ALI AYHAN GUNDER 2021400219

CMPE 160 HOMEWORK 2

Turkey navigation,

Brief information about project:

This project implements a navigational system for cities in Turkey, with provided input files includes city names and coordinates, and connections of neighbor cities. Program first reads data, parses information about cities and connections from input files.

Calculates shortest path by using Dijkstra's shortest path algorithm to efficiently determine the shortest path between start a destination city.

Finally, prints total distance of the shortest path and lists the cities along the path and visualizes Turkey map by StdDraw library and shows the blue path between desired cities.

Path Finding Algorithm Explanation:

In shortest Pathfinder method in my main class, I used Dijkstra's algorithm to find shortest way from start city to destination city. Dijkstra's algorithm is well known algorithm for finding the shortest path between two points in a graph.

Initially, the function starts by finding the start and destination city objects based on their names. If start and destination cities are the same, it returns path with zero distance, without redundant calculation. Otherwise, it initializes three data structures: distances (array list) that stores tentative distances from start city to other city; previous Cities (array list): that keeps track of the previous city in the shortest path found so far, each city; visited (array list): that mark which cities have already visited. Initially all cities marked as unvisited.

In main loop, algorithm iterates through all the cities (represented by their indexes) at most once. In each iteration, it finds the unvisited city with the shortest tentative distance from the start city by using a loop to compare distances.

If no such city exists, it means there are no reachable cities from the start city, and the loop breaks. Otherwise, the city is marked as visited and algorithm then relaxes the connections from the current city, it loops through all the connections of the current city.

For each connected city, it calculates the tentative distance if we travel from the start city through the current city and then to the connected city.

If this tentative distance is shorter than the currently stored distance for the connected city, the algorithm updates the distance and sets the previous city in the path to the current city.

After the main loop, the algorithm checks if a path exists by looking at the previous Cities list for the end city. If it's null, it means there's no path. Otherwise, it starts from the end city and iterates backward using the previous Cities list to reconstruct the actual path taken. It builds a new list (path) by adding on the names of the cities visited in the shortest path order.

Finally, the code prints the total distance and the reconstructed path from the start city to the end city.

Pseudo Code of my shortest Pathfinder method:

```
Function findShortestPath(cities, startCityName, endCityName)
    startCity = findCityByName(cities, startCityName)
    endCity = findCityByName(cities, endCityName)
    If startCity is null or endCity is null Then
        Return null
    End If
    If startCity is endCity Then
        path = new list
        Add startCityName to path
        Print "Total Distance: 0.00. Path: " + startCityName
        Return path
    Else
        distances = new list of size equal to cities, initialize all elements
to infinity
        previousCities = new list of size equal to cities, initialize all
elements to null
        visited = new list of size equal to cities, initialize all elements
to false
        Set distance of startCity to 0

        For i from 0 to size of cities Do
            currentCityIndex = -1
```

```
            For j from 0 to size of cities Do
                If city j is not visited and (currentCityIndex is -1 or
distance of city j is less than distance of currentCity) Then
                    currentCityIndex = j
                End If
            End For
            If distance of currentCity is infinity Then
                Break
            End If
            Mark currentCity as visited
            For each connection in connections of currentCity Do
                connectedCity = findCityByName(cities, connection)
                If connectedCity is not null Then
                    distance = distance of currentCity +
calculateDistance(currentCity, connectedCity)
                    If distance is less than distance of connectedCity Then
                        Set distance of connectedCity to distance
                        Set previousCity of connectedCity to currentCity
                    End If
                End If
            End For
        End For


        If previousCity of endCity is null Then
            Print "There is no path from " + startCityName + " to " +
endCityName
            Return null
        End If
        path = new list
        For city from endCity to null following previousCities Do
            Add city to path
        End For
        Reverse path
```

```
        Print "Total distance: " + distance of endCity + ". Path: " + path
        Return path
    End If
End Function
```

References:

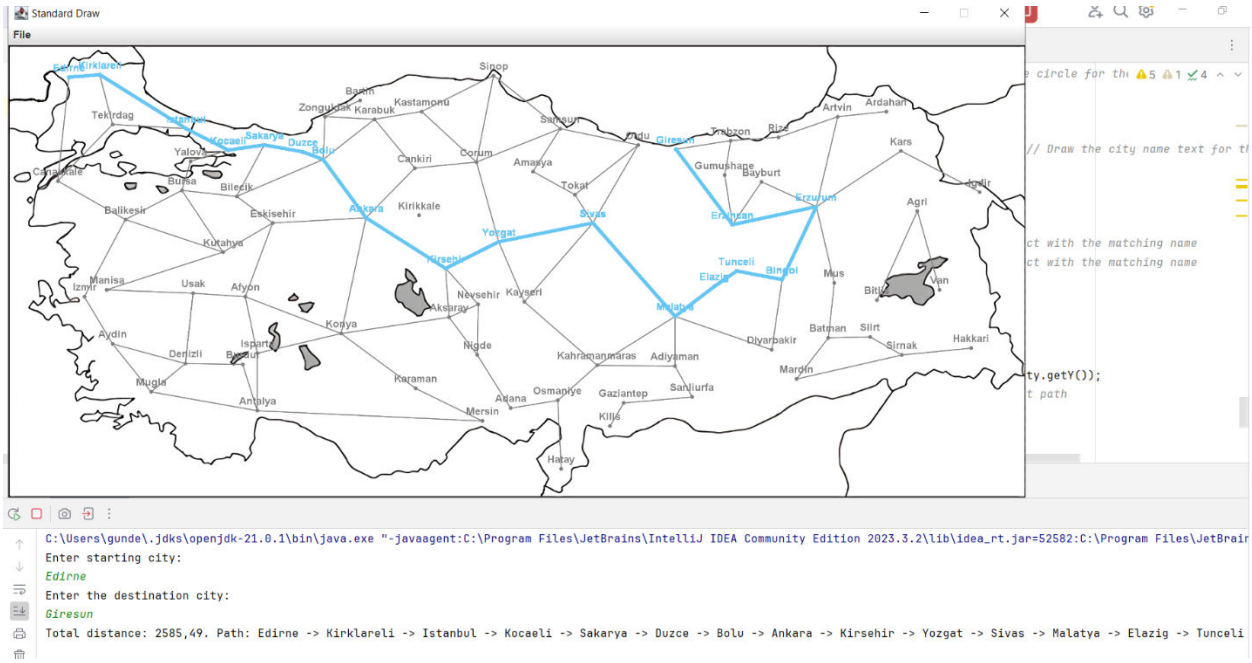
https://www.geeksforgeeks.org/

https://brilliant.org/

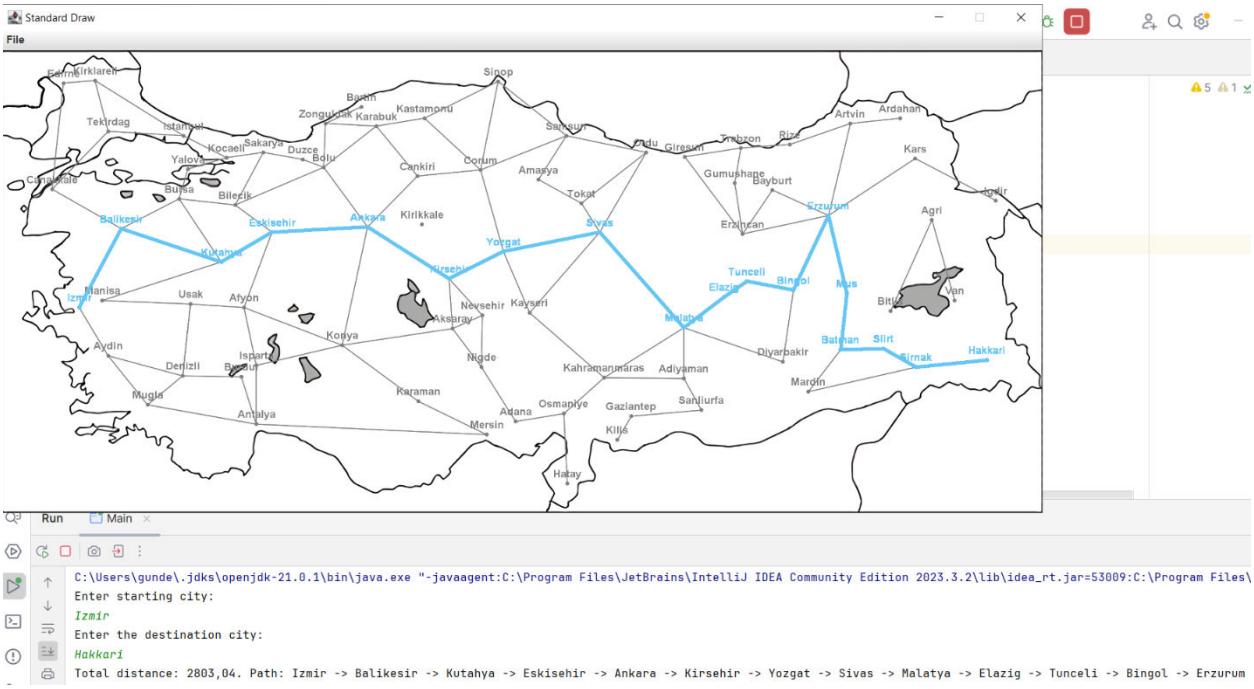
https://stackoverflow.com/

Screenshots of different cases:

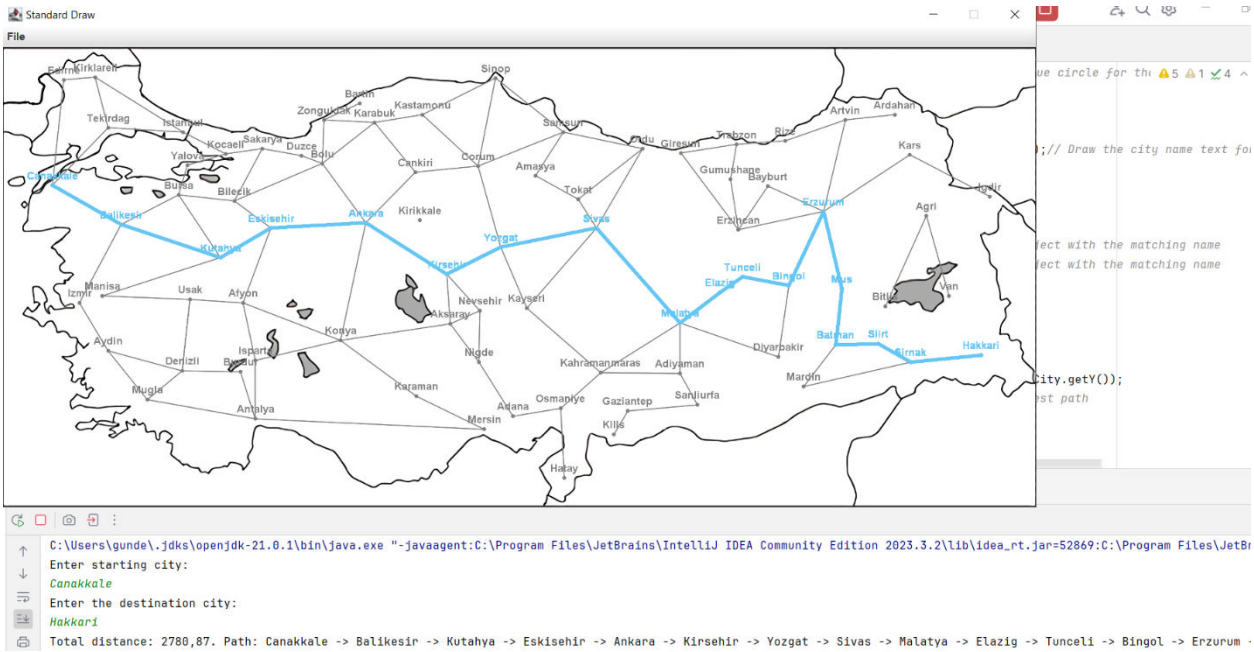
Case1 ex1:



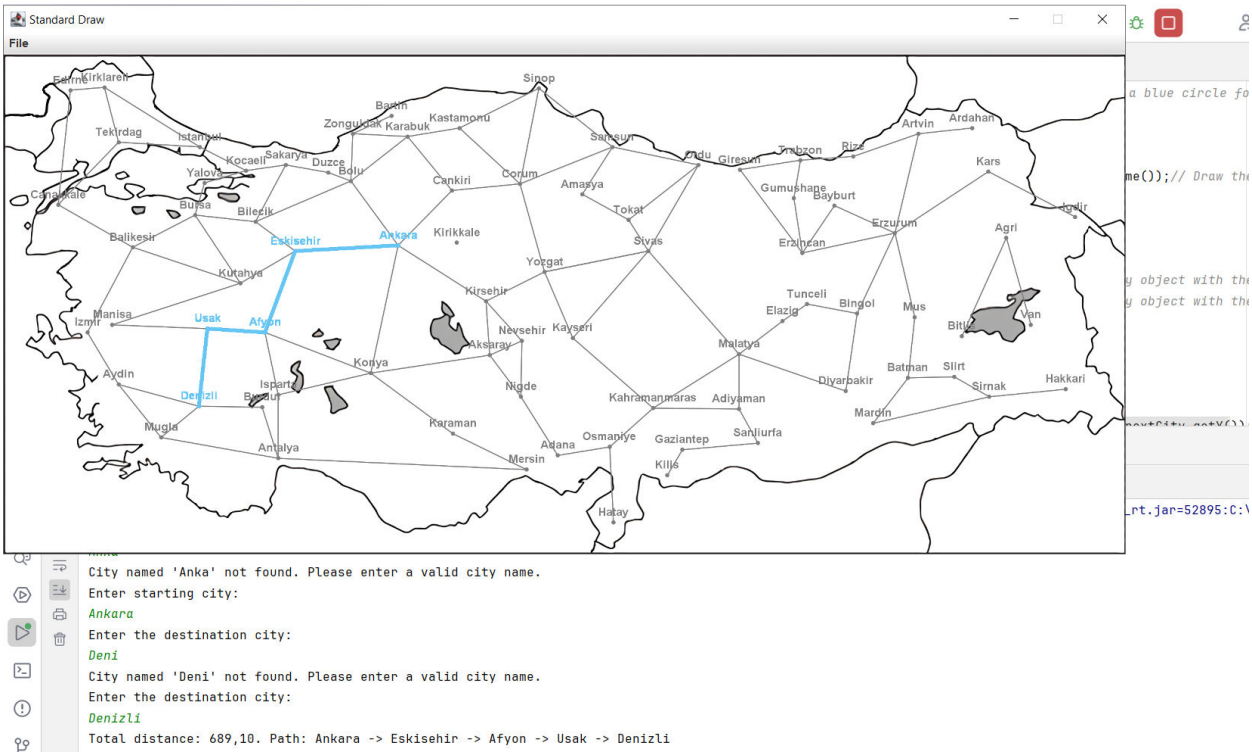
Case1 ex2:



