**Practical Work no. 1**

**Fătu Samuel**

**Group 912**

**Specifications**

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

The class Graph will provide the following methods:

Graph(self, n = 0);

Constructs a graph with n vertices

is\_vertex(self, vertex)

Returns true if the vertex is in the graph, false otherwise

is\_edge(self, vertex0, vertex1)

Returns true if (vertex0, vertex1) is an edge in the graph, false otherwise

count\_vertices(self)

Returns the number of vertices

count\_edges(self)

Returns the number of edges

get\_in\_degree(self, vertex)

Returns the in degree of a vertex

get\_out\_degree(self, vertex)

Returns the out degree of a vertex

set\_edge\_data(self, vertex0, vertex1, value)

Sets the cost of the edge (vertex0, vertex1) to value

get\_edge\_data(self, vertex0, vertex1)

Gets the cost of the edge (vertex0, vertex1)

add\_edge(self, vertex0, vertex1);

Adds an edge from vertex0 to vertex1, default cost is 0

remove\_edge(self, vertex0, vertex1);

Removes the edge from vertex0 to vertex1

add\_vertex(self, vertex);

Adds the given vertex to the graph

remove\_vertex(self, vertex);

Removes the given vertex from the graph

vertices\_iterator(self)

Returns a generator of the vertices in the graph

inbound\_iterator(self, vertex)

Returns a generator of the inbound vertices of `vertex`

outound\_iterator(self, vertex)

Returns a generator of the outbound vertices of `vertex`

read\_from\_file(filePath);

Reads a graph from the given file and returns it

write\_to\_file(graph, filePath);

Writes the given graph the to given file

create\_random\_graph(vertices, edges);

Generates a random graph with `vertices` vertices and `edges` edges

**Implementation**

The implementation of the class uses a set which represents the set of vertices and 3 dictionaries:

self.\_vertices = set()

self.\_inbound = dict()

self.\_outbound = dict()

self.\_edges = dict()