T.C. ESKİŞEHİR TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

BIM309 – Artificial Intelligence

HOMEWORK II - Report

A Python Implementation of Uniform Cost Search (UCS)

Kaouther MOUHEB

99926527616

I. Introduction

This is a brief report describing the UCS Algorithm and the steps I followed to implement the method on a part of the Turkish map to find the shortest path between two cities and some examples to test the application.

II. Uniform Cost Search

Uniform Cost Search or UCS is an informed search algorithm that aims to find an optimal path between an initial state and a goal state. UCS is a modified version of the Breadth First Search algorithm where the priority is given to the cost instead of the depth.

Pseudocode of UCS:

```
UCS( Graph, InitialState, Goal)

- Frontier <- New PriorityQueue()
- Explored <- New Set()
- Frontier.add( 0, InitialState, [InitialState])
- While Frontier is not empty:
    o currentState <- fetch element from Frontier
    o currentCost <- cost to currentState
    o currentPath <- path to currentState
    o If CurrentState == Goal:
          return currentCost, currentPath
    o For neighbor in currentState.neighbors
        ▪ If neighbor not in Explored:
        ▪ Frontier.add( currentCost + cost(currentState, neighbor) ,
                        neighbor,
                        currentPath + neighbor)
- Return ResultNotFound
```

Criteria of UCS:

Completeness: Complete if the solution has a finite cost

Time complexity: $O(b^{\frac{C'}{e}})$ | C' = cost of optimal solution & e = minimum cost for 1 action

Space complexity: $O(b^{\frac{C'}{e}})$ | C' = cost of optimal solution & e = minimum cost for 1 action

Optimality: optimal

Note: UCS does not stop once the goal is encountered, it keeps looking for possible shorter paths.

III. Implementation:

- First, I create a **Graph class** that has 2 attributes: one dictionary that associate each vertex with a list of its neighbors and one dictionary that associates each 2 neighbors with the distance between them. Since the graph is bidirectional (ab = ba), I store each distance only once using the 2 names alphabetically ordered as a key to each distance (cost). The graph also has functions to get the neighbors of a vertex and the distance between two vertices.
- Secondly, I implement the function to **build the graph**. This function takes the path as a parameter and opens the corresponding file encoded in UTF-8 to support Turkish Characters.
  I used the csv module's DictReader function to read the data. This will automatically skip the first line and consider its cells as column names (keys). Then row by row, I add the cities and distances between them to the graph. I also keep a list of all cities encountered to check the user input later.
- Then, I implement the **UCS function** that takes the start city and ending city and the graph as inputs and returns the shortest path and the distance between these two cities. I used python's built-in PriorityQueue to keep track of the frontier. I associated each element of the queue with its cost and the path that leads to it with that cost. I used a Set for the explored vertices.
- Next, I use these functions in the **main function** after I take the path and start and ending cities as **user inputs** using the input() function.
- Finally, to make the application robust to errors I use try – catch blocks to **handle errors** that can be faced because of invalid user input (FileNotFoundError, CityNotFoundError and SameCityError).

IV. Sample Results:

Solutions:

```
Welcome!
Please enter the path of the road map file: data/cities.csv
Please give your current city: İstanbul
Please give your destination city: Kayseri
The shortest path for you: ['İstanbul', 'Eskişehir', 'Konya', 'Kayseri']
Distance = 435
Thank you for using our app!
```

1. Shortest path between Istanbul and Kayseri

```
Welcome!
Please enter the path of the road map file: data/cities.csv
Please give your current city: Trabzon
Please give your destination city: İzmir
The shortest path for you: ['Trabzon', 'Samsun', 'Ankara', 'Eskişehir', 'İzmir']
Distance = 525
Thank you for using our app!
```

2. Shortest path between Trabzon and Izmir

```
Welcome!
Please enter the path of the road map file: data/cities.csv
Please give your current city: Çanakkale
Please give your destination city: Konya
The shortest path for you: ['Çanakkale', 'İstanbul', 'Eskişehir', 'Konya']
Distance = 375
Thank you for using our app!
```

3. Shortest path between Canakkale and Konya

```
Welcome!
Please enter the path of the road map file: data/cities.csv
Please give your current city: Balıkesir
Please give your destination city: Adana
The shortest path for you: ['Balıkesir', 'İzmir', 'Muğla', 'Antalya', 'Adana']
Distance = 490
Thank you for using our app!
```

4. Shortest path between Balikesir and Adana

Handling Errors:

```
Welcome!
Please enter the path of the road map file: data/cities.csv
Please give your current city: İstanbul
Please give your destination city: Paris
Paris does not exist
Please try with a valid City
Thank you for using our app!
```

1. Handling CityNotFound error (2nd city example)

```
Welcome!
Please enter the path of the road map file: data/cities.csv
Please give your current city: Seoul
Seoul does not exist
Please try with a valid City
Thank you for using our app!
```

2. Handling CityNotFound error (1st city example)

```
Welcome!
Please enter the path of the road map file: data/cities.csv
Please give your current city: Eskişehir
Please give your destination city: Eskişehir
Your Current and Destination cities should be different
Please try with different cities
Thank you for using our app!
```

3. Handling SameCityError

```
Welcome!
Please enter the path of the road map file: bonjour.file
File not found! Please try again
Thank you for using our app!
```

4. Handling FileNotFoundError