Penyajian Graph

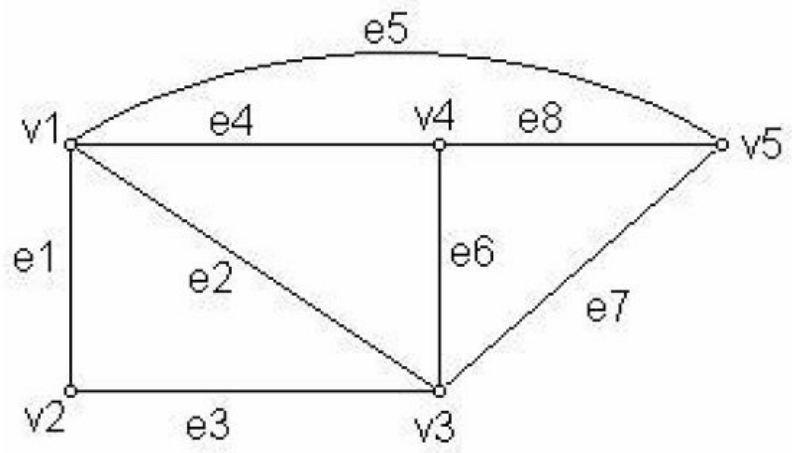
Graph Representasi Statis

- Implementasi dengan membuat matriks 2D menggunakan array 2D

Membuat matriks ruas

- Menyimpan ruas jalur dengan ukuran matriks 2xn, n adalah jumlah ruas.

```
int graph[2][8];
```



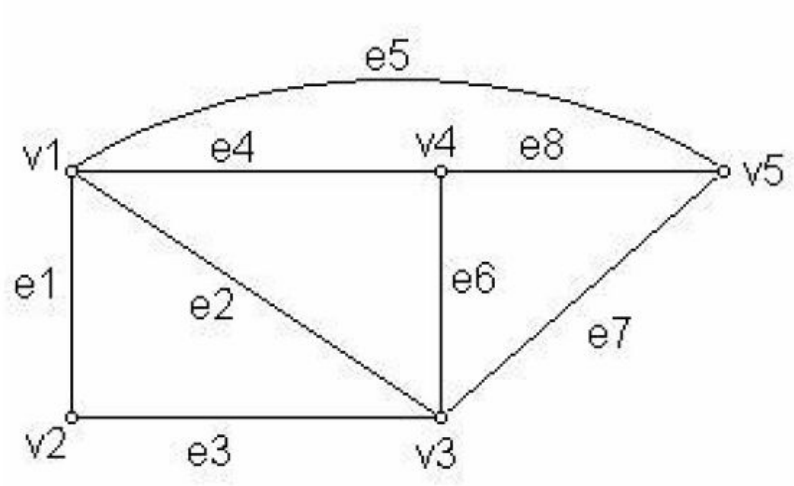
Matriks ruas

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

atau

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

Matriks Adjency



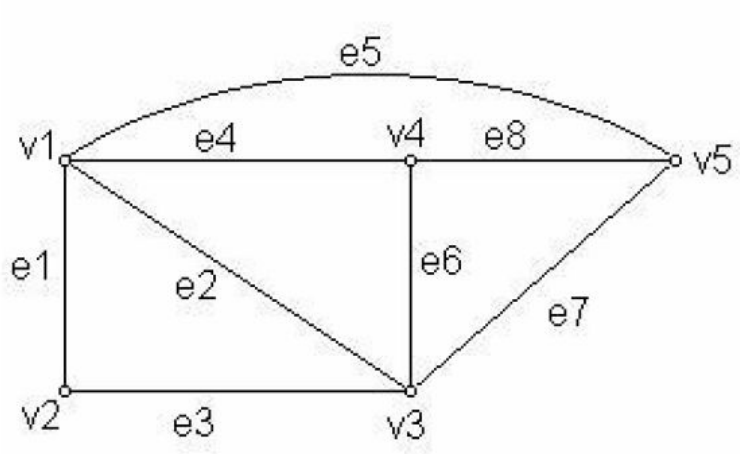

```
int graph
```

Simpul tujuan

Simpul awal	v1	v2	v3	v4	v5
v1	0	1	1	1	1
v2	1	0	1	0	0
v3	1	1	0	1	1
v4	1	0	1	0	1
v5	1	0	1	1	0

