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 without exposing the implementation details of the graph).

Additionally, create a map that associates to an edge an integer value (for instance, a cost).

Required operations:

- ✓ get the number of vertices;
- parse (iterate) the set 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;
- ✓ parse (iterate) 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).
- ✓ parse 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 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.
- Read the graph from a text file (as an external function); see the format below.
- Write the graph from a text file (as an external function); see the format below.
- Create a random graph with specified number of vertices and of edges (as an external function).

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.

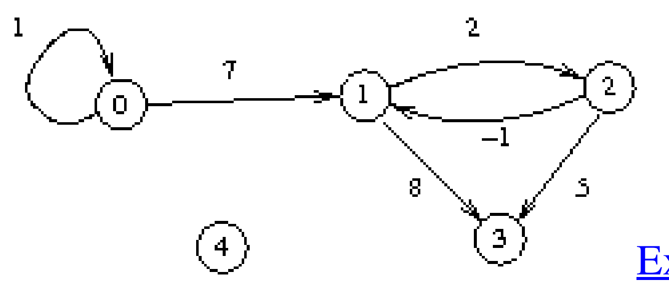
Other requirements:

- The object returned by the parse functions shall not allow modifying the graph through its public functions. So, don't return sets by reference. Return iterators.
- Generally, make sure the graph cannot be brought in an inconsistent state by applying public functions on various accessible objects.

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.



Example

Sample (partial) documentation

Random generator gen-digraph.cpp

Large input files:

- 1k vertices, 4k edges: graph1k.zip or graph1k.txt;
- 10k vertices, 40k edges: graph10k.zip or graph10k.txt;
- 100k vertices, 400k edges: graph100k.zip;
- 1m vertices, 4m edges: graph1m.zip;

Optional operations (bonus)

- Do the implementation in two distinct languages - one of them being Java (~2p), C# (~2p), C++ (~4p).