Graph Algorithms – Practical Work no. 1

Vlad Bogdan-Tudor, Computer Science, 1st Year, Group 917

We will define a class named **DirectedGraph**, which will represent a directed graph. As its internal representation, **DirectedGraph** uses 3 maps, each with the following purpose:

- **dict_in** – keeps track of the "in" relationships, i.e., the vertices of the graph will be the keys and the outbound neighbours (in a set) of the corresponding vertex will be the values.
- **dict_out**– keeps track of the "out" relationships, i.e., the vertices of the graph will be the keys and the inbound neighbours (in a set) of the corresponding vertex will be the value.
- **costs** – memorizes the edge costs in the graph. The keys are pairs representing edges, while the values will be the corresponding cost of each edge in the graph.

C++ implementation

The class DirectedGraph will provide the following methods:

explicit DirectedGraph(int nr_vertices = 0);

Constructor. Builds a graph with <nr_vertices> vertices using a triple map representation.

Initializes the sets of outbound and inbound neighbours for each vertex of the vertex.

Input: nr_vertices - Integer

Output: a new graph is generated

int get_nr_vertices() const;

Returns the number of vertices in the graph.

Output: How many vertices the graph has; Integer

int get_nr_edges() const;

Returns the number of edges in the graph.

Output: How many edges the graph has; Integer

std::pair<std::map<int, std::set<int>>::iterator, std::map<int, std::set<int>>::iterator> out_relations();

Returns 2 iterators in a pair: one at the start of the map which represents the outbound relationships (each vertex is mapped to a set of outbound vertices) and another iterator at the end of this map. This function can be used for getting all the vertices in the graph (by using the <.first> attribute of the iterators)

Output: A pair of 2 iterators, each being an iterator to a map between an Integer and a Set.

std::pair<std::map<std::pair<int, int>, int>::iterator, std::map<std::pair<int, int>, int>::iterator> get_all_edges();

Returns 2 iterators in a pair: one at the start of the map which represents the costs of the graph (each edge (as a pair of 2 Integers) is mapped to a cost, Integer) and another iterator at the end of this map.

Output: A pair of 2 iterators, each being an iterator to a map between a pair of Integers and an Integer.

bool is_edge_in_graph(int _from, int _to);

Checks if a given edge is in the graph.

Input: _from - Integer, starting vertex of an edge; _to - Integer, ending vertex of an edge

Output: true - If the given edge is present in the graph; false - Otherwise

int get_cost_of_edge(int _from, int _to);

Returns the cost of a given edge in the graph if it exists

Input: _from - Integer, starting vertex of an edge; _to - Integer, ending vertex of an edge

Output: The cost of the edge in the graph, Integer

Throws: An exception if the given edge is not in the graph or if one of the vertices is not in the graph

bool is_vertex_in_graph(int vertex);

Checks if a given vertex is in the graph.

Input: vertex - Integer

Output: true - If the given vertex is in the graph; false - Otherwise

int get_in_degree(int vertex);

Returns the in-degree of a vertex.

Input: vertex - Integer

Output: Integer - The number of vertices which come into <vertex>

Throws: An exception if the given vertex is not in the graph

int get_out_degree(int vertex);

Returns the out-degree of a vertex.

Input: vertex - Integer

Output: Integer - The number of vertices which <vertex> goes into

Throws: An exception if the given vertex is not in the graph

std::pair<std::set<int>::iterator, std::set<int>::iterator> get_outbound_neighbours(int vertex);

Returns a pair of 2 iterators: one at the start of the set of outbound neighbours of the given vertex and one at the end of this same set.

Input: vertex - Integer

Output: A pair of 2 iterators, each being an iterator to a set of Integers

Throws: Exception - if the given vertex is not in the graph

int get_nr_outbound_neighbours(int vertex);

Returns the number of outbound neighbours of a vertex.

Input: vertex - Integer

Output: Integer, the number of vertices which <vertex> goes into

Throws: Exception - if the given vertex is not in the graph

std::pair<std::set<int>::iterator, std::set<int>::iterator> get_inbound_neighbours(int vertex);

Returns a pair of 2 iterators: one at the start of the set of inbound neighbours of the given vertex and one at the nd of this same set.

Input: vertex - Integer

Output: A pair of 2 iterators, each being an iterator to a set of Integers

int get_nr_inbound_neighbours(int vertex);

Returns the number of inbound neighbours of a vertex.

Input: vertex - Integer

Output: Integer, the number of vertices which go into <vertex>

Throws: Exception - if the given vertex is not in the graph.

void add_vertex(int vertex);

Adds a new vertex to the graph.

Input: vertex - Integer

Throws: Exception - if the vertex is already in the graph

void remove_vertex(int vertex);

Removes a vertex from the graph.

Input: vertex - Integer

Throws: Exception - if the vertex is not in the graph

void add_edge(int _from, int _to, int _cost);

Adds a new edge in the graph

Input: _from, _to - Integers, starting and ending vertices of the edge; _cost - Integer, cost of the edge

Throws: Exception - if the edge is already in the graph, or if one of the vertices is NOT in the graph

void remove_edge(int _from, int _to);

Removes an edge from the graph.

Input: _from, _to - Integers, starting and ending vertices of the edge

Throws: Exception - if the edge is not in the graph, or if of of the vertices is NOT in the graph

void change_cost(int _from, int _to, int _new_cost);

Changes the cost of an edge.

Input: _from, _to - Integers, starting and ending vertices of the edge

Throws: Exception - if the edge is not in the graph, or if of of the vertices is NOT in the graph

DirectedGraph copy_graph();

Returns a deepcopy of the graph

The following are external functions (outside of the DirectedGraph class):

DirectedGraph read_graph_from_file(char *file_name);

Reads the graph from a file, builds that graph, and returns it.

Input: file_name - pointer to char, the name of the file

Output: a new instance of the class <DirectedGraph>; the graph from the file

void write_graph_to_file(char *file_name, DirectedGraph& graph);

Writes the given graph to a file.

Input: file_name - pointer to char, the name of the file

graph - a reference to a graph object

DirectedGraph generate_random_graph(int nr_vertices, int nr_edges);

Creates a random graph with <nr_vertices> vertices and <nr_edges> edges.

Input: nr_vertices - integer, the number of vertices

nr_edges - integer, the number of edges

Output: a new instance of the class <DirectedGraph>, the randomly generated graph