Specification

We shall define a class named `DoubleDictGraph` representing a *directed graph*.

The class `DoubleDictGraph` will provide the following methods:

```
def __init__(self, number_of_nodes)
```
Constructs a graph with *“number_of_nodes”* vertices and without arcs.

```
def get_vertices(self)
```
Returns an iterable containing all the vertices.

```
def get_number_of_vertices(self)
```
Returns the number of vertices in the graph.

```
def get_outbound_neighbours_of_vertex_X(self, x)
```
Returns a list containing the outbound neighbours of x.

```
def get_inbound_neighbours_of_vertex_X(self, x)
```
Returns a list containing the inbound neighbours of x.

```
def is_edge(self, x, y)
```
Returns True if there is an edge from x to y, False otherwise.

```
def add_edge(self, x, y, cost)
```
Adds an edge from x to y with the cost "cost".
Precondition: there is no edge from x to y

```
def remove_edge(self, x, y)
```
Removes the edge from x to y.
Precondition: there is an edge from x to y.

```
def get_in_and_out_degree(self, vertex_number)
```
Returns the in degree and out degree of a vertex in a tuple.

```
def parse_outbound_edges_of_vertex_x(self, vertex_x)
```
Returns a list with all the outbound edges starting from the vertex x.

```
def parse_outbound_edges_of_vertex_x(self, vertex_x)
```
Returns a list with all the inbound edges startpoints which end in the vertex x.

```
def retrieve_edge_cost(self, vertex_x, vertex_y)
```
Returns the cost of an edge.

```
def modify_edge_cost(self, vertex_x, vertex_y, new_cost)
```
Modifies the cost of an edge which is stored in the dictCosts dictionary of edges.

```
def add_vertex(self, vertex_x)
```
Adds a vertex to the graph.
Precondition: The vertex_x should not exist already.

```
def remove_vertex(self, vertex_x)
```
Removes a vertex from the graph and all the edges associated with it.

```
def copy(self)
```
Creates a copy of the graph and returns it.

```
@staticmethod
def read_graph_from_text_file(filename)
```
A static method which reads from a file a graph, creates it and returns it.

```
def write_graph_to_text_file(self, file_name)
```
A method which writes to a text file a graph.

```
@staticmethod
def create_random_graph(number_of_vertices, number_of_edges)
```
Creates a random graph with a certain number of vertices and edges and returns it.

Implementation

The implementation uses three dictionaries, one for the outbound edges of each node, another for the inbound edges, and the last for the costs of each edge.

```
class DoubleDictGraph:
      def __init__(self, number_of_nodes):
            self._dictOut = {}
            self._dictIn = {}
            self._dictCosts = {}

            for i in range(number_of_nodes):
                  self._dictOut[i] = []
                  self._dictIn[i] = []
```

Class `DoubleDictGraph` will therefore have the following data members:

`self._dictOut`
: stores the outbound edges of each node

`self._dictIn`
: stores the inbound edges of each node

`self._dictCosts`
: stores the costs of each edge