Membuat Matriks Incidence

- `int graph[2][8];`



`int graph[5][8];`

Matriks *incidence* :

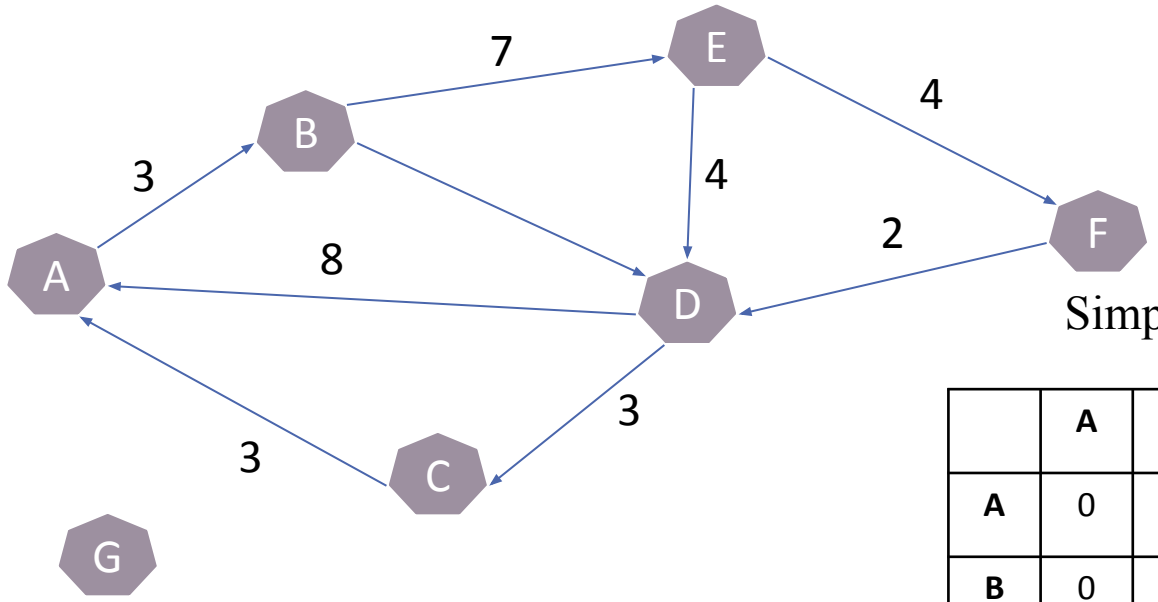
	e1	e2	e3	e4	e5	e6	e7	e8
v1	1	1	0	1	1	0	0	0
v2	1	0	1	0	0	0	0	0
v3	0	1	1	0	0	1	1	0
v4	0	0	0	1	0	1	0	1
v5	0	0	0	0	1	0	1	1

Insert value to array

```
int Graph[5][5];

int i, j; for(i=0; i<5; i++){
      for(j=0;j<5;j++)
    {
     printf("Enter value for Graph[%d][%d]:", i, j);
     scanf("%d", &Graph[i][j]);
    }
 }
```

