МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС
«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»
НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Практична робота №2

з курсу “Штучний інтелект”

Варіант 8

Виконала: студентка 4 курсу

групи КА-41

Лочман Я.В.

Київ – 2017

Завдання: Існує транспортна мережа між містами СНД. Мережа наведена у вигляді таблиці зв’язків між містами. Зв’язки є двосторонніми, тобто передбачають рух у двох напрямах. Потрібно побудувати маршрут проїзду із одного міста в інше. Відома топологія зв’язків між містами. Виконати:

1. пошук у ширину
2. пошук у глибину
3. пошук з обмеженням глибини
4. пошук у глибину з ітераційним збільшенням глибини
5. двонаправлений пошук

Зобразити рух по дереву пошуку на його графі та вказати складність кожного виду пошуку. Зробити висновки.

Номер варіанту	Пункт відправлення	Пункт призначення
8	Мурманськ	Сімферополь

Реалізація:

Була написана програма мовою програмування python з використанням бібліотек networkx, googlemaps, plotly.

Індексація міст для зручності:

{'Брест': 0,
'Волгоград': 1,
'Вороніж': 2,
'Вільнюс': 3,
'Вітебськ': 4,
'Даугавпілс': 5,
'Донецьк': 6,
'Житомир': 7,
'Казань': 8,
'Калінінград': 9,
'Каунас': 10,
'Кишинів': 11,
'Київ': 12,
'Москва': 13,
'Мурманськ': 14,
'Мінськ': 15,
'Нижній Новгород': 16,
'Одеса': 17,
'Орел': 18,
'Рига': 19,
'Самара': 20,
'Санкт-Петербург': 21,
'Сімферополь': 22,
'Таллінн': 23,
'Уфа': 24,
'Харків': 25,
'Ярославль': 26}

