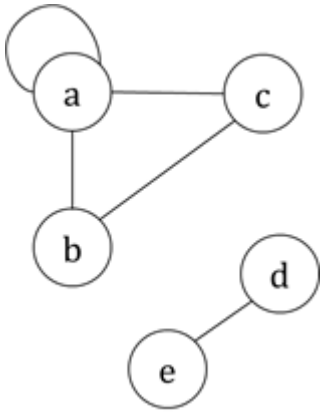
Catatan Perkuliahan

Pertemuan 12 Graf

1 Graf

Definisi 12.1. Suatu graf G dinyatakan dalam bentuk G = (V, E). V adalah himpunan verteks {v_1, v_2, …. , v_i}, sedangkan E adalah himpunan pasangan dua buah verteks {{v_1, v_2}, …, {v_i, v_j}} yang disebut sebagai *edge*.

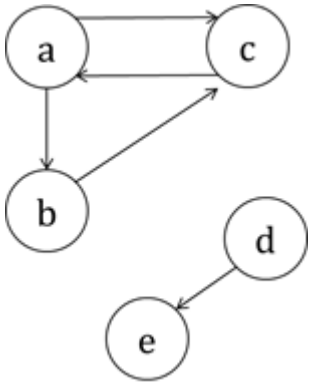
Contoh 12.1.
V = {a, b, c, d, e}
E = {{a, b}, {a, c}, {a, a}, {b, c}, {d, e}}
Graf tersebut dapat digambarkan dalam bentuk seperti pada Gambar 12.1



Gambar 12.1 Representasi graf pada contoh 12.1

Definisi 12.2. Graf berarah adalah graf yang memiliki edge dengan arah. Arah edge direpresentasikan dalam bentuk pasangan terurut (v_i, v_j) atau berupa tanda panah pada graf.

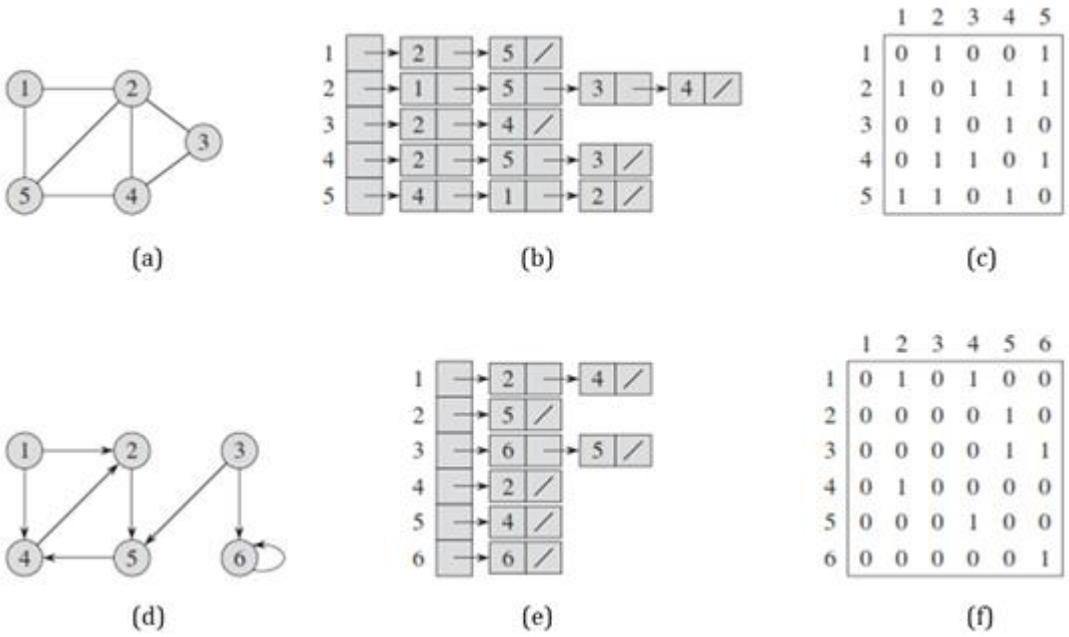
Contoh 12.2.
V = {a, b, c, d, e}
E = {(a, b}, (a, c), (c, a}, (b, c), (d, e)}
Graf tersebut dapat digambarkan dalam bentuk seperti pada Gambar 12.2



Gambar 12.2 Representasi graf berarah pada contoh 12.2

2 Representasi Graf

Dua cara standar untuk merepresentasikan graf ialah dengan menggunakan daftar ketetanggaan (*adjacency list*) dan matriks ketetanggaan (*adjacency matrix*). Daftar ketetanggaan cocok digunakan untuk graf renggang (*sparse graph*), yaitu graf yang memiliki |E| jauh lebih kecil daripada $|V|^2$. Apabila graf berbentuk rapat (*dense graph*) atau keterhubungan dua buah verteks perlu diketahui dengan cepat, bentuk matriks ketetanggaan lebih cocok untuk digunakan. Gambar 12.3 memperlihatkan bentuk daftar ketetanggaan dan matriks ketetanggaan dari graf berarah dan tidak berarah.