Matriks Adjacency

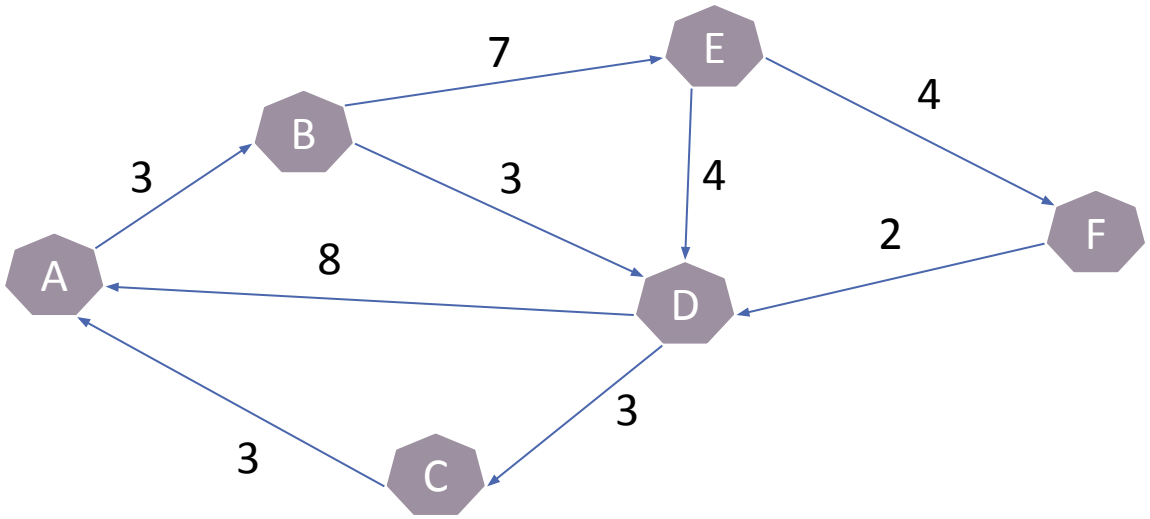


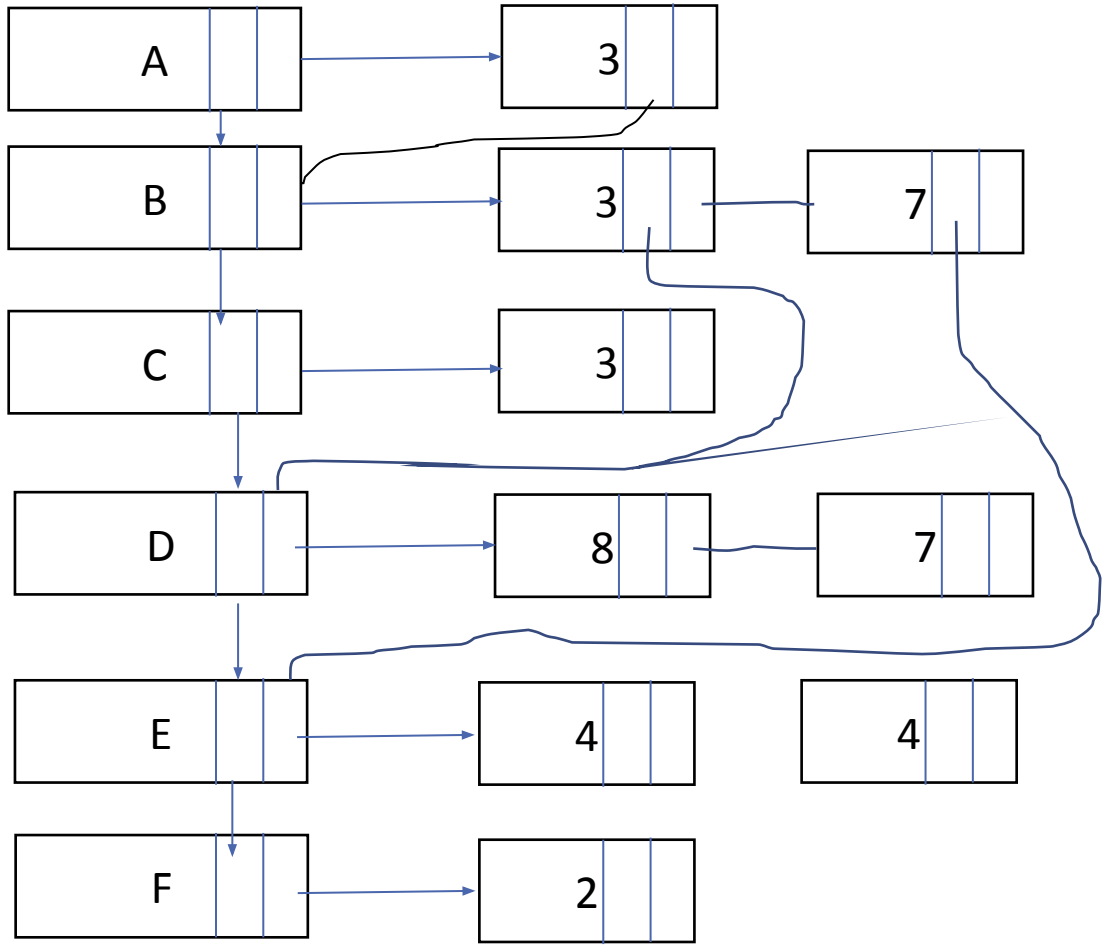
Simpul tujuan

Simpul awal

	A	B	C	D	E	F
A	0	3	0	0	0	0
B	0	0	0	3	7	0
C	3	0	0	0	0	0
D	8	0	3	0	0	0
E	0	0	0	4	0	4
F	0	0	0	2	0	0

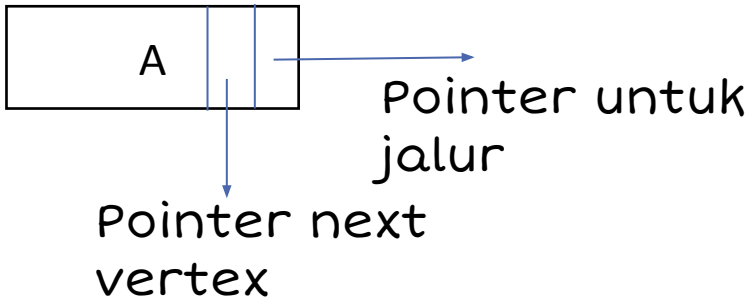
Graph Representasi Dinamis





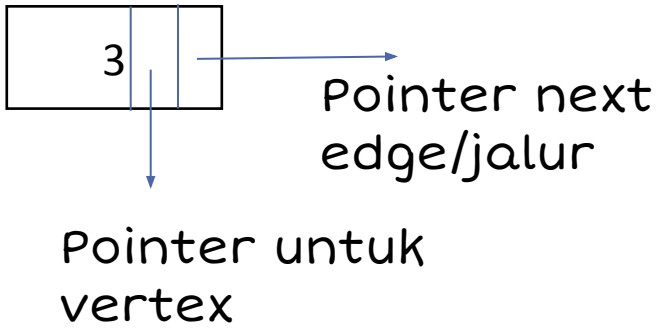
ADT untuk Graph

- ADT untuk menyimpan simpul/node/vertex




```
typedef struct simpul{
     char idSimpul;
     struct simpul * next;
     struct edge * jalur;
     }Simpul;
```

- ADT untuk menyimpan edge/jalur




```
typedef struct edge{
     int Cost;
     struct simpul * tujuan;
     struct edge * next_jalur;
     }Edge;
```

Create Graph

- Graph yang paling kecil adalah ketika memiliki minimal 1 node/simpul/vertex
- Untuk mempermudah, buat ADT untuk Graph

```
typedef struct Graph{
    Simpul * first;
}Graph;

