```python
from copy import deepcopy

class Graph:
    def __init__(self, n=0):
        self.__n = n
        self.__m = 0
        self.__vertices = list()
        self.__neighbors = dict()
        self.__costs = dict()

        for i in range(n):
            self.__vertices.append(i)
            self.__neighbors[i] = []

    def add_vertex(self, v:int):
        if self.valid_vertex(v) == False:
            raise Exception("Invalid vertex!\n")

        if self.existing_vertex(v) == True:
            raise Exception("Vertex already exists!\n")

        self.__vertices.append(v)
        self.__neighbors[v] = []
        self.__n = max(self.__n, v)

    def remove_vertex(self, v:int):
        if self.valid_vertex(v) == False:
            raise Exception("Invalid vertex!\n")

        if self.existing_vertex(v) == False:
            raise Exception("Vertex not found!\n")

        cnt = 0
        for neighbors in self.__neighbors.values():
            if v in neighbors:
                self.remove_edge(self.__vertices[cnt], v)
            cnt += 1

        self.__neighbors.pop(v)
        self.__vertices.remove(v)

    def add_edge(self, v_from:int, v_to:int, e_cost:int):
        if self.valid_edge(v_from, v_to) == False:
            raise Exception("Invalid edge!\n")

        if self.existing_edge(v_from, v_to) == True:
            raise Exception("Edge already exists!\n")

        self.__neighbors[v_to].append(v_from)
        self.__neighbors[v_from].append(v_to)
        self.__costs[(v_from, v_to)] = e_cost
        self.__costs[(v_to, v_from)] = e_cost
        self.__m += 1

    def remove_edge(self, v_from:int, v_to:int):
        if self.existing_edge(v_from, v_to) == False:
            raise Exception("Edge does not exist!\n")

        self.__neighbors[v_to].remove(v_from)
        self.__neighbors[v_from].remove(v_to)
        self.__costs.pop((v_from, v_to))
        self.__costs.pop((v_to, v_from))
        self.__m -= 1

    @property
    def vertices(self):
        return deepcopy(self.__vertices)

    def get_neighbors(self, node:int):
        vertices = deepcopy(self.__neighbors[node])
        neighbors = []

        for v in vertices:
            neighbors.append((v, self.__costs[(node, v)]))

        return neighbors

    @property
    def get_nr_of_vertices(self):
        return len(self.__vertices)

    @property
    def get_nr_of_edges(self):
        return self.__m

    def vertices_iterator(self):
        for vertex in self.__vertices:
            yield vertex
    def edges_iterator(self):
        visited = set()

        for v_from in self.__neighbors.keys():
            for v_to, cost in self.get_neighbors(v_from):
                if v_to not in visited:
                    yield (v_from, v_to, cost)

            visited.add(v_from)


    def copy(self):
        return deepcopy(self)

    def existing_vertex(self, v:int):
        return v in self.__vertices

    def valid_vertex(self, v:int):
        return  v >= 0

    def valid_edge(self, v_from:int, v_to:int):
        return self.valid_vertex(v_from) and self.valid_vertex(v_to) and self.existing_vertex(v_from) and self.existing_vertex(v_to)

    def existing_edge(self, v_from:int, v_to:int):
        return v_from in self.__neighbors.keys() and v_to in self.__neighbors[v_from]

    def clear(self):
        self.__n = 0
        self.__m = 0
        self.__vertices.clear()
        self.__neighbors.clear()
```