3National Research University Higher School of Economics (Higher School of Economics/HSE)
Faculty of Computer Science
Bachelor's Programme Data Science and Business Analytics
01.03.02 Applied Mathematics and Computer Science

Internship report

Fulfilled by

_____
(Surname, Given Name, Middle Name if any)

_____
(signature)

Checked by

_____ _____ _____
(job or academic title) (*surname, initials)* (signature)

Moscow

Table of content:

The problem: City Connector

Task:

For a given country, it is necessary to connect cities by roads so that you can get from any city to any other. Cities are given by their geographical coordinates, the roads are drawn directly and the length of each is the Euclidean distance between them. The target spanning tree should be visualised and obtained distances should be compared with existing.

Goals:

Learn how to manage data and use it for creating graphs, finding MST(minimal spanning tree) and using obtained information for maps. Also, this kind of project encourages us for team-work.

Preparations:

1. Obtaining data
2. Creating Graphs for different methods.

All data was collected by using geonames.org api. This one allows users to request a lot of information about countries or regions. Needed for us was list of cities with their latitude (lat) and length (len)

```
paa={'country':'UK','maxRows':'300', 'type':'json', 'username':"premium", 'cities':'cities1000'}
r =rs.get('http://api.geonames.org/search?', paa)
```

The form of our request. As you can see we request information about 300 cities in the UK with a population of at least 1000. Also, we would like to get this information as .json file.

The only restriction was limit of 25000 requests per day, but by saving downloaded information it can be easily avoided.

Next step is storing information in lists and .json files. Json file is used for storing all information and lists are the representation of our Graphs. For Prim's algorithm we will need NxN matrix with Euclidean distances between different cities stored in each cell. **Example:** suppose i = London and j = Cambridge, then Graph[i][j] - Euclidean distance between these cities.

However, for Kruskal's algorithm we will need Matrix, such that 1st cell would be city - start of the edge, 2nd - end of the edge, 3rd- distance between them. We will need this Matrix be sorted by distance, so we use lambda function.
Key for sorting by specific column would be:
key = lambda x:x[n], where n = number of column.

Prim's algorithm.

It is based on adding new vertices to the tree by taking from already included vertice edge with the minimum weight. It starts from one arbitrary taken vertice and ends up with minimal spanning tree. It also can be compared to Dijkstra algorithm.

```
def searchMin_Prims(Graph, visited):                    # Prims start
    min = max_of_list_in_list(Graph)
    for i in visited:
        graph_temp = enumerate(Graph[i])
        for j, elem in graph_temp:
            if elem<min and j not in visited:
                min = elem
                index_res = j
    return [min, index_res]

def Prims(Graph):
    visitPlan = list()
    for i in range(1, len(Graph)):
        visitPlan.append(i)
    visited, result = [0], [0]
    for i in visitPlan:
        weight, j = searchMin_Prims(Graph, visited)
        result.append(weight)
        visited.append(j)
    return result                                       # Prims End
```

Result:

We get a list of weights of edges, which are included in our spanning tree. To add it to map we just go through the whole primary matrix of distances and look for equalities.

Folium and its use:

Folium is the perfect library for creating and editing maps. It allows use to choose different styles of your map, create Clusters and divide by regions.

In my code it is used 2 times. First is creating the map, editing settings of clusters in it and adding all the cities obtained from geonames.org service.

```
map = folium.Map([53.139499, -1.685177], zoom_start=8, tiles='CartoDBdark_matter')             # doing all the map stuff
marker_cluster = MarkerCluster().add_to(map)
for i in data['geonames']:
    coor_temp = list()
    coor_temp.append(i['lat'])
    coor_temp.append(i['lng'])
    folium.CircleMarker(location=coor_temp, popup=i['toponymName'], radius=7, fill_color='ffa8af', color='white', icon=folium.Icon(colo
```

To add point we need its latitude, length and name. Also, there are different settings of visualisation.
Clusters are things which makes your map be visually attractive from any point of view. If you watch on the whole island of United Kingdom you will see only different points with numbers - unions of a few cities.
Second time we use it is creating lines between points. For it we need list of 2 points with their latitude and length.

```
for i in range(len(Graph_prim)):
    for j in range(len(Graph_prim[i])):
        if Graph_prim[i][j] in result_prim:
            points_to_connect_temp = list(list())
            temp_point_1 = (float(list_of_cities[i][1]), float(list_of_cities[i][2]))
            temp_point_2 = (float(list_of_cities[j][1]), float(list_of_cities[j][2]))
            coords_for_line  = [temp_point_1, temp_point_2]
            folium.PolyLine(locations = coords_for_line).add_to(map)
```

For prim's algorithm we go through our adjacency matrix and look for similarities with obtained result and distances between cities. Then, we group different lat and lng and create line.

One of the main purposes of our internship was learning new stuff about programming languages and also becoming used to work in team.

I enjoyed working on this project, searching for information and finding new ways of solving problems. It was a nice experience for me before my internship in another company. It taught me some new skills in coding, gave me knowledge and diligence.

Links:
GitHub - https://github.com/MaximusTVs/CityConnector
Used API - https://www.geonames.org