TRƯỜNG ĐẠI HỌC NGOẠI NGỮ - TIN HỌC THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

BÀI BÁO CÁO KẾT THÚC HỌC PHẦN
TRÍ TUỆ NHÂN TẠO

TRIỂN KHAI GIẢI THUẬT UNIFORM COST SEARCH VÀ ÁP DỤNG VÀO TÌM ĐƯỜNG ĐI NGẮN NHẤT

Giảng viên hướng dẫn: PGS.TS Dương Tuấn Anh

Sinh viên thực hiện:

1. Lê Phúc Tài 21DH111630
2. Nguyễn Ngọc Anh Thiên 21DH114521
3. Huỳnh Phan Minh Lợi 21DH112645

Thành phố Hồ Chí Minh, tháng 07/2023

MỤC LỤC

DANH MỤC BẢNG

DANH MỤC HÌNH ẢNH

1. Giải thuật Uniform-cost search:

1.1. Giới thiệu giải thuật:

Tìm kiếm theo chi phí đồng nhất (Uniform-cost search) là một thuật toán tìm kiếm được sử dụng để duyệt qua một cây hoặc đồ thị có trọng số. Thuật toán này phát huy tác dụng khi có sẵn một chi phí khác nhau cho mỗi cạnh. Mục tiêu chính của Uniform-cost search là tìm đường dẫn đến nút mục tiêu có chi phí tích lũy thấp nhất. Uniform-cost search mở rộng các nút theo chi phí đường dẫn của chúng tạo thành nút gốc. Nó có thể được sử dụng để giải quyết bất kỳ biểu đồ/ cây nào có nhu cầu về chi phí tối ưu. Thuật toán Uniform-cost search được thực hiện bởi hàng đợi ưu tiên. Nó ưu tiên tối đa cho chi phí tích lũy thấp nhất. Uniform-cost search tương đương với thuật toán BFS nếu chi phí đường dẫn của tất cả các cạnh là như nhau.

Giả sử chúng ta có một bản đồ trong đó có ghi khoảng cách trên cạnh nối giữa hai thành phố có đường nối. Và chúng ta muốn tìm lối đi ngắn nhất từ một thành phố khởi đầu đến một thành phố đích. Những khoảng cách có ghi trên bản đồ cung cấp một cơ sở để đánh giá lộ trình tốt nhất muốn tìm.

Một giải thuật có sử dụng thông tin như vậy để tìm lộ trình ngắn nhất được gọi là giải thuật ***uniform-cost search***.

1.2. Tác vụ then chốt:

Để có thể áp dụng breadth-first search trong giải thuật uniform-cost search, ta điều chỉnh để nó triển khai những nút mới mà có *tổng khoảng cách ngắn nhất* từ nút khởi đầu đến những nút đó. Như vậy mỗi khi tạo ra nút kế cận M của nút N, ta nên thực hiện:

(1) kiểm tra xem M có hiện diện trong list CLOSED, nếu có thì không xét nó,

(2) tính tổng khoảng cách lối đi từ nút M đến nút khởi đầu như sau

Temp = NodeDistance(N) + ArcDistance(M, N);

(3) kiểm tra xem M có hiện diện trong list OPEN, nếu có, ta nên so sánh giá trị hiện hành của NodeDistance(M) với Temp và nếu Temp nhỏ hơn thì xóa bỏ sự hiện diện trước đó của M trong OPEN; và

(4) cho NodeDistance(M) = Temp, và đưa M vào vị trí thích hợp trong list OPEN theo giá trị tăng dần của NodeDistance.

Ghi chú: List OPEN chứa các nút đang xét và List CLOSED chứa các nút đã được xét.

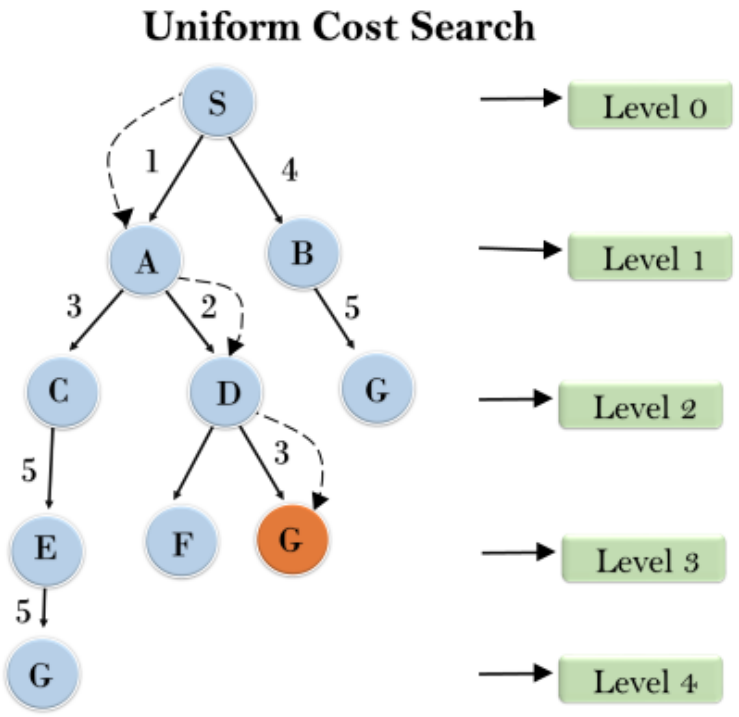
1.3. Ưu điểm:

Uniform-cost search là tối ưu vì ở mọi trạng thái, con đường có chi phí thấp nhất được chọn.

1.4. Nhược điểm:

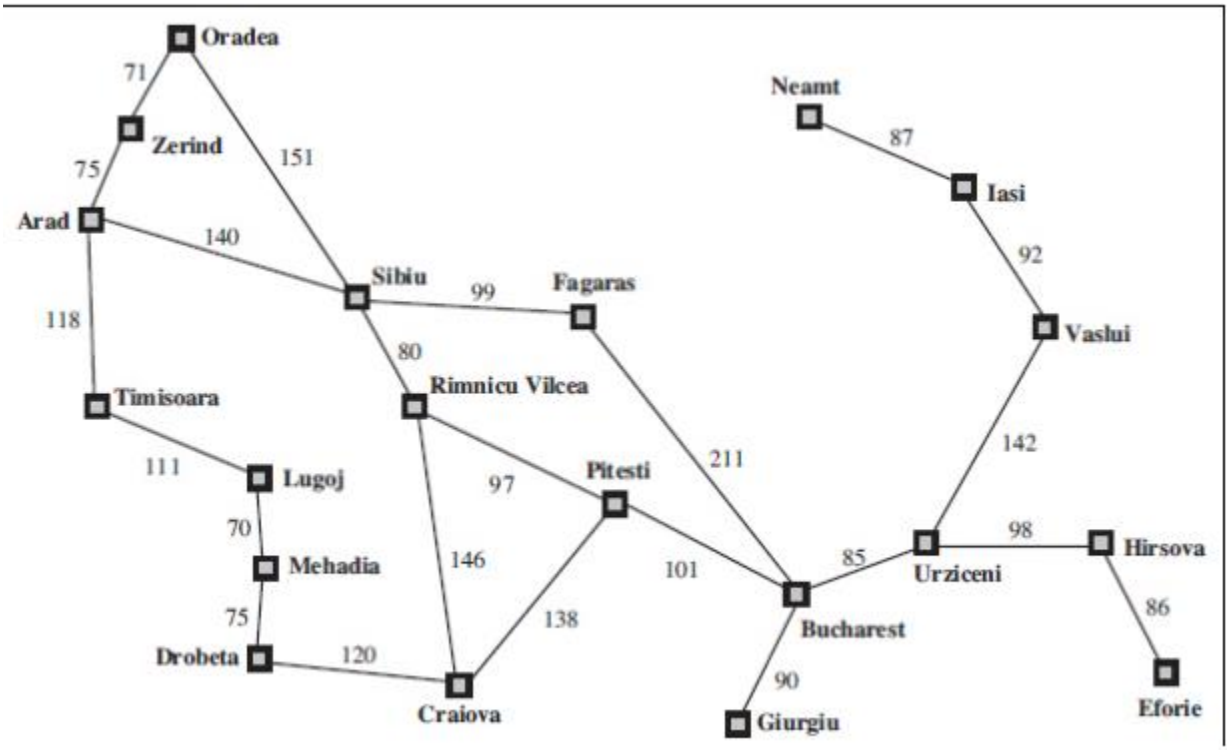
Giải thuật không quan tâm đến số bước liên quan đến việc tìm kiếm và chỉ quan tâm đến chi phí đường dẫn. Do đó thuật toán này có thể bị mắc kẹt trong một vòng lặp vô hạn.

Ví dụ về giải thuật Uniform-cost search:



Hình 1: Ví dụ về giải thuật Uniform-cost search

2. Mô tả dữ liệu:



Hình 2: Bản đồ thành phố ở Romania

Bản đồ thành phố ở Romania gồm 20 thành phố: Arad, Bucharest, Craiova, Drobeta, Eforie, Fagaras, Giurgiu, Hirsova, Iasi, Lugoj, Mehadia, Neamt, Oradea, Pitesti, Rimnicu Vilcea, Sibiu, Timisoara, Urziceni, Vaslui, Zerind.

Ta có thể biểu diễn đồ thị có trọng số bằng một tự điển có tên DISTANCE. Đây là một tự điển mà mỗi phần tử của nó lại chứa đựng một tự điển. Có một tự điển được dùng để chứa những khoảng cách từ một thành phố đến một số thành phố khác.

Các khoảng cách được ghi như sau trong tự điển:

```
DISTANCE = {}
DISTANCE['Arad'] = {'Sibiu':140,'Timisoara':118,'Zerind':75}
DISTANCE['Bucharest'] = {'Fagaras':211,'Giurgiu':90,'Pitesti':101,'Urziceni':85}
DISTANCE['Craiova'] = {'Drobeta':120,'Pitesti':138,'Rimnicu Vilcea':146}
DISTANCE['Drobeta'] = {'Craiova':120,'Mehadia':75}
DISTANCE['Eforie'] = {'Hirsova':86}
```

```
DISTANCE['Fagaras'] = {'Bucharest':211,'Sibiu':99}
DISTANCE['Giurgiu'] = {'Bucharest':90}
DISTANCE['Hirsova'] = {'Eforie':86,'Urziceni':98}
DISTANCE['Iasi'] = {'Neamt':87,'Vaslui':92}
DISTANCE['Lugoj'] = {'Mehadia':70,'Timisoara':111}
DISTANCE['Mehadia'] = {'Drobeta':75,'Lugoj':70}
DISTANCE['Neamt'] = {'Iasi':87}
DISTANCE['Oradea'] = {'Sibiu':151,'Zerind':71}
DISTANCE['Pitesti'] = {'Bucharest':101,'Craiova':138,'Rimnicu Vilcea':97}
DISTANCE['Rimnicu Vilcea'] = {'Craiova':146,'Pitesti':97,'Sibiu':80}
DISTANCE['Sibiu'] = {'Arad':140,'Fagaras':99,'Oradea':151,'Rimnicu Vilcea':80}
DISTANCE['Timisoara'] = {'Arad':118,'Lugoj':111}
DISTANCE['Urziceni'] = {'Bucharest':85,'Hirsova':98,'Vaslui':142}
DISTANCE['Vaslui'] = {'Iasi':92,'Urziceni':142}
DISTANCE['Zerind'] = {'Arad':75,'Oradea':71}
```

3. Áp dụng Uniform-cost search vào bài toán tìm đường đi ngắn nhất:

3.1. Tìm đường đi ngắn nhất từ thành phố ‘Arad’ đến ‘Bucharest’:

1. Khởi tạo: OPEN = [‘Arad’:0],
   CLOSED = []
2. OPEN = [‘Zerind’:75, ‘Timisoara’:118, ‘Sibiu’:140],
   CLOSED = [‘Arad’]
3. OPEN = [‘Timisoara’:118, ‘Sibiu’:140, ‘Oradea’: 75+71=146],
   CLOSED = [‘Arad’, ‘Zerind’]
4. OPEN = [‘Sibiu’:140, ‘Oradea’:146, ‘Lugoj’:118+111=229],
   CLOSED = [‘Arad’ , ‘Zerind’, ‘Timisoara’]
5. OPEN = [‘Oradea’:146, ‘Rimnicu Vilcea’:140+80=220, ‘Lugoj’:229,
   ‘Fagaras’:140+99=239],
   CLOSED = [‘Arad’, ‘Zerind’, ‘Timisoara’, ‘Sibiu’]

6. OPEN = [‘Rimnicu Vilcea’:220, ‘Lugoj’:229, ‘Fagaras’:239],
   CLOSED = [‘Arad’, ‘Zerind’, ‘Timisoara’, ‘Sibiu’, ‘Oradea’]
7. OPEN = [‘Lugoj’:229, ‘Fagaras’:239, ‘Pitesti’:220+97=317, ‘Craiova’:220+146=366],
   CLOSED = [‘Arad’, ‘Zerind’, ‘Timisoara’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’]
8. OPEN = [‘Fagaras’:239, ‘Mehadia’=229+70=299, ‘Pitesti’:317, ‘Craiova’:366],
   CLOSED = [‘Arad’, ‘Zerind’, ‘Timisoara’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’, ‘Lugoj’]
9. OPEN = [‘Mehadia’:299, ‘Pitesti’:317, ‘Craiova’:366, ‘Bucharest’:239+211=450],
   CLOSED = [‘Arad’, ‘Zerind’, ‘Timisoara’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’, ‘Lugoj’, ‘Fagaras’]
10. OPEN = [‘Pitesti’:317, ‘Craiova’:366, ‘Drobeta’:299+75=374, ‘Bucharest’:450],
    CLOSED = [‘Arad’, ‘Zerind’, ‘Timisoara’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’, ‘Lugoj’, ‘Fagaras’, ‘Mehadia’]
11. OPEN = [‘Craiova’:366, ‘Drobeta’:374, ‘Bucharest’:317+101=418],
    CLOSED = [‘Arad’, ‘Zerind’, ‘Timisoara’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’, ‘Lugoj’, ‘Fagaras’, ‘Mehadia’, ‘Pitesti’]
12. OPEN = [‘Drobeta’:374, ‘Bucharest’:418],
    CLOSED = [‘Arad’, ‘Zerind’, ‘Timisoara’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’, ‘Lugoj’, ‘Fagaras’, ‘Mehadia’, ‘Pitesti’, ‘Craiova’]
13. OPEN = [‘Bucharest’:418],
    CLOSED = [‘Arad’, ‘Zerind’, ‘Timisoara’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’, ‘Lugoj’, ‘Fagaras’, ‘Mehadia’, ‘Pitesti’, ‘Craiova’, ‘Drobeta’]

Lối đi tìm thấy: [‘Arad’, ‘Sibiu’, ‘Rimnicu Vilcea’, ‘Pitesti’, ‘Bucharest’]

Tổng số nút được xét là: 13

Tổng khoảng cách lối đi là: 418

3.2. Tìm đường đi ngắn nhất từ thành phố ‘Rimnicu Vilcea’ đến ‘Urziceni’:

1. Khởi tạo: OPEN = [‘Rimnicu Vilcea’:0],
   CLOSED = []
2. OPEN = [‘Sibiu’:80, ‘Pitesti’:97, ‘Craiova’:146],
   CLOSED = [‘Rimnicu Vilcea’]
3. OPEN = [‘Pitesti’:97, ‘Craiova’:146, ‘Fagaras’:80+99=179,
   ‘Arad’:80+140=220, ‘Oradea’:80+151=231],
   CLOSED = [‘Rimnicu Vilcea’, ‘Sibiu’]
4. OPEN = [‘Craiova’:146, ‘Fagaras’:179, ‘Bucharest’:97+101=198,
   ‘Arad’:220, ‘Oradea’:231],
   CLOSED = [‘Rimnicu Vilcea’, ‘Sibiu’, ‘Pitesti’]
5. OPEN = [‘Fagaras’:179, ‘Bucharest’:198, ‘Arad’:220, ‘Oradea’:231,
   ‘Drobeta’:146+120=266],
   CLOSED = [‘Rimnicu Vilcea’, ‘Sibiu’, ‘Pitesti’, ‘Craiova’]
6. OPEN = [‘Bucharest’:198, ‘Arad’:220, ‘Oradea’:231, ‘Drobeta’:266],
   CLOSED = [‘Rimnicu Vilcea’, ‘Sibiu’, ‘Pitesti’, ‘Craiova’, ‘Fagaras’]
7. OPEN = [‘Arad’:220, ‘Oradea’:231, ‘Drobeta’:266, ‘Urziceni’:198+85=283,
   ‘Giurgiu’:198+90=288],
   CLOSED = [‘Rimnicu Vilcea’, ‘Sibiu’, ‘Pitesti’, ‘Craiova’, ‘Fagaras’,
   ‘Bucharest’]
8. OPEN = [‘Oradea’:231, ‘Drobeta’:266, ‘Urziceni’:283, ‘Giurgiu’:288,
   ‘Zerind’:220+75=295, ‘Timisoara’:220+118=338],
   CLOSED = [‘Rimnicu Vilcea’, ‘Sibiu’, ‘Pitesti’, ‘Craiova’, ‘Fagaras’,
   ‘Bucharest’, ‘Arad’]
9. OPEN = [‘Drobeta’:266, ‘Urziceni’:283, ‘Giurgiu’:288, ‘Zerind’:295,
   ‘Timisoara’:338],
   CLOSED = [‘Rimnicu Vilcea’, ‘Sibiu’, ‘Pitesti’, ‘Craiova’, ‘Fagaras’,
   ‘Bucharest’, ‘Arad’, ‘Oradea’]

10. OPEN = [‘Urziceni’:283, ‘Giurgiu’:288, ‘Zerind’:295, ‘Timisoara’:338, ‘Mehadia’:266+75=341],
    CLOSED = [‘Rimnicu Vilcea’, ‘Sibiu’, ‘Pitesti’, ‘Craiova’, ‘Fagaras’, ‘Bucharest’, ‘Arad’, ‘Oradea’, ‘Drobeta’]

Lối đi tìm thấy: [‘Rimnicu Vilcea’, ‘Pitesti’, ‘Bucharest’, ‘Urziceni’]

Tổng số nút được xét là: 10

Tổng khoảng cách lối đi là: 283

3.3. Tìm đường đi ngắn nhất từ thành phố ‘Timisoara’ đến ‘Giurgiu’:

1. Khởi tạo: OPEN = [‘Timisoara’:0],
   CLOSED = []
2. OPEN = [‘Lugoj’:111, ‘Arad’:118],
   CLOSED = [‘Timisoara’]
3. OPEN = [‘Arad’:118, ‘Mehadia’:111+70=181],
   CLOSED = [‘Timisoara’, ‘Lugoj’]
4. OPEN = [‘Mehadia’:181, ‘Zerind’:118+75=193, ‘Sibiu’:118+140=258],
   CLOSED = [‘Timisoara’, ‘Lugoj’, ‘Arad’]
5. OPEN = [‘Zerind’:193, ‘Drobeta’:181+75=256 , ‘Sibiu’:258],
   CLOSED = [‘Timisoara’, ‘Lugoj’, ‘Arad’, ‘Mehadia’]
6. OPEN = [‘Drobeta’:256 , ‘Sibiu’:258, ‘Oradea’:193+71=264],
   CLOSED = [‘Timisoara’, ‘Lugoj’, ‘Arad’, ‘Mehadia’, ‘Zerind’]
7. OPEN = [‘Sibiu’:258, ‘Oradea’:264, ‘Craiova’:256+120=376],
   CLOSED = [‘Timisoara’, ‘Lugoj’, ‘Arad’, ‘Mehadia’, ‘Zerind’, ‘Drobeta’]
8. OPEN = [‘Oradea’:264, ‘Rimnicu Vilcea’:258+80=338, ‘Fagaras’:258+99=357, ‘Craiova’:376],
   CLOSED = [‘Timisoara’, ‘Lugoj’, ‘Arad’, ‘Mehadia’, ‘Zerind’, ‘Drobeta’, ‘Sibiu’]
9. OPEN = [‘Rimnicu Vilcea’:338, ‘Fagaras’:357, ‘Craiova’:376],
   CLOSED = [‘Timisoara’, ‘Lugoj’, ‘Arad’, ‘Mehadia’, ‘Zerind’, ‘Drobeta’, ‘Sibiu’, ‘Oradea’]

