Gutanu Nicola Carina, group 914, subgroup 914/1

I chose Python for my project.
In order to represent my graph, I chose 3 dictionaries. The first two have keys from 0 to n-1 vertices and have lists that indicate the in and out vertices for each key. The third dictionary is for costs, each edge in the graph is a tuple key and holds the cost to move from the beginning vertex to the end one.

Functions:
In the TripleDictGraph class:

1. def __init__(self, nr_of_vertices):
   - Input: number of vertices
   - Output: creates the graph with the three empty dictionaries

2. def is_edge(self, x, y):
   - You will input the two vertices between which you want to check if there is an edge
   - Return 1 if there exists and edge and 0 otherwise

3. def is_vertex(self, x):
   - Input: a vertex
   - Output: 1 if it's a vertex, 0 otherwise

4. def add_edge(self, x, y, c):
   - Input: an edge and its cost (ex: 2 3 100)
   - Precondition: the edge has to not already exist in the graph and the vertices have to exist
   - Output: will add the edge if everything is alright, the beginning and end vertices in each of the in and out dictionaries and then in the cost dictionary the tuple with its cost

5. def print_edges(self):
   - Output: will print all edges existent in the graph

6. def add_vertex(self, vertex):
   - Input: a vertex
   - Precondition: the vertex can't already exist
   - Output: a new key that is the new vertex will be added in the in and out dictionaries with an empty list

7. def remove_vertex(self, vertex):
   - Input: a vertex
   - Precondition: the vertex has to exist in the graph
   - Then the function will check if there are any edges that contain this vertex and delete them
   - Output: will remove the vertex key along with its list from the in and out dictionaries

8. def remove_edge(self, x, y):
   - Input: an edge (ex: 2 3)
   - Precondition: the edge has to exist in the graph
   - Then for each of the in and out vertices, they will be removed from the in and out dictionaries
   - Output: the tuple will be deleted from the cost dictionaries

9. def nr_of_vert(self):
   - Output: will return the number of vertices in the graph

10. def print_dicts(self):
    - Output: will print the dictionaries in the form that they are kept in the memory
    - For in and out dictionaries: key + list
    - For cost dictionary: the edge tuple and its cost

11. def degree(self, vertex):
    - Input: a vertex
    - Precondition: the vertex has to exist in the graph
    - Output the in and out degree of the vertex

12. def parse_vert(self):
    - Output: iterates the set of vertices

13. def parse_outbound(self, vertex):
    - Input: a vertex
    - Precondition: the vertex has to exist in the graph
    - Output: iterates the set of outbound edges of a given vertex

14. def parse_inbound(self, vertex):
    - Input: a vertex
    - Precondition: the vertex has to exist in the graph
    - Output: iterates the set of inbound edges of a given vertex

15. def change_cost(self, x, y, c):
    - Input: an edge and the new cost (ex: 2 3 100)
    - Precondition: the edge has to exist in the graph
    - Output: the cost of the edge will be updated

16. def copy_graph(self):

- Output: a copy of the graph

Outside the TripleDictGraph class:

1. def random_graph():
    - Input: number of vertices and number of edges
    - Output: a randomly generated graph

2. def print_menu():
    - Contains the menu with the commands

3. def run():
    - Input: file name
    - Prints the menu and calls the above mentioned commands

4. def save_changes(g):
    - Input: a graph and file name
    - Output: the graph is saved in the given file

5. def file_graph():
    - Input: file name
    - Output: the graph read from the file