndex=vertex1;
    e2->edgePtr=NULL;
    if(e)
    e->edgePtr=e2;
    else
    graph[vertex2].edgePtr=e2;
}
```

Note:

- edgePtr yang dimasukkan ke dalam vertex harus menunjuk ke NULL

Mencetak graph:

```
void printGraph()
{
    int i;
    struct edge *e;
    for(i=0;i<vertexCount;i++)
    {
        printf("%d(%d)",i,graph[i].vertexKey);
        e=graph[i].edgePtr;
        while(e)
        {
            printf("->%d",e->vertexIndex);
            e=e->edgePtr;
        }
        printf("\n");
    }
}
```

Jika diberi input:

```
main()
{
    InsertVertex(5);
    InsertVertex(3);
    InsertVertex(4);
    InsertVertex(2);
    InsertVertex(9);
    insertEdge(0,1);
    insertEdge(0,2);
    insertEdge(1,3);
    insertEdge(1,4);
    printGraph();
    return 0;
}
```

Maka output:

```
0(5)->1->2
1(3)->0->3->4
2(4)->0
3(2)->1
4(9)->1

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

BFS (Breadth First Search)

Pada metode BFS, semua node pada level n akan dikunjungi terlebih dahulu sebelum mengunjungi node-node pada level n+1. Pencarian dimulai dari node akar terus ke level ke-1 dari kiri ke kanan, kemudian berpindah ke level berikutnya demikian pula dari kiri ke kanan hingga ditemukannya solusi.

DFS (Depth First Search)

Pada DFS, proses pencarian akan dilakukan pada semua anaknya sebelum dilakukan ke node-node yang selevel. Pencarian dimulai dari node akar ke level yang lebih tinggi. Proses ini diulangi terus hingga ditemukannya solusi.

Struktur Data yang digunakan

```
vector<vector<int> > graph;
vector<int> visited;
```

Representasi Graph Dengan Adjacency List & Fungsi Main

```
int main()
{
    int V , E , a , b , start;

    scanf("%d%d",&V,&E);

    for(int i=0 ; i<=V ; i++){
        graph.push_back(vector<int>());
        visited.push_back(0);
    }

    for(int i=0 ; i<E ; i++){
        scanf("%d%d",&a,&b);
        graph[a].push_back(b);
        graph[b].push_back(a);
    }

    scanf("%d",&start);
    bfs(start);
    puts("");
    dfs(start);
    return 0;
}
```

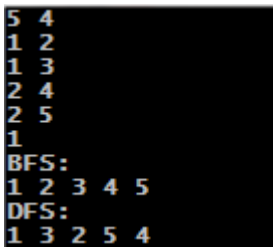
Fungsi BFS

```
void bfs(int start)
{
    queue<int> q;

    for(int i=1 ; i<=graph.size() ; i++)
        visited[i] = 0;

    q.push(start);
    while(!q.empty()){
        int now = q.front();
        q.pop();
        if(visited[now] == 0){
            printf("%d ",now);
            visited[now] = 1;
            for(int i=0 ; i<graph[now].size() ; i++){
                int next = graph[now][i];
                q.push(next);
            }
        }
    }
}
```

Fungsi DFS

```
void dfs(int start)
{
    stack<int> s;

    for(int i=1 ; i<=graph.size() ; i++)
        visited[i] = 0;

    s.push(start);
    while(!s.empty()){
        int now = s.top();
        s.pop();
        if(visited[now] == 0){
            printf("%d ",now);
            visited[now] = 1;
            for(int i=0 ; i<graph[now].size() ; i++){
                int next = graph[now][i];
                s.push(next);
            }
        }
    }
}
```

Contoh Jalan Program:



SOAL SHIFT

1. Terdapat 5 saklar, masing-masing memiliki status on/off (1/0). Jika salah satu ditekan maka status saklar yang ditekan tidak berubah sedangkan saklar lainnya akan berubah statusnya, yang tadinya on menjadi off dan begitu juga sebaliknya. Temukan langkah minimal untuk mengubah konfigurasi awal saklar menjadi konfigurasi akhir saklar yang diberikan.

Input :

Baris pertama terdapat 5 buah integer yang terdiri dari angka 1 dan 0 yang menyatakan status setiap saklar pada konfigurasi awal. aris kedua terdapat 5 buah integer yang terdiri dari angka 1 dan 0 yang menyatakan konfigurasi saklar akhir.

Output :

Keluarkan langkah minimal (jumlah penekanan saklar) untuk membuat konfigurasi saklar menjadi konfigurasi akhir dan Keluarkan 'Mustahil' apabila tidak ada cara untuk membuat konfigurasi saklar tersebut.

Contoh input:

1 1 0 0 0

0 0 0 0 0

Contoh output :

2

Contoh Input :

0 0 0 0 0

1 1 1 1 1

Contoh output :

Mustahil

2. "TC Travel " melayani perjalanan dari beberapa kota di Jawa Timur. untuk menghemat biaya pengeluaran, maka pihak TC Travel perlu mengetahui apakah suatu perjalanan dapat ditempuh dengan (tidak lebih dari) X jalur saja.

input:

N

S-1 D-1

S-2 D-2

.

.

.

SN DN

T

A-1 B-1 X-1

A-2 B-2 X-2

.

.

.

S-T B-T X-T

N: jumlah jalur

S-i D-i: ada jalur dari S-i ke D-i

T: jumlah testcase

A-i B-i X-i: apakah mungkin dari A-i ke B-i dapat dilewati dengan jumlah jalur <= X-i ?

contoh input:

6

1 2

1 5

2 3

2 5

3 4

4 5

3

1 2 1

5 3 1

1 4 2

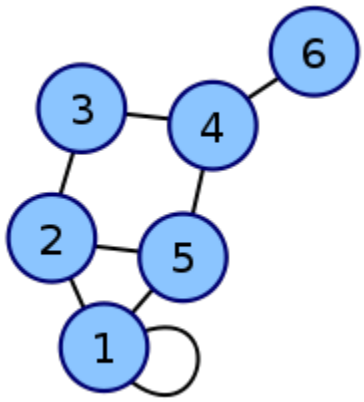
contoh output:

YA

TIDAK

YA

penjelasan:



dari kota 1 ke kota 2, dapat dilewati dengan kurang dari / sama dengan 1 jalur? YA (jalurnya: 1-2)

dari kota 5 ke kota 3, dapat dilewati dengan kurang dari / sama dengan 1 jalur? TIDAK (jalur paling minimumnya: 5-2-3 atau 5-4-3, 2 jalur)

dari kota 1 ke kota 4, dapat dilewati dengan kurang dari / sama dengan 2 jalur? YA (jalurnya: 1-5-4, 2 jalur)