void createGraph(Graph *G){
    G->first = NULL;
}
```

Fungsi dalam Graph

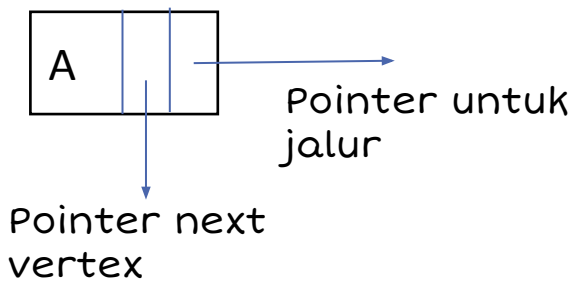
- Create Graph
- Add Vertex/Simpul
- Add Edge/Jalur
- Del Edge/jalur
- Find Vertex/Simpul
- dll

Menambahkan simpul baru

- prosedure ini akan mengcreate simpul baru
- Jika dalam graph belum ada simpul maka, simpul baru akan menjadi Graph<-first
- Jika dalam graph sudah ada simpul maka simpul baru akan ditambahkan pada next simpul

```
void createSimpul(Graph *G,char a ){
    Simpul * simpulBaru = (Simpul*) malloc (sizeof(Simpul));
        simpulBaru->idSimpul= a;
    simpulBaru-> jalur =NULL;
    simpulBaru-> next =NULL;
    if (G->first==NULL){
        G->first = SimpulBaru;
         }
    else{
    Simpul * akhir =G->first;
    while(akhir->next!=NULL){
        akhir = akhir->next;
        }
    akhir->next=simpulBaru;
    }
}
```

