**Ex3:Directed Weighted Graphs in python**

**Task:**

We want to create a python project that when given a json file such as A0.json (which you can find in this git project)

the program will be able to save the graph and perform changes on it such as: add a node (point), remove node, add edge, remove edge.

In addition the program will have the option to present the graph on screen via the class "gui"

Aside from the load function that will load the json file into a graph and save that will save changes made on the graph and present it on the json file the user will be able to run algorithms on the graph:

shortestPath: *Computes the the shortest path between src to dest - as an ordered List of nodes:  
src--> n1-->n2-->...dest*

Center: *Finds the NodeData which minimizes the max distance to all the other nodes.  
Assuming the graph isConnected, elese return null*

Tsp: *Computes a list of consecutive nodes which go over all the nodes in cities.  
the sum of the weights of all the consecutive (pairs) of nodes (directed) is the "cost" of the solution -  
the lower the better.*

**How the algorithm works?:**

The algorithm contain 6 classes:

**checks-**is a very basic class that i added in it you can find all the important imports already ready for use as well as an empty graph with an empty string all the user need to do is enter a json graph path and he will be able to freely use commands on the graph like add node remove node add edge or the more complex ones that are

Stated above in TASK

**MyEdges-**the edges of the graph each edge have a src(the starting node) dest(the node he is heading) and

Weights(the edge weight)

**MyNode**-the points each point have a x,yz to keep track of his location on the graph and an id

**MyGraph**- the graph which contains points(dict)-the points are saved in an dict given the node id as a key will give the node as a value.

Edged(dict)-the edges the key will be the src node id and the value will be another dict the key will be the edge destination and the value will be MyEdge which will contain all the data regarding the edge.

**MyGraphAlgo-**the algorithms the class contain a MyGraph graph all the algorithm are going to run on the given graph

**Gui-**this class will allow the user the get a gui of the graph on screen more information about how to use this class is presented in how to run section below

The classes are implements of the given interfaces:

GraphInterface

GraphAlgorithmInterface

**How the algorithm works?:**

Each class in the algorithm is used as a field in other class graphalgorithm require a graph the graph require nodes and edges

The most important class here is MyGraph as this is the class which create a graph for us to work on without this class we wont be able to do anything this class is implemented by 2 dict when the value of 1 of the given dict is another dict the reason the class is implemented that way is to allow the user to get the most important function that will build and present the graph in O(1) (which mean almost instantly) without need to worry about every little action to take a long time.

the algorithms that are presented in MyGraphAlgo are mostly known algorithms such as dijktra heres a link for more info about the algorithm:

<https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/>

Node and MyEdge are very simple classes that while taking important part in the process are relatively easy to understand

**How to run?:**

Running the algorithms is rather simple after you transfer all the classes stated in how the algorithm works section above you can use 2 classes in order to present and work on your graph the first class is checks by inputting json file path into “s“(which already presented there)

The class will transfer the json in to a graph and put him in the algorithm class so you will be able to run all the algorithms stated in task section above by simply enter the commands you want to perform on the graph.

in order to use a complex algorithm from mygraphalgo just use “algo.(enter command after dot)” and in order to perform an action on the graph use “algo.graph.(enter command after dot)”

the second class is gui this class is much simpler all that needed to be done is to run the class after running it the user will be asked to input a json graph file path after inputting the file the graph will be presented on screen

**Performance**

The program is able to present up to 100000(and possibly more) in a matter of seconds

**Important for the checker of this task!! Read before checking the code:**

Unfortunately I wasn’t able to complete everything in time the functions:

center,tsp will not work