10. OPEN = [‘Fagaras’:357, ‘Craiova’:376, ‘Pitesti’:338+97=435],
    CLOSED = [‘Timisoara’, ‘Lugoj’, ‘Arad’, ‘Mehadia’, ‘Zerind’, ‘Drobeta’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’]
11. OPEN = [‘Craiova’:376, ‘Pitesti’:435, ‘Bucharest’:357+211=568],
    CLOSED = [‘Timisoara’, ‘Lugoj’, ‘Arad’, ‘Mehadia’, ‘Zerind’, ‘Drobeta’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’, ‘Fagaras’]
12. OPEN = [‘Pitesti’:435, ‘Bucharest’:568],
    CLOSED = [‘Timisoara’, ‘Lugoj’, ‘Arad’, ‘Mehadia’, ‘Zerind’, ‘Drobeta’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’, ‘Fagaras’, ‘Craiova’]
13. OPEN = [‘Bucharest’:435+101=536],
    CLOSED = [‘Timisoara’, ‘Lugoj’, ‘Arad’, ‘Mehadia’, ‘Zerind’, ‘Drobeta’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’, ‘Fagaras’, ‘Craiova’, ‘Pitesti’]
14. OPEN = [‘Urziceni’:536+85=621, ‘Giurgiu’:536+90=626],
    CLOSED = [‘Timisoara’, ‘Lugoj’, ‘Arad’, ‘Mehadia’, ‘Zerind’, ‘Drobeta’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’, ‘Fagaras’, ‘Craiova’, ‘Pitesti’, ‘Bucharest’]
15. OPEN = [‘Giurgiu’:626, ‘Hirsova’:621+98=719, ‘Vaslui’:621+142=763],
    CLOSED = [‘Timisoara’, ‘Lugoj’, ‘Arad’, ‘Mehadia’, ‘Zerind’, ‘Drobeta’, ‘Sibiu’, ‘Oradea’, ‘Rimnicu Vilcea’, ‘Fagaras’, ‘Craiova’, ‘Pitesti’, ‘Bucharest’, ‘Urziceni’]

Lối đi tìm thấy: [‘Timisoara’, ‘Arad’, ‘Sibiu’, ‘Rimnicu Vilcea’, ‘Pitesti’, ‘Bucharest’, ‘Giurgiu’]

Tổng số nút được xét là: 15

Tổng khoảng cách lối đi là: 626

4. Mã nguồn:

4.1. Khởi tạo và lưu khoảng cách vào tự điển:

```
DISTANCE = {}
DISTANCE['Arad'] = {'Sibiu':140,'Timisoara':118,'Zerind':75}
DISTANCE['Bucharest'] = {'Fagaras':211,'Giurgiu':90,'Pitesti':101,'Urziceni':85}
DISTANCE['Craiova'] = {'Drobeta':120,'Pitesti':138,'Rimnicu Vilcea':146}
DISTANCE['Drobeta'] = {'Craiova':120,'Mehadia':75}
DISTANCE['Eforie'] = {'Hirsova':86}
DISTANCE['Fagaras'] = {'Bucharest':211,'Sibiu':99}
DISTANCE['Giurgiu'] = {'Bucharest':90}
DISTANCE['Hirsova'] = {'Eforie':86,'Urziceni':98}
DISTANCE['Iasi'] = {'Neamt':87,'Vaslui':92}
DISTANCE['Lugoj'] = {'Mehadia':70,'Timisoara':111}
DISTANCE['Mehadia'] = {'Drobeta':75,'Lugoj':70}
DISTANCE['Neamt'] = {'Iasi':87}
DISTANCE['Oradea'] = {'Sibiu':151,'Zerind':71}
DISTANCE['Pitesti'] = {'Bucharest':101,'Craiova':138,'Rimnicu Vilcea':97}
DISTANCE['Rimnicu Vilcea'] = {'Craiova':146,'Pitesti':97,'Sibiu':80}
DISTANCE['Sibiu'] = {'Arad':140,'Fagaras':99,'Oradea':151,'Rimnicu Vilcea':80}
DISTANCE['Timisoara'] = {'Arad':118,'Lugoj':111}
DISTANCE['Urziceni'] = {'Bucharest':85,'Hirsova':98,'Vaslui':142}
DISTANCE['Vaslui'] = {'Iasi':92,'Urziceni':142}
DISTANCE['Zerind'] = {'Arad':75,'Oradea':71}
```

Hình 3: Khởi tạo và lưu khoảng cách vào tự điển

4.2. Hàm mô tả giải thuật Uniform-cost search:

```
def unifcost(start, goal):
    OPEN = [start]
    CLOSED = []
    PREDECESSOR[start] = None
    FVALUE[start] = 0
    while OPEN != []:
        n = deleteMin(OPEN, FVALUE)
        CLOSED.append(n)
        global COUNT
        COUNT += 1
        if n == goal:
            return extractPath(n)
        lst = successors(n)
        lst = listDifference(lst, CLOSED)
        for elt in lst:
            temp = f(elt, n)
            if elt in OPEN:
                if temp < FVALUE[elt]:
                    FVALUE[elt] = temp
                    PREDECESSOR[elt] = n
                    OPEN.remove(elt)
                    OPEN = insert(elt, OPEN)
            else:
                if not elt in CLOSED:
                    FVALUE[elt] = temp
                    PREDECESSOR[elt] = n
                    OPEN = insert(elt, OPEN)
```

Hình 4: Hàm unifcost(start, goal)

Hàm ‘unifcost’ nhận hai tham số đầu vào là ‘start’ và ‘goal’, đại diện cho thành phố khởi đầu và thành phố kết thúc của bài toán tìm kiếm. Giải thuật tìm kiếm đường đi tối ưu từ trạng thái bắt đầu (start) đến trạng thái kết thúc (goal) trong một bài toán tìm kiếm.

4.3. Các hàm hỗ trợ:

4.3.1. Hàm successors(city)

```
def successors(city):
    return DISTANCE[city]
```

Hình 5: Hàm successors(city)

Hàm ‘successors(city)’ dùng để trả về danh sách các thành phố kế cận của ‘city’.

4.3.2. Hàm listDifference(list1, list2):

```
def listDifference(list1, list2):
    list3 = list(list1)
    for elt in list2:
        if elt in list3:
            list3.remove(elt)
    return list3
```

Hình 6: Hàm listDifference(list1, list2)

Hàm 'listDifference(list1, list2)' trả về một bản sao của list1 trong đó các phần tử có xuất hiện trong list2 bị loại bỏ.

4.3.3. Hàm extractPath(n):

```
def extractPath(n):
    if PREDECESSOR[n]==None: return [n]
    return extractPath(PREDECESSOR[n]) + [n]
```

Hình 7: Hàm extractPath(n)

Hàm 'extractPath(n)' trả về đường đi tìm thấy của giải thuật tìm kiếm Uniform-cost search.

4.3.4. Hàm f(m, n):

```
def f(m, n):
    return FVALUE[n] + DISTANCE[m][n]
```

Hình 8: Hàm f(m, n)

Hàm 'f(m, n)' dùng để tính khoảng cách từ nút khởi đầu đến nút m.

4.3.5. Hàm deleteMin(lst, fn):

```
def deleteMin(lst, fn):
    minVal = 9999
    minElt = None
    for e in lst:
        temp = FVALUE[e]
        if temp < minVal: minVal = temp; minElt = e
    lst.remove(minElt)
    return minElt
```

Hình 9: Hàm deleteMin(lst, fn)

Hàm *deleteMin* để xác định phần tử "tốt nhất" dựa vào một hàm đánh giá. Hàm sẽ xóa bỏ phần tử này ra khỏi danh sách và trả nó về trình gọi.

4.3.6. Hàm insert(elt, lst):

```
def insert(elt, lst):
    if lst == []:
        return [elt]
    if FVALUE[elt] < FVALUE[lst[0]]:
        return [elt] + lst
    else:
        return [lst[0]] + insert(elt, lst[1:])
```

Hình 10: Hàm insert(elt, lst)

Hàm insert(elt, lst) để chèn một phần tử mới vào danh sách dựa theo giá trị 'FVALUE' của nó.

4.4. Hàm test:

4.4.1. Hàm test1(): Đường đi ngắn nhất từ 'Arad' đến 'Bucharest'

```
def test1():
    start = 'Arad'
    goal = 'Bucharest'
    print(f"Đường đi ngắn nhất từ thành phố {start} đến {goal} là:")
    path = unifcost(start, goal)
    print (path)
    global COUNT
    print('Tổng số nút được xét là: ' + str(COUNT))
    print('Tổng khoảng cách lối đi là: ' + str(FVALUE[goal]))
```

Hình 11: Hàm test1()

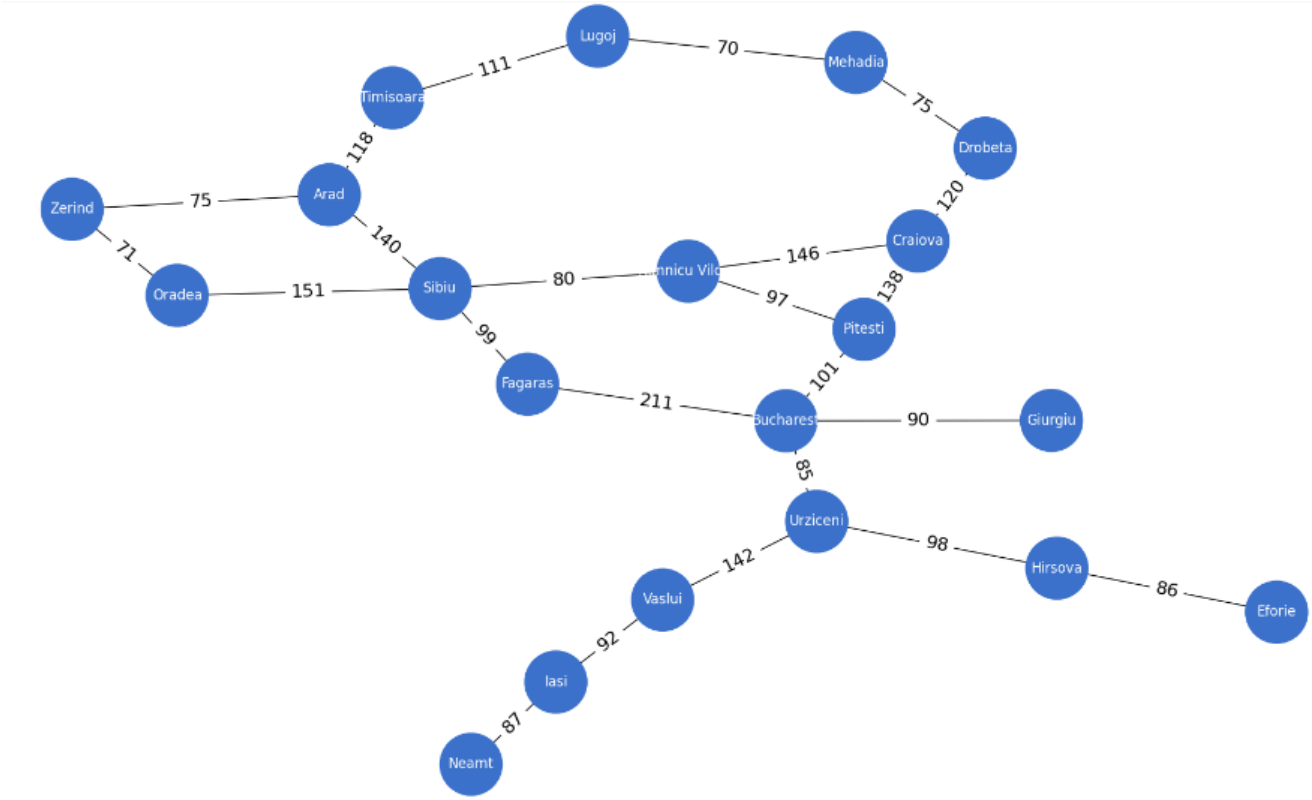
→ Kết quả khi gọi hàm test1():

```
Đường đi ngắn nhất từ thành phố Arad đến Bucharest là:
['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
Tổng số nút được xét là: 13
Tổng khoảng cách lối đi là: 418
```

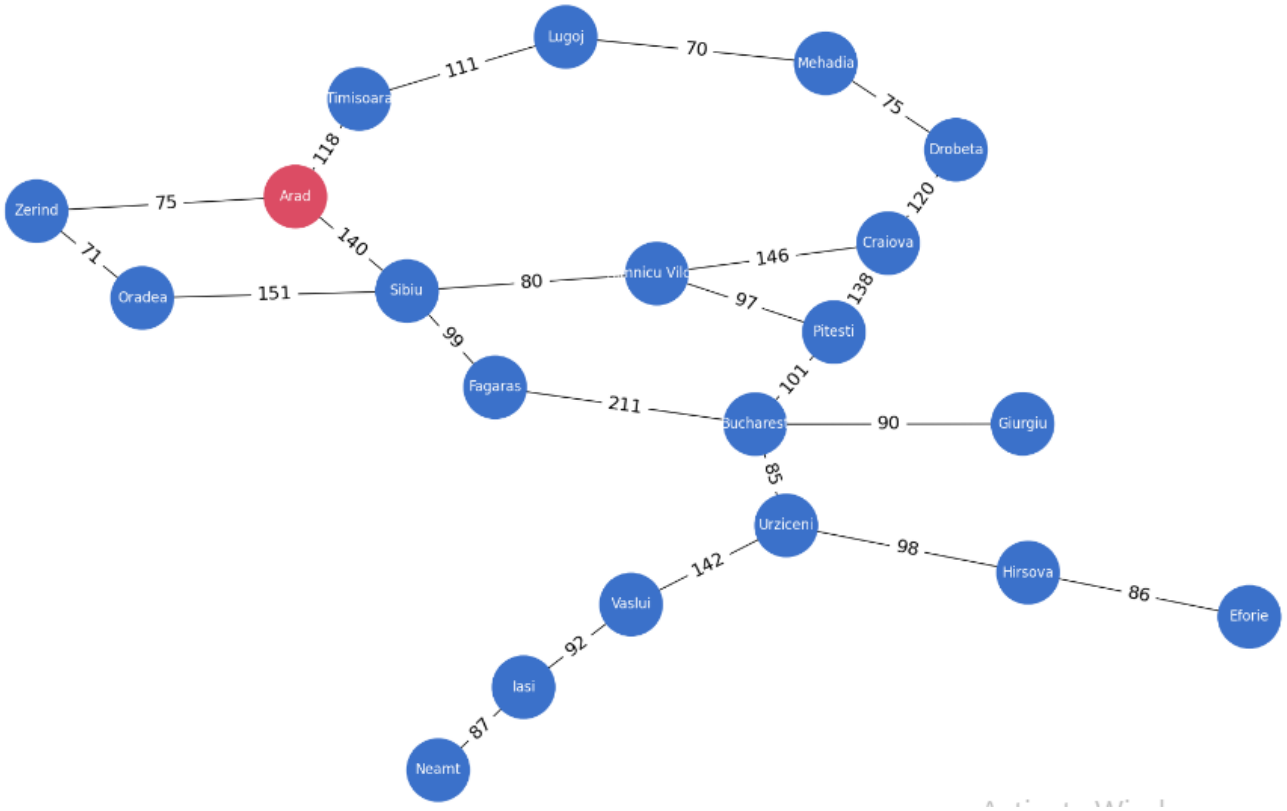
Hình 12: Kết quả hàm test1()

Minh họa các bước tìm đường đi ngắn nhất từ thành phố 'Arad' đến 'Bucharest':

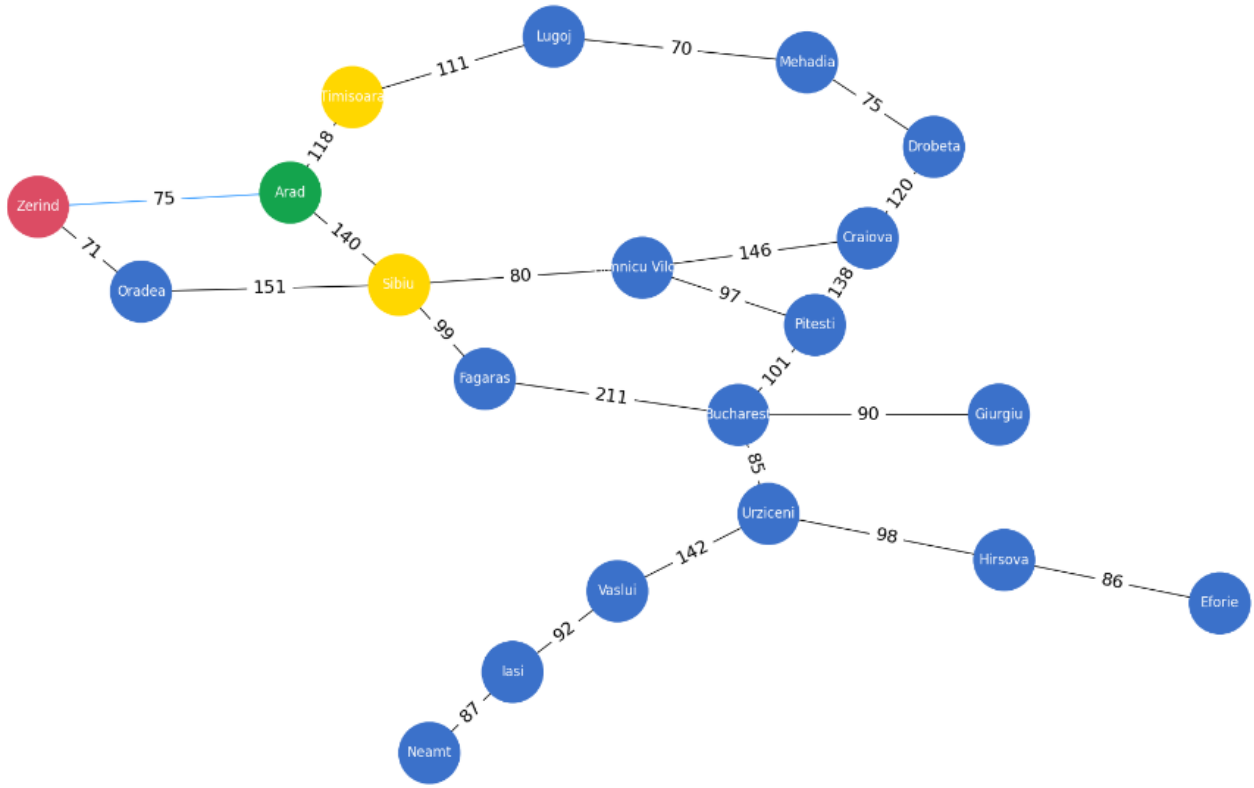
Chú thích:
- : Thành phố đang xét
- : Thành phố trong hàng đợi
- : Thành phố đã xét
- : Thành phố chưa xét



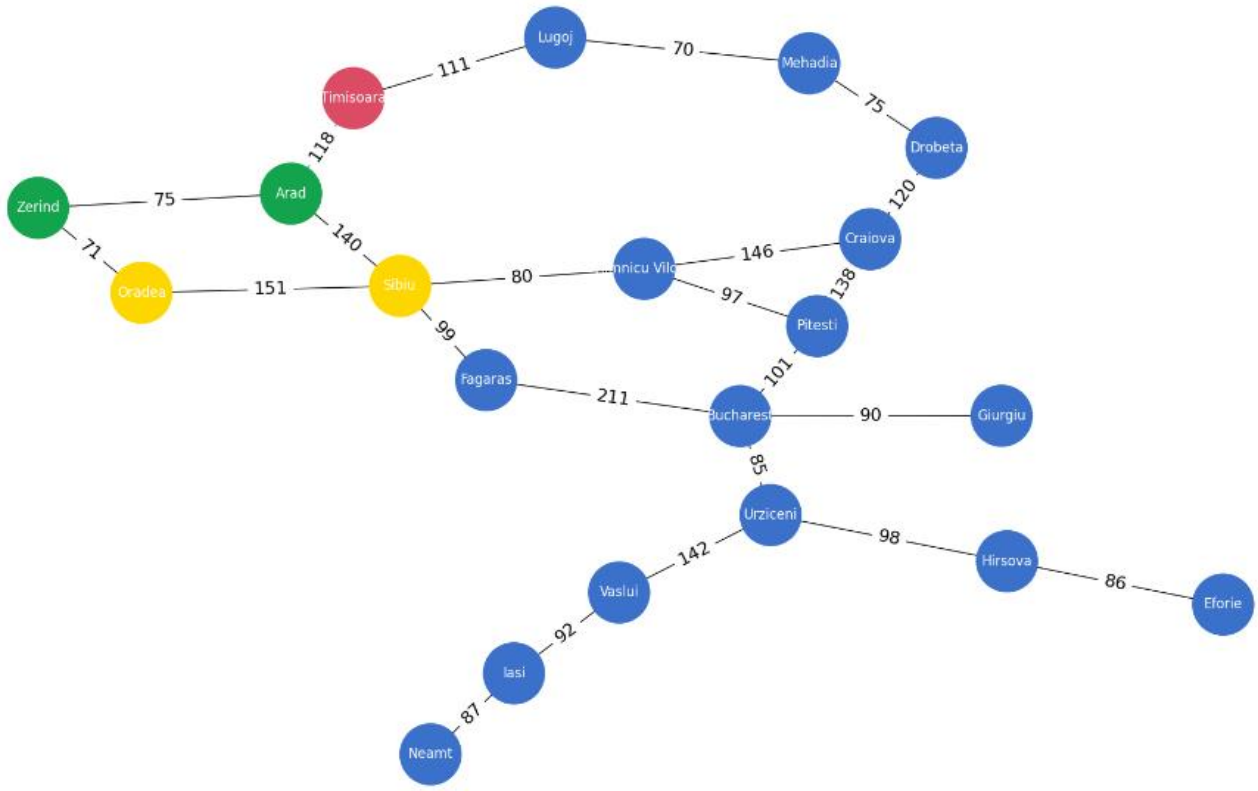
Hình 13: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(1)