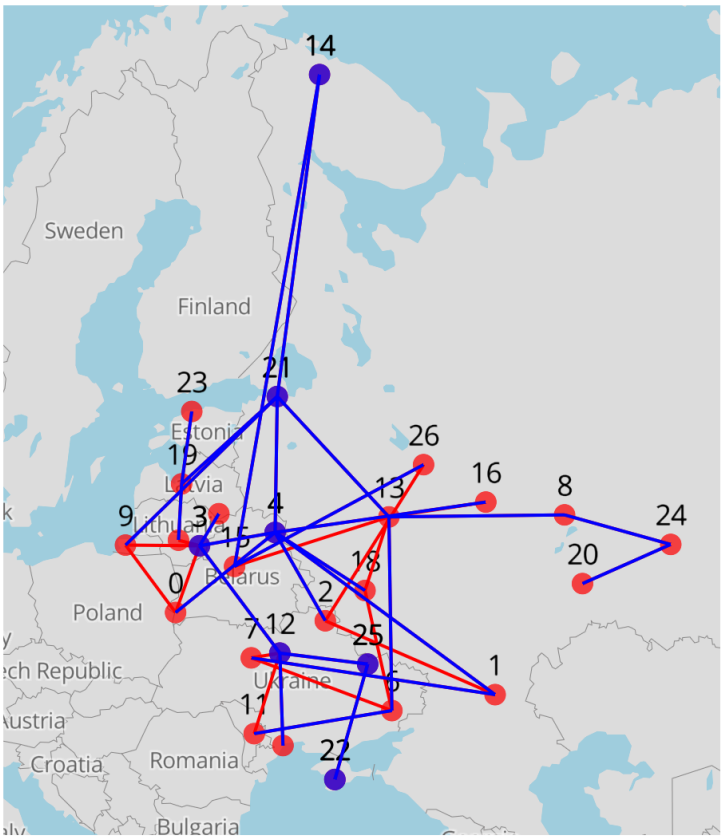
1

Пошук у ширину (Breadth-First Search)

```
def bfs(Graph, start, end):
bfs(Graph, start, end)
```

```
[('Мурманськ', 'Санкт-Петербург'),
 ('Мурманськ', 'Мінськ'),
 ('Санкт-Петербург', 'Вітебськ'),
 ('Санкт-Петербург', 'Калінінград'),
 ('Санкт-Петербург', 'Москва'),
 ('Санкт-Петербург', 'Рига'),
 ('Мінськ', 'Ярославль'),
 ('Вітебськ', 'Брест'),
 ('Вітебськ', 'Волгоград'),
 ('Вітебськ', 'Вороніж'),
 ('Вітебськ', 'Вільнюс'),
 ('Вітебськ', 'Нижній Новгород'),
 ('Вітебськ', 'Орел'),
 ('Москва', 'Донецьк'),
 ('Москва', 'Казань'),
 ('Рига', 'Каунас'),
 ('Рига', 'Таллінн'),
 ('Волгоград', 'Житомир'),
 ('Вільнюс', 'Даугавпілс'),
 ('Вільнюс', 'Київ'),
 ('Донецьк', 'Кишинів'),
 ('Казань', 'Уфа'),
 ('Київ', 'Одеса'),
 ('Київ', 'Харків'),
 ('Уфа', 'Самара'),
 ('Харків', 'Сімферополь')]
```

Шлях = ['Мурманськ', 'Санкт-Петербург', 'Вітебськ', 'Вільнюс', 'Київ', 'Харків', 'Сімферополь']



