Assignment 2 Report 04.04.2024

Cmpe 160

Mehmet Kaan ÜNSEL

Here is a detailed explanation for my path-finding algorithm:

I used **Dijkstra's algorithm** because it is efficient for finding the shortest path in graphs without negative weight edges. It's particularly useful in real-world applications like mapping and navigation systems, where the graph represents a network of locations connected by paths of various distances.

Initialization

- The algorithm begins by initializing the graph with cities as **nodes**. Each city has a list of connections to other cities.

- Initially, all cities are marked unvisited, and the distance to each city from the start city is set to infinity, except for the start city itself, which is set to a distance of zero. This setup reflects that, at the beginning, the shortest distance to every city is unknown.

Core Loop

- The main part of the algorithm repeatedly examines unvisited cities, starting with the start city, to update the shortest distances from the start city to all other cities.

- For the current city, the algorithm considers all its unvisited neighbors (i.e., directly connected cities) and calculates the distance from the start city to each neighbor by summing the distance from the start city to the current city and the distance from the current city to the neighbor.

- If this total distance to a neighbor is less than the previously recorded distance, the algorithm updates the shortest distance to this neighbor and records the current city as the **"previous node"** on the path to this neighbor. This step ensures that the algorithm always keeps the shortest path to each city updated.

- After examining and updating distances for all neighbors of the current city, the city is marked as visited, meaning it will not be checked again, and its distance from the start city is finalized.

- The algorithm then selects the next city to examine from the set of unvisited cities. This next city is the one with the **smallest** recorded distance from the start city. By always choosing the unvisited city with the smallest distance, Dijkstra's algorithm ensures that it follows the shortest possible path step by step.

Path Reconstruction

- Once the algorithm has visited all cities or found the target city, it has effectively calculated the shortest path from the start city to every other city in the graph.

- To reconstruct the shortest path from the start city to the destination city, the algorithm traces back from the destination city using the **"previous node"** information stored at each city. By moving from the destination city to its recorded previous node, then to that node's previous node, and so on, the algorithm retraces the steps of the shortest path back to the start city.

Outcome

- The algorithm outputs the total distance of the shortest path from the start city to the destination city, along with the sequence of cities that form this path. This gives both the length and the route of the shortest path between the two specified cities.

Here is the pseudo code for my path-finding algorithm:

```
// cities, city_names, and connections are already populated
initialize cities list
initialize city_names list
initialize unvisitedCities list

idCount = 0
for each line in city_coordinates.txt:
    create city object with name = line[0], x = line[1], y = line[2], id = idCount
    add city object to cities and unvisitedCities
    add line[0] to city_names
    idCount++

prompt for start_city and end_city
validate start_city and end_city

load connections from city_connections.txt

for each connection:
    find cities by name
    update connections in cities list

find and assign start_city_object and end_city_object from cities
set start_city_object.distance = 0
set nextCity = start_city_object
```

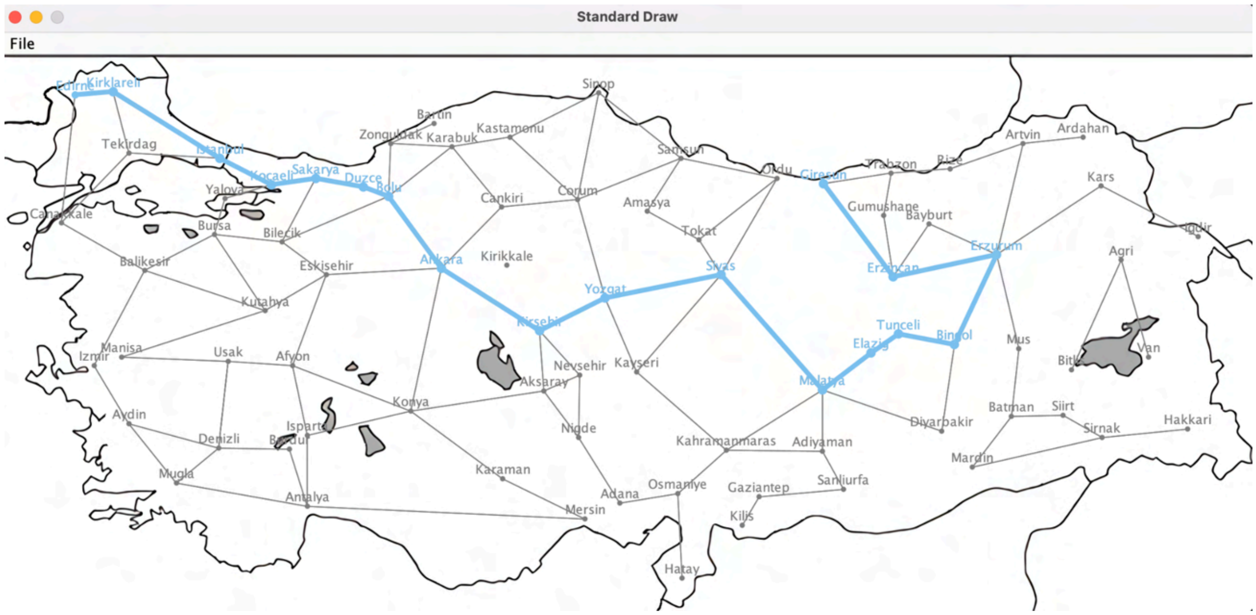
```
while unvisitedCities is not empty and nextCity.distance != Infinity:
    mark nextCity as visited
    remove nextCity from unvisitedCities

    for each connection of nextCity:
        if connection is not visited:
            calculate realDistance
            if realDistance < connection.distance:
                update connection.prev_node to nextCity
                update connection.distance to realDistance

    find city in unvisitedCities with minimum distance
    set this city as nextCity

if path to end_city_object found:
    trace back from end_city_object using prev_node
    display path and total distance
else:
    print "No path could be found."
```