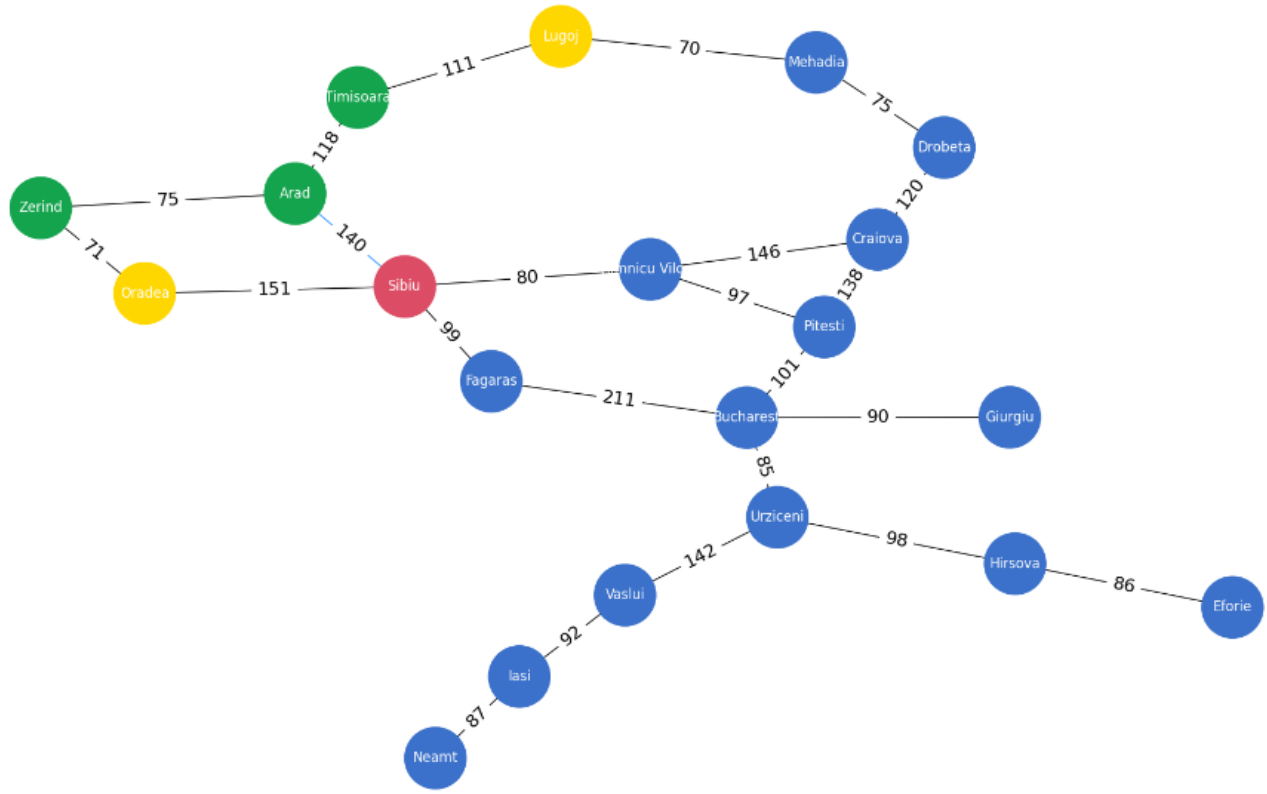
Hình 14: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(2)



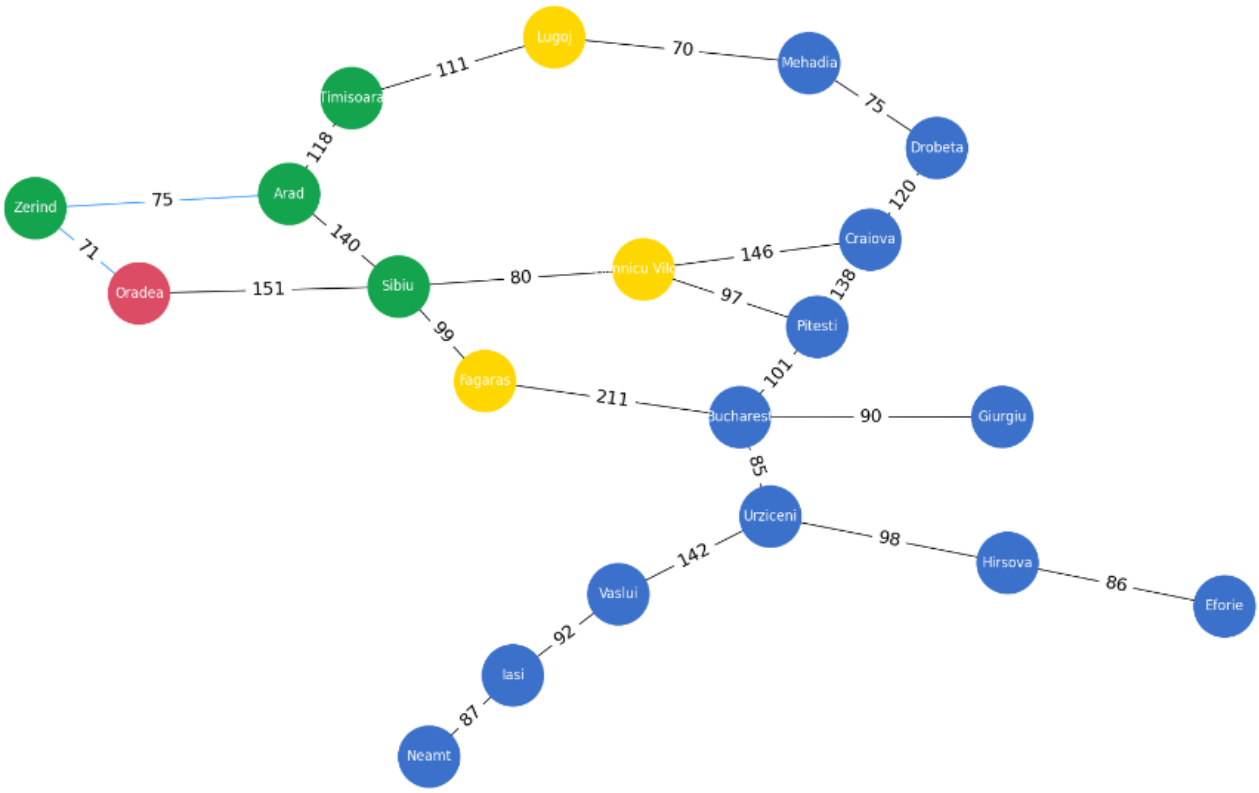
Hình 15: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(3)



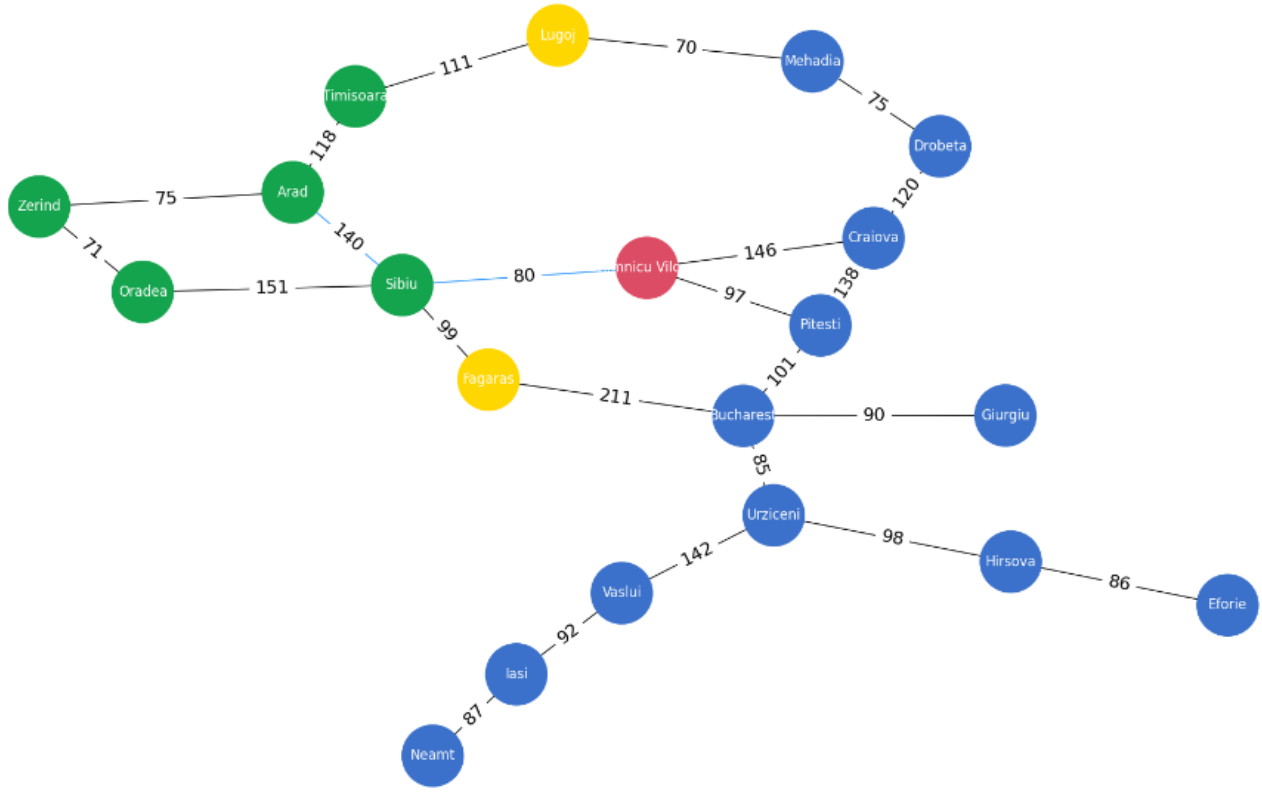
Hình 16: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(4)



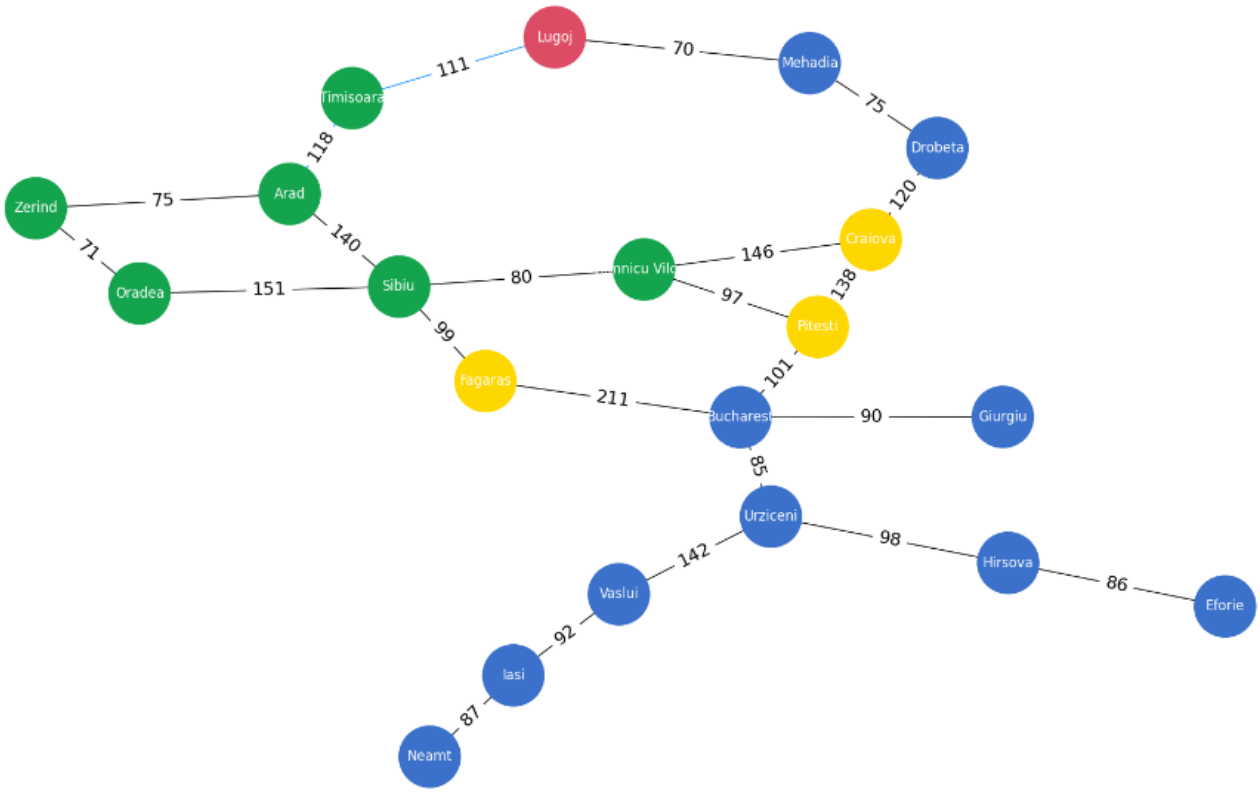
Hình 17: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(5)



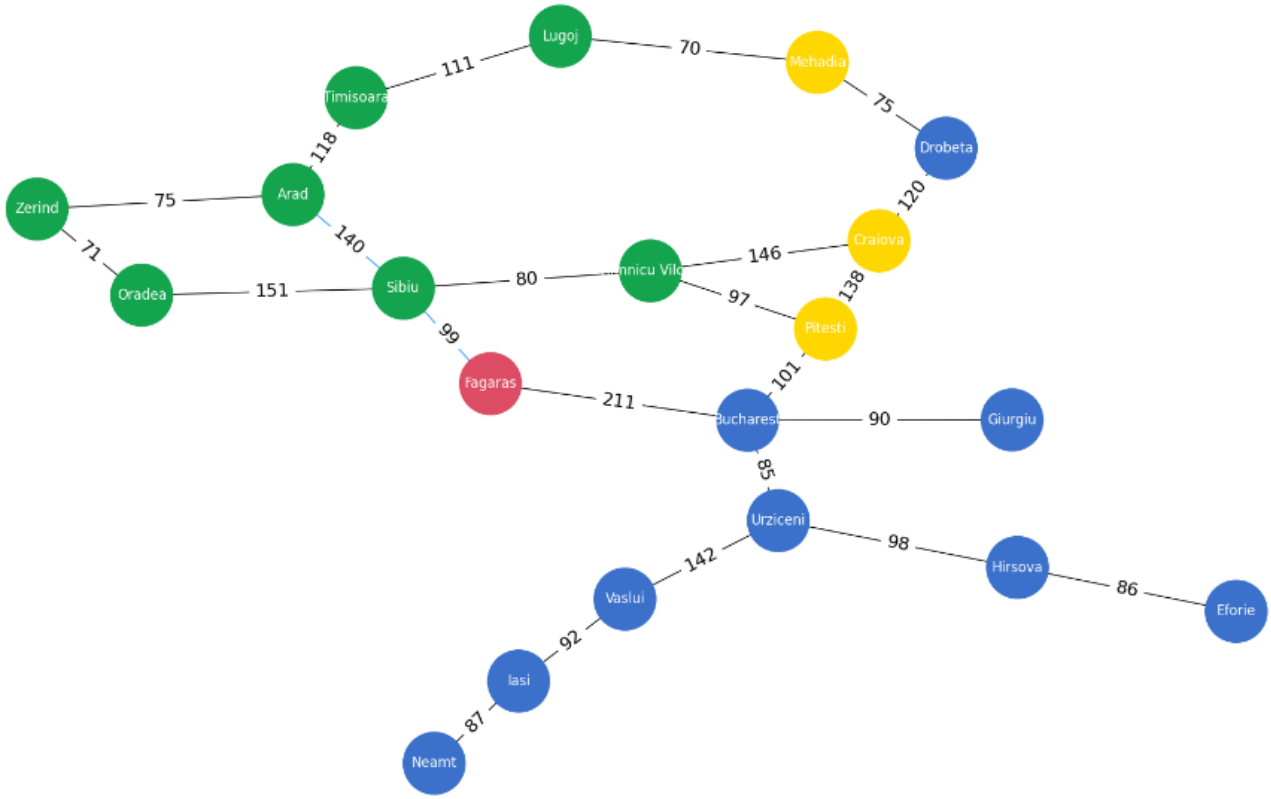
Hình 18: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(6)



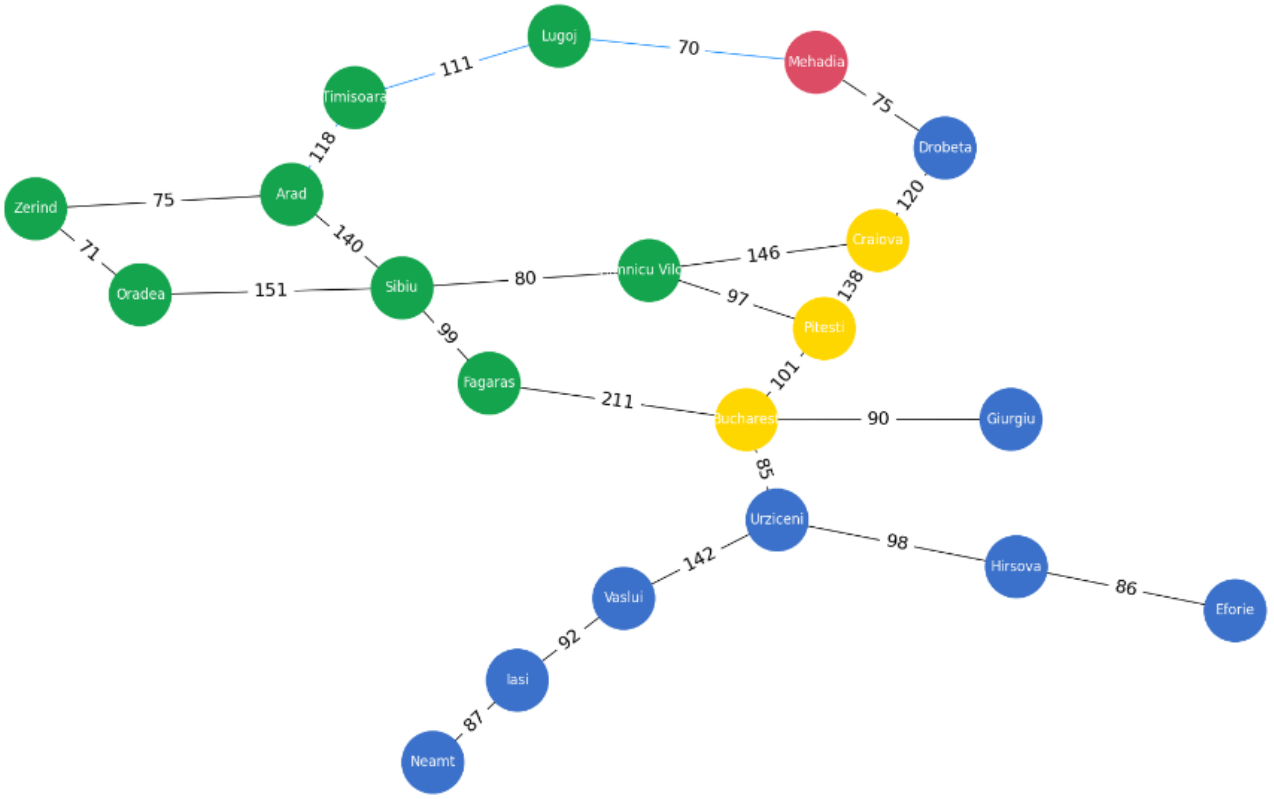
Hình 19: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(7)



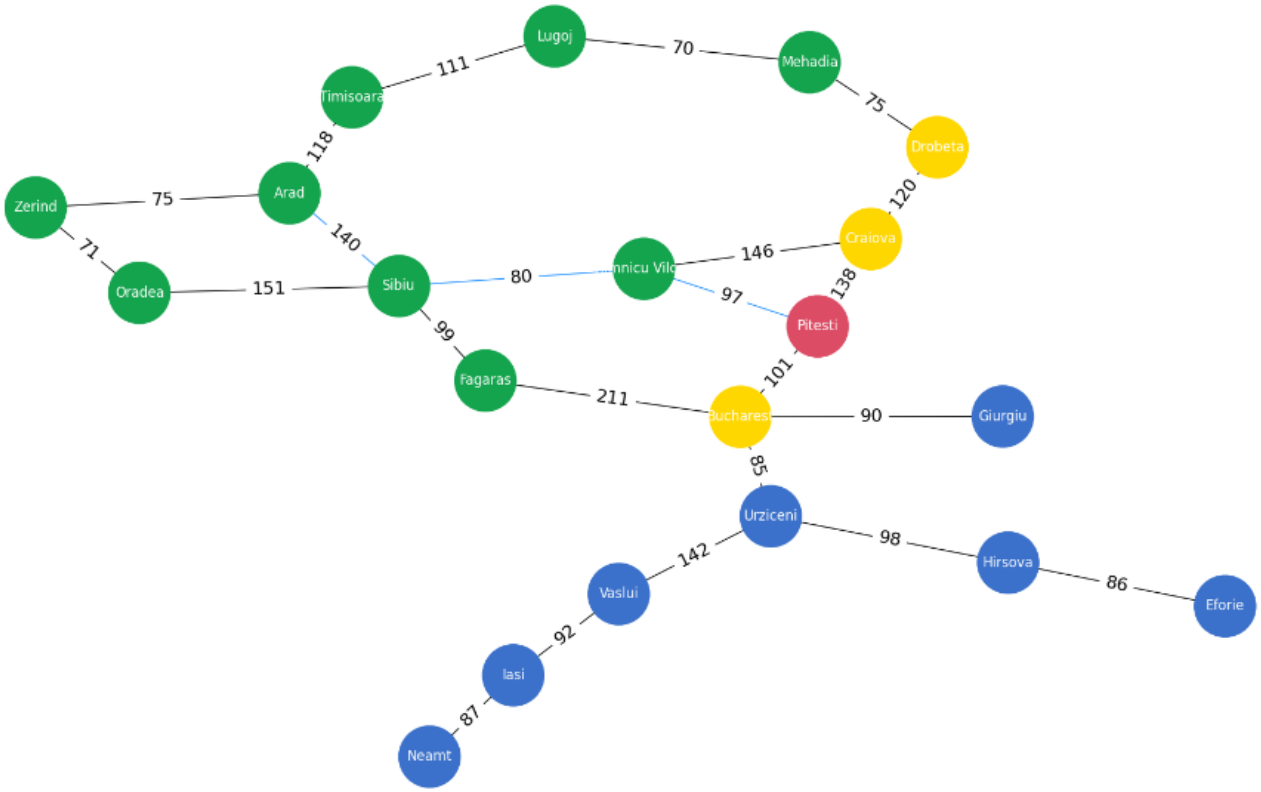
Hình 20: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(8)



