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

We need two auxiliary classes:

FileManager:
Helps save, load to a file and generate a random graph
VertexIter:
Is used to iterate over the vertices of the graph
The class Graph will provide the following methods:

__init__()
Constructs a graph without vertices and without arcs.
GetVertexNr(self)
Returns the number of vertices
AddEdge(self,x,y,z)
Adds an edge from x to y, with cost z associated. Return True on success, False if the edge already exist
s. Precond: x and y exists
RemoveEdge(self,x,y)
Removes the edge between vertex x and y. Return True on success, False if the edge doesn't exist.
AddVertex(self,x)
Adds a vertex with the key x. Return True on success, False if the vertex already exists.
RemoveVertex(self,x)
Removes the vertex with the key x. Return True on success, False if the vertex doesn't exist.
GetCost(self,x,y)
Returns the cost of the edge between x and y. Precond: An edge exists between x and y
ModifyCost(self,x,y,z)
Changes the cost of the edge between x and y to the value of z. Precond: An edge exists between x and
y
__copy__(self)
Returns a copy of the current graph.
parseNOut(self,x)
Returns the outgoing edges of the vertex x.
parseNIn(self,x)
Returns the outgoing incoming of the vertex x.
isEdge(self,x,y)
Returns true if there is an edge between x and y, returns false otherwise.
GetVertices(self)
Returns a list of all vertices.
__iter__(self)
Is used to iterate over the vertices of the graph.

Class Graph will have the following data members:
nr_vertices - represents the number of vertices
nr_edges - represents the number of edges
out_neighbours - a dictionary where the values of a key represent that vertex's outbound neighbours
in_neighbours - a dictionary where the values of a key represent that vertex's inbound neighbours
cost_map - a dictionary where the keys are a tuple of two vertices, the value representing the cost betwee
n the two vertices