Ilustrasi Simpul



Menambahkan edge baru

- Edge/jalur menghubungkan 2 node/simpul/vertex
- Parameter yang harus ada : simpul awal, simpul akhir, beban/nilai (untuk grap bernilai)
- Periksa apakah simpul awal sudah mempunyai jalur/belum
- Jika simpul awal belum memiliki edge/jalur, sambungkan ke simpul
- Jika simpul awal sudah memiliki edge/jalur maka sambungkan edge/jalur ke jalur tersebut dengan menyimpan alamat jalur ke jalur sebelumnya
- Sambungkan jalur ke simpul akhir

```
void addEdge(Simpul * awal, Simpul * akhir, int nilaiJalur){
    //buat new jalur
    Edge * newEdge = (Edge *)malloc(sizeof(Edge));
    //lakukan inisialisasi dan masukan nilai edge
    //pada saat inisiali sambungkan newEdge->tujuan dengan simpul akhir
    //periksa awal->jalur
    //jika belum, tambahkan newEdge ke simpul awal
    //jika sudah, tambahkan diakhir edge pada simpul awal tsb


}
```

Menambahkan Edge baru

```
void addEdge(Simpul * awal, Simpul *akhir, int nilaiJalur){
    //buat new jalur
    Edge * newEdge = (Edge*)malloc(sizeof(Edge));
    //lakukan inisialisasi dan masukan nilai edge
    newEdge->Cost = nilaiJalur;
    newEdge->next_jalur=NULL;
    newEdge->tujuan = akhir;
    //periksa awal->jalur
    //jika belum, tambahkan newEdge ke simpul awal
    if(awal->jalur==NULL){
        awal->jalur=newEdge;
    }else{//jika sudah, periksa jalur akhir pada simpul awal tsb
        Edge * jalurAkhir = awal->jalur;
        while (jalurAkhir->next_jalur!=NULL){
            jalurAkhir = jalurAkhir->next_jalur;
        }
        //tambahkan jalur akhir diujung jalur
        jalurAkhir->next_jalur=newEdge;
    }

    }
```

Prosedur Mencetak Graph

- Prosedur ini digunakan untuk mencetak graph dengan melakukan penelusuran simpul/node/vertex dan jalur/edge pada graph tsb
- Penelusuran dimulai dari simpul awal (setiap edgenya) sampai simpul akhir
- Proses penelusuran membutuhkan temporary untuk simpul/vertex maupun jalur/edge.
- Parameternya adalah graph yang akan ditelusuri

```
void cetakGraph(struct graph *G){
        struct simpul * tempSimpul = G->first;
        if (bantu ==NULL){ //jika graph tidak kosong lakukan penelusuran simpul
        while (bantu!=NULL){
            //print simpul tsb
            //lakukan pengecekan dan penelusuran jalur
            //print setiap jalur dan simpul akhir jalur tsb
        }
        else {//selainnya graph kosong }
        }
    }
```

Prosedur Mencetak Graph

```
void cetakGraph(struct graph *G){
        struct simpul * tempSimpul = G->first;
        if (tempSimpul ==NULL){ //if graph tdk kosong lakukan penelusuran
                //simpul
        while (tempSimpul!=NULL){
            //print simpul tsb
            printf("Simpul %c\n", &bantu->idSimpul);
            //lakukan pengecekan dan penelusuran jalur
            struct edge * tempEdge = bantu->jalur;
            while (tempEdge ->jalur!=NULL){
                //print setiap jalur dan simpul akhir jalur tsb
                printf("Terhubung dengan simpul %c, dengan Cost %d",
                    &tempEdge->tujuan,&temptEdge->Cost);
                tempEdge= tempEdge->next_jalur;
            }
            tempSimpul=tempSimpul->next;
        }
        else {//selainnya graph kosong
            printf("Graph Kosong");
            }
        }
    }
```

Fungsi Mencari simpul

- Fungsi ini akan mengembalikan tipe data ADT sipul
- Mencari simpul dengan parameter masukannya adalah Graph yang akan ditelusuri dan id dari simpul tersebut
- Penelusuran dilakukan dengan menggunakan simpul->next

Prosedur Menghapus Simpul

- Prosedur menghapus simpul membutuhkan parameter idSimpul yang akan dihapus
- Lakukan penelusuran untuk mencari simpul yang akan dihapus
- Dengan menghapus simpul maka harus menghapus edge/jalur yang berasal dari simpul tersebut dan yang mengarah ke simpul tersebut

Gambarkan informasi berikut menjadi Grap dan implementasikan dalam graph dinamis

- Jakarta – Bandung 300
- Surabaya – Bandung 1200
- Yogyakarta – Bandung 1000
- Jakarta – Solo 1000
- Jakarta – Surabaya 1500
- Surabaya – Yogyakarta 300
- Bandung – Jakarta 400
- Solo – Bandung 700