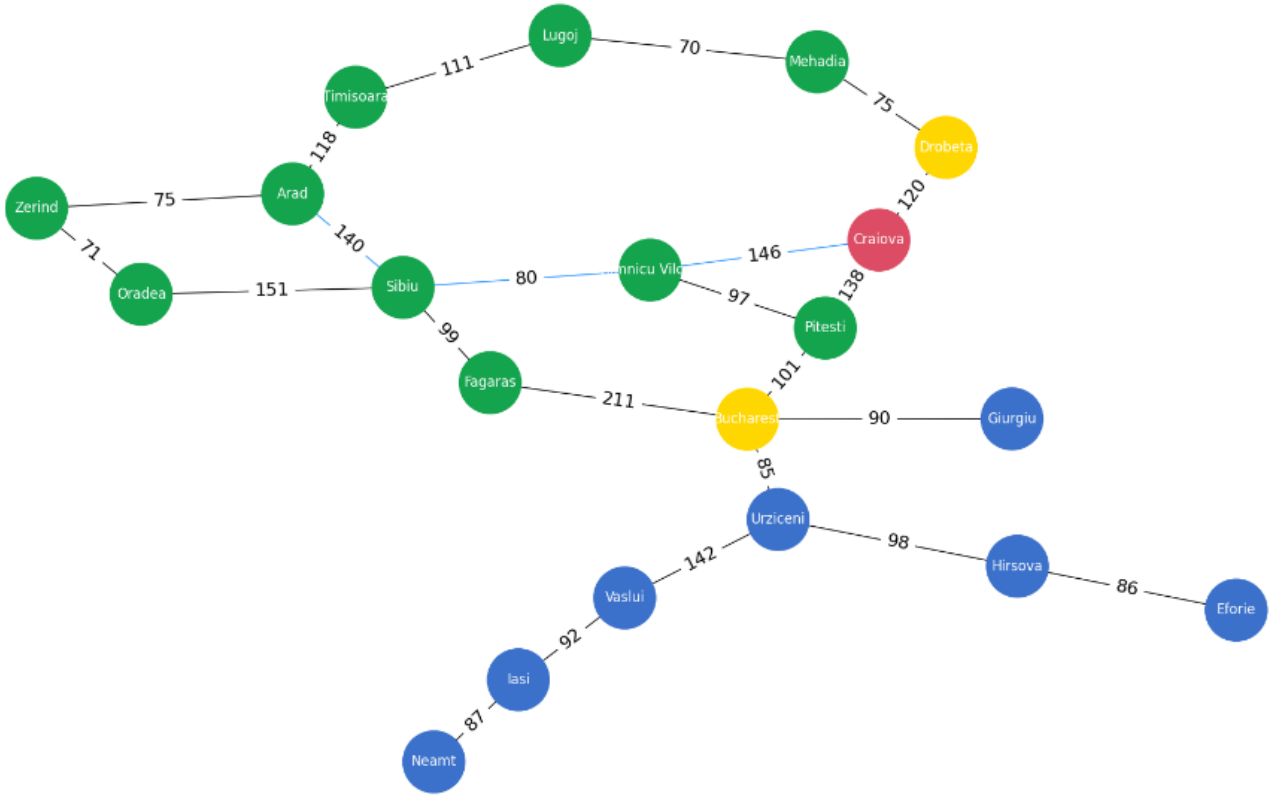
Hình 21: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(9)



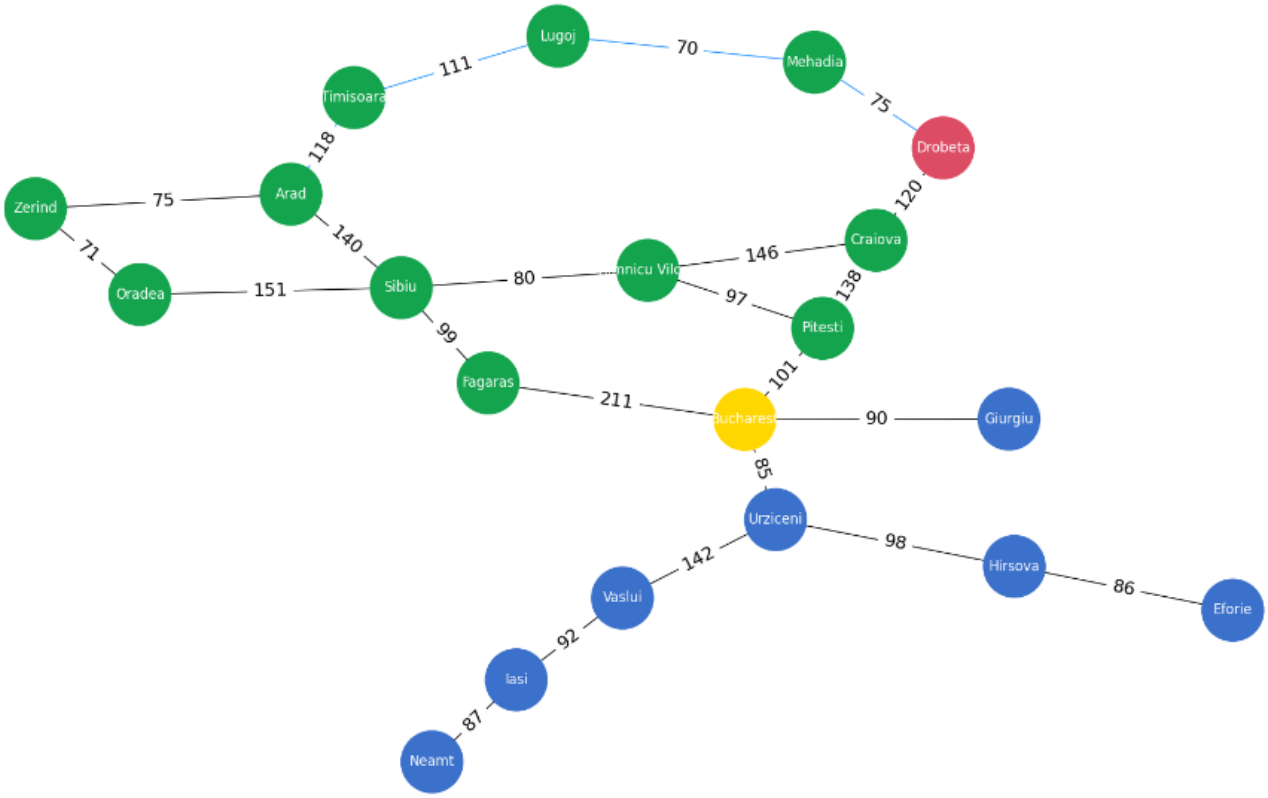
Hình 22: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(10)



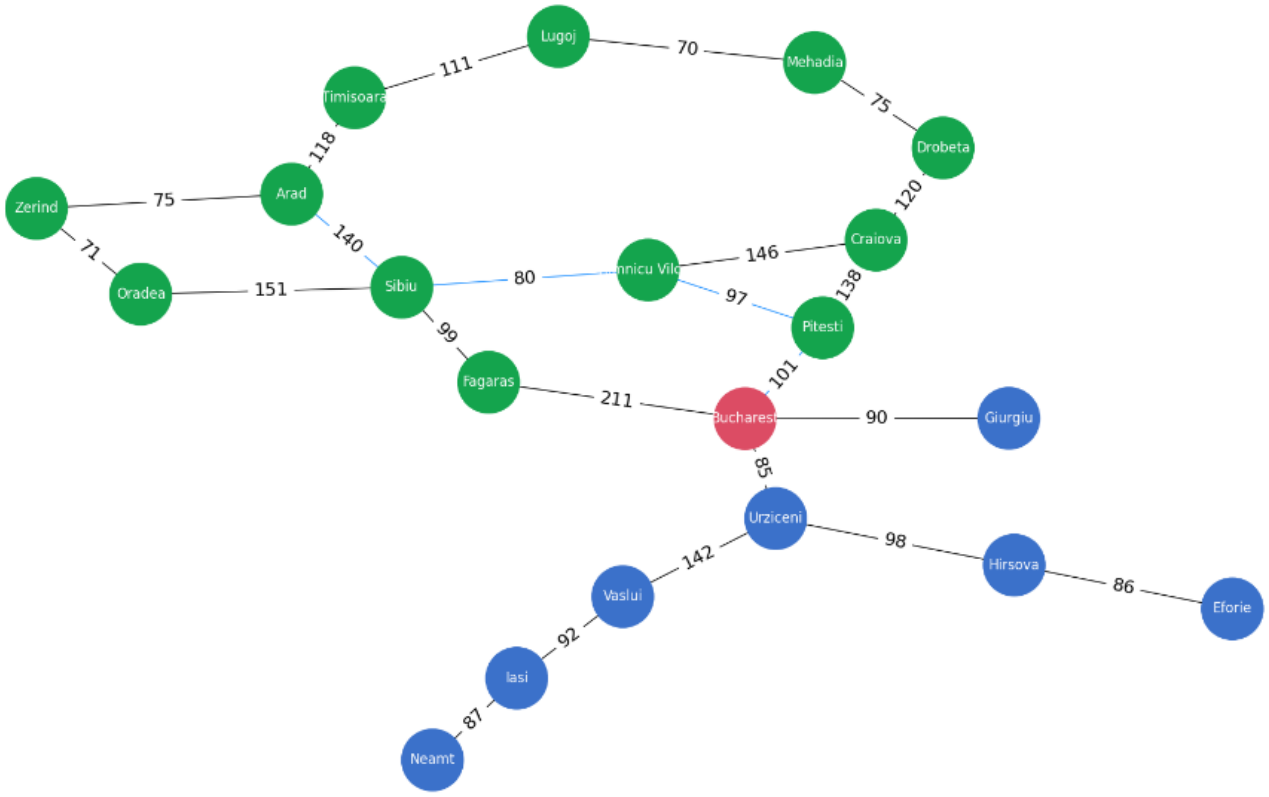
Hình 23: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(11)



Hình 24: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(12)



Hình 25: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(13)



Hình 26: Minh họa tìm đường đi ngắn nhất từ 'Arad' đến 'Bucharest'(14)

4.4.2. Hàm test2(): Đường đi ngắn nhất từ ‘Rimnicu Vilcea’ đến ‘Urziceni’

```
def test2():
    start = 'Rimnicu Vilcea'
    goal = 'Urziceni'
    print(f"Đường đi ngắn nhất từ thành phố {start} đến {goal} là:")
    path = unifcost(start, goal)
    print (path)
    global COUNT
    print('Tổng số nút được xét là: ' + str(COUNT))
    print('Tổng khoảng cách lối đi là: ' + str(FVALUE[goal]))
```

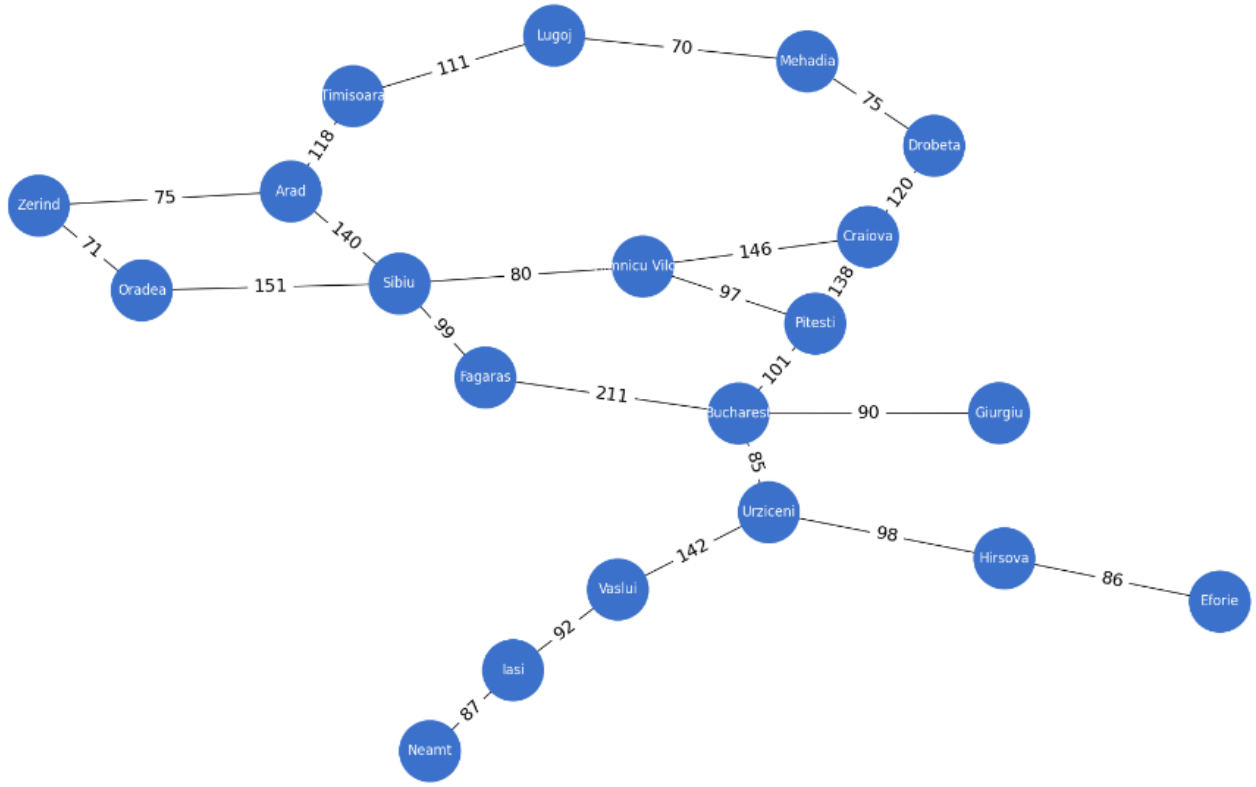
Hình 27: Hàm test2()

→ Kết quả khi gọi hàm test2():

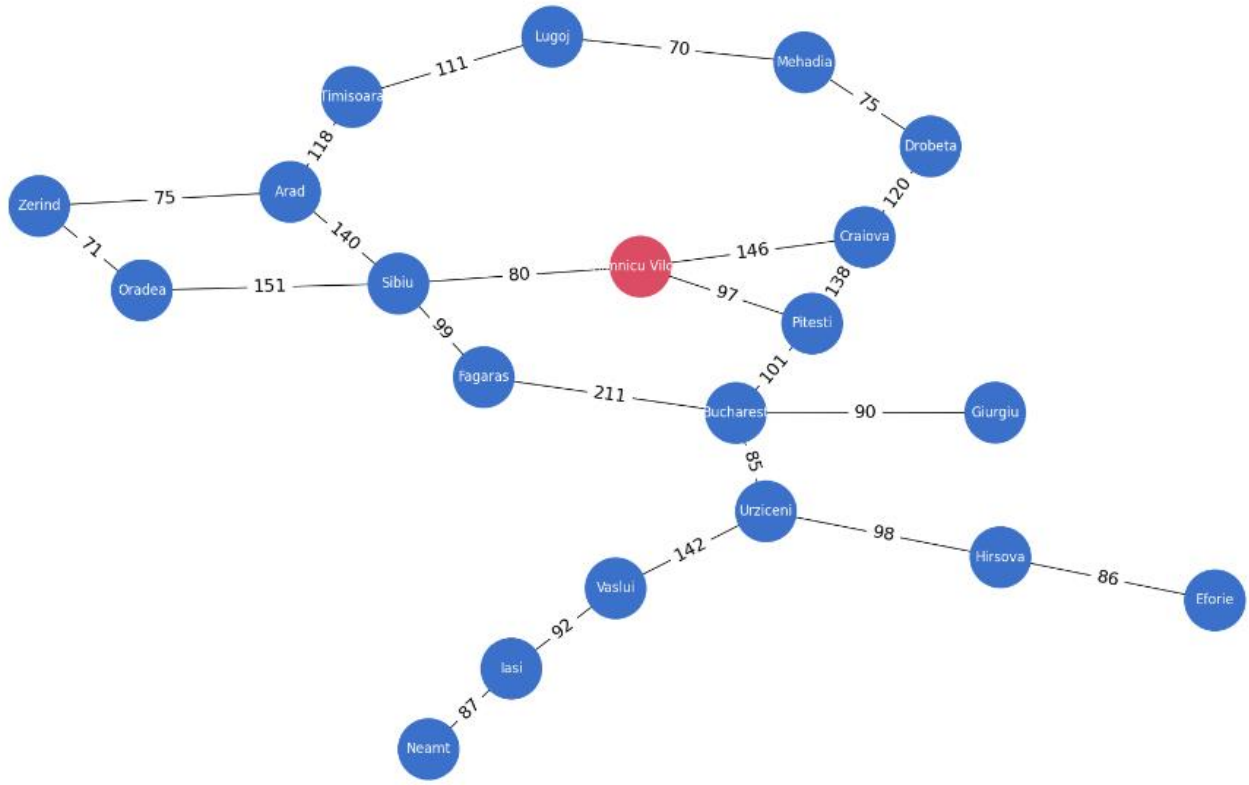
```
Đường đi ngắn nhất từ thành phố Rimnicu Vilcea đến Urziceni là:
['Rimnicu Vilcea', 'Pitesti', 'Bucharest', 'Urziceni']
Tổng số nút được xét là: 10
Tổng khoảng cách lối đi là: 283
```

Hình 28: Kết quả hàm test2()

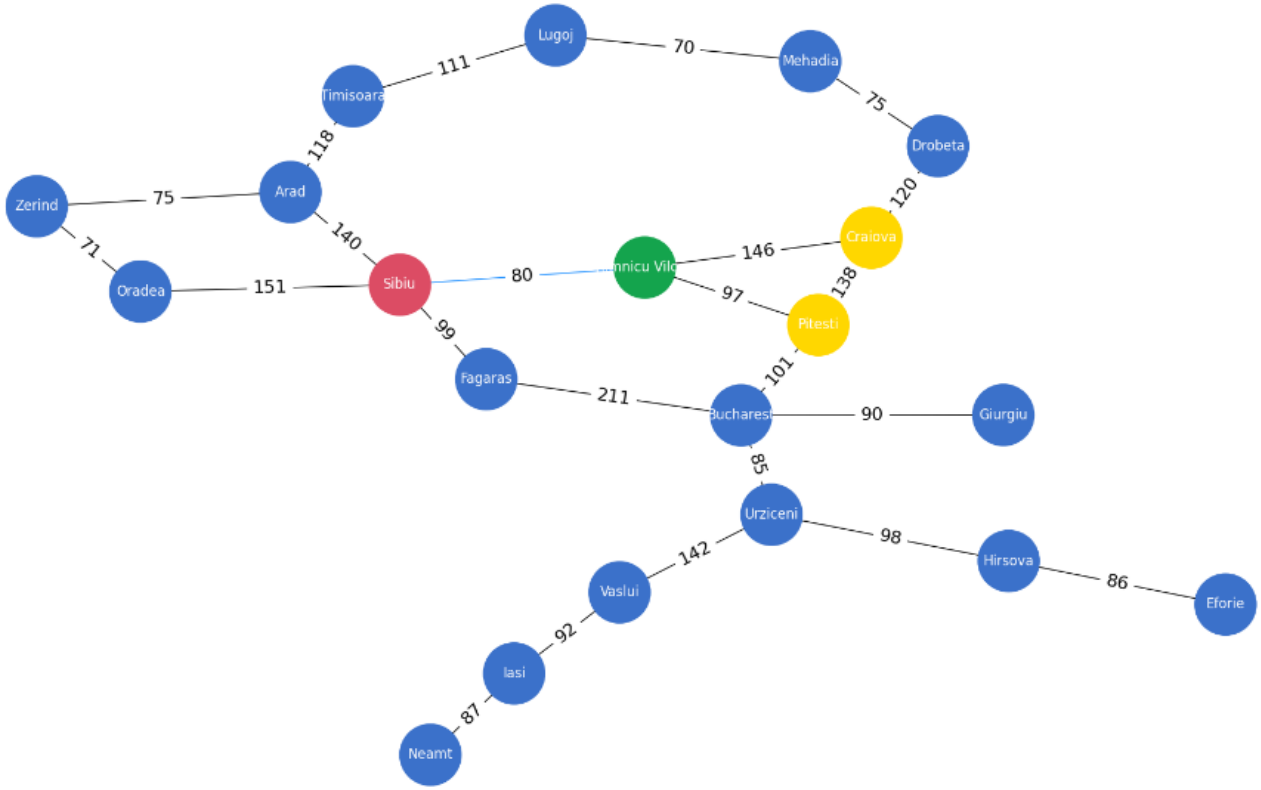
Minh họa các bước tìm đường đi ngắn nhất từ thành phố ‘Rimnicu Vilcea’ đến ‘Urziceni’:



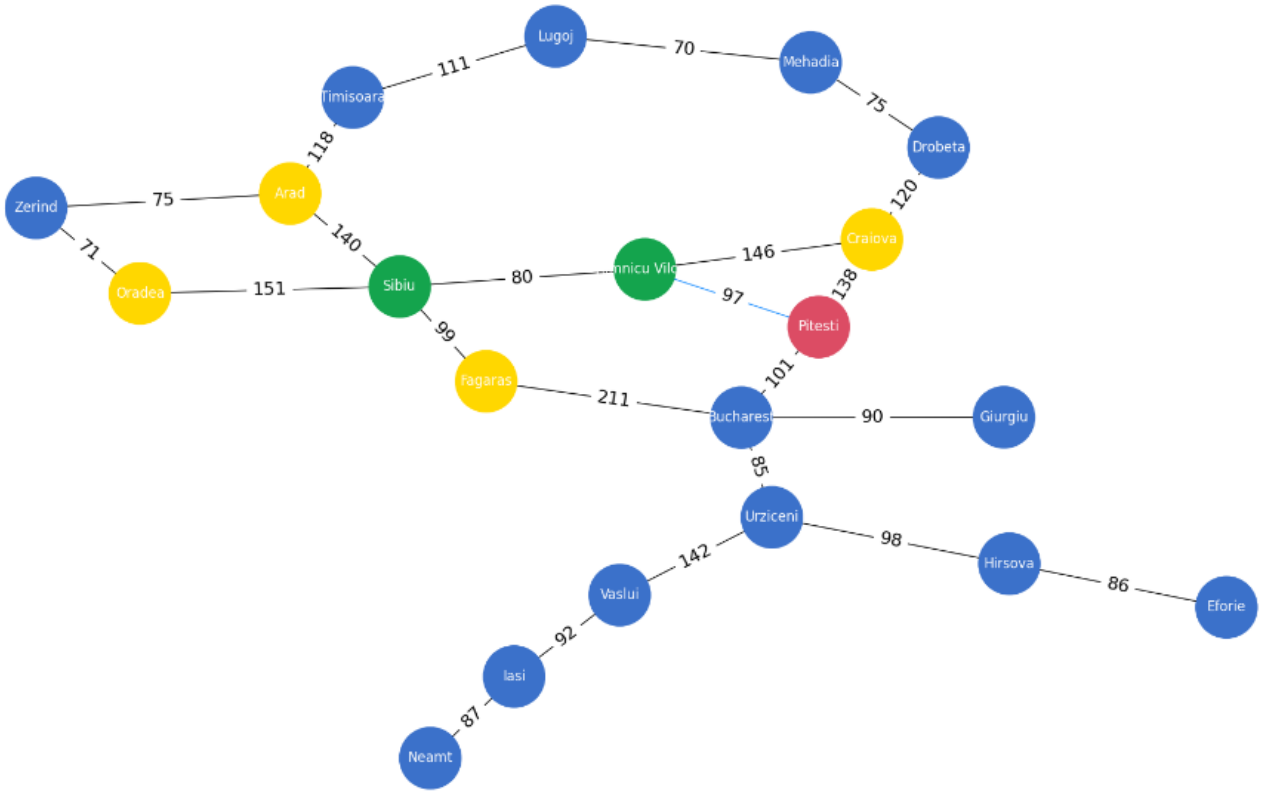
Hình 29: Minh họa tìm đường đi ngắn nhất từ ‘Rimnicu Vilcea’ đến ‘Urziceni’(1)



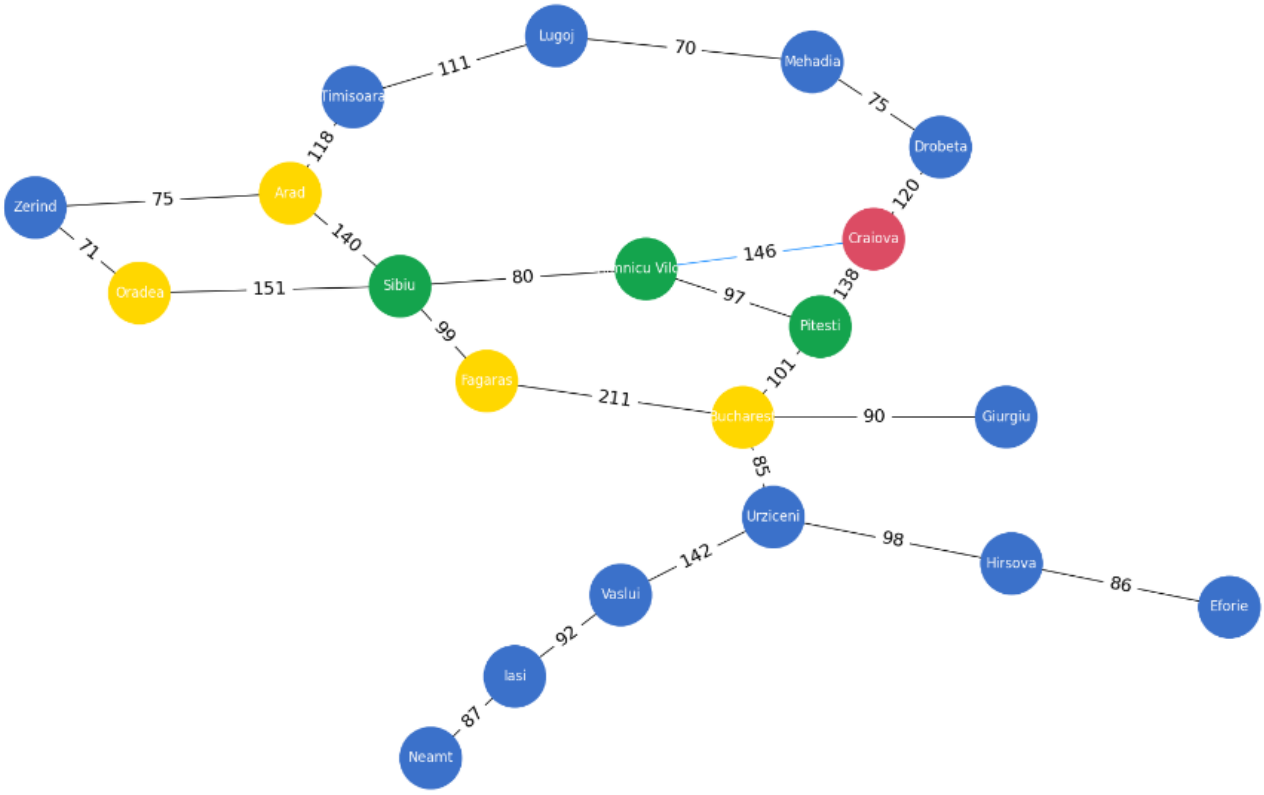
Hình 30: Minh họa tìm đường đi ngắn nhất từ 'Rimnicu Vilcea' đến 'Urziceni'(2)



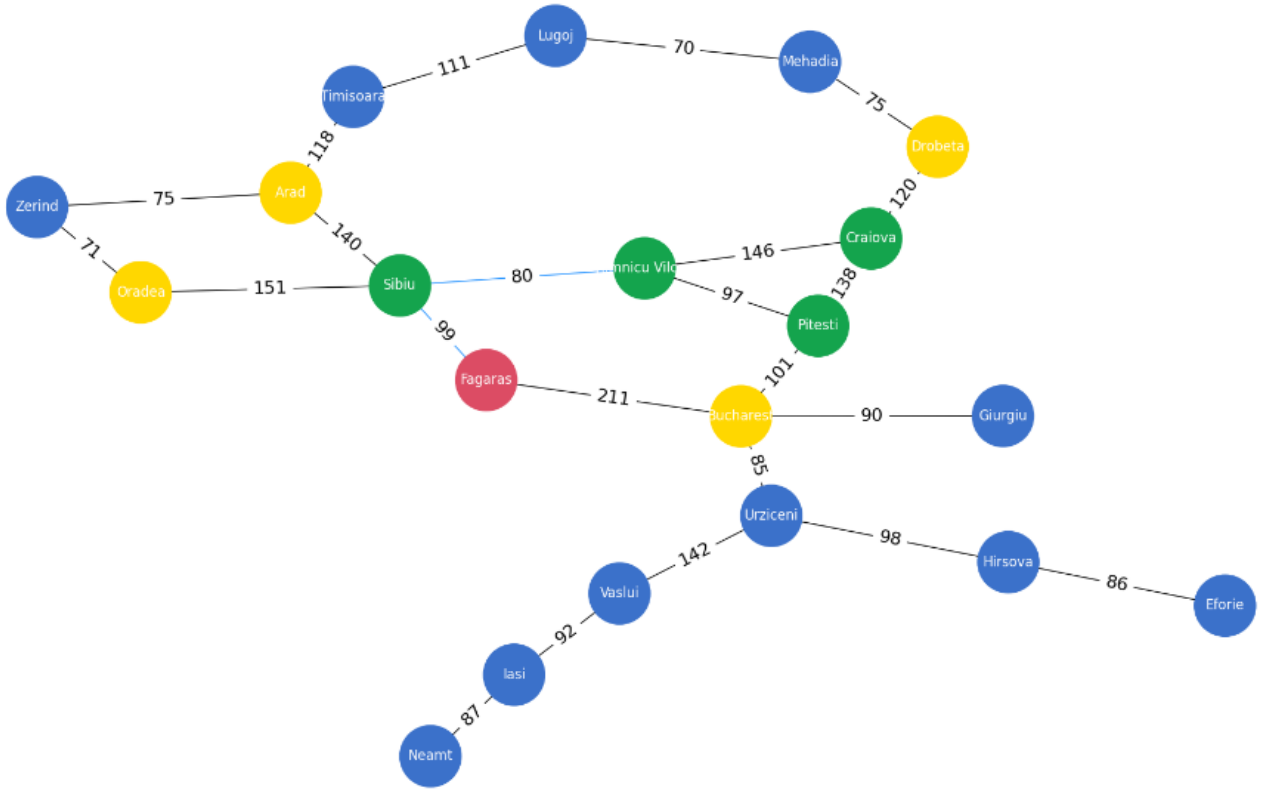
Hình 31: Minh họa tìm đường đi ngắn nhất từ 'Rimnicu Vilcea' đến 'Urziceni'(3)



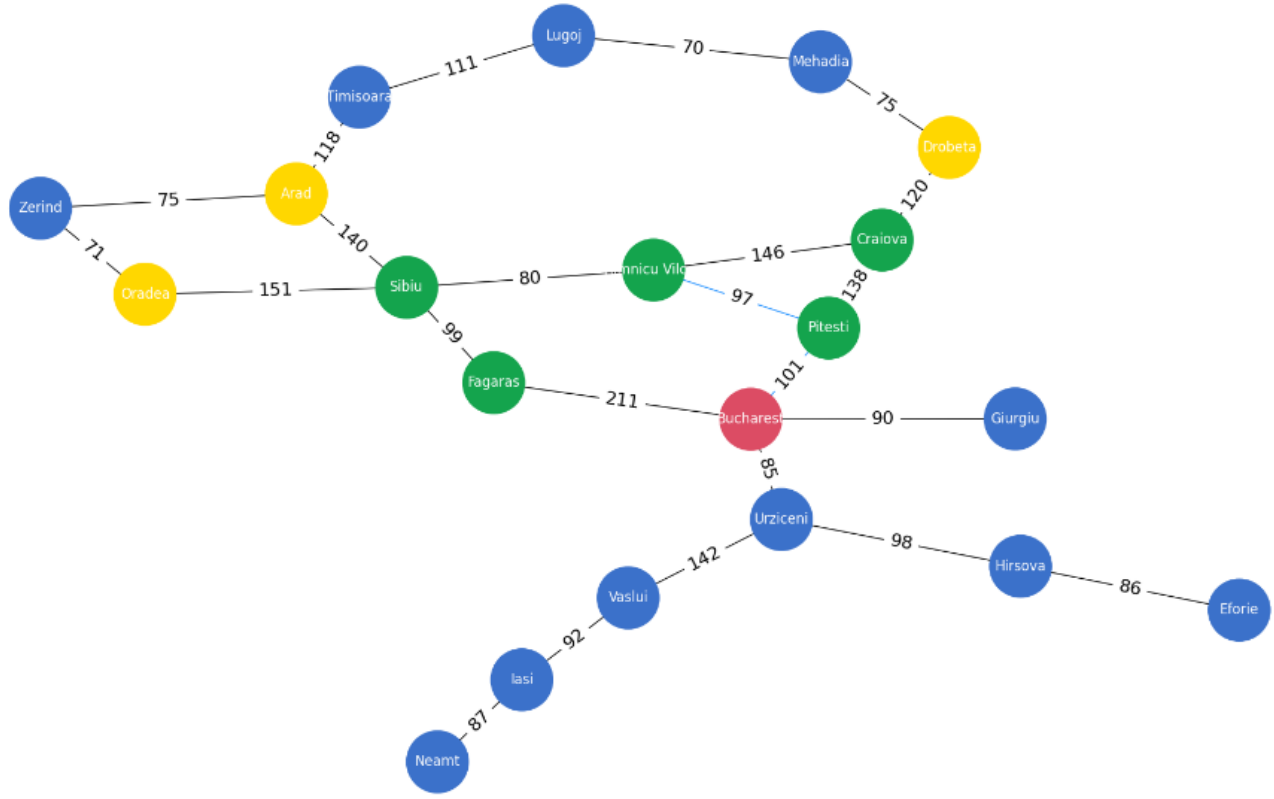
Hình 32: Minh họa tìm đường đi ngắn nhất từ 'Rimnicu Vilcea' đến 'Urziceni'(4)



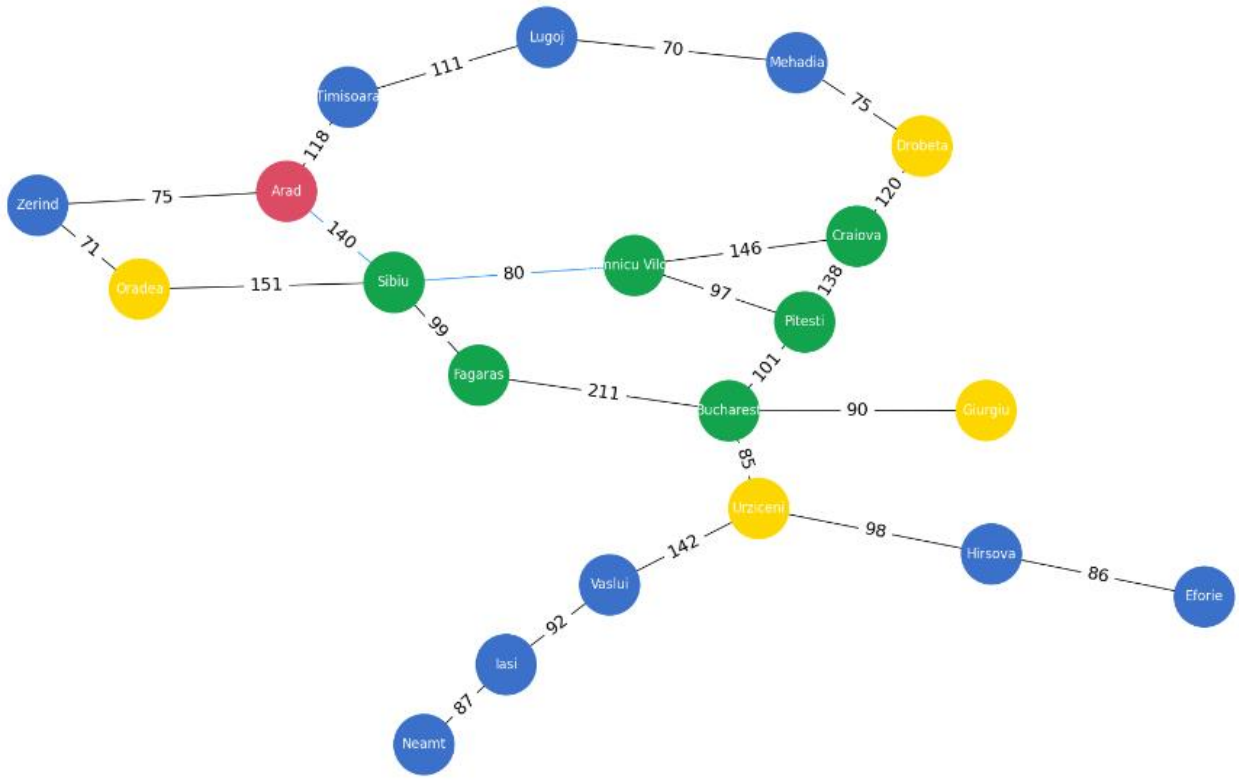
Hình 33: Minh họa tìm đường đi ngắn nhất từ 'Rimnicu Vilcea' đến 'Urziceni'(5)



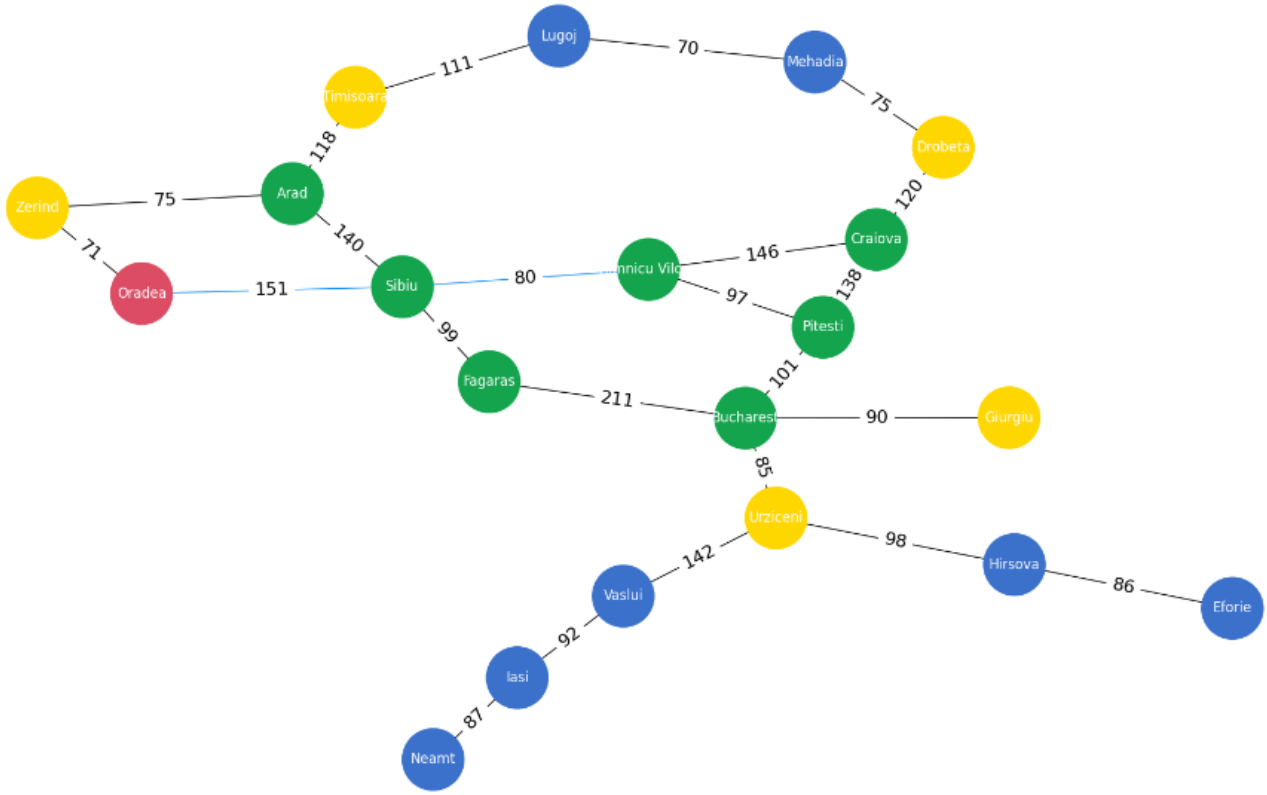
Hình 34: Minh họa tìm đường đi ngắn nhất từ 'Rimnicu Vilcea' đến 'Urziceni'(6)