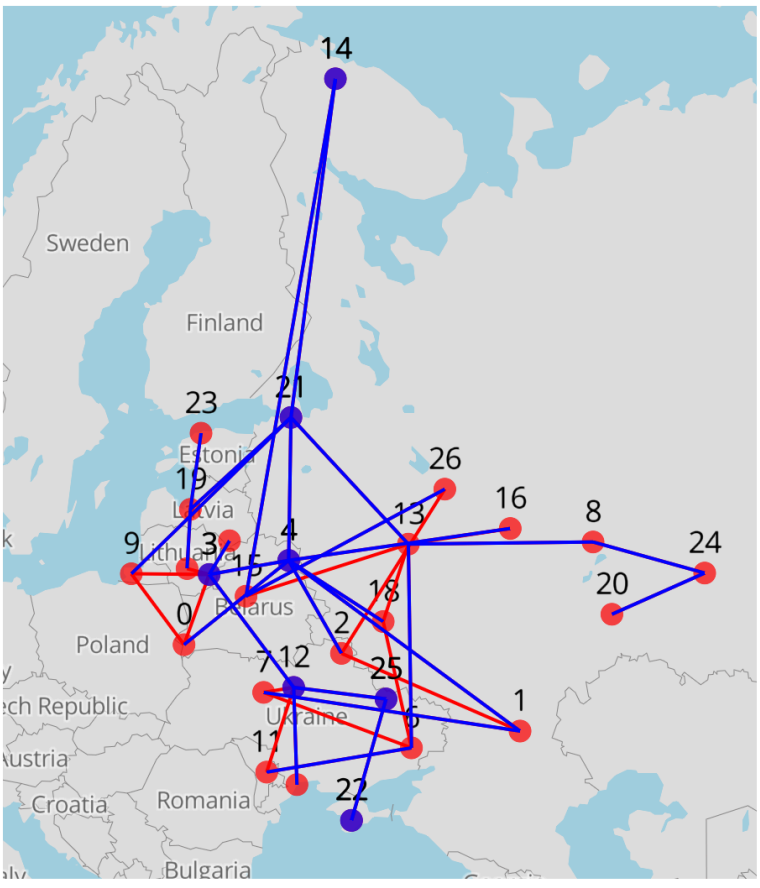
2

Пошук у глибину (Depth-First Search)

```
def dfs(Graph, start, end):
dfs(Graph, start, end)
```

```
[('Мурманськ', 'Санкт-Петербург'),
 ('Санкт-Петербург', 'Вітебськ'),
 ('Вітебськ', 'Брест'),
 ('Брест', 'Вільнюс'),
 ('Вільнюс', 'Даугавпілс'),
 ('Вільнюс', 'Калінінград'),
 ('Вільнюс', 'Каунас'),
 ('Каунас', 'Рига'),
 ('Рига', 'Таллінн'),
 ('Вільнюс', 'Київ'),
 ('Київ', 'Житомир'),
 ('Житомир', 'Волгоград'),
 ('Волгоград', 'Вороніж'),
 ('Вороніж', 'Ярославль'),
 ('Ярославль', 'Мінськ'),
 ('Мінськ', 'Москва'),
 ('Москва', 'Донецьк'),
 ('Донецьк', 'Кишинів'),
 ('Донецьк', 'Орел'),
 ('Москва', 'Казань'),
 ('Казань', 'Уфа'),
 ('Уфа', 'Самара'),
 ('Москва', 'Нижній Новгород'),
 ('Київ', 'Одеса'),
 ('Київ', 'Харків'),
 ('Харків', 'Сімферополь')]
```

Шлях = ['Мурманськ', 'Санкт-Петербург', 'Вітебськ', 'Брест', 'Вільнюс', 'Київ', 'Харків', 'Сімферополь']



