# Turkey Navigation

CMPE 160 Assignment 2

Salih Can ERER
2022400174

Boğaziçi University

# Contents

Chapter 1

# Code Elaborations

## 1.1 Referances

This particular Youtube video assisted me to grasp the fundamentals of the Dijkstra's Algorithm's purpose, function and significance.

FelixTechTips. (2020, September 26). *Dijkstras Shortest Path Algorithm Explained | With Example | Graph Theory* [Video]. YouTube. https://www.youtube.com/watch?v=bZkzH5x0SKU

## 1.2 Explanations

The implemented algorithm is based on Dijkstra's Algorithm and efficiently computes the shortest path from a source city to all other cities within a graph. The graph is represented as an adjacency matrix, and the algorithm focuses on visualizing the shortest path from the source to a specified destination city.

**Initialization**

- The graph is represented as an adjacency matrix `adjacencyMatrix`, where `adjacencyMatrix[i][j]` stores the distance from city $i$ to city $j$. Distances to non-directly connected cities are set to infinity (`Double.MAX_VALUE`).
- Three key data structures are initialized:
  1. `distanceArray`: Stores the shortest known distances from the source to each vertex. Initially, all distances are set to infinity, except for the source, set to 0.
  2. `sptSet`: A Boolean array indicating if a vertex's shortest distance has been finalized. Initially set to false for all vertices.
  3. `parent`: Stores each vertex's predecessor in the shortest path from the source, enabling path reconstruction.

**Algorithm Execution**

1. The algorithm iteratively selects the non-finalized vertex with the minimum distance, marking it as processed.
2. For each neighboring vertex of the selected vertex, it checks if a shorter path is found through the current vertex. If so, it updates the neighboring vertex's distance and records the current vertex as its predecessor.
3. Using the `parent` array, the shortest path from the source to the destination is reconstructed.

**Visualization**

- Draws lines between consecutive cities on the path.
- Marks cities on the path with filled circles and labels them.

This visualization aids in intuitively understanding the chosen shortest route.

**Path and Distance Output**

The algorithm outputs the total distance of the shortest path and the sequence of cities along this path, from the source to the destination. This output provides both a verification of the algorithm's effectiveness and a clear, human-readable representation of the path found.

**Conclusion**

The implementation of Dijkstra's Algorithm in this project not only computes shortest paths efficiently but also incorporates a visualization aspect, enhancing the comprehensibility and accessibility of the algorithm's results. This approach is especially beneficial in applications requiring geographical visualization and detailed path information, such as mapping software or logistical planning tools.

## 1.3 Pseudocode for the algorithm

In this section, the pseudeocode of the algorithm, and also the pseudeocode of the other methods will be provided.

**Algorithm 1** Dijkstra's Algorithm for Shortest Path and Visualization

```
1: procedure DIJKSTRA(adjacencyMatrix, source, destination, cities)
2:     vertexCount ← length(adjacencyMatrix)
3:     Declare distanceArray[vertexCount]
4:     Declare sptSet[vertexCount]
5:     Declare parent[vertexCount]
6:     for i ← 0 to vertexCount − 1 do
7:         distanceArray[i] ← ∞
8:         sptSet[i] ← false
9:     end for
10:    parent[source] ← −1
11:    distanceArray[source] ← 0
12:    for count ← 0 to vertexCount − 1 do
13:        u ← MINDISTANCE(distanceArray, sptSet, vertexCount)
14:        sptSet[u] ← true
15:        for v ← 0 to vertexCount − 1 do
16:            if ¬sptSet[v] and adjacencyMatrix[u][v] ≠ ∞ and distanceArray[u] +
   adjacencyMatrix[u][v] < distanceArray[v] then
17:                parent[v] ← u
18:                distanceArray[v] ← distanceArray[u] + adjacencyMatrix[u][v]
19:            end if
20:        end for
21:    end for
22:    if distanceArray[destination] ≠ ∞ then
23:        PRINTSOLUTION(destination, distanceArray, parent, cities)
24:    else
25:        Print "No path could be found."
26:    end if
27: end procedure
28: function MINDISTANCE(distanceArray, sptSet, vertexCount)
29:    minValue ← ∞
30:    minIndex ← −1
31:    for v ← 0 to vertexCount − 1 do
32:        if ¬sptSet[v] and distanceArray[v] ≤ minValue then
33:            minValue ← distanceArray[v]
34:            minIndex ← v
35:        end if
36:    end for
37:    return minIndex
38: end function
```

**Algorithm 2** Utility Functions Used in the Code

```
1: procedure PrintSolution(destination, distanceArray, parent, cities)
2:     Print "Total Distance: ", distanceArray[destination], ". Path: "
3:     Set pen color and radius for visualization
4:     PrintPath(destination, parent, cities)
5: end procedure
6: procedure PrintPath(j, parent, cities)
7:     if parent[j] = −1 then
8:         Print cities[j].cityName
9:         Visualize the city on the map return
10:    end if
11:    PrintPath(parent[j], parent, cities)
12:    Print " -> ", cities[j].cityName
13:    Visualize the path and city on the map
14: end procedure
```