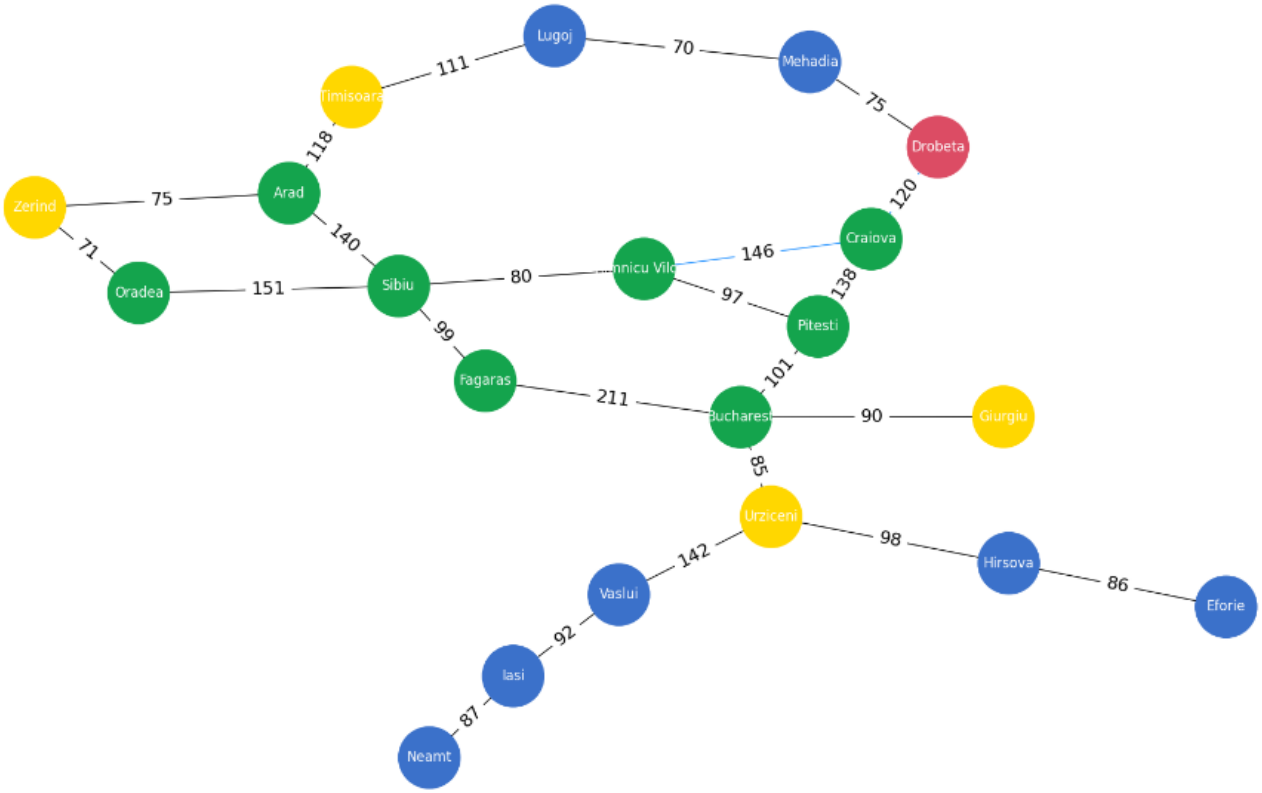
Hình 35: Minh họa tìm đường đi ngắn nhất từ 'Rimnicu Vilcea' đến 'Urziceni'(7)



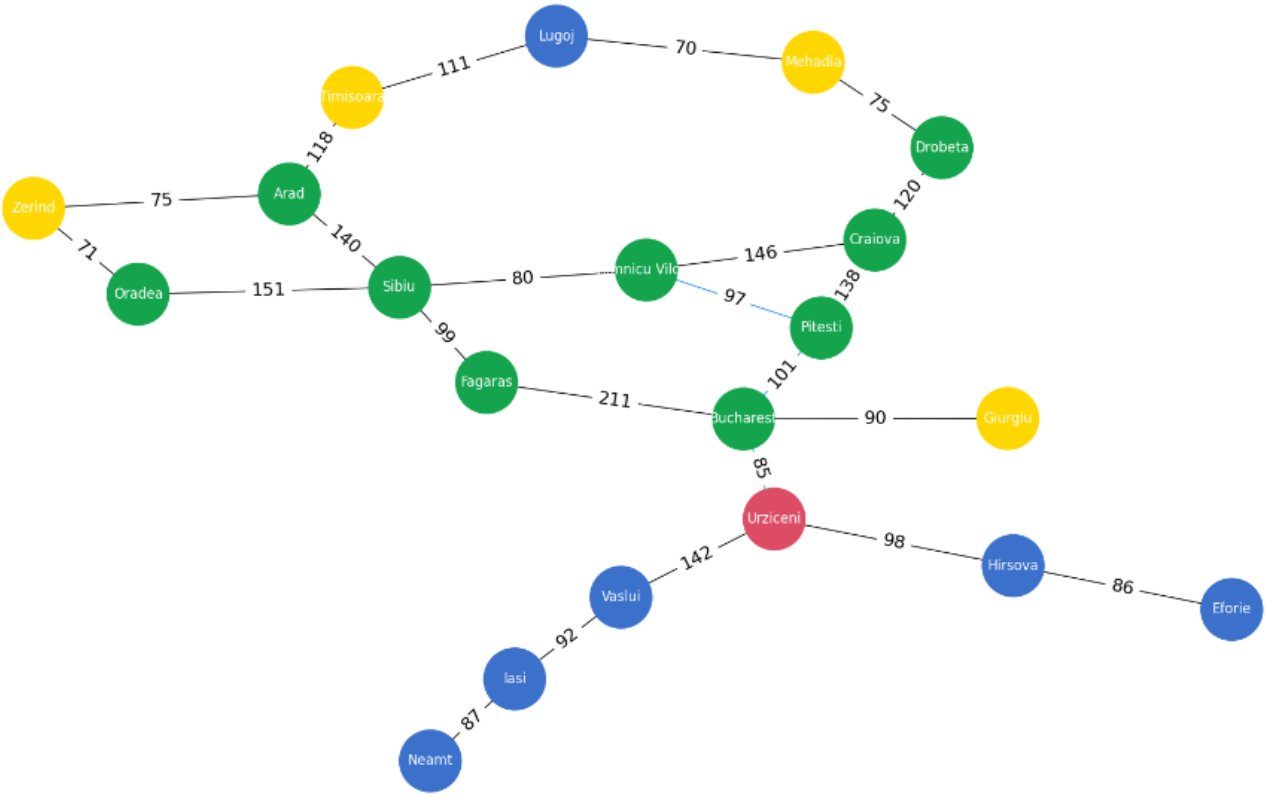
Hình 36: Minh họa tìm đường đi ngắn nhất từ 'Rimnicu Vilcea' đến 'Urziceni'(8)



Hình 37: Minh họa tìm đường đi ngắn nhất từ 'Rimnicu Vilcea' đến 'Urziceni'(9)



Hình 38: Minh họa tìm đường đi ngắn nhất từ 'Rimnicu Vilcea' đến 'Urziceni'(10)



Hình 39: Minh họa tìm đường đi ngắn nhất từ 'Rimnicu Vilcea' đến 'Urziceni'(11)

4.4.3. Hàm test3(): Đường đi ngắn nhất từ ‘Timisoara’ đến ‘Giurgiu’

```
def test3():
    start = 'Timisoara'
    goal = 'Giurgiu'
    print(f"Đường đi ngắn nhất từ thành phố {start} đến {goal} là:")
    path = unifcost(start, goal)
    print (path)
    global COUNT
    print('Tổng số nút được xét là: ' + str(COUNT))
    print('Tổng khoảng cách lối đi là: ' + str(FVALUE[goal]))
```

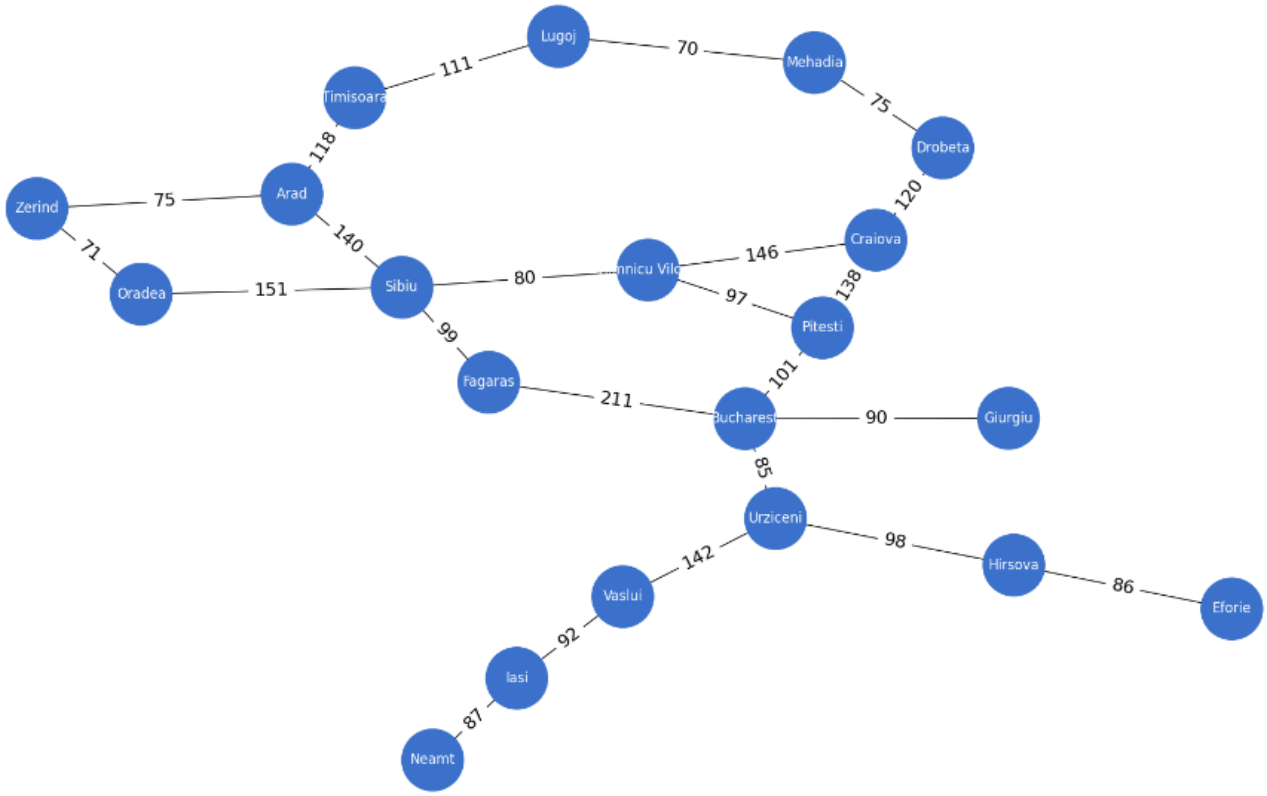
Hình 40: Hàm test3()

→ Kết quả khi gọi hàm test3():

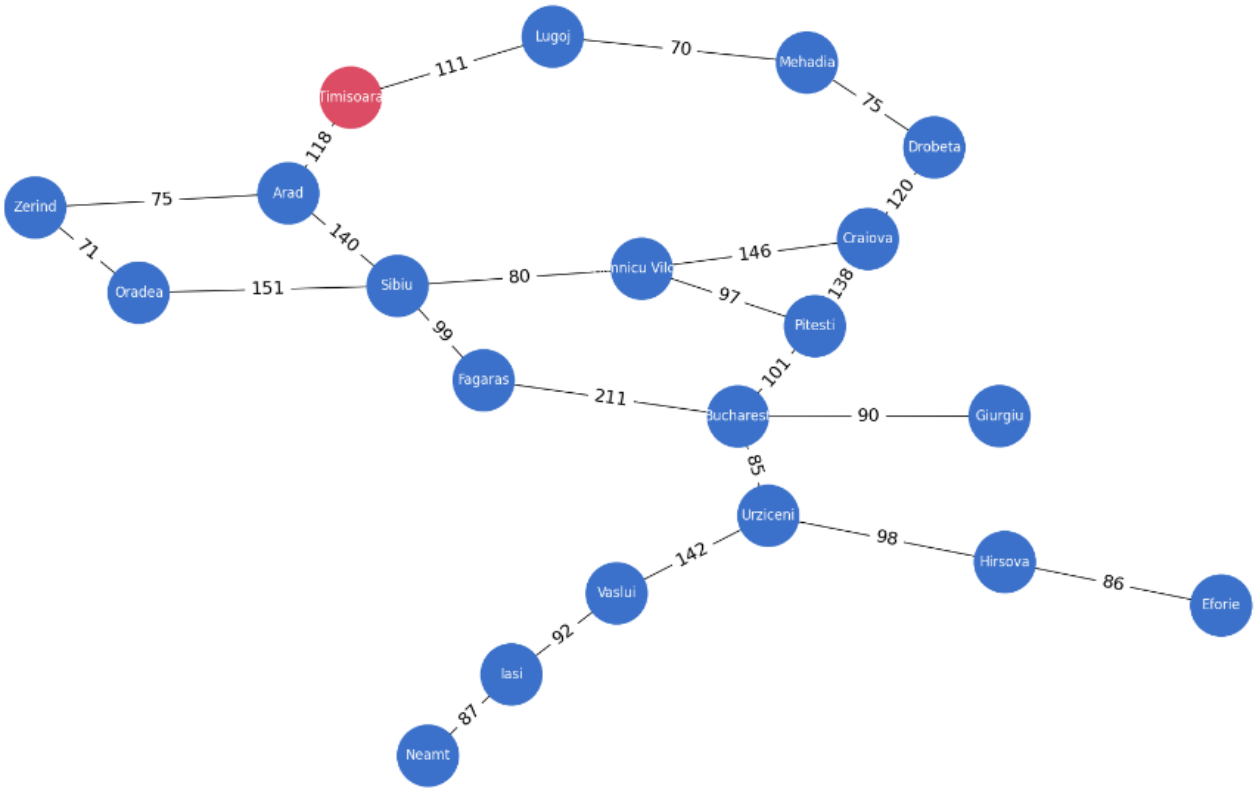
```
Đường đi ngắn nhất từ thành phố Timisoara đến Giurgiu là:
['Timisoara', 'Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest', 'Giurgiu']
Tổng số nút được xét là: 15
Tổng khoảng cách lối đi là: 626
```

Hình 41: Kết quả hàm test3()

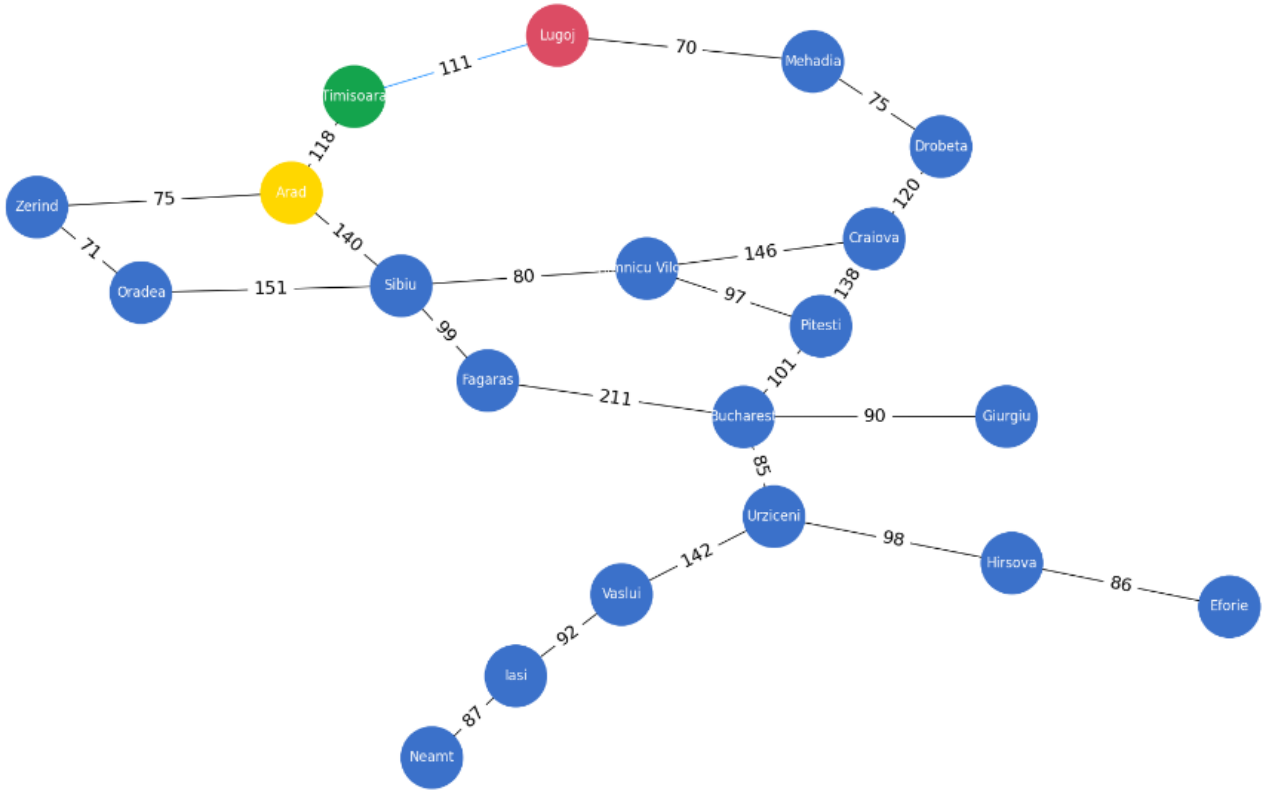
Minh họa các bước tìm đường đi ngắn nhất từ thành phố ‘Timisoara’ đến ‘Giurgiu’:



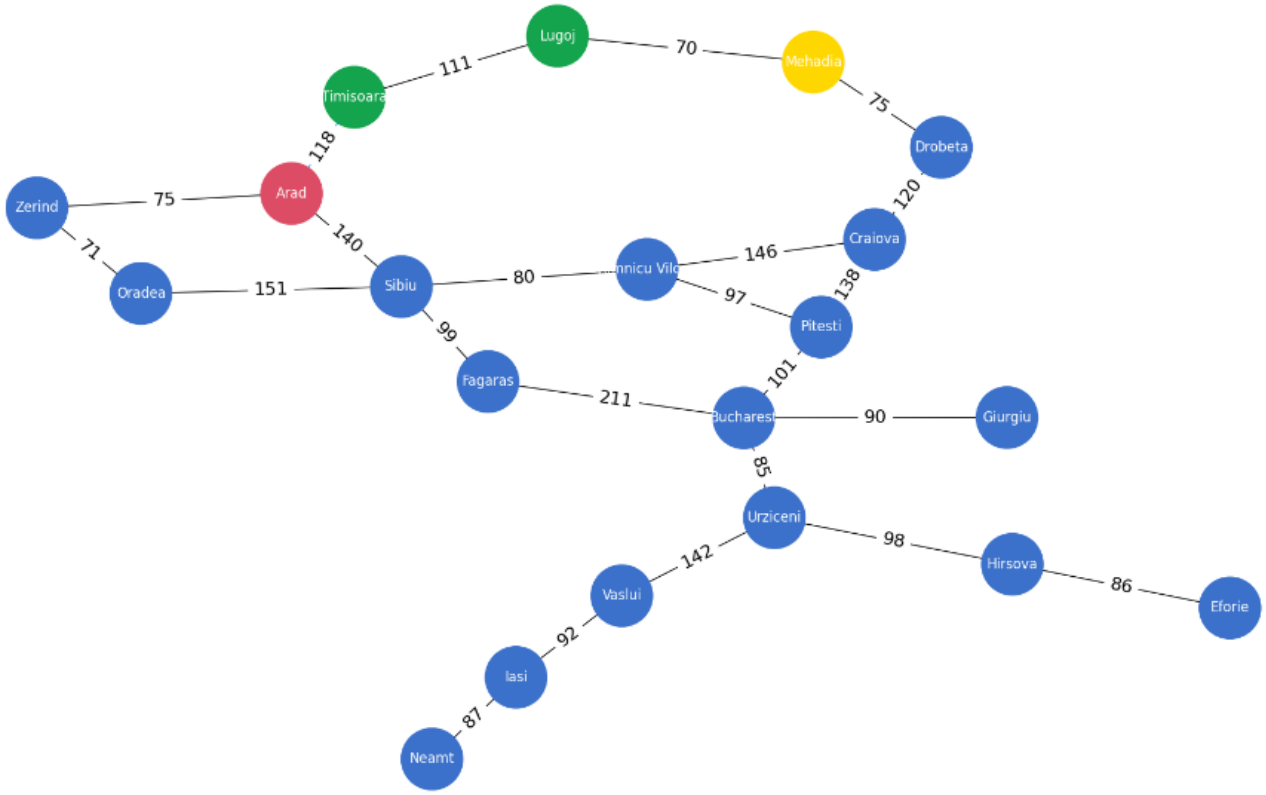
Hình 42: Minh họa tìm đường đi ngắn nhất từ ‘Timisoara’ đến ‘Giurgiu’(1)



