GRAPH CLASS

self.outbound = dict(), vertices as keys, list as value

self.inbound = dict() vertices as keys, list as value  
self.costs = dict() edges as keys, integer as value

def \_\_init\_\_(self, m=set(), n=set()):  
 """  
 this function is the constructor for the graph class  
 :param m: set of vertices, empty if not specified in the constructor function  
 :param n: set of edges, empty if not specified in the constructor function  
 """

def \_\_init\_\_(self, m=set(), n=set()):  
 """  
 this function is the constructor for the graph class  
 :param m: set of vertices, empty if not specified in the constructor function  
 :param n: set of edges, empty if not specified in the constructor function  
 """

def add\_vertex(self, vertex):  
 """  
 it adds the vertex as a key in both the inbound and outbound dictionaries an initializes their values with an empty list  
 :param vertex:  
 :return: nothing  
 """

def add\_edge(self, starting\_point, ending\_point, value):  
 """  
 checks if the vertices are part of the graph(aka are keys in the inbound and outbound dictionaries), adds them  
 if necessary and then adds the edge to the costs dictionary with the value assigned  
 :param starting\_point: vertex  
 :param ending\_point: vertex  
 :param value: integer  
 :return: raises ValueError if edge already exists  
 """

def set\_cost(self, starting\_point, ending\_point, value):  
 """  
 this function assigns a value in the costs dictionary to a specified edge(specified points)  
 :param starting\_point: vertex  
 :param ending\_point: vertex  
 :param value: integer  
 :return: raises ValueError if the edge doesnt exist  
 """

def get\_cost(self, starting\_point, ending\_point):  
 """  
 this function returns the cost of an edge  
 :param starting\_point: vertex  
 :param ending\_point: vertex  
 :return: integer  
 """

def get\_number\_vertices(self):  
 """  
 :return: Returns the number of vertices.  
 """

def parse\_vertices(self):  
 """  
 :return: returns a list of all the edges in the graph  
 """

def parse\_nout(self, x):  
 """  
 :param x: vertex  
 :return: Returns the outbound list of the vertex x  
 """

def parse\_nin(self, x):  
 """  
 :param x: vertex  
 :return: Returns the inbound list of x  
 """

def get\_in\_out\_degree(self, x):  
 """  
 :param x: vertex  
 :return: Returns the in and out degrees of a given vertex. Raises ValueError if vertex does not exist.  
 """

def parse\_graph(self):  
 """  
 This function parses the inbound and outbound lists of all vertices in the graph and show how much time it took  
 :return:  
 """

def remove\_edge(self, starting\_point, ending\_point):  
 """  
 Removes an edge from the graph. The endpoints lose each other from their in/outbound lists.  
 Raises ValueError if the edge does not exist.  
 :param starting\_point: vertex  
 :param ending\_point: vertex  
 :return: none  
 """

def remove\_vertex(self, vertex):  
 """  
 removes a vertex from the graph as well as all the edges to or from that vertex from the costs dictionary  
 :param vertex: a vertex of the graph  
 :return:  
 """

def is\_edge(self, x, y):  
 """  
 checks in (x,y) is an edge  
 :param x: vertex  
 :param y:vertex  
 :return: true or false  
 """

def get\_all\_str(self):  
 """  
 This function returns the contents of the repository in form of a list of strings  
 :return: a list of strings  
 """

def get\_all\_edges(self):  
 """  
 This function returns the contents of the repository in form of a list of objects  
 :return: a list of objects  
 """

def copy\_graph(self):  
 """  
 this function creates another graph object with the same edges, vertices and costs of an instance  
 :return: graph object  
 """

OUTSIDE GRAPH CLASS

def build\_random\_graph(n, m):  
 """  
 builds a random graph  
 :param n: integer  
 :param m: integer  
 :return: a graph  
 """

def set\_cost\_to\_edge():  
 """  
 this function returns a random integer from 1 to 100 which can be assigned to an edge as its cost  
 :return: integer  
 """

def load\_file(file\_name):  
 """  
 Reads the information about a graph from a file, in the required form.  
 :param file\_name: a string, a valid file name  
 :return: a graph object  
 """

def save\_file(graph, file\_name):  
 """  
 Writes the information about a graph in a file, in the required form.  
 :param graph: a graph object which contains a dictionary with edges as keys  
 :param file\_name: a string, a valid file name  
 :return: none  
 """

def valid(file\_name):  
 """  
 Checks if a file\_name is valid aka .txt appears in its name  
 file\_name: a string  
 :return true or false  
 """