Detailed Explanation of the Algorithm

Dijkstra's algorithm which is commonly used for finding the shortest path from a vertex to another is used for this project. Algorithm starts by marking all the cities as unvisited since they have not been checked by the algorithm yet. We need to store all the current distances of cities from the source in a distance array. We set distance of the source to zero and we initialize the distance of every other city to infinity. We start our search by looking at the neighbors of the source city. We find the shortest distances to these cities from the source and change their distances to these numbers from the infinity. Since we checked the neighbors of the source, we mark it as visited. Then we look at other cities which are still unvisited and find the one with current minimum distance from the source. That city is going to be our new current city. We again check the unvisited neighbors of the city and find the distances of these cities from our current city. We calculate the sum of the distance of the neighbor from the current city and distance of the current city from the source. We set the distance of that neighbor to the distance we calculated. This distance is not permanent distance of that city from the source but rather a temporary distance. We mark our current city that we have just checked the neighbors of as visited. We find the next minimum distant city and we check the neighbors of it as well. If the new distance we find for a city is shorter than before, we update the distance of that city. Algorithm goes in this order until all the cities are visited.

Pseudocode of the Path-finding Algorithm

This script is going to find the shortest distance and the shortest path between any given cities. It consists of two methods.

METHOD findShortestPath:
Method takes parameters: startingCity(integer), destinationCity(integer), cities(City array), cityConnections(two dimensional City array)

Declare list of City list variable: path

FOR 0 to (length of cities array)-1
    ADD path null

Declare City array variable: arrayForStarting

ADD starting City to arrayForStarting

SET path for starting city as arrayForStarting

Declare boolean array: visited

Declare double array: distance

SET distance of starting City to 0

FOR i = 0 to i = (length of cities array)-1

IF i equals to startingCity

continue

ELSE

SET distance of i to infinity

FOR i = 0 to i = (length of cities array)-1

Declare integer variable: minDistCity = CALL findMinDistCity

SET visited of minDistCity to true;

FOR j = 0 to j = (length of cities array)-1

Declare list variable: pathOfJ

Declare boolean variable: ifNeighbor = false

FOR k = 0 to k = (length of cityConnections)-1

IF first or second City's name of that connection = name of current city

IF first or second City's name of that connection = name minDistCity

SET ifNeighbor to true

Break

Declare double variable: d = distance between minDistCity, current City

IF (d is not 0) and (visited is false for current City) and (distance of minDistCity is not infinity) and (ifNeighbor is true)

Declare double variable: newDist = distance of minDistCity + d

IF newDist < current distance

SET distance to newDist

SET pathOfJ to copy of path of minDistCity

ADD current city to pathOfJ

SET path of current city to PathOfJ

METHOD findMİnDistCity :
Method takes parameters: distance(double array), visited(boolean array)

Declare integer variable: minCity = -1

FOR i = 0 to i = (length of distance array)-1

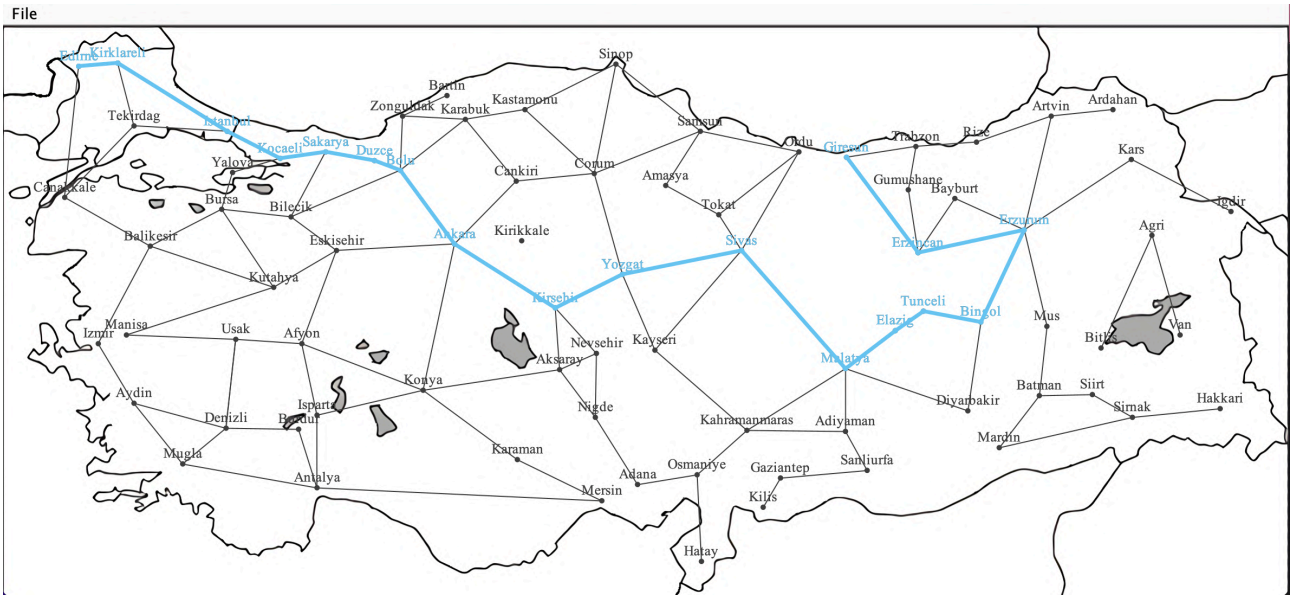
IF (visited is false) and (minCity is -1 or distance < distance of minCity)

