CMPE 160 ASSIGNMENT 2

Turkey Navigation

Name: Ahmet Erdem Bulut

Student ID: 2022400093

Path Finding Method Explanation:

The ‘findPath’ method is designed to compute the shortest path between two specified cities by using a simplified version of Dijkstra’s algorithm, which is one of the common ways to find the shortest path from a starting point to all other vertices.

The algorithm first checks if the start and end cities are the same. If so, it returns a path containing just the start city, else the main algorithm starts. We start with initializing three main arrays to store the states of each city during the algorithm’s execution. These arrays are:

- Visited: Keeps track of whether a city has been visited
- Distances: Stores the shortest known distance from the start city to every other city
- Previous: Stores the preceding city on the shortest path from the start city

Main Loop:

The core of the method iterates over the cities, continuously selecting the unvisited city with the minimum distance from the start city. For each such city, it updates the distances to its neighboring cities based on the connections. This process repeats until all cities have been visited or the destination city is reached. If a shorter path to a city via another city is found, the method updates the distance to reflect this shorter path and records the city from which this shorter path arrived in the ‘previous’ array.

Path Reconstruction:

When the algorithm finishes, the method constructs the shortest path from the destination city back to the start city using the ‘previous’ array. If the end city is unreachable, which is indicated by ‘null’ in the ‘previous’ array for the end city, it returns ‘null’. Otherwise, the method returns a list of cities forming the shortest path from the start city to the destination city.

Pseudocode:

```
algorithm findPath is
	input: list of cities, list of connections, starting city, destination city
	output: list of cities forming the shortest path from the starting city to the destination city

	IF starting city is the same as destination city THEN
		SET sameCityPath as a new list of cities
		ADD starting city to sameCityPath
		RETURN sameCityPath
	ENDIF

	INITIALIZE array Visited to mark whether each city has been visited, initially SET to false for all cities
	INITIALIZE array Distances with infinite values for all cities
	INITIALIZE array Previous to store the preceding city on the path for each city, initially SET to a null equivalent

	SET distance of starting city in Distances to 0

	WHILE there is an unvisited city with a finite distance DO
		Find the unvisited city with the smallest distance (let it be CurrentCity)
		Mark CurrentCity as visited

		FOR each connection from CurrentCity to a neighboring city (Neighbor) DO
			Calculate the potential new distance as the sum of the distance to CurrentCity and the distance from CurrentCity to Neighbor.
			IF this potential new distance is less than the current distance to Neighbor in Distances THEN
				Update Neighbor's distance in Distances to this new lower distance.
				Update Neighbor's preceding city in Previous to CurrentCity
			ENDIF
		ENDFOR
	ENDWHILE

	IF the destination city has no preceding city in Previous THEN
		RETURN null or an equivalent to indicate no path exists
	ENDIF

	INITIALIZE an empty list of cities ShortestPath
	Start with the destination city as the current city to examine and trace back through Previous to reach the starting city
	WHILE the current city is not null or an equivalent to indicate current city has no preceding city DO
		INSERT the current city at the beginning of ShortestPath
		SET current city to its previous
	ENDWHILE

	RETURN ShortestPath
```

Pseudocode for my helper method indexOfCity:

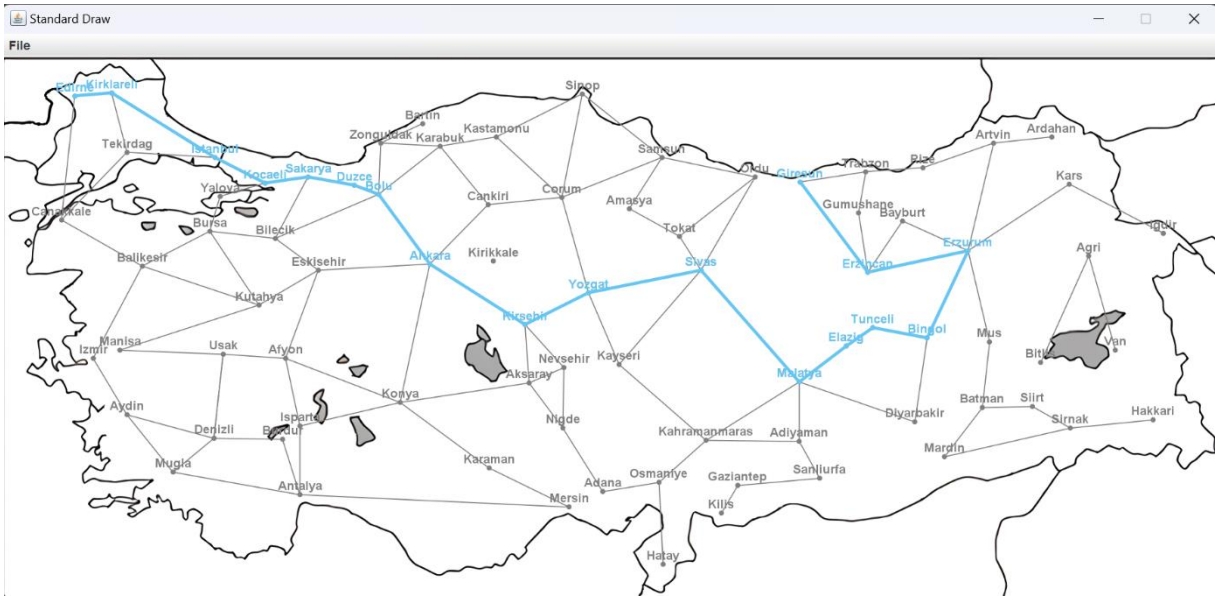
```
algorithm indexOfCity
	input: a list of cities ‘cities’, a city ‘city’ to find
	output: the index of ‘city’ in the ‘cities’ list, or -1 if ‘city’ is not found

	FOR each city in cities with index i DO
		IF city at index i equals to ‘city’ THEN
			RETURN i
		ENDIF
	ENDFOR

	RETURN -1 (indicates ‘city’ was not found in ‘cities’)
```

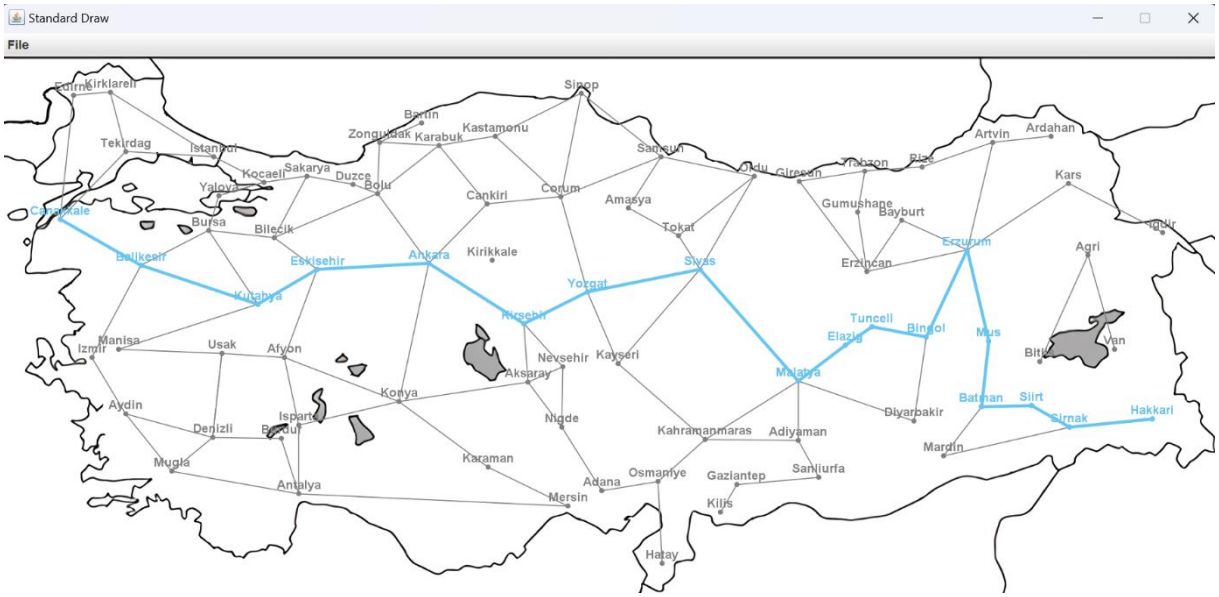
Some images for the assignment are on the following pages.

