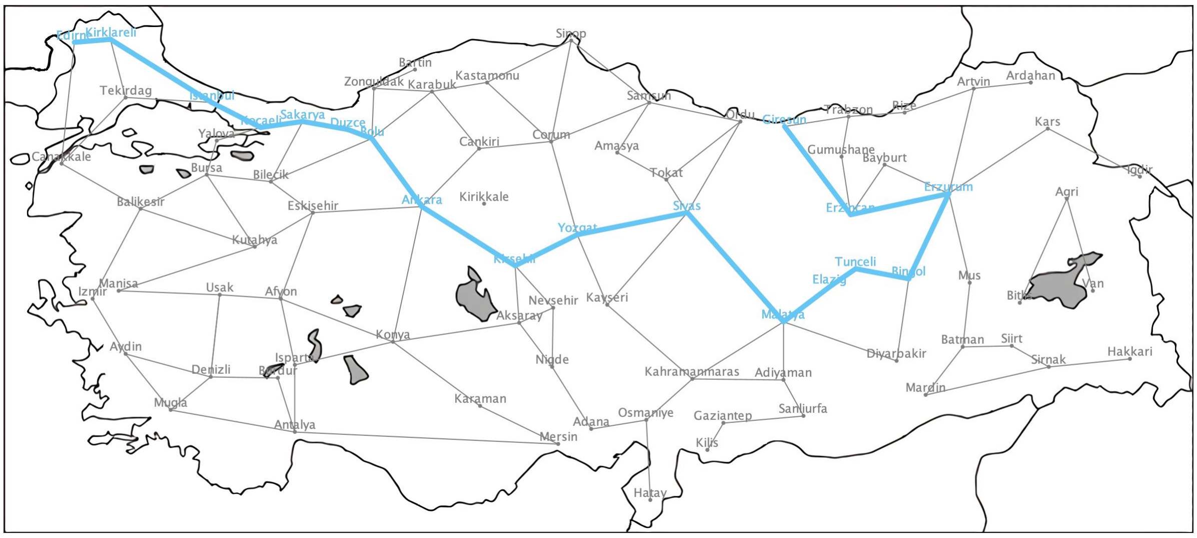
Assignment#2: Turkey Navigation

Name & Surname: Berk Sel
Student ID: 2022400147

Case 1: Edirne to Giresun
<u>**Console output:**</u>

```
Enter starting city: Edirne
Enter destination city: Giresun
Total distance: 2585.49. Path: Edirne -> Kirklareli -> Istanbul -> Kocaeli -> Sakarya -> Duzce -> Bolu -> Ankara -> Kirsehir -> Yozgat -> Sivas -> Malatya
-> Elazig -> Tunceli -> Bingol -> Erzurum -> Erzincan -> Giresun
```

