Specification

We are defining a class called “Graph” to represent a directed graph.

In order to represent our data, we are using three maps, one for the edges that goes out from a vertex, one for edges that goes in a vertex and the third map retain the cost between between two vertices;

We are using the following functions:

Graph();

The constructor for the graph class. The number of edges and the number of vertices are equal to 0, inittialy.

Graph(const Graph& graph);

Copy constructor for the graph class

void add\_edge(int node\_from, int node\_to, int cost);

Add an edge between 2 vertices, if the vertices does not exists or if the edge already exists, it throws an error.

void add\_node(int node);

Add an node if is possible, otherwise throws an exception.

void delete\_edge(int node\_from, int node\_to);

Delete an edge between two vertices, if one of the vertices does not exists it throws an exception

bool is\_edge(int node\_from, int node\_to);

Return true if there is an edge between two vertices, otherwise false. If one of the vertices does not exists it throws an exception;

int get\_out\_degree\_of\_node(int node);

Return the out degree of a node. If the node does not exists it throws an exception

int get\_in\_degree\_of\_node(int node);

Return the in degree of a node. If the node does not exists it throws an exception;

std::vector<int>nodes;

int get\_number\_of\_nodes();

Return the numbers of nodes;

void remove\_node(int node);

Remove a node. If the node does not exists, it throw an exception

int get\_the\_cost\_between\_two\_nodes(int node\_from, int node\_to);

Return the cost between two nodes. If one of the nodes does not exists, it throws an exception

void set\_the\_cost\_between\_two\_nodes(int node\_from, int node\_to, int new\_value);

Set the cost between two nodes. If one the nodes does not exists, it throws an exception

std::pair<std::vector<int>::iterator, std::vector<int>::iterator> get\_iterator\_for\_nodes();

Returns an iterator for nodes.

std::pair<std::vector<int>::iterator, std::vector<int>::iterator> get\_iterator\_for\_outbounds\_of\_a\_node(int node);

Returns an iterator for outbounds of a node

std::pair<std::vector<int>::iterator, std::vector<int>::iterator> get\_iterator\_for\_inbounds\_of\_a\_node(int node);

Returns an iterator for inbounds of a node