Documentation

We will define a class Graph alongside the methods for randomly generation one and to load/save to a file(+testing). The Graph class is used to represent a directed graph with the ability to select the number of edges and vertices. Initializing means creating three dictionaries that each represent: the outbound connections of a vertex meaning all the vertices it has an edge going to from it, the inbound connections of a vertex meaning all the vertices that have an edge going to it, and the costs for all edges(in this case all source, destination vertex pairs).

Next we will present the methods used. Unless specified otherwise, the methods presented each receive as parameters the current instance of the Graph class (self). The methods defined in the Graph class are:

- __init__ : initializes the graph and receives as parameter the number of vertices of the graph

- vert_iter : iterates through the vertices of the graph

- is_vertex : returns True if a certain value is a vertex belonging to the graph or False otherwise, receiving as parameter a value ver

- count_vertices: returns the number of the vertices present in the graph

- edge_iter : the iterator for the edges of the graph

- is_an_edge : returns True if there exists an edge going from a vertex src to another vertex dest and False otherwise, receiving as parameters two values src and dst

- count_edges : returns the number of edges in the graph

- copy : copies the current state of the graph(edges, vertices, costs) and returns it

- source_iter : iterates through the list of vertices that are outbound vertices for a certain value ver, receiving as parameter a value ver. Also throws an exception if ver is not a valid vertex of the graph

- destination_iter : iterates through the list of vertices that have edges going into a certain value ver, receiving as parameter a value ver. Also throws an exception if ver is not a valid vertex of the graph

- degree_inbound : returns how many vertices have an edge going into the vertex ver, reveiving as a parameter a value ver. Throws an exception if ver is not a valid vertex of the graph

- degree_outbound : returns how many vertices have an edge going into the vertex ver, reveiving as a parameter a value ver. Throws an exception if ver is not a valid vertex of the graph

- get_cost : returns the cost of the edge going from a vertex src to a vertex dst, receiving as parameters two values src and dst. Throws an exception if there is no edge from src to dst

- set_cost : sets the cost of the edge going from a vertex src to a vertex dst, receiving as parameters two values src and dst. Throws an exception if there is no edge from src to dst

- add_vertex : adds a vertex ver to the list of vertices of the graph, receiving as parameter a value ver. Throws an exception if ver is already an existent vertex

- remove_vertex : removes a vertex ver from the list of vertices of the graph, alongside any existent edges going from or to it, receiving as parameter a value ver. Throws an exception if ver is not a valid vertex of the graph

- add_edge : creates an edge going from a vertex src to a vertex dst, also associating a cost to the edge, receiving as parameters three values : src, dst and cost. Throws an exception if there already exists an edge from src to dst or if either src or dst are not valid vertices of the graph

- remove_edge : removes an edge going from a vertex src to a vertex dst, receiving as parameters two values src and dst. Throws an exception if there is no existent edge going from src to dst

In order to implement the Graph class, we used Python specific dynamically allocated lists, one unordered set for the vertices of the graph, as well as dictionaries to retain the inbound/outbound vertices of every vertex and also the cost of every edge