SET minCity to i

RETURN minCİty

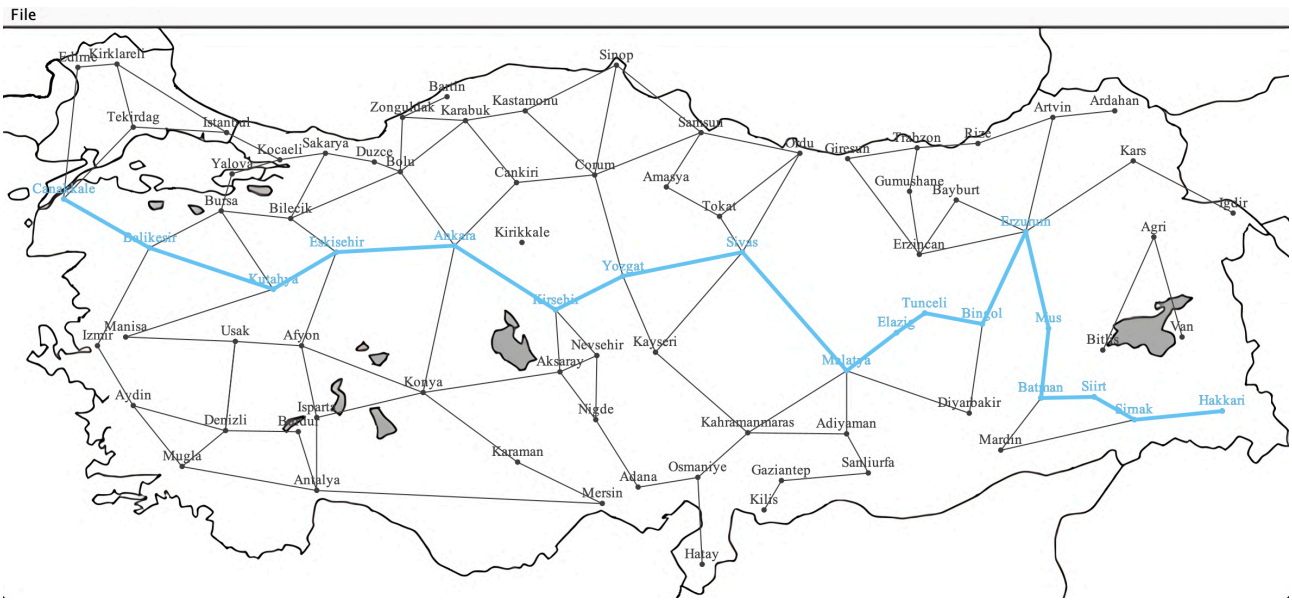
Cases

Case 1: Edirne to Giresun




```
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
```

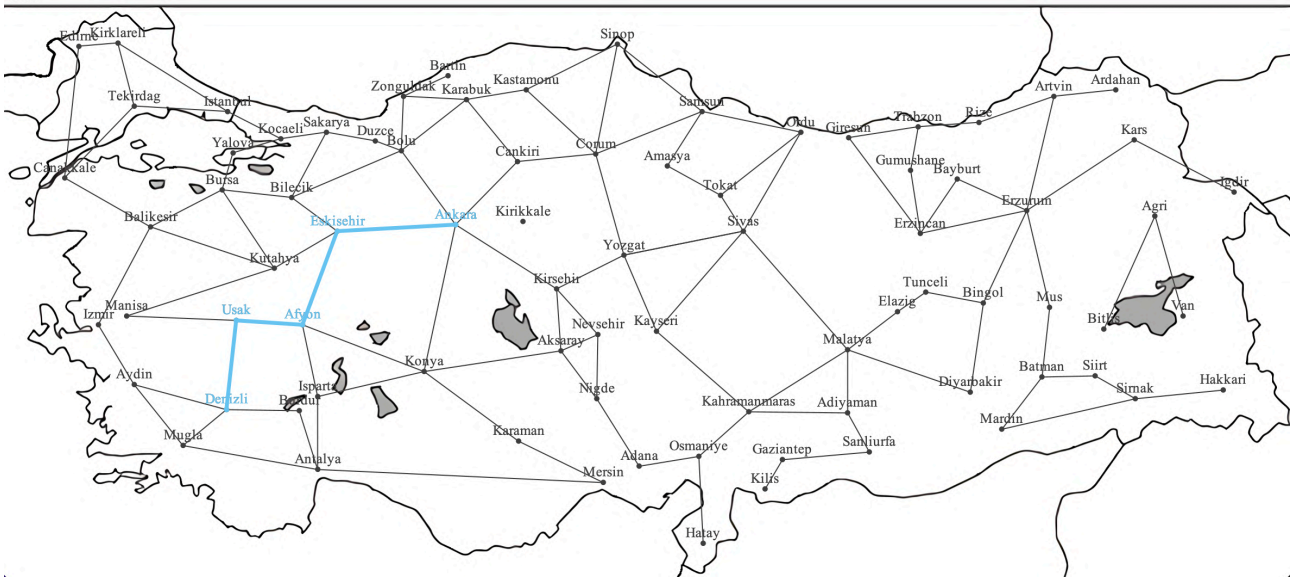
Case 2: Çanakkale to Hakkari



```
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
```

