**Practical work no. 1**

Specification

We shall define a class named Graph representing a directed graph.

The class Graph will provide the following methods:

Graph()

* Constructs an empty graph.

Graph(std::vector<int> vertices, std::map< int, std::pair<int, int>> edgesindexed, std::map<int, int> costsindexed)

* Constructs a graph containing the vertices in the vector vertices, the edges in the map edgesindexed and the costs in the map costsindexed.

int numberofvertices()

* Returns the number of vertices of the graph.

bool is\_edge(int x, int y)

* Precondition: Both vertices exist in the graph.
* Returns true if the edge (x,y) is present in the graph, else false.

int vertex\_out\_degree(int x)

* Precondition: Vertex x exists in the graph.
* Returns the out degree of vertex x.

int vertex\_in\_degree(int x)

* Precondition: Vertex x exists in the graph.
* Returns the in degree of vertex x.

std::vector<int> parse\_vertices()

* Returns a vector containing all the vertices of the graph.

std::vector<int> parse\_nout(int x)

* Precondition: Vertex x exists in the graph.
* Returns a vector of all vertices that are on outbound edges from x.

std::vector<int> parse\_nin(int x)

* Precondition: Vertex x exists in the graph.
* Returns a vector of all vertices that are on inbound edges in x.

bool modify\_cost(int x, int y, int c)

* Precondition: Vertices x and y exist in the graph, as well as the edge (x,y).
* Sets the cost of edge (x,y) to c.
* Returns a vector of all vertices that are on inbound edges in x.

bool get\_cost(int x, int y)

* Precondition: Vertices x and y exist in the graph, as well as the edge (x,y).
* Gets the cost of edge (x,y).

bool add\_edge(int x, int y)

* Precondition: Vertices x and y exist in the graph, and edge (x,y) doesn’t.
* Adds edge (x,y) to the graph.

bool remove\_edge(int x, int y)

* Precondition: Vertices x and y exist in the graph, as well as the edge (x,y).
* Removes edge (x,y) from the graph.

bool add\_vertex(int x)

* Precondition: Vertex x not in graph.
* Adds vertex x to the graph.

bool remove\_vertex(int x)

* Precondition: Vertex x exists in the graph.
* Removes vertex x from the graph, along with all its associated edges.

Graph& copy\_graph(const Graph& g)

* Returns a copy of the graph.

Utility methods

Graph buildRandomGraph(int v, int e)

* Precondition: v^2 >= e.
* Builds a random graph with vertices 0, 1, …, v-1 and e edges.

Graph readGraphFile(std::string filename)

* Precondition: filename is a valid file.
* Reads a graph from a file.

Graph writeGraphFile(Graph g, std::string filename)

* Precondition: filename is a valid file.
* Writes graph g to a file.

Graph readGraphFile2(std::string filename)

* Precondition: filename is a valid file.
* Reads a graph from a file using the second format.

Graph writeGraphFile2(Graph g, std::string filename)

* Precondition: filename is a valid file.
* Writes graph g to a file using the second format.

Implementation

Each edge is a member of three maps, a map where each value is a vector of all edges that are outbound from the key vertex, a map where each value is a vector of all edges that are inbound to the key vertex and a map that assigns each edge a cost.

Class Graph will have the following data members:

std::map<int, std::vector<int> > nin; // The inbound map

std::map<int, std::vector<int> > nout; // The outbound map

std::map<std::pair<int, int>, int> costs; // The costs map