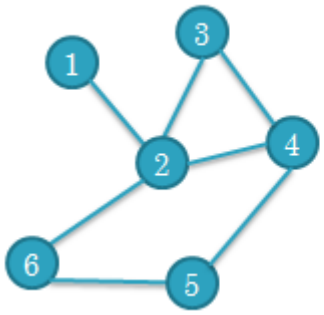
Praktikum DAA: Graph Algorithm

Representasi Graph dengan Adjacency Matrix

Adjacency matrix adalah matriks berukuran *n x n* yang berisi nilai boolean yang merepresentasikan sebuah graph. Nilai dari baris *v* dan kolom *w* adalah 1 apabila ada sebuah edge yang menghubungkan vertex *v* dan vertex *w* di dalam graph, dan nilai 0 berarti sebaliknya. Cara ini dapat juga digunakan untuk menyimpan graph yang memiliki bobot, di mana matrix diisi dengan bobot tiap edge.

Contoh graph tidak berbobot:



	1	2	3	4	5	6
1	0	1	0	0	0	0
2	1	0	1	1	0	1
3	0	1	0	1	0	0
4	0	1	1	0	1	0
5	0	0	0	1	0	1
6	0	1	0	0	1	0

Latihan: Membuat Representasi Graph dengan Adjacency Matrix

1. Untuk merepresentasikan sebuah graph menggunakan adjacency matrix, kita membutuhkan sebuah array 2 dimensi. Atribut lain yang berguna adalah jumlah vertex pada graph tersebut. Buatlah sebuah kelas Graph dengan atribut sebuah array dua dimensi (int[][]) bernama matrix, dan sebuah integer bernama N. Tambahkan constructor yang menginisialisasi kedua atribut tersebut.

```
public class Graph{
    int N;  //number of vertices
    int[][] matrix;

    public Graph(int N){
        this.N = N;
        this.matrix = new int[N][N];
    }
}
```

2. Untuk menambahkan sebuah edge di dalam Graph, buatlah method *addEdge(int, int)*. Kedua parameter merupakan dua buah vertex yang akan dihubungkan dengan edge tersebut.

```
public void addEdge(int v1, int v2){
    this.matrix[v1][v2] = 1;
    this.matrix[v2][v1] = 1;
}
```

Tambahkan juga method *addEdge(int, int, int)* yang berguna untuk menambah edge dengan bobot bukan 1, berguna untuk representasi graph berbobot.

```
public void addEdge(int v1, int v2, int bobot){
        this.matrix[v1][v2] = bobot;
        this.matrix[v2][v1] = bobot;
}
```

3. Untuk menghapus sebuah edge yang menghubungkan dua buah vertex, buatlah method *removeEdge(int, int).* Dengan parameter adalah kedua buah vertex yang edge-nya yang akan dihapus.

```
public void removeEdge(int v1, int v2){
    this.matrix[v1][v2] = 0;
    this.matrix[v2][v1] = 0;
}
```

4. Edge pada graph melambangkan relasi antar vertex-vertexnya. Setelah membuat mutator untuk edge, kita juga perlu membuat accessornya. Buatlah sebuah method *isEdgeExist(int, int)* sebagai accessor edge pada graph. Method ini mengembalikan 0 jika tidak ada edge yang menghubungkan kedua vertex, atau bobot edge jika kedua vertex terhubung.

```
public int isEdgeExist(int v1, int v2){
    return this.matrix[v1][v2];
}
```

5. Untuk menguji bahwa method-method pada kelas Graph telah bekerja dengan benar, buatlah sebuah kelas *GraphTester*. Untuk memudahkan pengujian, override method *toString* supaya dapat mencetak semua isi array *matrix*. Kode berikut ini contoh graph yang sama seperti pada gambar graph di atas. Tambahkan kode untuk menguji method lainnya!

```
class TesterGraph{
    public static void main(String[] args){
        Graph g = new Graph(6);
        g.addEdge(0,1);
        g.addEdge(1,2);
        g.addEdge(2,3);
        g.addEdge(1,3);
        g.addEdge(5,1);
        g.addEdge(1,3);
        g.addEdge(5,4);
        g.addEdge(3,4);

        System.out.println(g);
    }
}
```

Graph Traversal: DFS dan BFS

Implementasikanlah Graph Traversal dengan algoritma DFS dan BFS. Pertama-tama buatlah kelas Graph seperti dijelaskan pada modul, kemudian tambahkan beberapa method seperti yang ditunjukkan oleh kerangka di bawah ini.

```
public void DFS(){
    //...
}

private void DFSRecursive(int idx){
    //...
}

public void BFS(){
    //...
}

private void process(int v){
    System.out.print(v + " ");
}
```

Method *DFS* berguna untuk memulai traversal secara DFS dari vertex 0. Method ini bertugas untuk menginisialisasi variable-variable yang digunakan, kemudian memanggil method *DFSRecursive* pada setiap *root* node. Method BFS digunakan untuk melakukan traversal secara BFS. Method process dapat diisi dengan kode-kode untuk memproses vertex yang sedang dikunjungi. Pada contoh ini, nomor vertex dicetak ke layar.

Untuk melakukan DFS/BFS, tambahkan atribut *ctr*, array of *ord,* dan array of *parent* pada kelas Graph. Jangan lupa kedua array ini perlu dibuat pada constructor, sesuai dengan banyak vertex.