Case2:

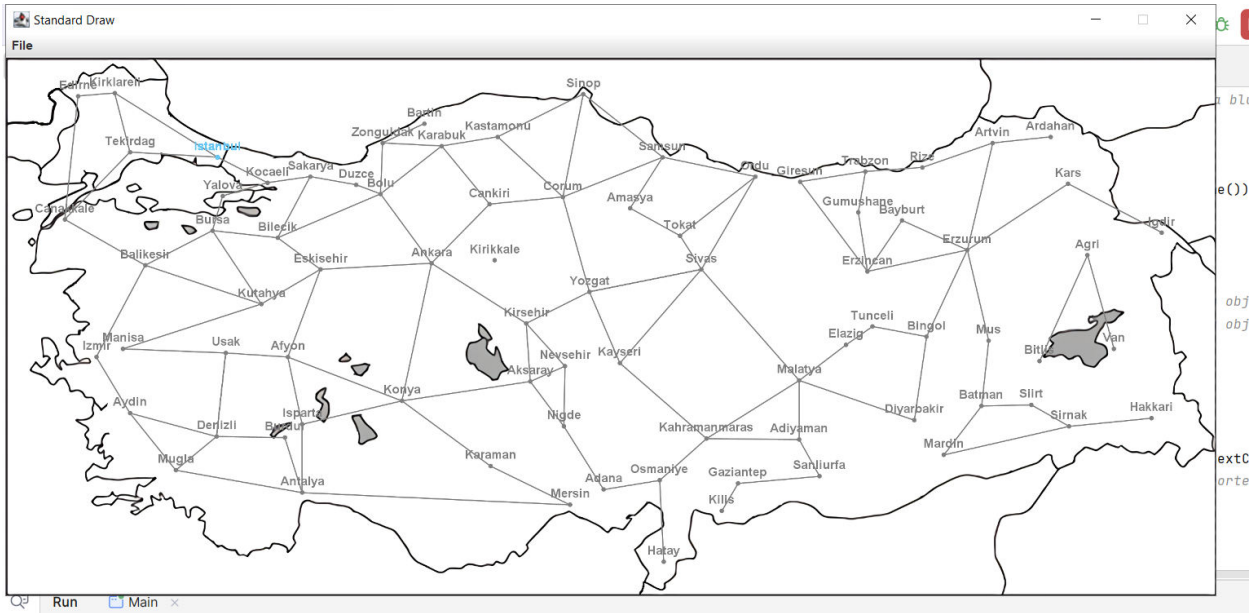


Case3:




```
City named 'Anka' not found. Please enter a valid city name.
Enter starting city:
Ankara
Enter the destination city:
Deni
City named 'Deni' not found. Please enter a valid city name.
Enter the destination city:
Denizli
Total distance: 689,10. Path: Ankara -> Eskisehir -> Afyon -> Usak -> Denizli
```

Case4:




```
C:\Users\gunde\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.3.2\lib\idea_rt.j
Enter starting city:
Istanbul
Enter the destination city:
Istanbul
Total Distance: 0.00. Path: Istanbul
```

Case5:

```
Enter starting city:
Izmir
Enter the destination city:
Van
No path could be found.
```

No visual output.