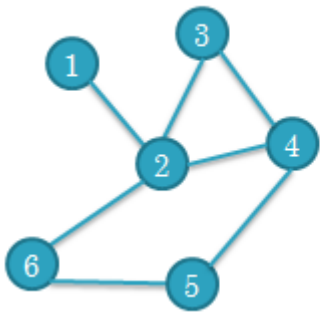
Praktikum DAA: Graph Algorithm

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.

Contoh:



	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 adjacencyMatrix, dan sebuah integer bernama numberOfVertices. Tambahkan constructor yang menginisialisasi kedua atribut tersebut.-

```
public class Graph{
    int numberOfVertices;
    int[][] adjacencyMatrix;

public Graph(int numberOfVertices){
    this.numberOfVertices = numberOfVertices;
    this.adjacencyMatrix=new int[numberOfVertices][numberOfVertices];
}
```

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 idxVertex1, int idxVertex2){
 this.adjacencyMatrix[idxVertex1][idxVertex2]=1;
 this.adjacencyMatrix[idxVertex2][idxVertex1]=1;
}
```

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 idxVertex1, int idxVertex2){
 this.adjacencyMatrix[idxVertex1][idxVertex2]=0;
 this.adjacencyMatrix[idxVertex2][idxVertex1]=0;
}
```

4. Edge pada graph melambangkan relasi antar vertex-vertexnya. Setelah membuat mutator untuk edge, kita juga perlu membuat accessornya. Buatlah sebuah method edgeIsExist(int, int) sebagai accessor edge pada graph.

```
public boolean edgeIsExist(int vertex1Idx, int vertex2Idx){
 return this.adjacencyMatrix[vertex1Idx][vertex2Idx]==1;
}
```

5. Untuk menguji bahwa method-method pada kelas Graph telah bekerja dengan benar, buatlah sebuah kelas Tester. Untuk memudahkan pengujian, tambahkan method untuk mencetak semua isi array adjacencyMatrix. Method ini hanya digunakan untuk keperluan debugging, karena itu boleh dihapus setelah class selesai diuji.

```
import java.util.Scanner;
public class Tester{
    public static void main(String[] args){
        Scanner sc=new Scanner(System.in);
        //Buat Graph Dengan Jumlah Vertex=9
        Graph G=new Graph(9);
        //Tambahkan beberapa edges
        G.addEdge(0,1);
        G.addEdge(0,7);
        G.addEdge(7,8);
        G.addEdge(8,5);
        G.addEdge(5,4);
        G.addEdge(4,3);
        G.addEdge(5,1);
        G.addEdge(5,6);
        G.addEdge(1,2);
        G.addEdge(2,6);
        G.addEdge(1,6);

        //print adjacencyMatrix
        G.printAdjacencyMatrix();

        //hapus beberapa edges
        G.removeEdge(1,2);
        G.removeEdge(5,6);
        System.out.println("");
        //print adjacencyMatrix
        G.printAdjacencyMatrix();

    }
}
```

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 LinkedList DFSTraversal(){
    //...
}

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

public LinkedList BFSTraversal(){
    //...
 }

private void process(int vertex){
    this.result.add(vertex);
}
```

Method `DFSTraversal` mengembalikan sebuah LinkedList yang berisi urutan traversal node-node pada graph secara DFS. Untuk itu, di awal traversal, kita perlu membuat sebuah `LinkedList` kosong, yang kemudian diisi oleh method `process`. Pada akhir traversal, `LinkedList` yang sudah terisi ini di-return. Mirip dengan `DFSTraversal`, `BFSTraversal` juga mengembalikan `LinkedList` yang berisi urutan traversal node-node secara BFS.