Case 1: Edirne to Giresun



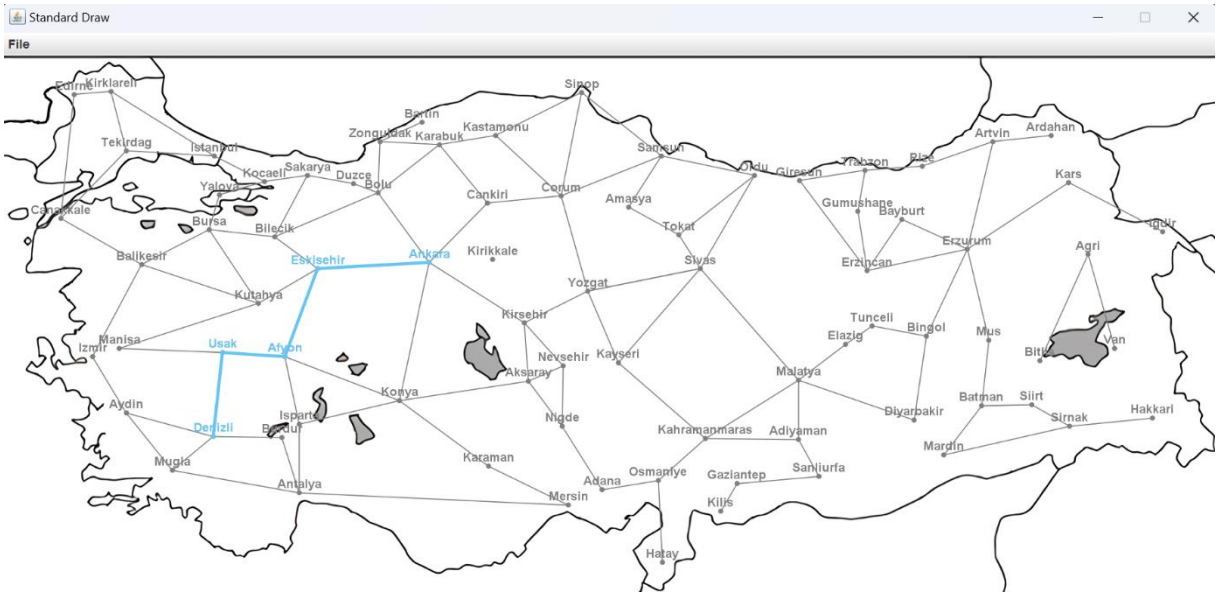

```
Enter starting city: Edirne
Enter destination city: Giresun
Total distance: 2585.49. Path: Edirne -> Kirklareli -> Istanbul -> Kocaeli -> Sakarya -> Duzce -> Bolu -> Ankara -> Kirsehir -> Yozgat -> Sivas -> Malatya -> Elazig -> Tunceli -> Bingol ->
Erzurum -> Erzincan -> Giresun
```