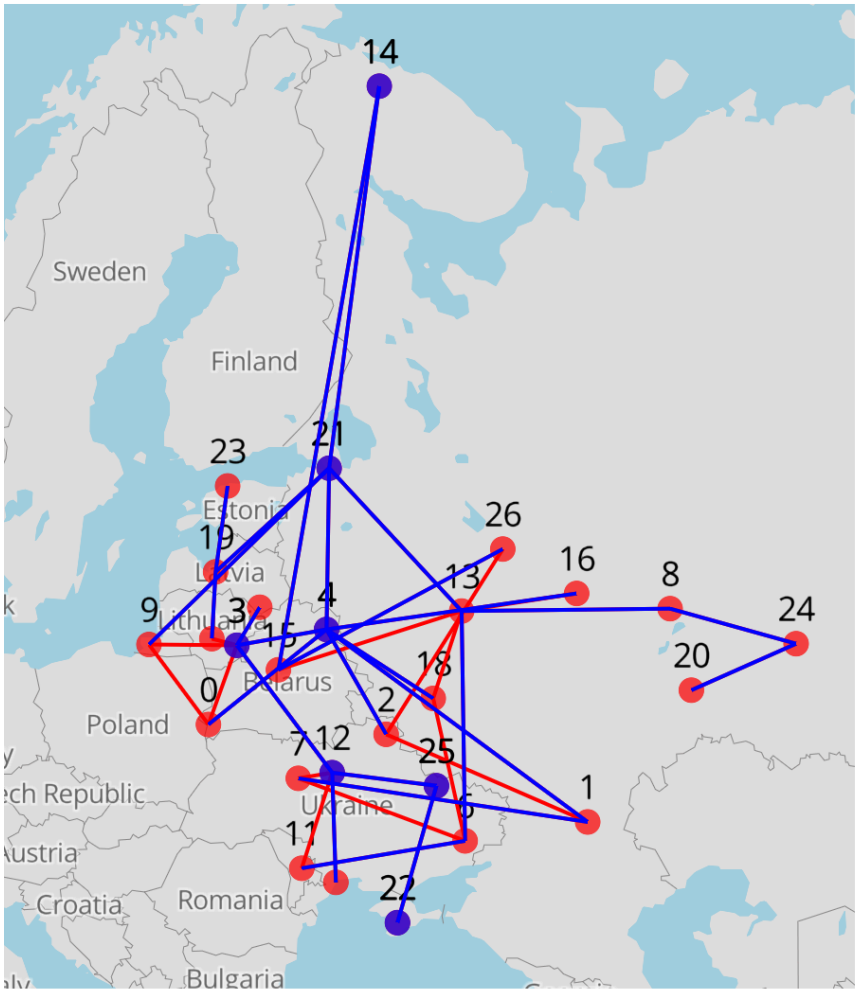
3

Пошук з обмеженням глибини (Depth-Limited Search)

```
def dls(Graph, start, end, limit):↔
dls(Graph, start, end, 8)
```

```
[('Мурманськ', 'Санкт-Петербург'),
 ('Санкт-Петербург', 'Вітебськ'),
 ('Вітебськ', 'Брест'),
 ('Брест', 'Вільнюс'),
 ('Вільнюс', 'Даугавпілс'),
 ('Вільнюс', 'Калінінград'),
 ('Вільнюс', 'Каунас'),
 ('Каунас', 'Рига'),
 ('Рига', 'Таллінн'),
 ('Вільнюс', 'Київ'),
 ('Київ', 'Житомир'),
 ('Житомир', 'Волгоград'),
 ('Волгоград', 'Вороніж'),
 ('Житомир', 'Донецьк'),
 ('Донецьк', 'Кишинів'),
 ('Донецьк', 'Москва'),
 ('Донецьк', 'Орел'),
 ('Київ', 'Одеса'),
 ('Київ', 'Харків'),
 ('Харків', 'Сімферополь')]
```

Шлях = [' Мурманськ', 'Санкт-Петербург', 'Вітебськ', 'Брест', 'Вільнюс', 'Київ', 'Харків', 'Сімферополь']



