Specification

We shall define a class named Graph representing a *directed graph*.

This class uses three dictionaries to store the graph:

- One for the outbound vertices
- One fot the inbound vertices
- And one for the edges between two vertices and the cost associated with that edge

Representation examples:

Ex graph list:

8 17

0 1 0

1 7 67

5 0 12

1 0 87

3 1 21

6 0 15

1 4 64

6 2 17

6 6 47

4 4 62

5 6 4

1 3 22

5 7 38

5 2 41

2 2 65

6 7 9

7 4 48

The resulting dictionaries are :

self.dout = {

0: [1],

1: [7, 0, 4, 3],

2: [2],

3: [1],

4: [4],

5: [0, 6, 7, 2],

6: [0, 2, 6, 7],

7: [4]

}

self.din = {

0: [5, 1, 6],

1: [0, 3],

2: [6, 5, 2],

3: [1],

4: [1, 4, 7],

5: [],

6: [6, 5],

7: [1, 5, 6]

}

self.dcosts = {

(0, 1): 0, (1, 7): 67,

(5, 0): 12, (1, 0): 87,

(3, 1): 21, (6, 0): 15,

(1, 4): 64, (6, 2): 17,

(6, 6): 47, (4, 4): 62,

(5, 6): 4, (1, 3): 22,

(5, 7): 38, (5, 2): 41,

(2, 2): 65, (6, 7): 9,

(7, 4): 48

}

The class Graph will provide the following methods:

```
class Directed_Graph:
    def __init__(self, graph_list):
        '''
        input: list of edges with costs in the form : <vertex 1> <vertex 2> <value associated with the edge>
        '''
    def check_vertex_validity(self, vertex):
        '''
        input: vertex of a graph
        out: true if the vertex is valid
        '''
    def save_to_file(self, file_name):
        '''
        input: name of file to save tha graph
        '''
    def check_edge_validity(self, vertex1, vertex2):
        '''
        pre: valid verteces
        input: 2 verteces
        out: true if there is an edge between vertex1 and vertex2
        '''
    def set_dicts(self):
        '''
        Initialize the dictionaries
        '''
    def add_vertex(self, vertex):
        '''
        pre: vertex not already in graph
        input: vertex to add
        '''
    def get_cost(self, vertex1, vertex2):
        '''
        pre: vertex1 and vertex2 are valid
        input: two verteces
        out: the cost associated with the edge between vertex1 and vertex2
```

```
        '''
    def set_cost(self, vertex1, vertex2, cost):
        '''
        pre: vertex1 and vertex2 are valid and there exists and edge between them
        input: two verteces and a cost
        '''
    def remove_vertex(self, vertex):
        '''
        pre: vertex1 is valid
        input: vertex to remove
        '''
    def remove_edge(self, vertex1, vertex2):
        '''
        pre: vertex1 and vertex2 are valid and there exists and edge between them
        input: two verteces
        '''
    def add_edge(self, vertex1, vertex2, cost):
        '''
        pre: vertex1 and vertex2 are valid and there does not exists and edge between them
        input: two verteces and a cost
        '''
    def edge_between(self, vertex1, vertex2):
        '''
        pre: vertex1 and vertex2 are valid
        input: two verteces
        out: True if there is an edge between the vertices
        '''

def out_degree(self, vertex):
        '''
        pre: vertex is valid
        input: vertex
        out: the outbound degree of the given vertex
        '''
    def in_degree(self, vertex):
        '''
        pre: vertex is valid
        input: vertex
        out: the inbound degree of the given vertex
        '''
    def out_vertices(self, vertex):
        '''
        pre: vertex is valid
        input: vertex
        out: the outbound vertices of the given vertex
        '''
    def in_vertices(self, vertex):
        '''
        pre: vertex is valid
        input: vertex
        out: the inbound vertices of the given vertex
        '''
    @property
    def Number_of_vertices(self):
        '''
        Returns the number of vertices
        '''
    @property
    def Number_of_edges(self):
        '''
        Returns the number of edges
        '''
    @property
    def Vertices(self):
        '''
        out: the list of vertices
        '''

    def copy_Graph(self):

        '''
        out: Returns a copy of the graph
        '''
```