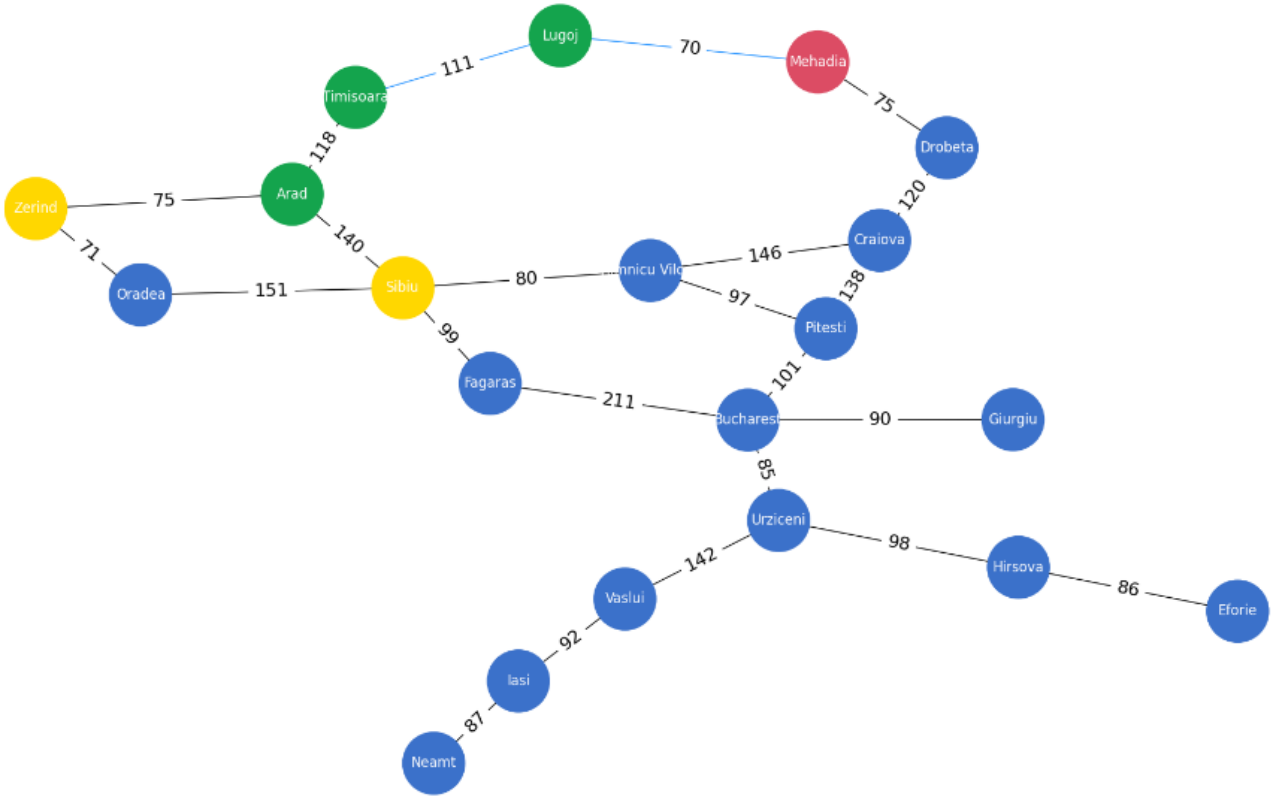
Hình 43: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(2)



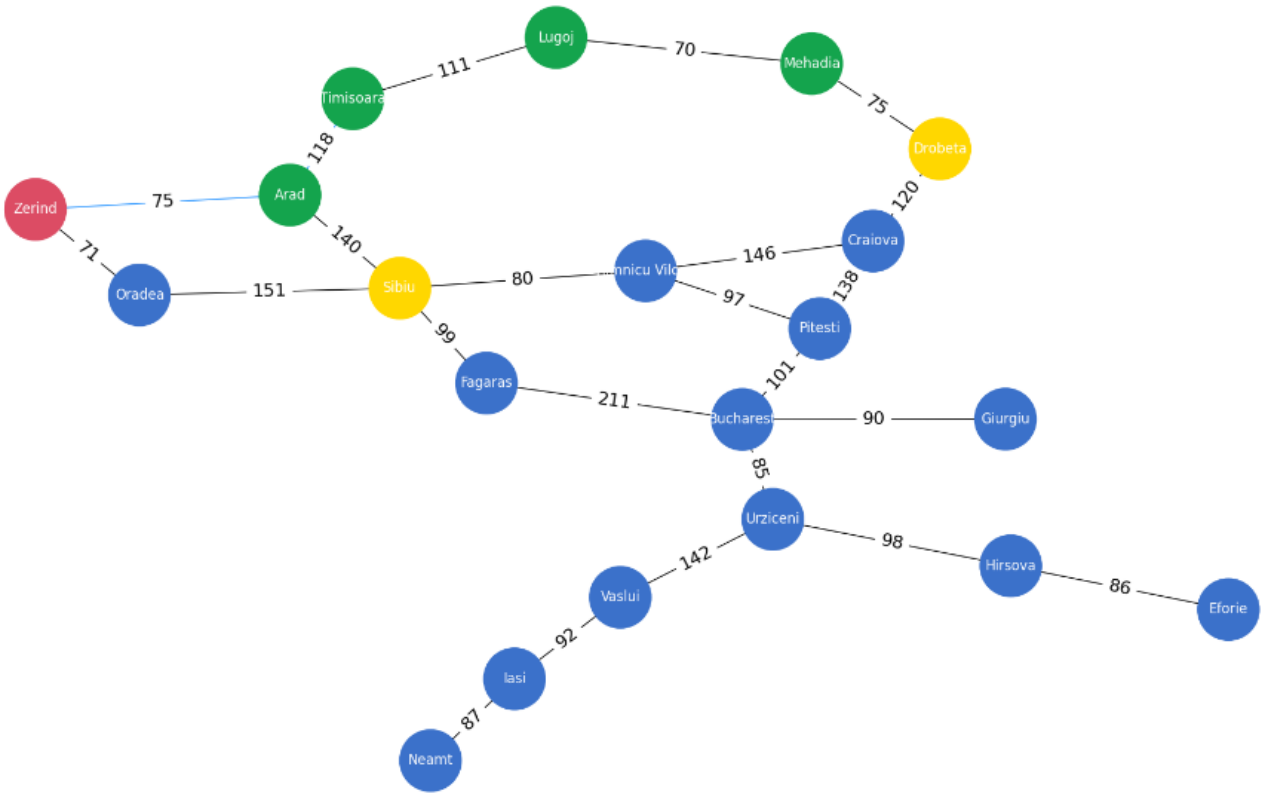
Hình 44: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(3)



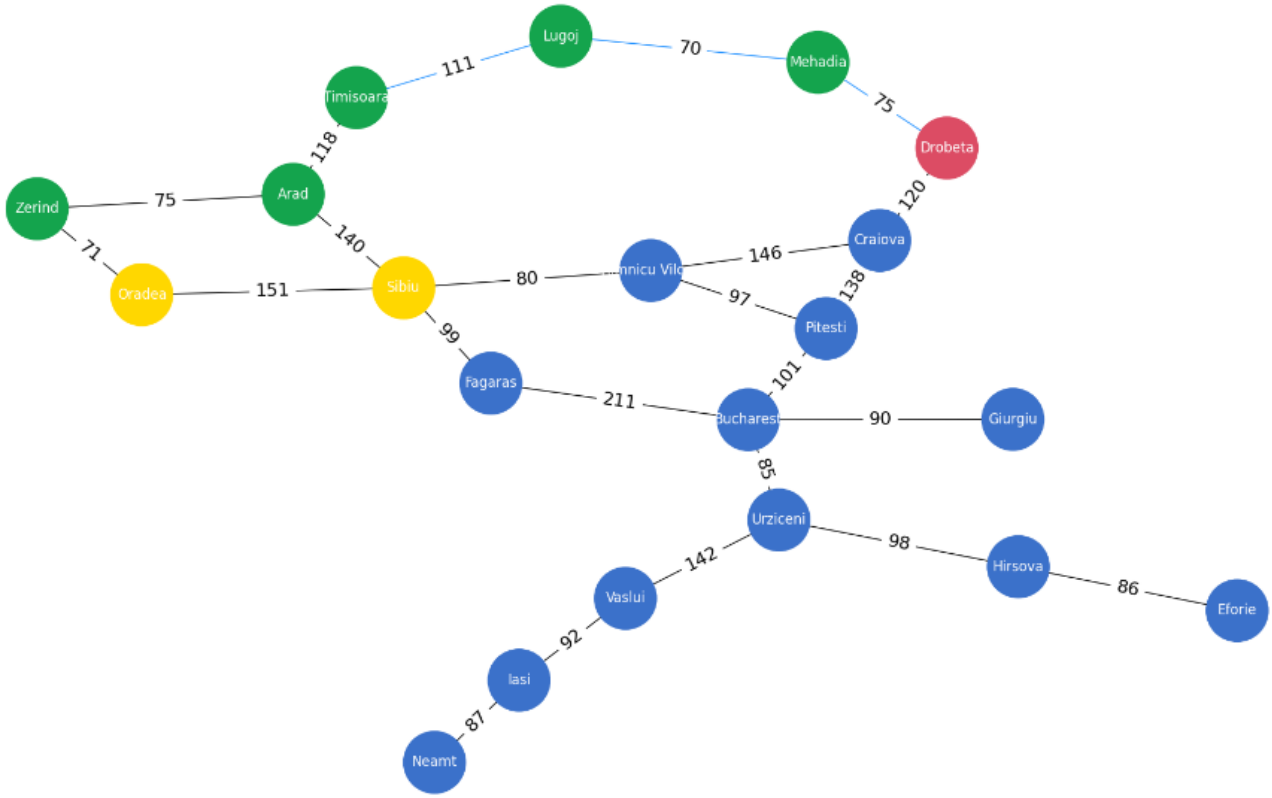
Hình 45: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(4)



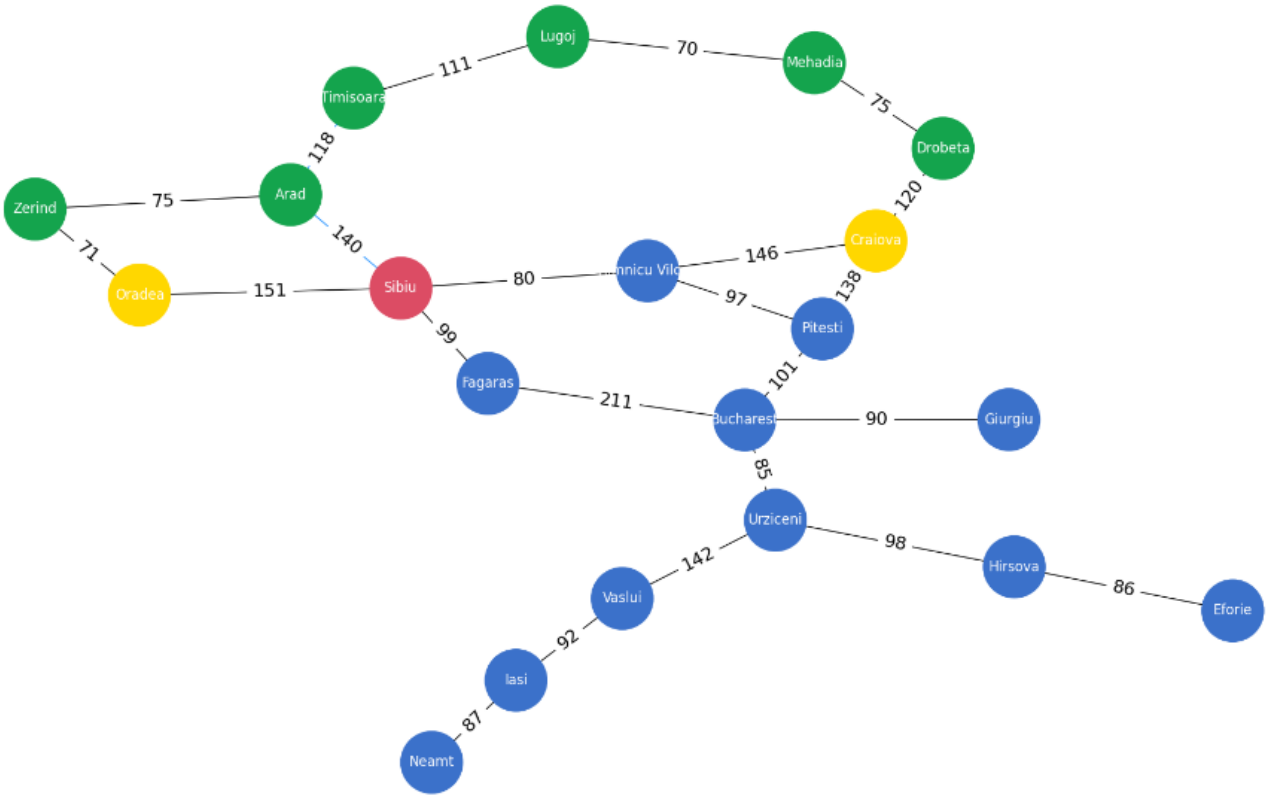
Hình 46: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(5)



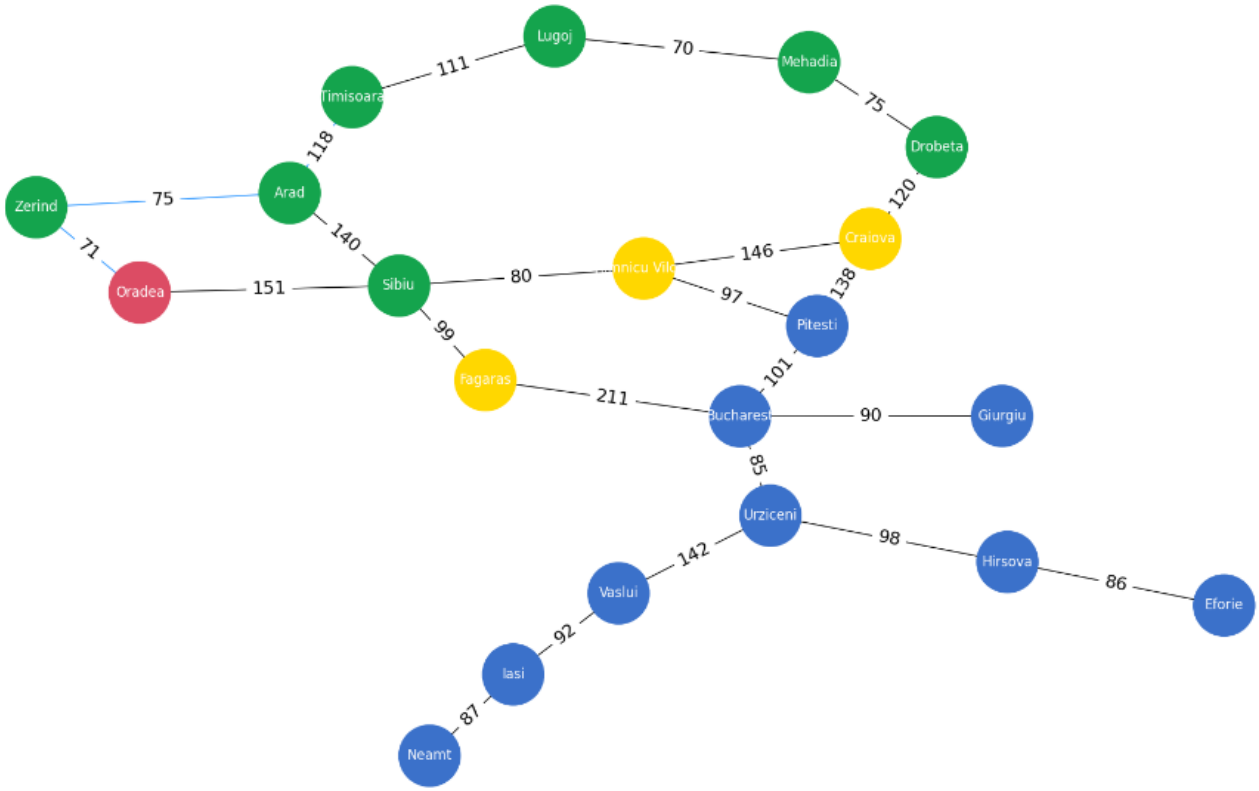
Hình 47: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(6)



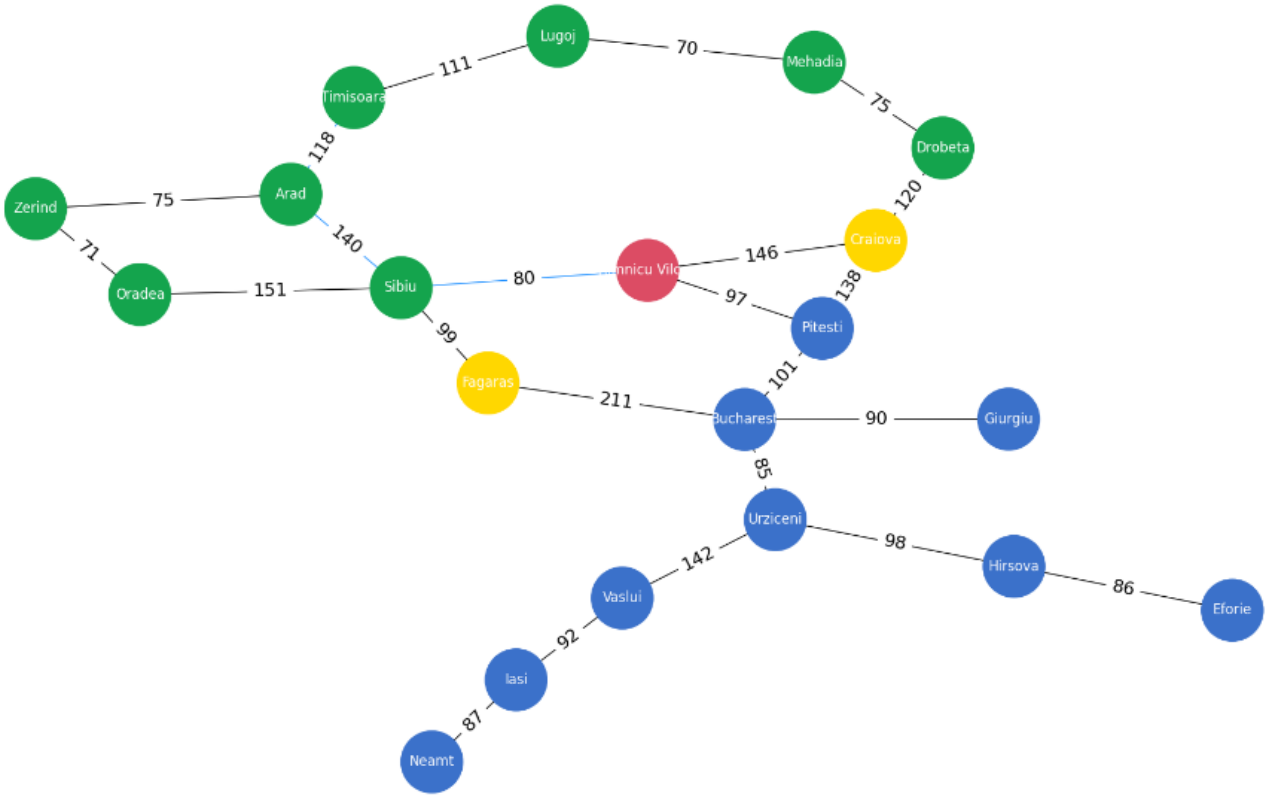
Hình 48: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(7)



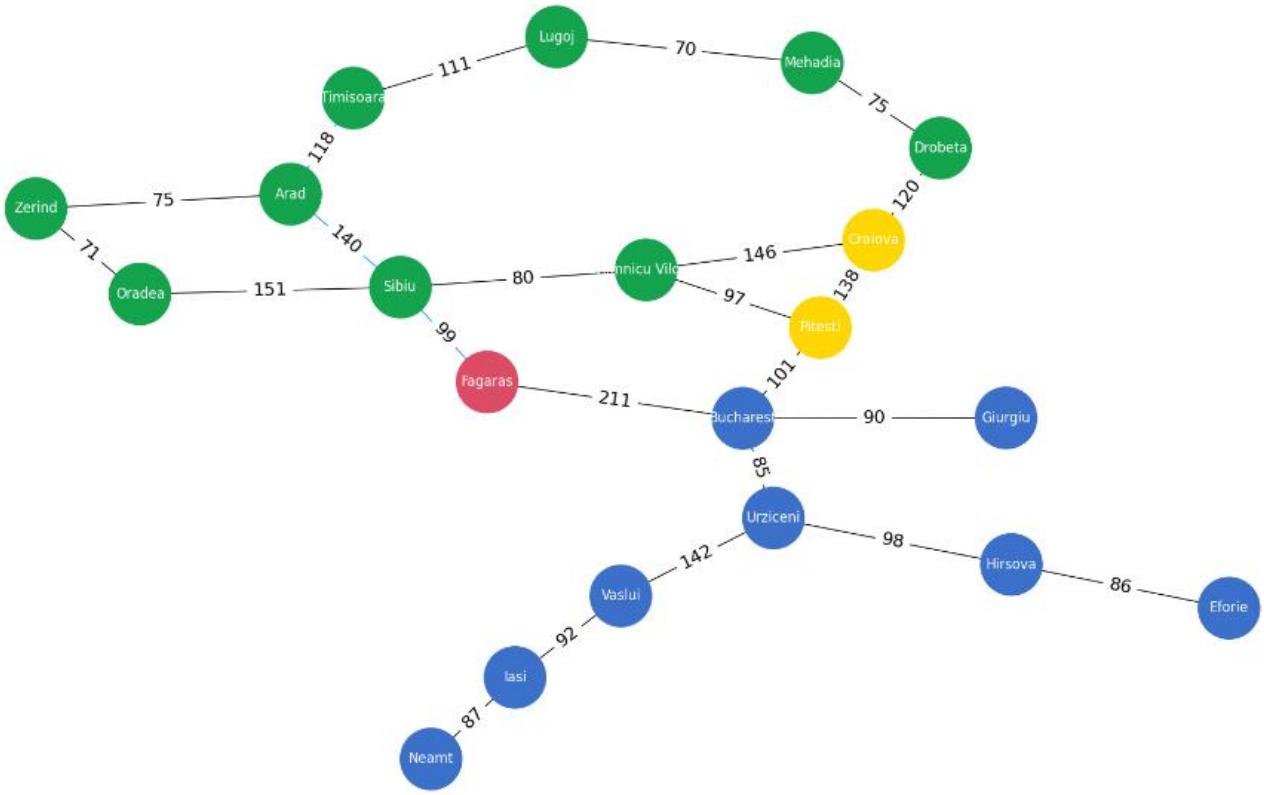
Hình 49: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(8)



