```
# The class that handles the graph implementation.
class Graph:

      # The graph constructor.
      def __init__(self, verticesCount : int)

      """
      :returns The number of vertices of the graph.
      """
      def get_vertices_count(self) -> int

      """
      :returns The number of edges of the graph.
      """
      def get_edge_count(self) -> int

      """
      :returns The list of vertices of the graph.
      """
      def parse_vertices(self) -> list

      """
      :vertex The vertex for which to get the neighbours.
            :precondition: Must be a vertex of the graph.
      :returns The inbound neighbours of that vertex.
      """
      def parse_inbound_neighbours(self, vertex : int) -> list

      """
      :vertex The vertex for which to get the neighbours.
      :precondition: Must be a vertex of the graph.
      :returns The outbound neighbours of that vertex.
      """
      def parse_outbound_neighbours(self, vertex : int) -> list

      """
      :vertex The vertex for which to get the edges.
      :precondition: Must be a vertex of the graph.
      :returns The inbound edges of that vertex.
      """
      def parse_inbound_edges(self, vertex : int) -> list

      """
      :vertex The vertex for which to get the edges.
      :precondition: Must be a vertex of the graph.
      :returns The outbound edges of that vertex.
      """
      def parse_outbound_edges(self, vertex : int) -> list
```

```
"""
:vertex The vertex for which to get the degree.
      :precondition: Must be a vertex of the graph.
:returns The in degree of that vertex.
"""
def in_degree(self, vertex : int) -> int

"""
:vertex The vertex for which to get the degree.
:precondition: Must be a vertex of the graph.
:returns The out degree of that vertex.
"""
def out_degree(self, vertex : int) -> int

"""
Adds the specified vertex to the graph.
:vertex The vertex that needs adding.
"""
def add_vertex(self, vertex : int)

"""
Removes the specified vertex from the graph.
:vertex The vertex that needs removing.
      :precondition: Must be a vertex of the graph.
"""
def remove_vertex(self, vertex : int) -> int

"""
Adds the specified edge with the specified cost to the graph.
:source The source of the edge.
      :precondition: Must be a vertex of the graph.
:target The target of the edge.
      :precondition: Must be a vertex of the graph.
:cost The cost of the edge.
"""
def add_edge(self, source : int, target : int, cost : int)

"""
Removes the specified edge from the graph.
:source The source of the edge.
      :precondition: Must be a vertex of the graph.
:target The target of the edge.
      :precondition: Must be a vertex of the graph.
"""
def remove_edge(self, source : int, target : int)
```

```
"""
Checks if there is an edge between the source and the target.
:source The source of the possible edge.
      :precondition: Must be a vertex of the graph.
:target The target of the possible edge.
      :precondition: Must be a vertex of the graph.
:returns True if there is an edge, false otherwise.
"""
def is_edge(self, source : int, target : int) -> bool

"""
Checks if the vertex exists.
:vertex The possible vertex.
:returns True if the vertex exists, false otherwise.
"""
def is_vertex(self, vertex : int) -> bool

"""
Sets the cost of the specified edge.
:source The source of the possible edge.
      :precondition: Must be a vertex of the graph.
:target The target of the possible edge.
      :precondition: Must be a vertex of the graph.
:cost The new cost of the possible edge.
"""
def set_cost(self, source : int, target : int, cost : int)

"""
Gets the cost of the specified edge.
:source The source of the possible edge.
      :precondition: Must be a vertex of the graph.
:target The target of the possible edge.
      :precondition: Must be a vertex of the graph.
:returns The cost of the possible edge.
"""
def get_cost(self, source : int, target : int) -> int

"""
:returns a deep copy of the graph.
"""
def copy_graph(self)
```