Case 2: Çanakkale to Hakkari



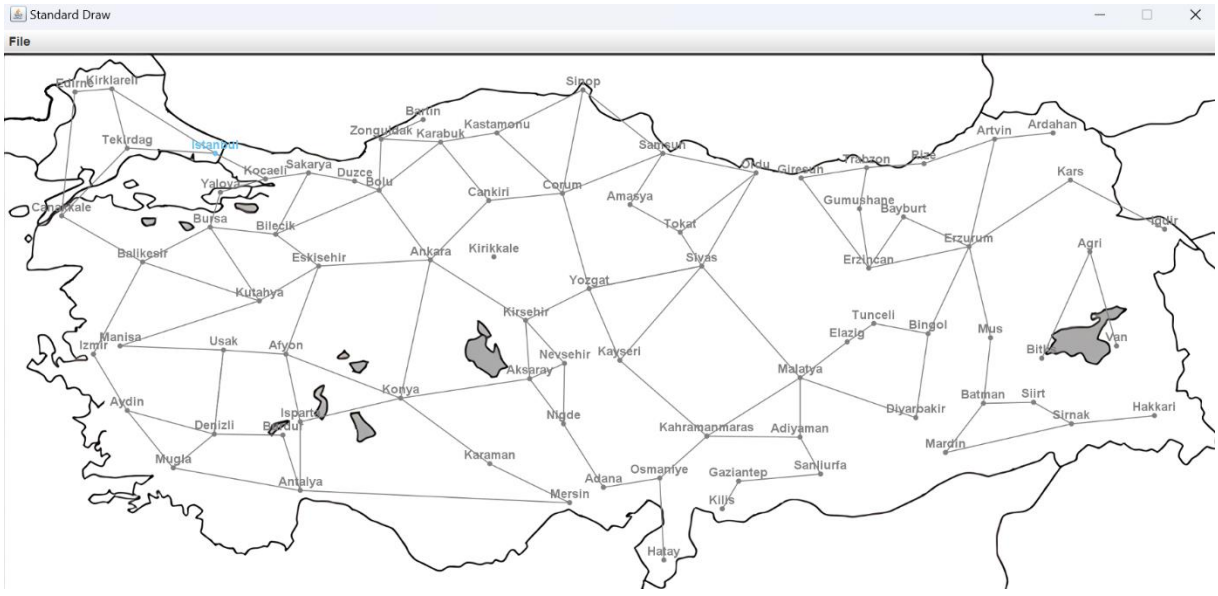

```
Enter starting city: Canakkale
Enter destination city: Hakkari
Total distance: 2780.87. Path: Canakkale -> Balikesir -> Kutahya -> Eskisehir -> Ankara -> Kirsehir -> Yozgat -> Sivas -> Malatya -> Elazig -> Tunceli -> Bingol -> Erzurum -> Mus -> Batman ->
Siirt -> Sirnak -> Hakkari
```

Case 3: Invalid city names




```
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

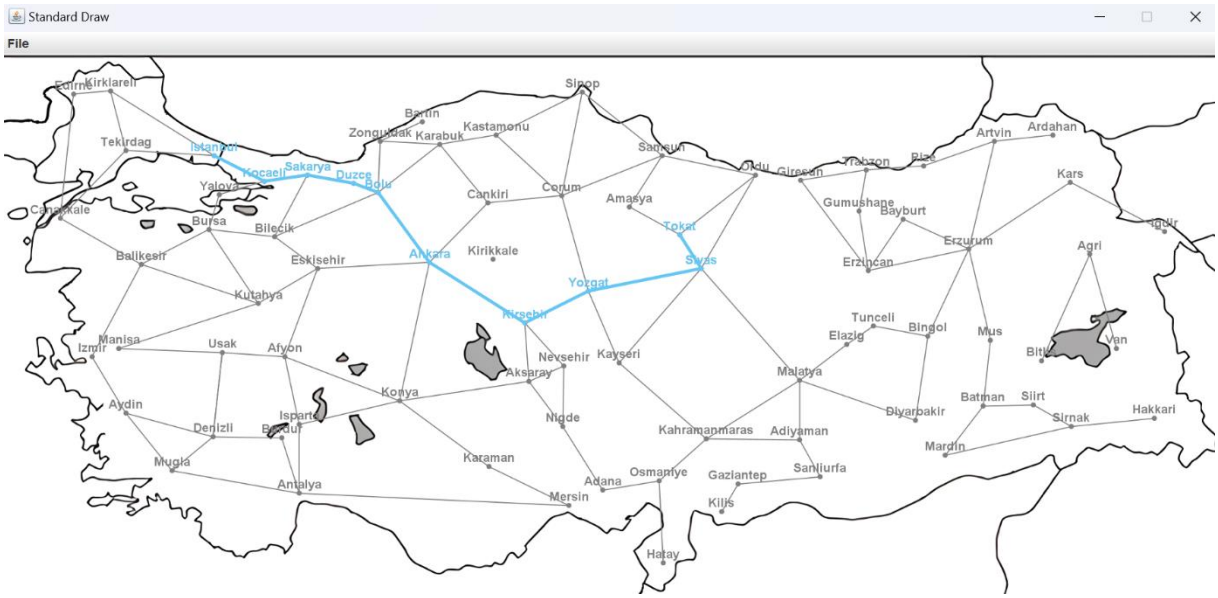