Chapter 2

# Example Screenshots

**Case 1:** Edirne to Giresun

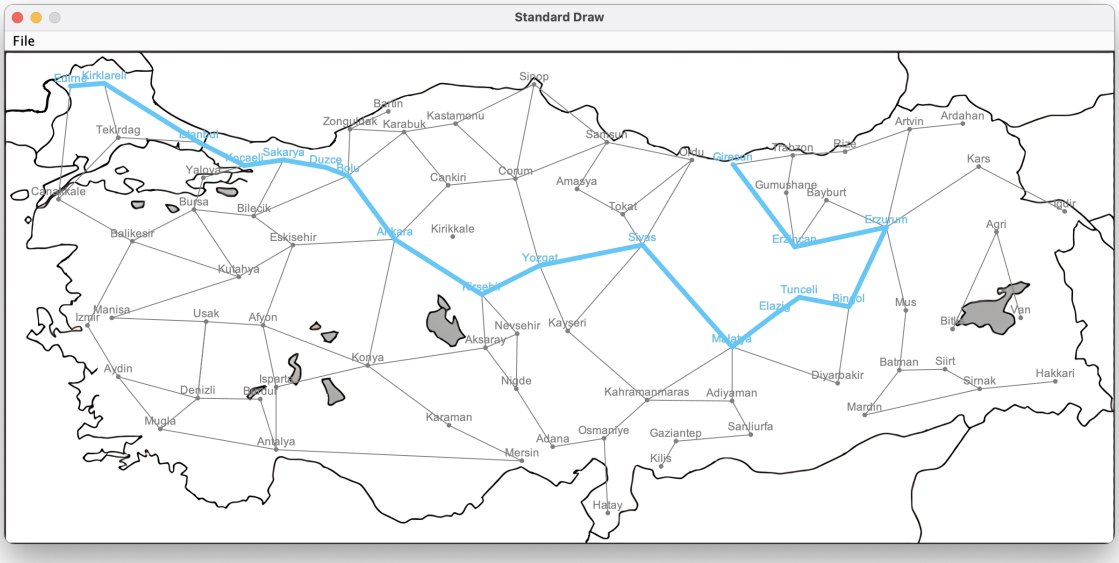


**Fig. 2.1:** Case 1

Please enter starting city: Edirne
Please 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

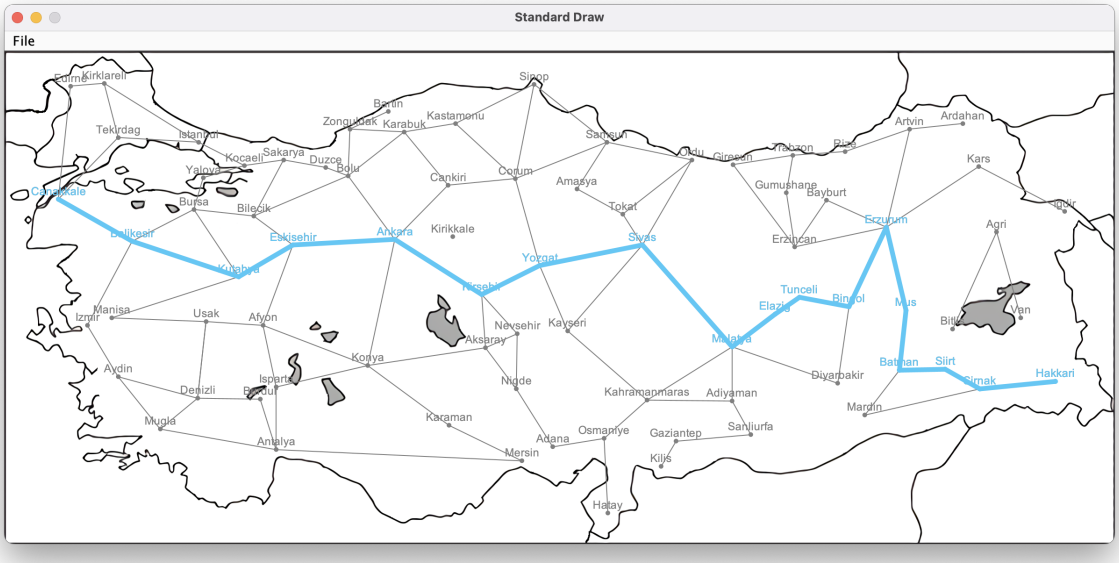


**Fig. 2.2:** Case 2

Please enter starting city: Canakkale
Please 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: User should be prompted again to enter a valid city name.