Case 3: Invalid city names

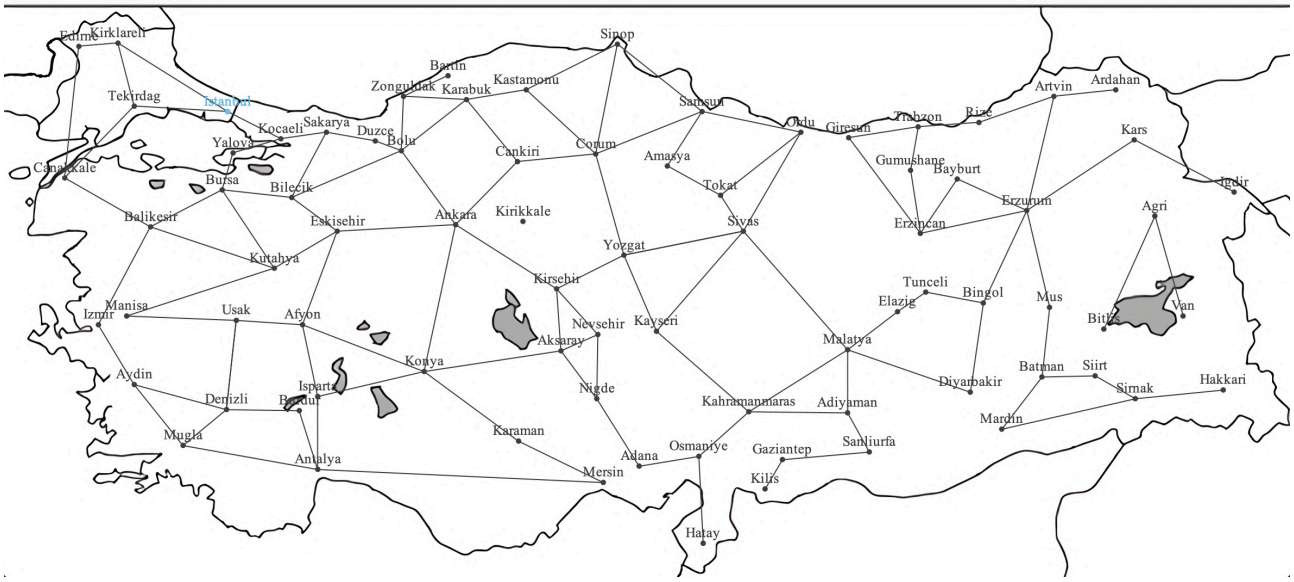
File




```
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
```

Case 4: Path to the same city

File




```
Enter starting city: Istanbul
Enter destination city: Istanbul
Total Distance: 0.00. Path: Istanbul
```

Case 5: Unreachable city pairs

```
Enter starting city: Izmir
Enter destination city: Van
No path could be found.
Process finished with exit code 0
```

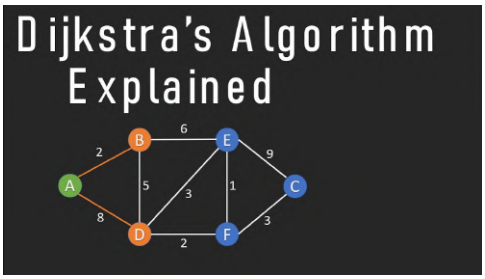
References

Links for websites:

https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm&ved=2ahUKEwi1-ffzlpyFAxVdYPEDHRZYAIMQFnoECBsQAQ&usg=AOvVaw14iNMPx1mzwSQQ4UNocSSt

https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php#:~:text=Dijkstra's%20algorithm%20finds%20the%20shortest,all%20the%20unvisited%20neighboring%20vertices.&ved=2ahUKEwjJocX7jJyFAxWFafEDHb9DB-4QFnoECCQQAw&usg=AOvVaw1EjL0BICnKZMTbeyRZZ0sV

Link for YouTube video:



Author: Aysu Keskin
Student Number: 2023400042

Date:30.03.2024