Practical work nr.2

```
This function reads an undirected graph.
def undirected_load(self):
    with open(self.__file_name, "r") as content:
        for lines in content:
            line = lines[:-1]
            data = line.split(" ")
            if len(data) == 2:
                self.__vertices = int(data[0])
                self.__edges = int(data[1])
            else:
                if int(data[0]) not in self.__dicin.keys():
                    self.__dicin[int(data[0])] = []
                if int(data[0]) not in self.__dicout.keys():
                    self.__dicout[int(data[0])] = []
                if int(data[1]) not in self.__dicin.keys():
                    self.__dicin[int(data[1])] = []
                if int(data[1]) not in self.__dicout.keys():
                    self.__dicout[int(data[1])] = []
                self.add_edge(int(data[1]), int(data[0]), int(data[2]))
                self.add_edge(int(data[0]),int(data[1]),int(data[2]))


    def DFS(self, visited, vertex, recent_connected_components: list):
        """
        :param visited: list, that holds the truth value, whether a vertex was visited or not
        :param vertex: current visited vertex
        :param recent_connected_components: list that holds the current connected components
        :return:
        """
        visited[vertex] = True  #The vertex is marked as visited
        recent_connected_components.append(vertex) # The vertex is added to the current
connected component list
        for v in self.__dicin[vertex]:
            if not visited[v]:
                recent_connected_components = self.DFS(visited, v, recent_connected_components) #
We run recursively this modified DFS for a vertex that is connected to the initial one, and is not
visited yet.
        return recent_connected_components # We return the current connected component as a
list

    def connected_components(self):
        visited = [False for _ in self.__dicin.keys()] # we mark every vertex as not visited
        connected_components_list = [] #in this list we store the connected components as lists
        graphs=[] # in this list we store the connected components as graphs
        for vertex in self.__dicin.keys():
            if not visited[vertex]:
```

```
            cc = [] # This will hold the current connected component
            graph_1=Graph(0,"empty.txt") #We create an empty graph object, to store the
connected components as a graph
            connected_components_list.append(self.DFS(visited, vertex, cc)) # We add to the
connected component list the connected component we just found
            for vertex_1 in cc: # in this for we add the vertices and edges to the graph from the
connected component
                if vertex_1 not in graph_1.__dicout.keys():
                    graph_1.__dicin[vertex_1]=[]
                    graph_1.__dicout[vertex_1]=[]
                for v in self.__dicin[vertex_1]:
                    if v not in graph_1.__dicout.keys():
                        graph_1.__dicout[v]=[]
                        graph_1.__dicin[v]=[]
                    graph_1.add_edge(vertex_1,v,1)
            graphs.append(graph_1)
        return connected_components_list,graphs
```

In main, we can call this search, by inputing a filename

```
        undirected_file=str(input("What is the file: "))
        graph=Graph(0,undirected_file) # This is the undirected graph
        graph.undirected_load() # We load the undirected graph
        cc,graphs=graph.connected_components()
        for graph in graphs: #For each connected component, we print its vertices and edges
                print("Connected component: ")
                print(graph.return_vertices())
                print("Edges: ")
        for v in graph.return_edges():
                print(v)
```

Here are the source codes in a format which can be copied:
The plus lines in main

```
if option==5:
    undirected_file=str(input("What is the file: "))
    graph=Graph(0,undirected_file) # This is the undirected graph
    graph.undirected_load() # We load the undirected graph
    cc,graphs=graph.connected_components()
    for graph in graphs: #For each connected component, we print its vertices
and edges
        print("Connected component: ")
        print(graph.return_vertices())
        print("Edges: ")
        for v in graph.return_edges():
            print(v)
```

```
def connected_components(self):
    visited = [False for _ in self.__dicin.keys()]  # we mark every vertex as
not visited
    connected_components_list = [] #in this list we store the connected
components as lists
    graphs = [] # in this list we store the connected components as graphs
    for vertex in self.__dicin.keys():
        if not visited[vertex]:
            cc = [] # This will hold the current connected component
            graph_1 = Graph(0, "empty.txt") #We create an empty graph object, to
store the connected components as a graph
            connected_components_list.append(self.DFS(visited, vertex, cc)) # We
add to the connected component list the connected component we just found
            for vertex_1 in cc: # in this for we add the vertices and edges to
the graph from the connected component
                if vertex 1 not in graph 1.  dicout.keys():
                    graph_1.__dicin[vertex_1] = []
                    graph_1.__dicout[vertex_1] = []
                for v in self.__dicin[vertex_1]:
                    if v not in graph_1.__dicout.keys():
                        graph_1.__dicout[v] = []
                        graph_1.__dicin[v] = []
                    graph_1.add_edge(vertex_1, v, 1)
            graphs.append(graph_1)
    return connected_components_list, graphs
```

```
def DFS(self, visited, vertex, recent_connected_components: list):
    """

    :param visited: list, that holds the truth value, whether a vertex was
visited or not
    :param vertex: current visited vertex
    :param recent_connected_components: list that holds the current connected
components
    :return:
    """
    visited[vertex] = True  # The vertex is marked as visited
    recent_connected_components.append(vertex)  # The vertex is added to the
current connected component list
    for v in self.__dicin[vertex]:
        if not visited[v]:
            recent_connected_components = self.DFS(visited, v,
recent_connected_components) # We run recursively this modified DFS for a vertex
that is connected to the initial one, and is not visited yet.
    return recent_connected_components # We return the current connected
component as a list
```