4

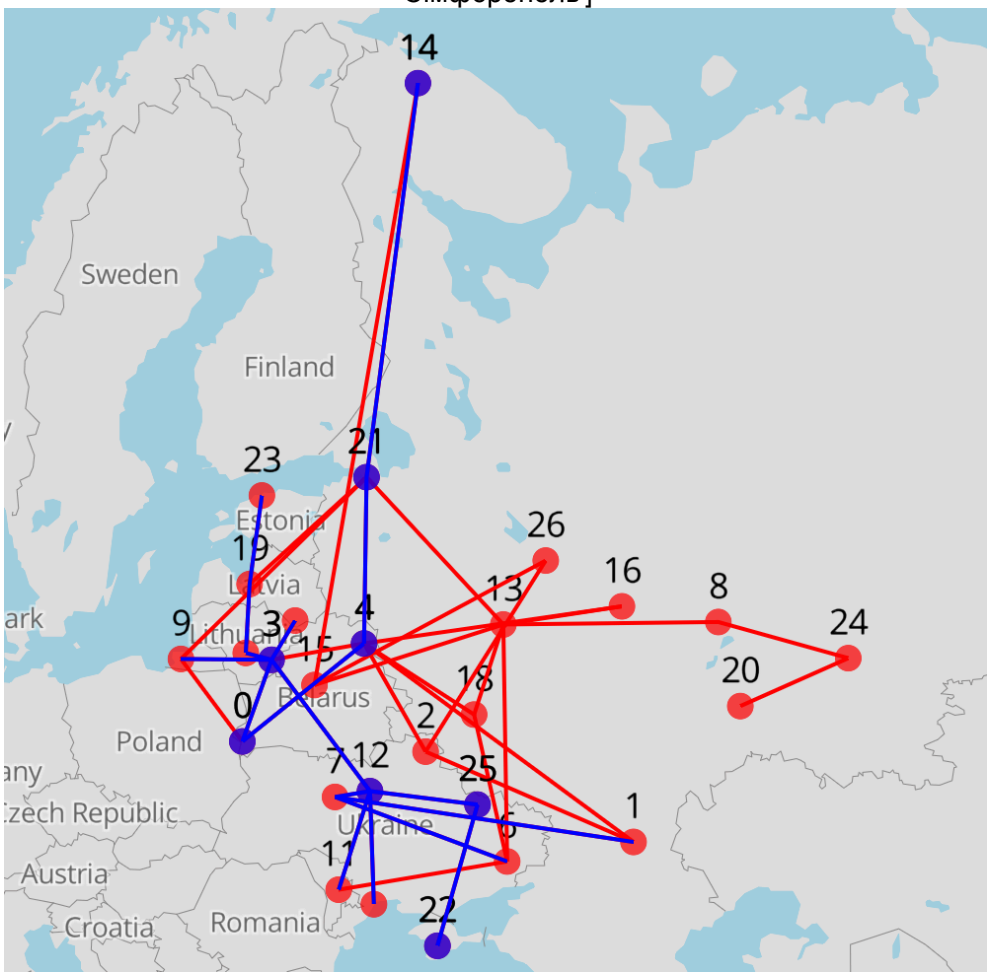
Пошук з ітраційним збільшенням глибини

(Iterative deepening depth-first search)

```
def iddfs(Graph, start, end):↔
iddfs(Graph, start, end)
```

```
[('Мурманськ', 'Санкт-Петербург'),
 ('Санкт-Петербург', 'Вітебськ'),
 ('Вітебськ', 'Брест'),
 ('Брест', 'Вільнюс'),
 ('Вільнюс', 'Даугавпілс'),
 ('Вільнюс', 'Калінінград'),
 ('Вільнюс', 'Каунас'),
 ('Каунас', 'Рига'),
 ('Рига', 'Таллінн'),
 ('Вільнюс', 'Київ'),
 ('Київ', 'Житомир'),
 ('Житомир', 'Волгоград'),
 ('Житомир', 'Донецьк'),
 ('Київ', 'Кишинів'),
 ('Київ', 'Одеса'),
 ('Київ', 'Харків'),
 ('Харків', 'Сімферополь')]
```

Шлях = ['Мурманськ', 'Санкт-Петербург', 'Вітебськ', 'Брест', 'Вільнюс', 'Київ', 'Харків', 'Сімферополь']



