As you know, a **graph** G *is* a set *V* of **vertices** and a collection *E* of pairs of vertices from *V*, called **edges**. The aim of this worksheet is to make an implementation of the Graph ADT based on the **adjacency map** representation.

As illustrated in figure 1, with this representation the set **V of vertices** are stored in a **map** and for each vertex *v* its outgoing edges are represented also in a **map**.
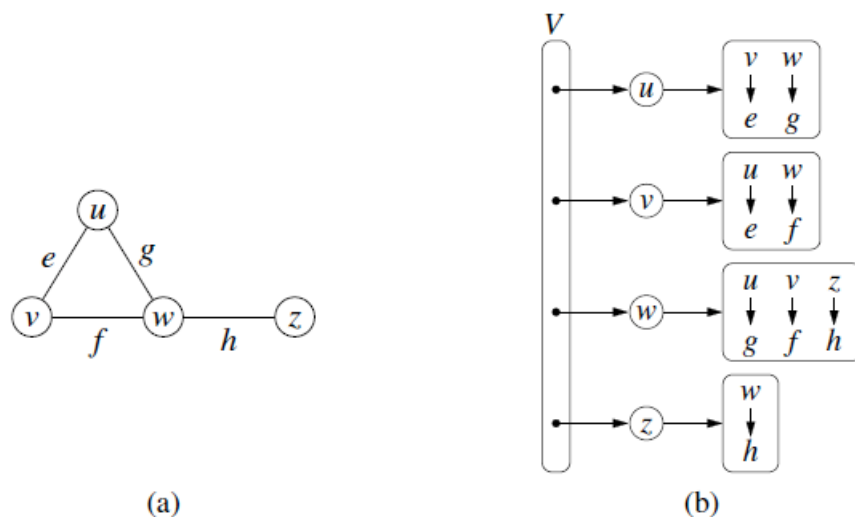

Figure1 - (a) An undirected graph G;   (b) a schematic representation of the adjacency map structure for G

Download the project Graph from the Moodle and analyse the classes.

All the classes use generic parameters V and E to designate the element type stored respectively at vertices and edges.

The Edge class stores the information associated with the edge, the weight of the edge (that may represent a distance, time, or a capacity) and its both endpoints vertices.

The Vertex class stores the information associated with the vertex, a numeric key of vertex and a map with its outgoing edges.

A graph instance maintains the number of vertices and edges of the graph, a boolean variable that designates whether the graph is directed and maintains the map with all its vertices.

Part 1

The generic class Graph<V,E> implements the Graph ADT that includes the following methods:

```
public interface GraphInterface<V,E> {
    int numVertices();
    Iterable<V> vertices();
    int numEdges();
    Iterable<Edge<V,E>> edges();
    Edge<V,E> getEdge(V vOrig, V vDest);
    V[] endVertices(Edge<V,E> edge);
    V opposite(V vert, Edge<V,E> edge);
    int outDegree(V vert) ;
    int inDegree(V vert) ;
    Iterable<Edge<V,E>> outgoingEdges (V vert);
    Iterable<Edge<V,E>> incomingEdges(V vert);
    boolean insertVertex(V newVert);
    boolean insertEdge(V vOrig, V vDest, E edge, double eWeight);
    boolean removeVertex(V vert);
    boolean removeEdge(V vOrig, V vDest);
}
```

1. Complete the generic class Graph<V,E> implementing the following methods:

```
Iterable<Edge<E>> edges();
Iterable<Edge<V,E>> incomingEdges(Vertex<V,E> v);
```

2. Test the methods.

Part 2

In the GraphAlgorithms class develop the following methods:

1. Depth-first search of a graph starting in a vertex with a given information
2. All paths in a graph between two vertices with a given information
3. Shortest-path from a source vertex to all reachable vertices of the graph, using Dijkstra's algorithm
4. Test the methods.