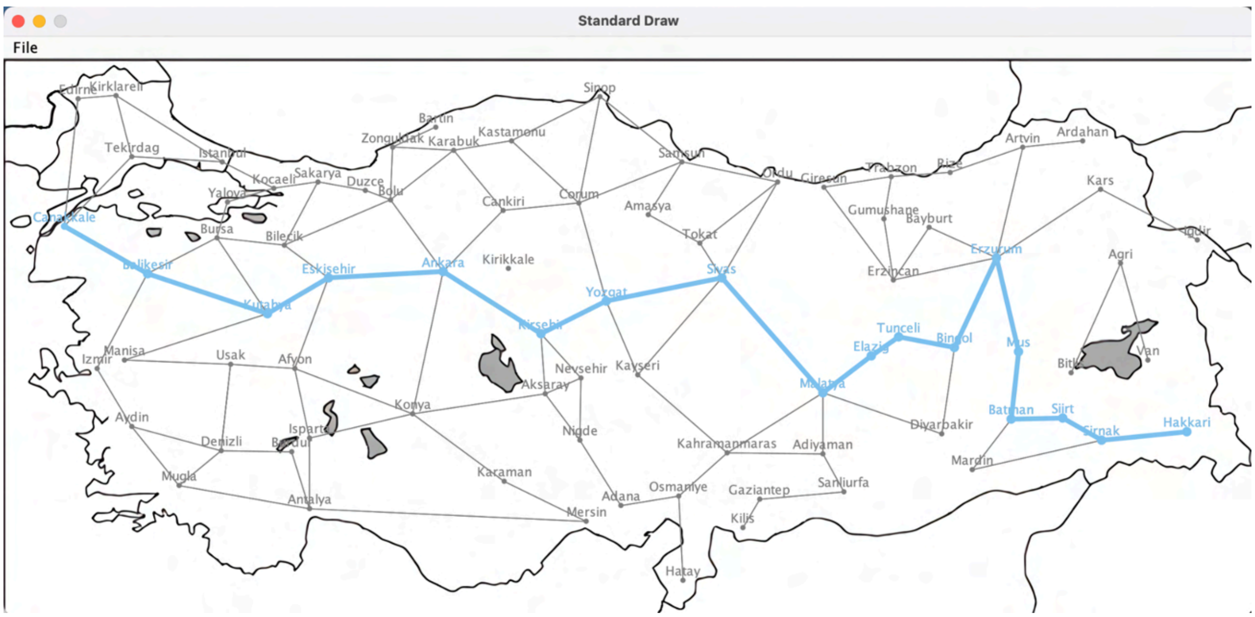
Case 1: Edirne to Giresun



Console Output:

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

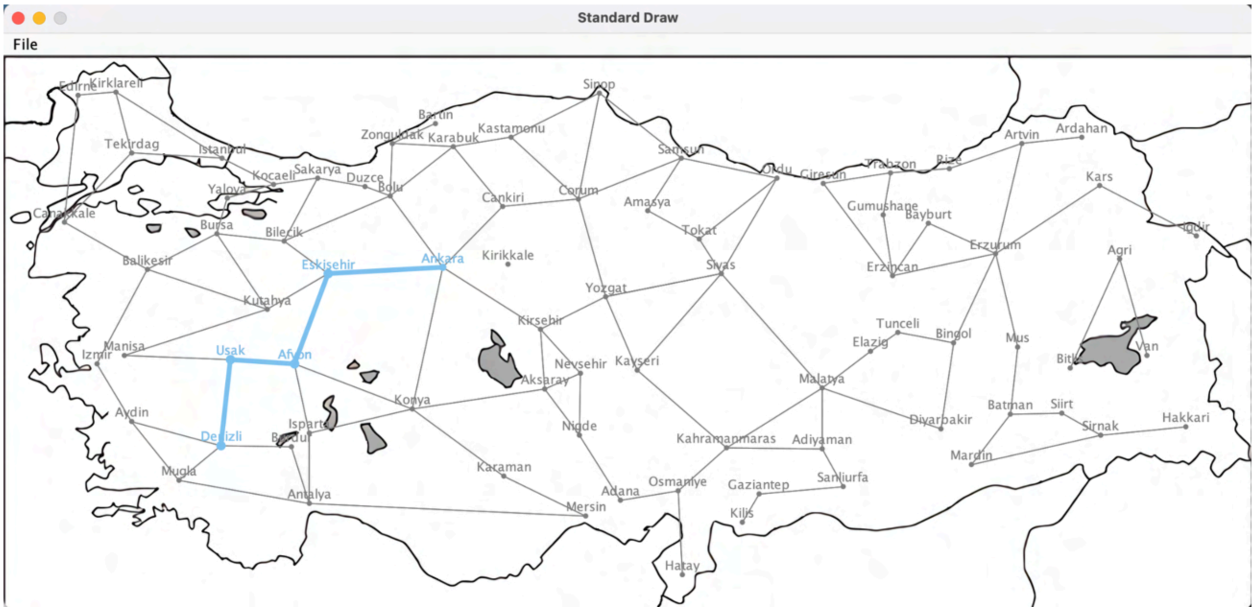
Case 2: Çanakkale to Hakkari



Console Output:

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

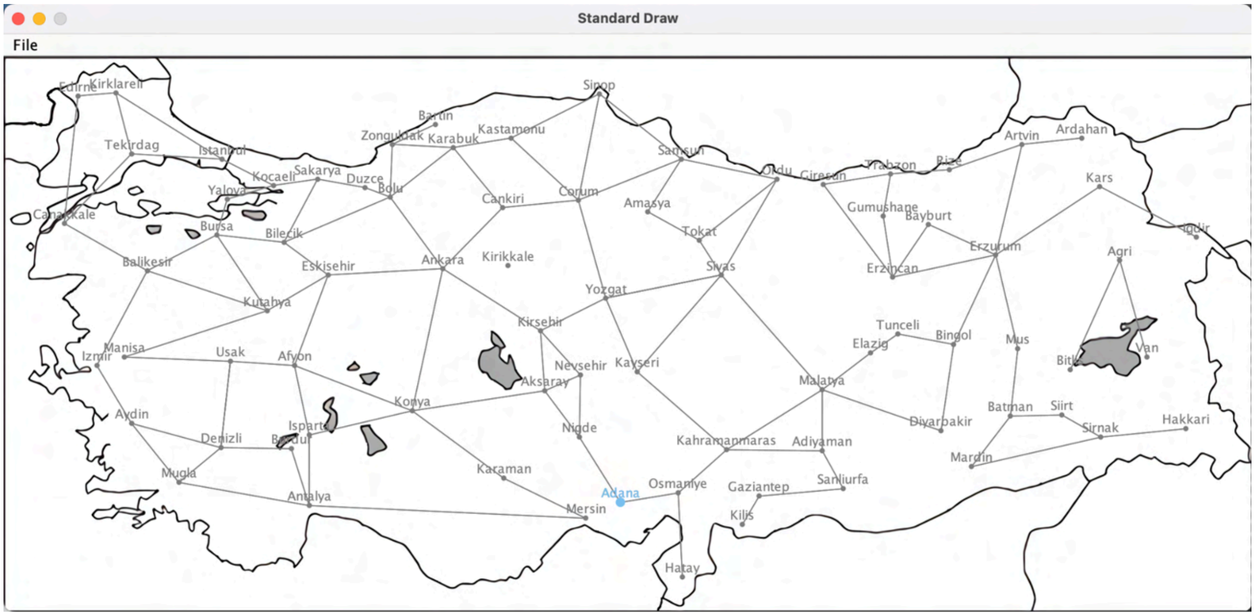
Case 3: Invalid city names: User should be prompted again to enter a valid city name.



Console Output:

Enter starting city:
Anakara
City named 'Anakara' not found. Please enter a valid city name.
Enter starting city:
Ankara
Enter destination city:
Denizlilili
City named 'Denizlilili' 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



Console Output:

Enter starting city:
Adana
Enter destination city:
Adana
Total Distance: 0.00. Path: Adana

Case 5: Unreachable city pairs

Console Output:

Enter starting city:

Izmir

Enter destination city:

Van

No path could be found.

References

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

https://www.youtube.com/watch?v=pVfj6mxhdMw