



Computer Engineering Department

BIM 309 – Artificial Intelligence

UCS - Report

Prepared By
Necip Şükrü Aşık

Date: 04.12.2021

```

with open('cities.csv', 'r', encoding='utf8') as outp:
    next(outp, None)
    1 for sprt in outp:
        line = sprt.split(',')
        city1 = line[0]
        2 if not city1 in graph:
            graph[city1] = {}
            for i in range(1, len(line)-1, 2):
                graph[city1][line[i]] = int(line[i+1])
            city2 = line[1]
            if not city2 in graph:
                graph[city2] = {}

```

Nam	Type	Size	Value
city1	str	1	Balıkesir
g	classes.graph.Graph	1	Graph object of networkx.classes.graph module
graph	dict	0	{}
line	List	3	['Balıkesir', 'Çanakkale', '95']
outp	TextIOWrapper	1	TextIOWrapper object of _io module
path	dict	0	{}
sprt	str	1	Balıkesir,Çanakkale,95

Firstly, I started with converting csv to dictionary.

1. I separate lines with “,” so I can take values with their indexes. I take Line[0] , which is node, and copy it to city1.
2. Graph dict is empty, now I copy city1 to graph.

```

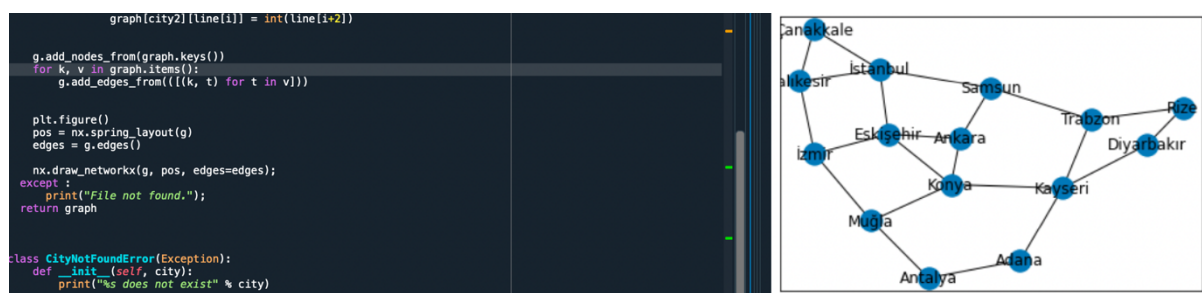
with open('cities.csv', 'r', encoding='utf8') as outp:
    next(outp, None)
    for sprt in outp:
        line = sprt.split(',')
        city1 = line[0]
        if not city1 in graph:
            graph[city1] = {}
            1 for i in range(1, len(line)-1, 2):
                graph[city1][line[i]] = int(line[i+1])
            2 city2 = line[1]
            if not city2 in graph:
                graph[city2] = {}
            3 for i in range(0, len(line)-2, 2):
                graph[city2][line[i]] = int(line[i+2])

```

Nam	Type	Size	Value
city1	str	1	Balıkesir
city2	str	1	Çanakkale
g	classes.graph.Graph	1	Graph object of networkx.classes.graph module
graph	dict	2	{'Balıkesir': {'Çanakkale': 95}, 'Çanakkale': {}}
i	int	1	0
line	list	3	['Balıkesir', 'Çanakkale', '95']
outp	TextIOWrapper	1	TextIOWrapper object of _io module
path	dict	2	{'Balıkesir': {'Çanakkale': 95}, 'Çanakkale': {}}
sprt	str	1	Balıkesir,Çanakkale,95

Then ,

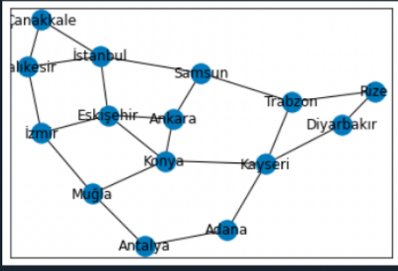
1. I convert last variable of line (distances) into int and added it to graph.
2. This part makes graph bidirectional. If city2 is not in graph, then add it to graph. Without this part, cities will be 12 not 15.
3. Same as 1.step, add distances to graph.