5

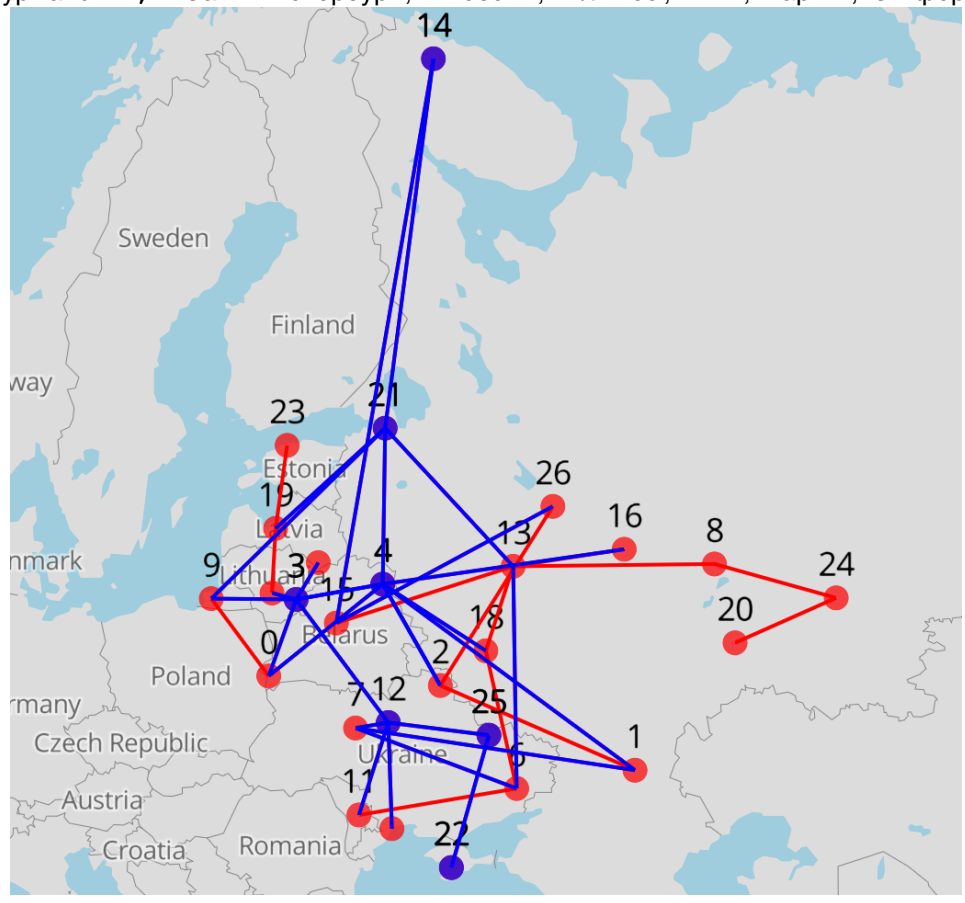
Двонапрямлений пошук (Bidirectional search)

```
]: def bds(Graph, start, end):↔

   bds(Graph, start, end)
```

```
]: [('Мурманськ', 'Санкт-Петербург'),
    ('Мурманськ', 'Мінськ'),
    ('Санкт-Петербург', 'Вітебськ'),
    ('Санкт-Петербург', 'Калінінград'),
    ('Санкт-Петербург', 'Москва'),
    ('Санкт-Петербург', 'Рига'),
    ('Мінськ', 'Ярославль'),
    ('Вітебськ', 'Брест'),
    ('Вітебськ', 'Волгоград'),
    ('Вітебськ', 'Вороніж'),
    ('Вітебськ', 'Вільнюс'),
    ('Вітебськ', 'Нижній Новгород'),
    ('Вітебськ', 'Орел'),
    ('Москва', 'Донецьк'),
    ('Вороніж', 'Вітебськ'),
    ('Донецьк', 'Житомир'),
    ('Волгоград', 'Житомир'),
    ('Каунас', 'Вільнюс'),
    ('Калінінград', 'Вільнюс'),
    ('Даугавпілс', 'Вільнюс'),
    ('Вітебськ', 'Вільнюс'),
    ('Брест', 'Вільнюс'),
    ('Одеса', 'Київ'),
    ('Кишинів', 'Київ'),
    ('Житомир', 'Київ'),
    ('Вільнюс', 'Київ'),
    ('Київ', 'Харків'),
    ('Харків', 'Сімферополь')]
```

Шлях = ['Мурманськ' , 'Санкт-Петербург', 'Вітебськ', 'Вільнюс', 'Київ', 'Харків', 'Сімферополь']