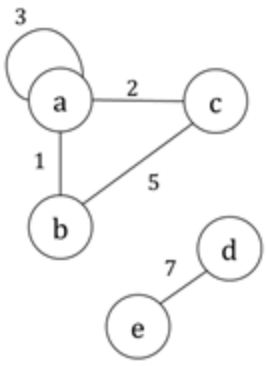


Gambar 12.3 (a) Graf tidak berarah dan (b) representasi daftar ketetanggaannya serta (c) representasi matriks ketetanggaannya. (d) Graf berarah beserta (e) daftar ketetanggaannya dan (f) matriks ketetangaannya (Cormen *et al*. 2009)

3 Atribut Graf

Verteks atau *edge* pada graf dapat memiliki atribut. Atribut *d* pada verteks *v* dinotasikan sebagai *v.d*. Atribut *f* pada *edge* (*u, v*) dinotasikan sebagai (*u, v*)*.f.* Implementasi atribut baik dalam bentuk daftar atau matriks ketetanggaan dapat beragam. Salah satu cara ialah dengan membuat *array* tambahan untuk menyimpan atribut setiap verteks. Atribut ini dapat berupa apa pun, misalnya berupa label pada verteks atau jarak atau biaya pada *edge*.

Contoh 12.3
Pada Gambar 12.4, disajikan sebuah graf tak berarah. Setiap verteks memiliki atribut nama kota dan setiap *edge* memiliki atribut jarak dalam satuan 10 km.



(a)

	a	b	c	d	e
a	3	1	2	0	0
b	1	0	5	0	0
c	2	5	0	0	0
d	0	0	0	0	7
e	0	0	0	7	0

(b)

a	"Jakarta"
b	"Bogor"
c	"Bandung"
d	"Medan"
e	"Tapanuli"

(c)

Gambar 12.4 (a) Graf yang memiliki atribut, (b) representasi matriks ketetanggaan, (c) tabel untuk menyimpan nilai yang dimiliki oleh atribut.

Implementasi kode dalam bahasa pemrograman dapat beragam. Dalam bahasa berorientasi objek, misalnya, verteks dapat dibuat menjadi sebuah kelas yang memiliki atribut. Dalam C++, kita dapat membuat sebuah kelas graf sederhana yang memiliki |V| verteks dan *adjacency list* berikut:

```
class Graph
{
    int V;
    list<int> *adj;
public:
    Graph(int V);
    void addEdge(int v, int w);
    void BFS(int s);
    void DFS(int s);
};
```

Untuk membuat sebuah graf baru, informasi jumlah verteks pada graf diperlukan.

```
Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}
```

Penambahan *edge* dari pada graf dapat dilakukan dengan menambahkan nilai

```
void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}
```

4 Breadth-First Search

Breadth-first search (BFS) adalah algoritme paling sederhana untuk menelusuri graf dan merupakan bentuk dasar dari banyak algoritme pada graf seperti algoritme Prim pada *minimum*

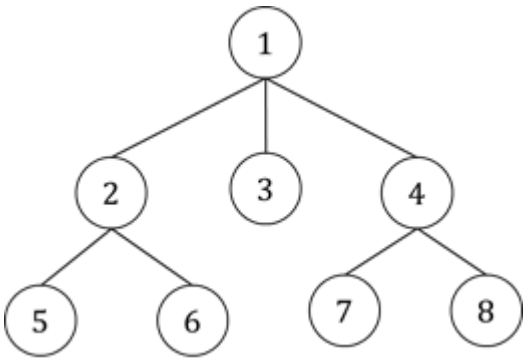
spanning tree dan algoritme Dijkstra untuk masalah *single-source shortest-path*. BFS menelusuri graf G = (V, E) dari sebuah verteks sumber *s* dan menghitung jarak (jumlah *edge* minimum) dari *s* ke seluruh verteks yang dapat dikunjungi. *Breadth* pada BFS berasal dari sifat pencarian BFS yang mengunjungi seluruh verteks dengan jarak *k* dari *s* sebelum mengunjungi verteks dengan jarak *k* + 1.

Setelah *s* dikunjungi, *s* dimasukkan ke dalam sebuah *queue*. Selama *queue* berisi, sebuah node *v* akan dikeluarkan dari *queue* dan seluruh node tetangganya yang belum dikunjungi dimasukkan ke dalam *queue*. Algoritme 12.1 memperlihatkan pseudokode dari proses tersebut.

Algoritme 12.1 – Breadth First Search

```
BFS(s)
 1 for each vertex u ∈ G.V
 2      visited[u] = false
 3 visited[s] = true
 4 queue.push_back(s)
 5 while(!queue.empty())
 6    v = queue.front()
 7    queue.pop_front()
 8    for each i ∈ G.Adj[s]
 9         if visited[i] == false
10            visited[i] = true
11                queue.push_back(i)
```

Sebagai contoh, BFS akan menelusuri graf pada Gambar 12.5 akan mengunjungi graf dengan urutan 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8.



Gambar 12.5 Contoh graf

5 Depth-First Search

Depth-first search (DFS) mencari graf sedalam mungkin dengan terus menelusuri *edge* yang keluar dari verteks yang terakhir dikunjungi. Ketika penelusuran tidak dapat lagi dilakukan, DFS akan melakukan *backtrack* untuk mengeksplorasi *edge* lain yang belum ditelusuri.

Algoritme 12.2 – Depth First Search

```
DFS(s)
 1 for each vertex u ∈ G.V
 2      visited[u] = false
 3 DFS-Visit(v, visited)

DFS-Visit(v, visited)
 1 visited[v] = true
 2 for each i ∈ G.Adj[s]
 3    if visited[i] == false
 4          DFS-Visit(I, visited)
```

Sebagai contoh, DFS akan menelusuri graf pada Gambar 12.5 akan mengunjungi graf dengan urutan 1 – 2 – 5 – 6 – 3 – 4 – 7 – 8.

Referensi

Cormen TH, Leiserson CE, Rivest RL, Stein C. 2009. Introduction to Algorithms. Ed ke-3. Cambridge(US): MIT Press.