Lab 1 Documentation

I used Python to implement the graph. I defined a class called Graph for the graph itself, a class UI for printing the menu and using the operations defined in graph and an error class for error handling. I also used 2 additional functions to save the graph to a file and one to load a graph from a file.

The class Graph has the following functions:

vertices\_iterator(self):

Iterator for the vertices

neighbours\_iterator(self, vertex)

Iterator for the neighbours of a vertex

edges\_iterator(self)

*Iterator for the edges*

is\_vertex(self, vertex)

*Check if a vertex belongs to the graph*

is\_edge(self, vertex1, vertex2)

*Check if the edge of vertex1 and vertex2 belongs to the graph*

count\_vertices(self)

***return****: Number of vertices in the graph*

count\_edges(self)

***return****: Number of edges in the graph*

in\_degree(self, vertex)

***return****: Number of edges with endpoint vertex*

out\_degree(self, vertex)

***return****: Number of edges with startpoint vertex*

get\_edge\_cost(self, vertex1, vertex2)

***return****: The cost of an edge*

set\_edge\_cost(self, vertex1, vertex2, new\_cost)

*Set the cost of an edge*

add\_vertex(self, vertex)

*Add a vertex to the graph*

add\_edge(self, vertex1, vertex2, edge\_cost = 0)

*Add an edge to the graph*

remove\_edge(self, vertex1, vertex2)

*Remove edge from the graph*

remove\_vertex(self, vertex)

*Remove a vertex from the graph*

copy(self)

***return****: Deepcopy of the graph*

The graph is initialized with: self.\_vertices = the number of vertices of the graph (a set)

* Self.\_neighbours = number of neighbours of a vertex(a dict)
* Self.\_transpose = number of inbound neighbours of a vertex(a dict)
* Self.\_cost = cost of an edge (a dict)

Additional functions:

read\_file(file\_path)

Reads a graph from a file

write\_file(file\_path, g)

Saves a graph to a file

random\_graph(vertices\_no, edges\_no)

Creates a random graph with a given number of vertices and edges