In this part, I draw a graph with networkx. I take nodes from graph's keys and edges from their neighbors.

```
# Implement this function to perform uniform cost on the graph.
def uniform_cost_search(graph, start, end):
    try:
        queue = Q.PriorityQueue()
        queue.put((0, [start]))
        while not queue.empty():
            node = queue.get()
            current = node[1][len(node[1]) - 1]
            if end in node[1]:
                print("Valid Path: " + str(node[1]) + ", Total Distance = " + str(node[0]))
                break
            distance = node[0]
            for neighbors in graph[current]:
                temp = node[1][:]
                temp.append(neighbors)
                queue.put((distance + graph[current][neighbors], temp))
    except:
        if start not in graph:
            CityNotFoundError(start)
        if end not in graph:
            CityNotFoundError(end)
        else:
            print("Path is not found.")

# Implement main to call functions with appropriate try-except blocks
if __name__ == "__main__":
    build_graph(graph)
    uniform_cost_search(graph, 'Istanbul', 'Eskişehir')
```



Variable explorer Help Plots Files

Console 1/A

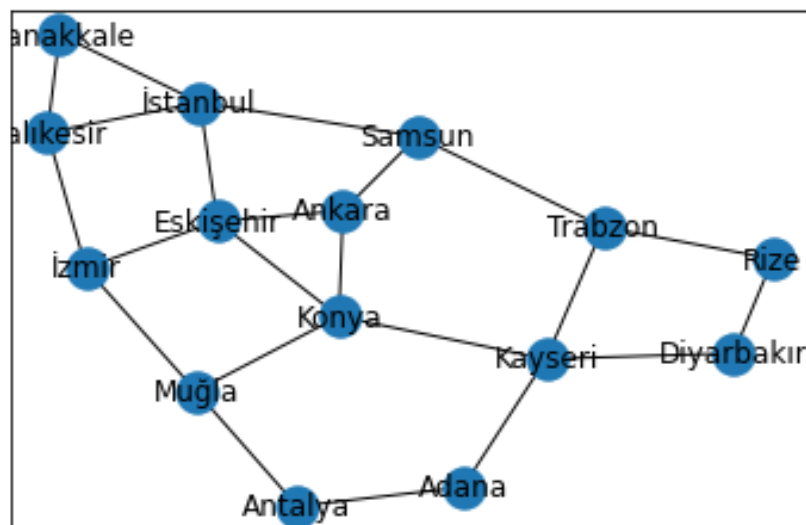
```
In [29]: runfile('/Users/eco/Desktop/submission/deneme2.py', wdir=
Desktop/submission')
Valid Path: ['Istanbul', 'Eskişehir'], Total Distance = 130
In [30]:
```

1. After putting start into queue, we get it and make it node. Current variable change until find the end. Then we take city1 from node and find city2 and print path and distance.

```
# Implement this function to perform uniform cost on the graph.
def uniform_cost_search(graph, start, end):
    try:
        queue = Q.PriorityQueue()
        queue.put((0, [start]))
        while not queue.empty():
            node = queue.get()
            current = node[1][len(node[1]) - 1]
            if end in node[1]:
                print("Valid Path: " + str(node[1]) + ", Total Distance = " + str(node[0]))
                break
            distance = node[0]
            for neighbors in graph[current]:
                temp = node[1][:]
                temp.append(neighbors)
                queue.put((distance + graph[current][neighbors], temp))
```

Name	Type	Size	Value
current	str	1	Istanbul
distance	int	1	0
end	str	1	Eskişehir
g	classes.graph.Graph	1	Graph object of networkx.classes.graph modul
graph	dict	15	{'Balıkesir': {'Çanakkale': 95, 'İstanbul': 155, 'İzmir': 140}, 'Çanakkale ...
neighbors	str	1	Çanakkale
node	tuple	2	(0, ['Istanbul'])
queue	PriorityQueue	1	PriorityQueue object of queue module
start	str	1	Istanbul