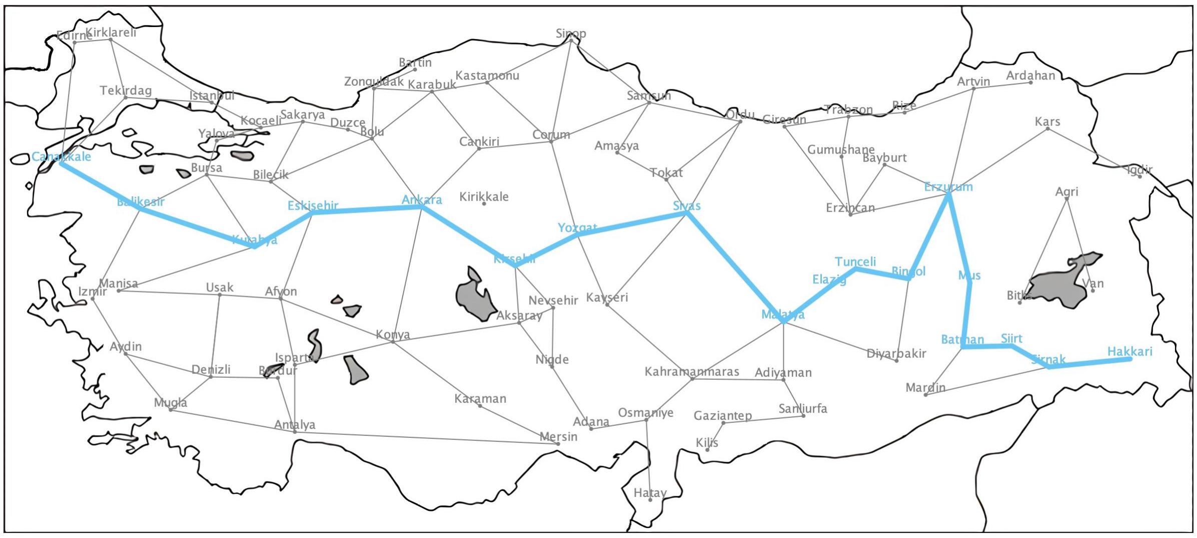
<u>**Graphical output:**</u>



Case 2: Canakkale to Hakkari
<u>**Console output:**</u>

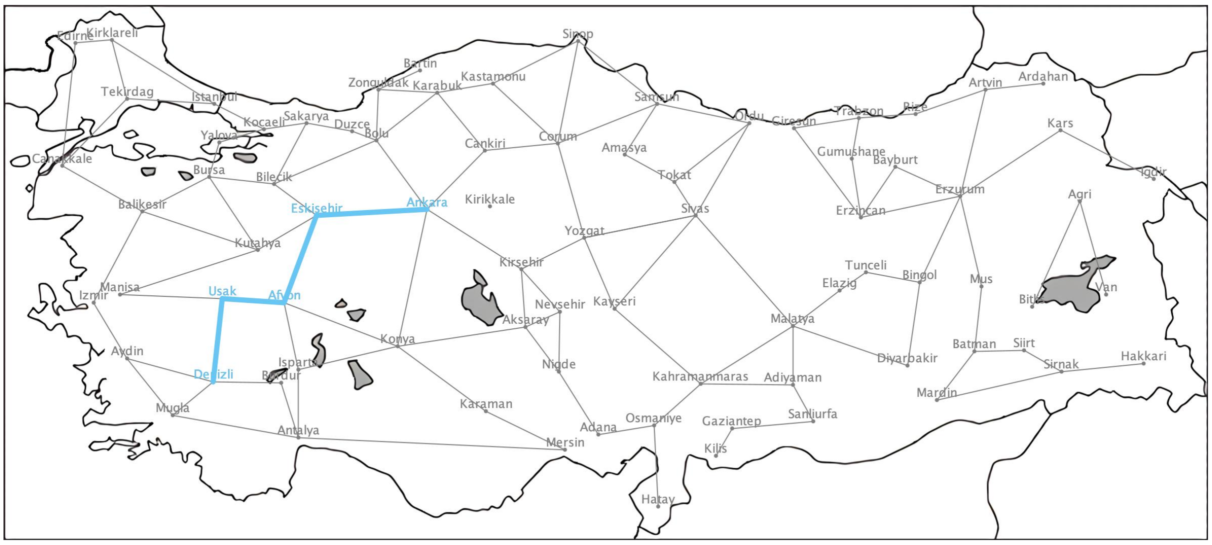
```
Enter starting city: Canakkale
Enter destination city: Hakkari
Total distance: 2780.87. Path: Canakkale -> Balikesir -> Kutahya -> Eskisehir -> Ankara -> Kirsehir -> Yozgat -> Sivas -> Malatya -> Elazig -> Tunceli ->
Bingol -> Erzurum -> Mus -> Batman -> Siirt -> Sirnak -> Hakkari
```