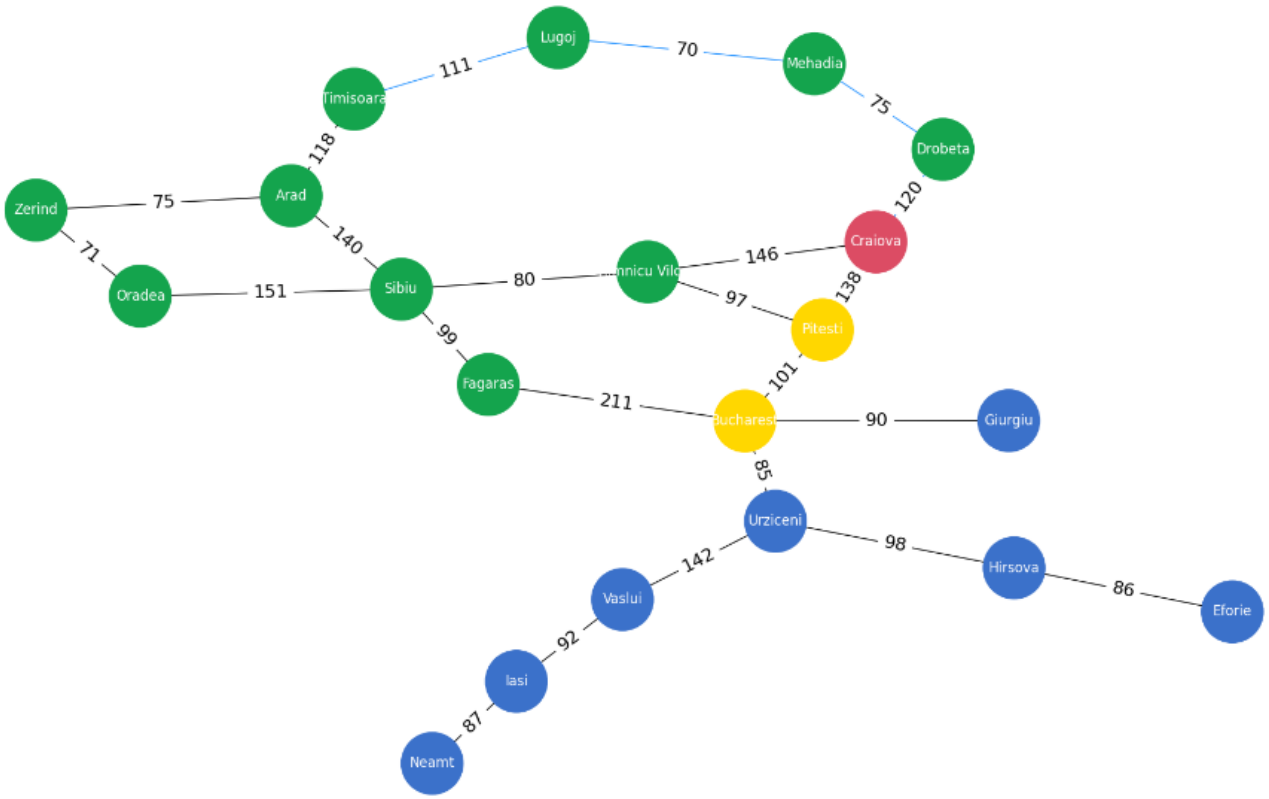
Hình 50: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(9)



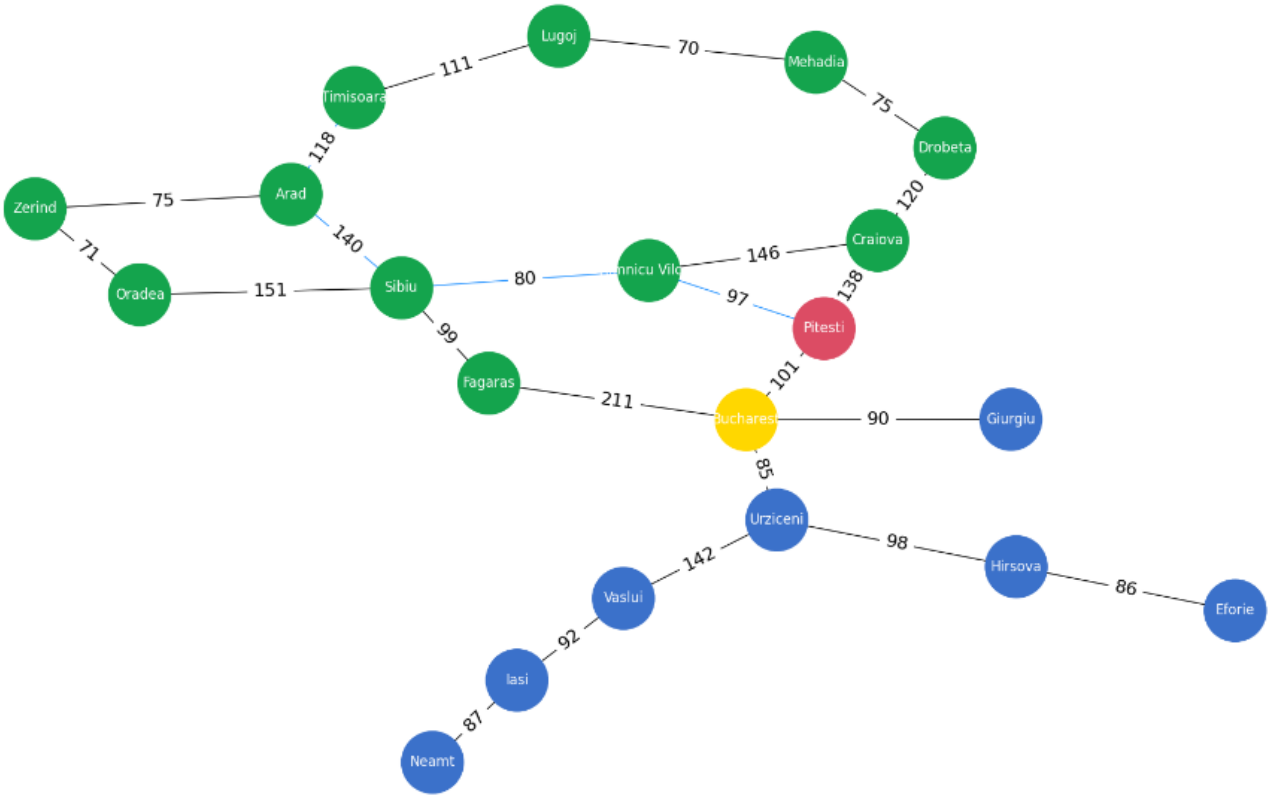
Hình 51: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(10)



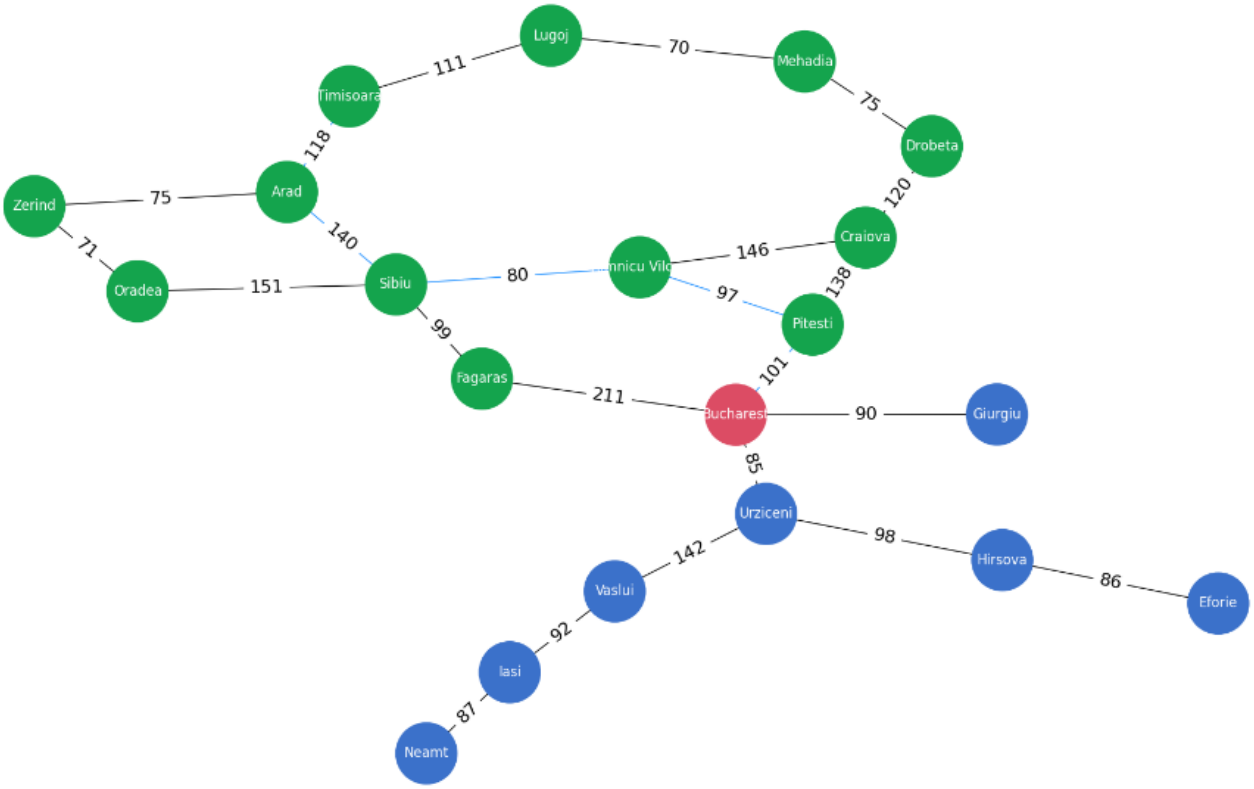
Hình 52: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(11)



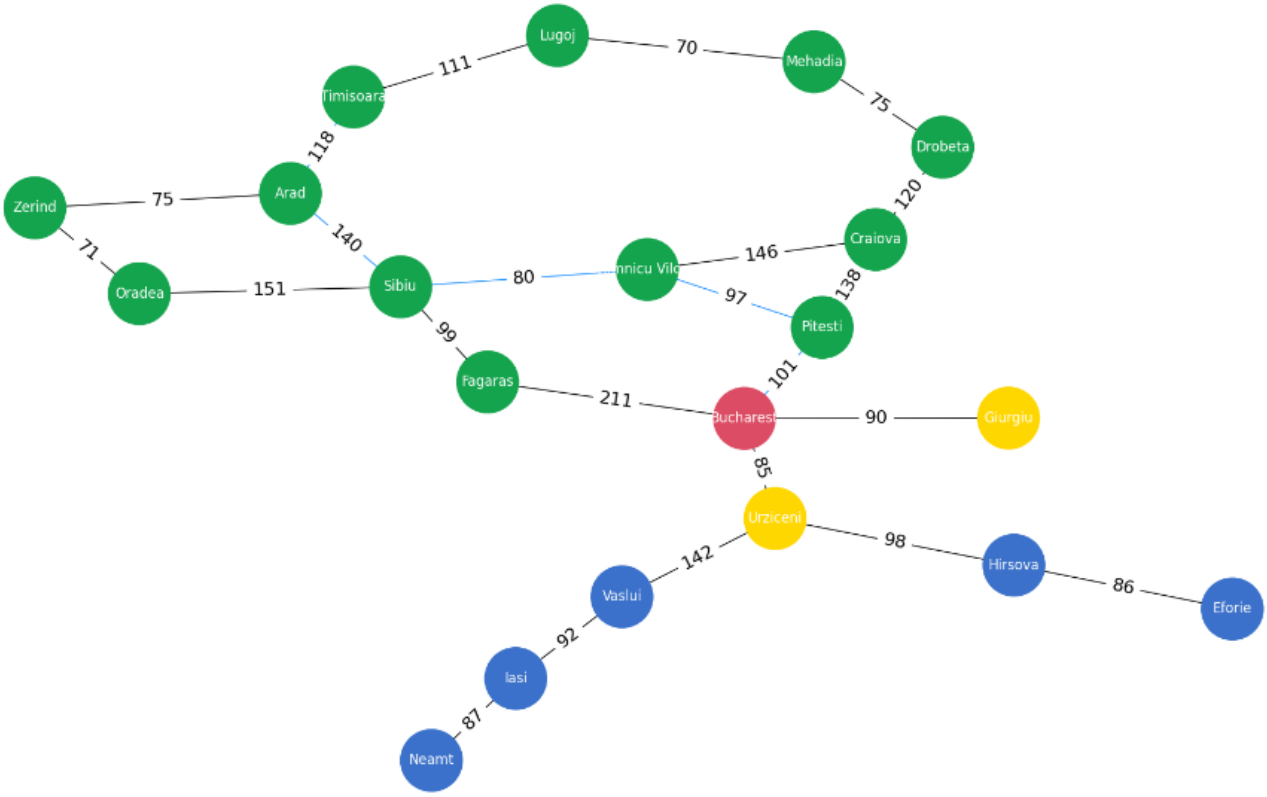
Hình 53: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(12)



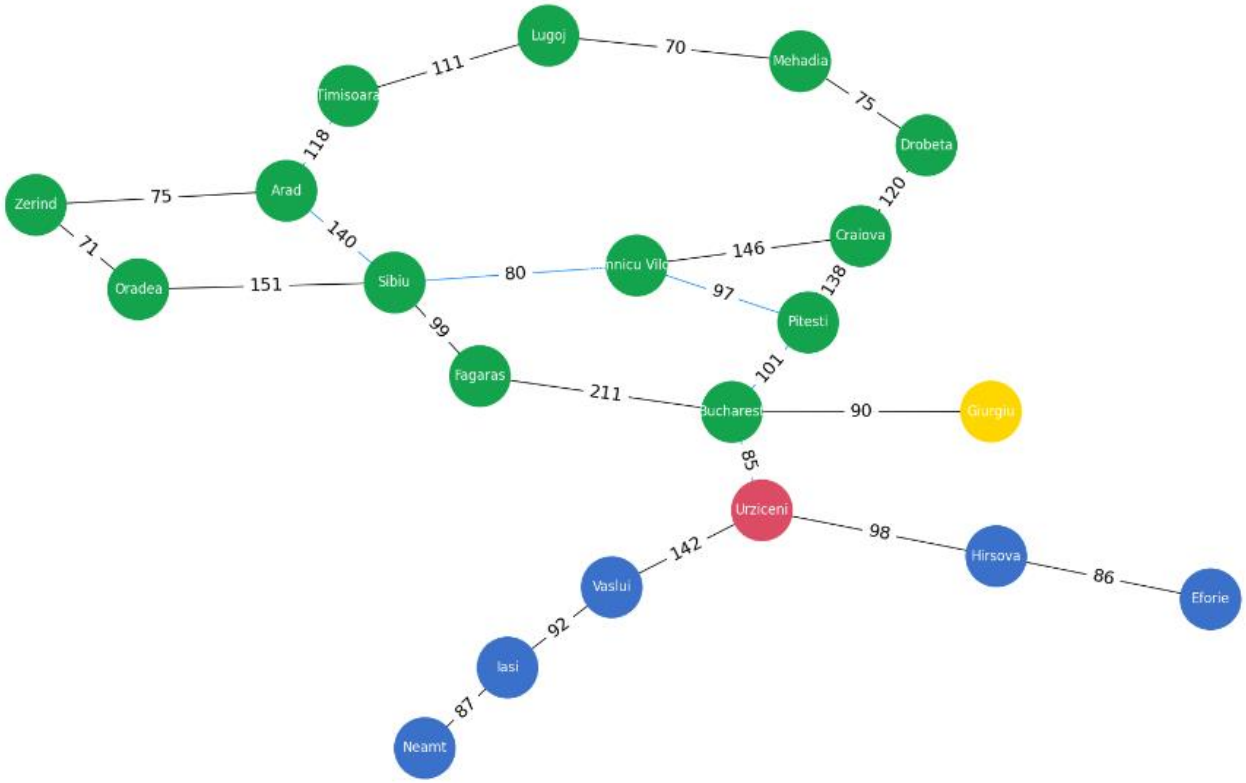
Hình 54: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(13)



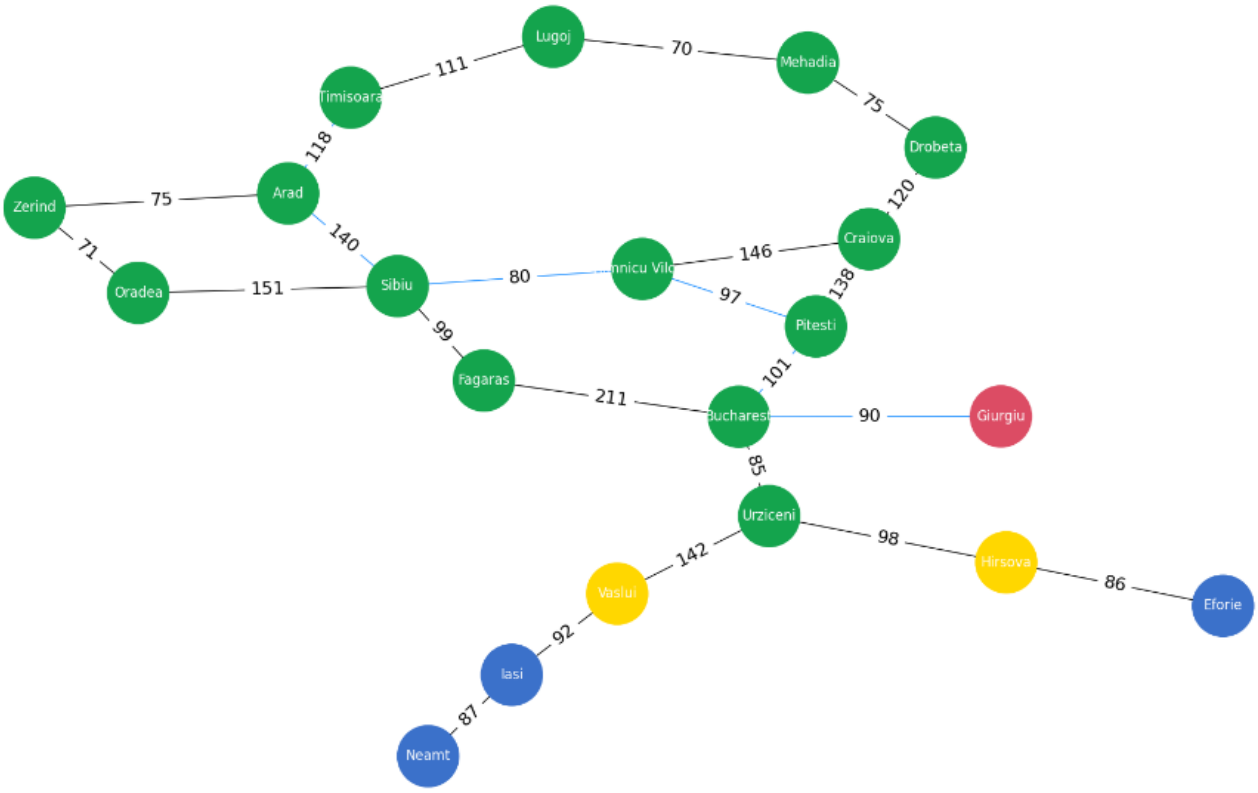
Hình 55: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(14)



Hình 56: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(15)



Hình 57: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(16)



Hình 58: Minh họa tìm đường đi ngắn nhất từ 'Timisoara' đến 'Giurgiu'(17)

5. Kết luận:

- Tìm kiếm trên không gian trạng thái có thể giúp giải quyết một số bài toán trong AI.
- Uniform cost search thuộc loại tìm kiếm dựa vào heuristic, có sử dụng thông tin hỗ trợ để di chuyển tới trạng thái đích nhanh hơn.

6. Tài liệu tham khảo:

[1] PGS.TS Dương Tuấn Anh, Chương 3 – Tìm kiếm trên không gian trạng thái.