Код програми

```
# coding: utf-8

# In[521]:

import networkx as nx

import googlemaps
gmaps = googlemaps.Client(key='AIzaSyA0mht5h9QKEY3HrW0OQn9r2SYDJ1eOb58')

import plotly
import plotly.plotly as py
from plotly.offline import download_plotlyjs, init_notebook_mode
init_notebook_mode(connected=True)
from plotly.graph_objs import *


# In[522]:

#cities = open('Cities.txt').read().split('\n')[1:]
distances = pd.read_csv('Distances.txt', sep=',')
cities = set(distances['Місто1']).union(set(distances['Місто2']))
cities = list(cities)
cities.sort()
cities_dict = {cities[i]:i for i in range(len(cities))}

cities_locs = [gmaps.geocode(city)[0]['geometry']['location'] for city in cities]


# In[523]:

graph = [dict() for i in range(len(cities))]
for row in distances.values:
    graph[cities_dict[row[0]]].update({cities_dict[row[1]]:{'weight':row[2]}})
    graph[cities_dict[row[1]]].update({cities_dict[row[0]]:{'weight':row[2]}})
    #graph[cities_dict[row[0]]].update({row[1]:row[2]})
    #graph[cities_dict[row[1]]].update({row[0]:row[2]})

graph_encoded = {cities[j]:{cities[i] : graph[j][i] for i in graph[j].keys()} for
j in range(len(graph))}
graph_dict = {j:{i : graph[j][i] for i in graph[j].keys()} for j in
range(len(graph))}


# # Build graph using NetworkX

# In[524]:

def graph_info(G):
    print('graph: ',G.adj)
    print(G.number_of_nodes(), ' nodes: ',G.nodes())
    print(G.number_of_edges(), ' edges: ',G.edges())
    print('degrees of nodes: ',G.degree())

Graph = nx.Graph(graph_dict)
#Graph = nx.Graph(graph_encoded)

#graph_info(Graph)
```

```
# In[525]:

start_city, end_city = 'Мурманськ', 'Сімферополь'
start, end = cities_dict[start_city], cities_dict[end_city]
#start, end = start_city, end_city
print('start: ',start)
print('  end: ',end)


# # Visualization on map

# In[526]:

def plot_map(cities_locs, path, coordinates, col):
    mapbox_access_token =
"pk.eyJ1IjoidmFsZW50eW4xOTk3IiwiYSI6ImNqNHlubm03cjFpc3EzM21nbW1rdGhvNmwifQ.xuQ0fbW
NXjOi3y03MYbqAQ"
    path_locs = [cities_locs[p] for p in
path]#[gmaps.geocode(cities[p])[0]['geometry']['location'] for p in path]
    verticies = Scattermapbox(
            lat = [city['lat'] for city in path_locs],
            lon = [city['lng'] for city in path_locs],
            mode = 'markers+text',
            marker = Marker(
                size = 14,
                color = col,
                opacity = 0.7
            ),
            text = path,
            textposition = 'top',
            textfont = dict(
                family = 'sans serif bold',
                size = 18,
                color = 'black'
            )
        )

    edges = [Scattermapbox(
            lat = [cities_locs[x[0]]['lat'], cities_locs[x[1]]['lat']] ,
            lon = [cities_locs[x[0]]['lng'], cities_locs[x[1]]['lng']],
            mode = 'lines',
            marker = Marker(
                    size = 14,
                    color = col,
                    opacity = 0.7
                ),
        ) for x in coordinates]

    layout = Layout(
        title = 'Map',
        autosize = True,
        hovermode = 'closest',
        showlegend = False,
        height = 700,
        mapbox = dict(
            accesstoken = mapbox_access_token,
            bearing = 0,
            center = dict(
                lat = [city['lat'] for city in path_locs][0],
                lon = [city['lng'] for city in path_locs][0]
            ),
```

```
            pitch = 1,
            zoom = 3
        ),
    )
    return verticies, edges, layout


# In[527]:

v1, e1, l1 = plot_map(cities_locs, list(range(len(cities))),
list(Graph_ind.edges), 'rgb(255, 0, 0)')
fig1 = dict(data=Data([v1] + e1), layout=l1)


# # Implement algorithms

# ## Пошук у ширину (Breadth-First Search)

# In[528]:

#%timeit pass
def BFS(Graph, start, end):
    steps = list(nx.bfs_edges(Graph, start))
    index_end = [pair[1] for pair in steps].index(end)