```
Enter starting city: Istanbul
Enter destination city: Istanbul
Total distance: 0.00. Path: Istanbul
```

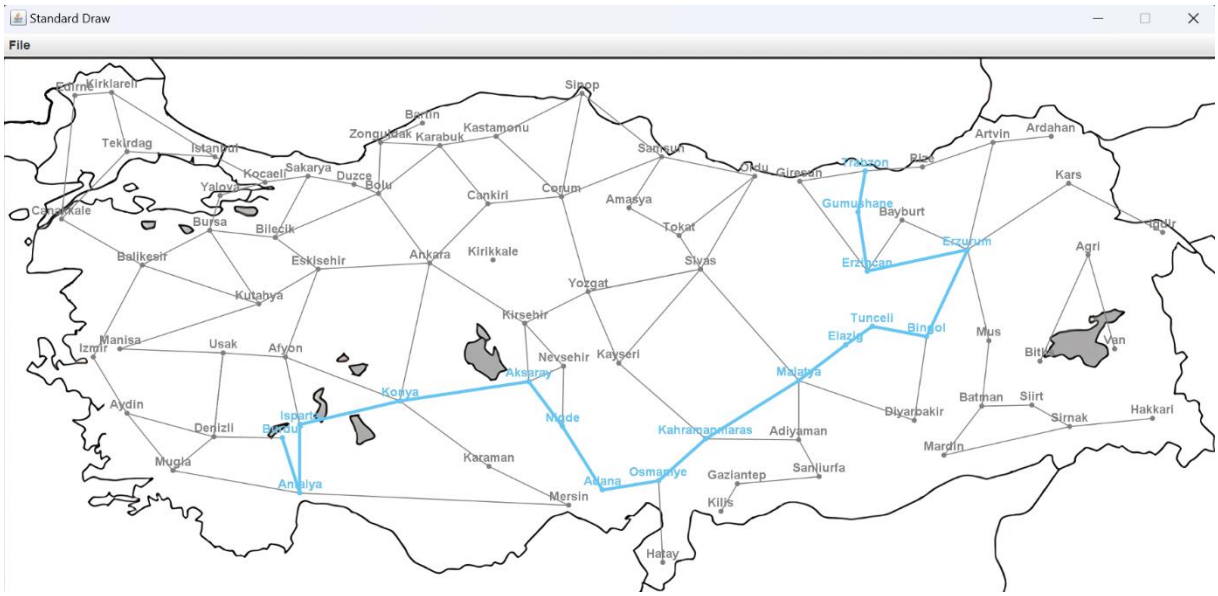
Case 5: Unreachable city pairs

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

Some Custom Cases:




```
Enter starting city: Istanbul
Enter destination city: Tokat
Total distance: 1167.85. Path: Istanbul -> Kocaeli -> Sakarya -> Duzce -> Bolu -> Ankara -> Kirsehir -> Yozgat -> Sivas -> Tokat
```




```
Enter starting city: Trabzon
Enter destination city: Burdur
Total distance: 2283.01. Path: Trabzon -> Gumushane -> Erzincan -> Erzurum -> Bingol -> Tunceli -> Elazig -> Malatya -> Kahramanmaras -> Osmaniye -> Adana -> Nigde -> Aksaray -> Konya -> Isparta
ç -> Antalya -> Burdur
```

Sources:

1. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
2. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
3. https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php#:~:text=Dijkstra%27s%20algorithm%20finds%20the%20shortest,all%20the%20unvisited%20neighboring%20vertices.
4. https://en.wikipedia.org/wiki/Pseudocode
5. https://builtin.com/data-science/pseudocode
6. https://www.youtube.com/watch?v=PwGA4Lm8zuE