<u>**Graphical output:**</u>



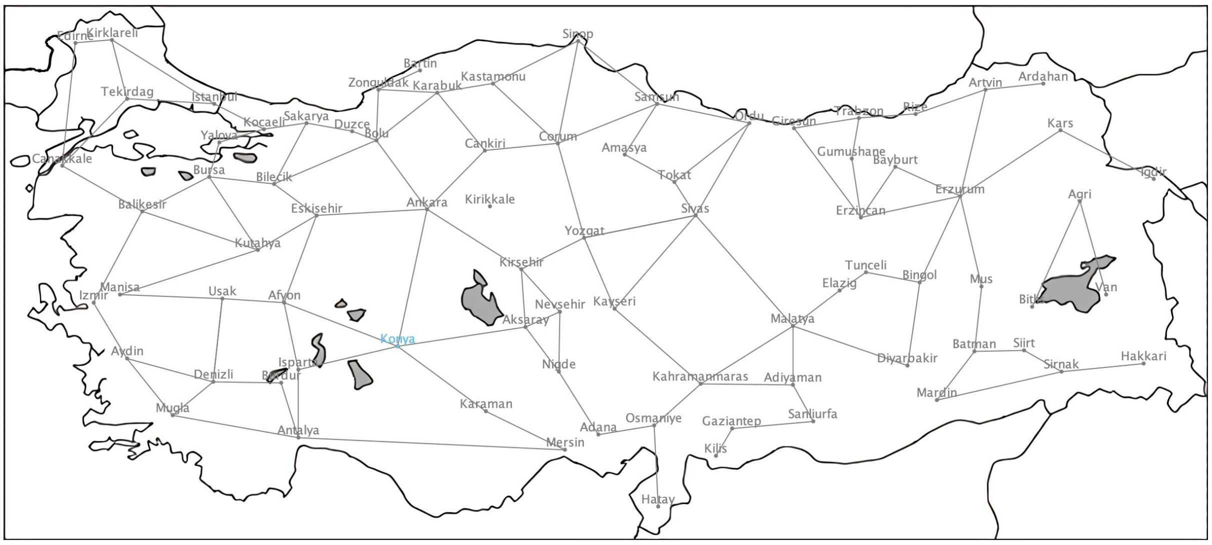
Case 3: Input validation
Console output:
```
Enter starting city: Anka
City named 'Anka' not found. Please enter a valid city name.
Enter starting city: Ankara
Enter destination city: Deni
City named 'Deni' not found. Please enter a valid city name.
Enter destination city: Denizli
Total distance: 689.10. Path: Ankara -> Eskisehir -> Afyon -> Usak -> Denizli
```

Graphical output:



Case 4: Path to same city
Console output:
```
Enter starting city: Konya
Enter destination city: Konya
Total distance: 0.00. Path: Konya
```

Graphical output:



Case 5: Unreachable city pairs
Console output:
```
Enter starting city: Bilecik
Enter destination city: Agri
No path could be found.
```

Graphical output:
No output is produced.

The Algorithm

The key idea behind the algorithm derives from the following questions: "Can a city be used to connect two cities, which are not already connected before?" or "Can a city be used to connect cities so that with the use of this city along the path, the distance becomes smaller?."

Let's call these type of cities “connector cities”. Our distance array initially shows only the direct connections' length, if they exist, and a very big number to indicate that two cities are not connected.

By using cities as connectors, we will establish new connections or update the existing ones.
For every possible connector city, we will look at all city pairs if they can be connected by our connector.

When can we use a connector city to form a connection between cities a and b?

First of all, a current connection should exist between the connector city with both cities a, b (not necessarily a direct connection). Also, connector city must reduce the current distance between a and b. Notice that, we do not need to check whether the cities a and b are already connected or not since the distance between a and b is very large initially, any distance will be less than the current distance and will be updated.

If these two conditions are satisfied, our new path will include the connector city.
Then, we can calculate the new distance by simply adding the distance between the start and connector and the distance between the connector and destination, which we know they exist by conditions.

Also, if the connector is in the path from start and destination, we can update our path.
secondCityIndex[start][destination] gives us the index of the city after start on the shortest path from start to destination. If our current shortest path includes connector, we can update the next value to secondCityIndex[start][connector] as our current path suggests going to connector first.

Finally, after examining all cities as connectors, we are done with updating.
Then, we find the indexes of the two cities and directly access the distance from minimum distance array. If the distance turns out to be the large value we assigned in the beginning, that means no path exists.

If the path exists, to find the path, we follow the second city of the current path and then update the path’s starting city to the second city until we arrive to the destination city.

Pseudocode of the algorithm

```
// initialize the minDistance array
for each i from 0 to numberOfCities - 1 do
    for each j from 0 to numberOfCities - 1 do
        if i == j then
            minDistance[i][j] = 0   // if the cities are the same, distance is zero.
        else
            minDistance[i][j] = INF // the impossible case.
        end if
    end for
end for


for each connectorCity from 0 to numberOfCities - 1 do
    for each fromCity from 0 to numberOfCities - 1 do
        for each toCity from 0 to numberOfCities - 1 do
            // Check if going through the current connectorCity shortens the distance from fromCity to toCity
            if minDistance[fromCity][toCity] > minDistance[fromCity][connectorCity] + minDistance[connectorCity][toCity] then
                // Update minDistance and secondCityIndex accordingly
                minDistance[fromCity][toCity] = minDistance[fromCity][connectorCity] + minDistance[connectorCity][toCity]
                secondCityIndex[fromCity][toCity] = secondCityIndex[fromCity][connectorCity]
            end if
        end for
    end for
end for
```

References:

https://en.wikipedia.org/wiki/Floyd–Warshall_algorithm