Practical work no. 5

First approach to get a vertex cover of an graph:

The algorithm picks a random edge and adds the corresponding vertices to the result set. Then removes all the edges that are incident with the added vertices. The complexity of the algorithm is O(V + E) and it returns a vertex cover that is twice the minimum cover of the graph.

```
def approximateVertexCover(self):
    result = set()
    while self.__E != []:
        # choose random (u,v) edge
        (u, v) = choice(self.__E)

        # add u and v to the result set, if they aren't already
in there
        if u not in result:
            result.add(u)
        if v not in result:
            result.add(v)

        # delete all the edges having as endpoints u or v

        ok = True
        while ok:
            ok = False
            for i in range(len(self.__E)):
                if self.__E[i][0] == u or self.__E[i][1] == u or
self.__E[i][0] == v or self.__E[i][1] == v:
                    del self.__E[i]
                    ok = True
                    break

    return result
```

The second approach is using a greedy algorithm. We first determine the vertex with maximum degree and then we perform the same operation. Add it to the result and then remove this vertex from the dictionary of edges. The complexity of this algorithm is O(logn). In some cases will reutrn the minimum vertex cover and in some cases it will not return the minimum vertex cover.

```python
def greedyVertexCover(self):
    solution = set()
    while self.__dictEdges != {}:
        # choose the vertex with the max degree
        v = self.getMaxDegreeVertex()
        # check is not isolated
        if self.getDegree(v) == 0:
            break
        # add vertex to solution
        solution.add(v)

        #delete all edges having connection with the vertex v
        del self.__dictEdges[v]
        ok = True
        while ok:
            ok = False
            for x in self.__dictEdges:
                if v in self.__dictEdges[x]:
                    self.__dictEdges[x].remove(v)
                    ok = True
                    break
    return solution
```