    steps = steps[:index_end + 1]
    return steps, nx.shortest_path(nx.Graph(steps), start, end)

steps, path = BFS(Graph, start, end)
print(path)

v2, e2, _ = plot_map(cities_locs, path, steps, 'rgb(0, 0, 255)')
plotly.offline.iplot(dict(data=fig1['data'] + Data([v2] + e2), layout=l1),
                     filename='Map')


# ## Пошук у глибину (Depth-First Search)

# In[529]:

#%timeit pass
def DFS(Graph, start, end):
    steps = list(nx.dfs_edges(Graph, start))
    index_end = [pair[1] for pair in steps].index(end)
    steps = steps[:index_end + 1]
    return steps, nx.shortest_path(nx.Graph(steps), start, end)

steps, path = DFS(Graph, start, end)
print(path)

v2, e2, _ = plot_map(cities_locs, path, steps, 'rgb(0, 0, 255)')
plotly.offline.iplot(dict(data=fig1['data'] + Data([v2] + e2), layout=l1),
filename='Map')


# ## Пошук з обмеженням глибини (Depth-Limited Search)

# In[530]:

#%timeit pass
def DLS(Graph, start, end, limit):
    steps = list(nx.dfs_edges(Graph, start, limit))
```

```
    index_end = [pair[1] for pair in steps].index(end)
    steps = steps[:index_end + 1]
    return steps, nx.shortest_path(nx.Graph(steps), start, end)

steps, path = DLS(Graph, start, end, 7)
print(path)

v2, e2, _ = plot_map(cities_locs, path, steps, 'rgb(0, 0, 255)')
plotly.offline.iplot(dict(data=fig1['data'] + Data([v2] + e2), layout=l1),
filename='Map')


# ##  Пошук з ітераційним збільшенням глибини (Iterative deepening depth-first
search)

# In[531]:

#%timeit pass
def IDDFS(Graph, start, end):
    error = True
    limit = 0
    while error:
        limit += 1
        steps = list(nx.dfs_edges(Graph, start, limit))
        try:
            index_end = [pair[1] for pair in steps].index(end)
            error = False
        except:
            pass
    steps = steps[:index_end + 1]
    return steps, nx.shortest_path(nx.Graph(steps), start, end)

steps, path = IDDFS(Graph, start, end)
print(path)

v2, e2, _ = plot_map(cities_locs, path, steps, 'rgb(0, 0, 255)')
plotly.offline.iplot(dict(data=fig1['data'] + Data([v2] + e2), layout=l1),
filename='Map')


# ## Двонапрямлений пошук (Bidirectional search)

# In[532]:

#%timeit pass
def BDS(Graph, start, end):
    start_steps = list(nx.bfs_edges(Graph, start))
    index_end = [pair[1] for pair in start_steps].index(end)
    start_steps = start_steps[:index_end + 1]

    end_steps = list(nx.bfs_edges(Graph, end))
    index_start = [pair[1] for pair in end_steps].index(start)
    end_steps = end_steps[:index_start + 1]

    #end_steps = end_steps[::-1]
    i = 0
    while set.intersection(set(start_steps[:i]),(set(end_steps[:i]))) == set():
        i += 1
    steps = start_steps[:i]
    steps.extend([item[::-1] for item in end_steps[:i][::-1]])
    return steps, nx.shortest_path(nx.Graph(steps), start, end)
```

```
steps, path = BDS(Graph, start, end)
print(path)

v2, e2, _ = plot_map(cities_locs, path, steps, 'rgb(0, 0, 255)')
plotly.offline.iplot(dict(data=fig1['data'] + Data([v2] + e2), layout=l1),
filename='Map')
```

Оцінка складності стратегій неінформативного пошуку

Стратегія	Повнота	Часова складність (експериментально)	Часова складність (теоретично)	Оптимальність
Пошук у ширину	Так	10.8 ns	O(|v| + |e|)	Так
Пошук у глибину	Ні	12.2 ns	O(|v| + |e|)	Ні
Пошук з обмеженням глибини	Ні	10.9 ns	$O(b^e)$	Ні
Пошук з ітераційним збільшенням глибини	Ні	10.1 ns	$O(b^d)$	Так
Двонаправлений пошук	Ні	11.1 ns	$O(b^{d/2})$	Так

, де: |v| = 27 – кількість вузлів, |e| = 39 – кількість зв’язків, b = 3 – коефіцієнт розгалуження, e = 8 – границя глибини, d = 7 – глибина найбільш поверхового розв’язку.

Висновок

В ході виконання роботи я навчилася будувати різні моделі пошуку на графі. За експериментальними оцінками складності найгіршим, що не дивно, виявився пошук у глибину. Найкращим – пошук з ітераційним збільшенням глибини. Пошук у ширину і двонаправлений пошук дали також один з кращих результатів, враховуючи ще й довжину шляху.