2. With travel current's all neighbors, we found shortest path of the start and end after a few steps.



Test Samples:

ISTANBUL - KAYSERI

```
# Implement main to call functions with appropriate try-except blocks
if __name__ == "__main__":
    build_graph(graph)
    uniform_cost_search(graph, 'Istanbul', 'Kayseri')
```

```
ipdb> !runfile('/Users/eco/Desktop/submission/deneme2.py', wdir='/Users/eco/Desktop/submission')
Valid Path: ['Istanbul', 'Eskişehir', 'Konya', 'Kayseri'], Total Distance = 435
ipdb>
```

TRABZON- IZMIR

```
# Implement main to call functions with appropriate try-except blocks
if __name__ == "__main__":
    build_graph(graph)
    uniform_cost_search(graph, 'Trabzon', 'Izmir')
```

```
ipdb> !runfile('/Users/eco/Desktop/submission/deneme2.py', wdir='/Users/eco/Desktop/submission')
Valid Path: ['Istanbul', 'Eskişehir', 'Konya', 'Kayseri'], Total Distance = 435
ipdb> !runfile('/Users/eco/Desktop/submission/deneme2.py', wdir='/Users/eco/Desktop/submission')
Valid Path: ['Trabzon', 'Samsun', 'Ankara', 'Eskişehir', 'Izmir'], Total Distance = 525
ipdb>
```

ÇANAKKALE - KONYA

```
# Implement main to call functions with appropriate try-except blocks
if __name__ == "__main__":
    build_graph(graph)
    uniform_cost_search(graph, 'Çanakkale', 'Konya')
```

```
Valid Path: ['Trabzon', 'Samsun', 'Ankara', 'Eskişehir', 'Izmir'], Total Distance = 525
ipdb> !runfile('/Users/eco/Desktop/submission/deneme2.py', wdir='/Users/eco/Desktop/submission')
Valid Path: ['Çanakkale', 'Istanbul', 'Eskişehir', 'Konya'], Total Distance = 375
ipdb>
```

BALIKESİR- ADANA

```
# Implement main to call functions with appropriate try-except blocks
if __name__ == "__main__":
    build_graph(graph)
    uniform_cost_search(graph, 'Balıkesir', 'Adana')
```

```
IPython 7.16.1 -- An enhanced Interactive Python.
In [1]: runfile('/Users/eco/Desktop/submission/deneme2.py', wdir='/Users/eco/Desktop/submission')
Valid Path: ['Balıkesir', 'Izmir', 'Muğla', 'Antalya', 'Adana'], Total Distance = 490

Figures now render in the Plots pane by default. To make them also appear inline in the Console, uncheck "Mute Inline Plotting" under the Plots pane options menu.
```

ISTANBUL - PARIS

```
# Implement main to call functions with appropriate try-except blocks
if __name__ == "__main__":
    build_graph(graph)
    uniform_cost_search(graph, 'Istanbul', 'Paris')
```

```
IPython 7.16.1 -- An enhanced Interactive Python.
In [1]: runfile('/Users/eco/Desktop/submission/deneme2.py', wdir='/Users/eco/Desktop/submission')
Paris does not exist

Figures now render in the Plots pane by default. To make them also appear inline in the Console, uncheck "Mute Inline Plotting" under the Plots pane options menu.
```

FILE NOT FOUND

```
with open('cities1.csv', 'r', encoding='utf8') as outp:
    next(outp, None)
    for spirt in outp:
        line = spirt.split(',')
        city1 = line[0]
        if not city1 in graph:
            graph[city1] = {}
        for i in range(1, len(line)-1, 2):
            graph[city1][line[i]] = int(line[i+1])
```

```
In [4]: runfile('/Users/eco/Desktop/submission/deneme2.py', wdir='/Users/eco/Desktop/submission')
File not found.
Istanbul does not exist
Izmir does not exist
In [5]:
```