**Graph algorithms - practical work no. 1**

**Due:** week 5-6.

Design and implement an abstract data type *directed graph* and a function (either a member function or an external one, as your choice) for reading a directed graph from a text file.

The vertices will be specified as integers from 0 to *n*-1, where *n* is the number of vertices.

Edges may be specified either by the two endpoints (that is, by the source and target), or by some abstract data type *Edge\_id* (that data type may be a pointer or reference to the edge representation, but then care should be taken not to expose the implementation details of the graph).

Each edge will have an integer value (for instance, a cost) attached to it. The *directed graph* data type shall allow its users to retrieve and modify that integer and shall not interpret or restrain it in any way.

**Required operations:**

* get the number of vertices;
* given two vertices, find out whether there is an edge from the first one to the second one, and retrieve the *Edge\_id* if there is an edge (the latter is not required if an edge is represented simply as a pair of vertex identifiers);
* get the in degree and the out degree of a specified vertex;
* iterate through the set of outbound edges of a specified vertex (that is, provide an iterator). For each outbound edge, the iterator shall provide the *Edge\_id* of the curren edge (or the target vertex, if no *Edge\_id* is used).
* iterate through the set of inbound edges of a specified vertex (as above);
* get the endpoints of an edge specified by an *Edge\_id* (if applicable);
* retrieve or modify the information (the integer) attached to a specified edge.

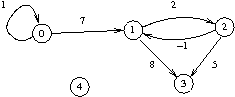
The operations must take no more than:

* O(deg(*x*)+deg(*y*)) for: verifying the existence of an edge and for retrieving the edge between two given vertices.
* O(1) for: getting the first or the next edge, inbound or outbound to a given vertex; get the endpoints, get or set the attached integer for an edge (given by an *Edge\_id* or, if no *Edge\_id* is defined, then given by its source and target); get the total number of vertices or edges; get the in-degree or the out-degree of a given vertex.

**Note:** You are allowed to use, from existing libraries, data structures such as linked lists, double-linked lists, maps, etc. However, you are not allowed to use already-implemented graphs (though, you are encouraged to take a look at them).

**Text file format:** the graph will be read from a text file having the following format:

* On the first line, the number *n* of vertices and the number *m* of edges;
* On each of the following *m* lines, three numbers, *x*, *y* and *c*, describing an edge: the origin, the target and the cost of that edge.

[[](http://www.cs.ubbcluj.ro/~rlupsa/edu/grafe/digraph-ex1.txt)Example](digraph-ex1.txt)

[Sample (partial) documentation](Documentation%20example.docx)

Random generator <gen-digraph.cpp>

Large input files:

* 1k vertices, 4k edges: [graph1k.zip](Large%20input%20files/graph1k.zip);
* 10k vertices, 40k edges: [graph10k.zip](Large%20input%20files/graph10k.zip);
* 100k vertices, 400k edges: [graph100k.zip](Large%20input%20files/graph100k.zip);
* 1m vertices, 4m edges: [graph1m.zip](Large%20input%20files/graph1m.zip);

**Optional operations (bonus)**

* The property (cost) attached to the edges should be external to the graph. It shall be a template/generic class, parametrised on the property type. An instance of this Edge\_property class shall behave like a map (dictionary) attaching a value to each edge of the graph. It shall be possible to create any number of instances, at any time, without modifying the graph.
* The graph shall be modifiable: it shall be possible to add and remove an edge, and to add and remove a vertex. Think about what should happen with the properties of existing edges and with the identification of remaining vertices. You may use an abstract Vertex\_id instead of an int in order to identify vertices; in this case, provide a way of iterating the vertices of the graph.
* The graph shall be copyable, that is, it should be possible to make an exact copy of a graph, so that the original can be then modified independently of its copy. Think about the desirable behaviour of an Edge\_property attached to the original graph, when a copy is made.