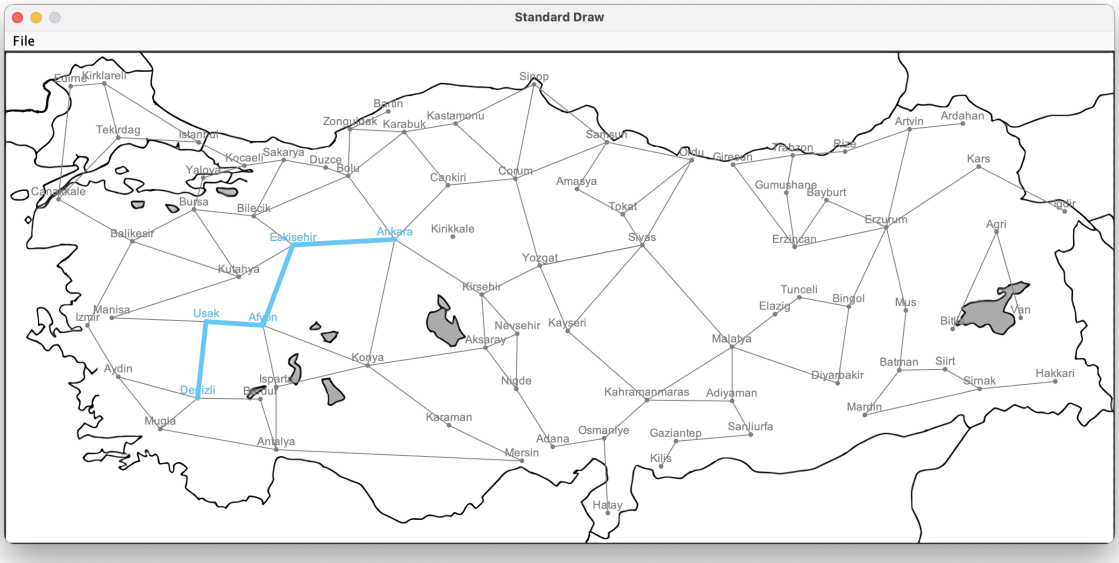


**Fig. 2.3:** Case 3

Please enter starting city: Anka
City named 'Anka' not found. Please enter a valid city name.
Please enter starting city: Ankara
Please enter destination city: Deni
City named 'Deni' not found. Please enter a valid city name.
Please enter destination city: Denizli
Total Distance: 689.10. Path: Ankara -> Eskisehir -> Afyon -> Usak -> Denizli

**Case 4:** Path to the same city

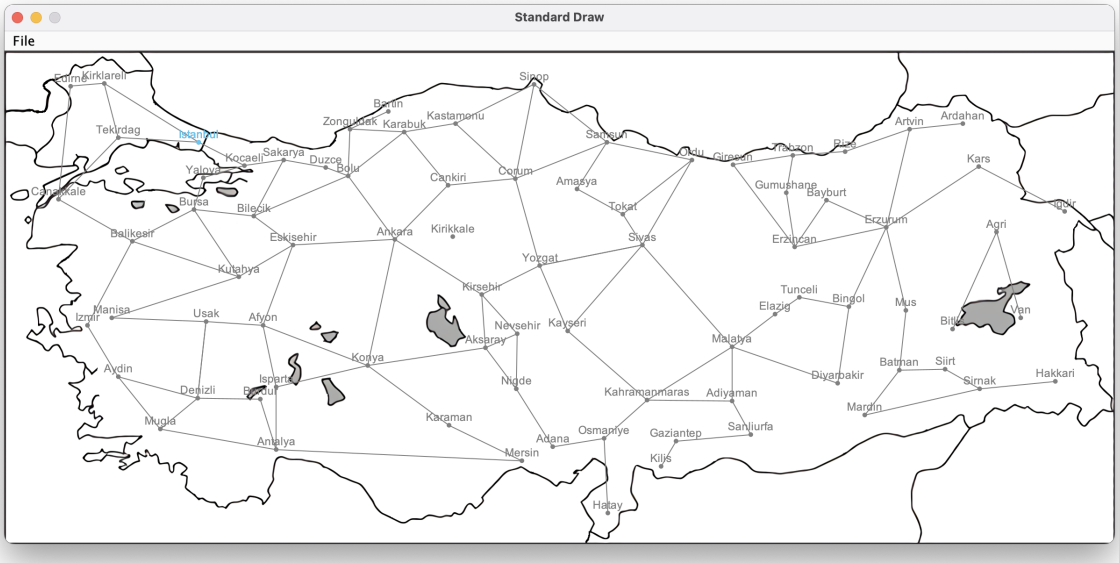


**Fig. 2.4:** Case 4

Please enter starting city: Istanbul
Please enter destination city: Istanbul
Total Distance: 0.00. Path: Istanbul

**Case 5:** Unreachable city pairs: There may be no path from the starting city to the destination city. In this case, no graphical output is produced and console output should be "No path could be found".

Please enter starting city: Izmir
Please enter destination city: Van
No path could be found.