

Bài tập tuần 3

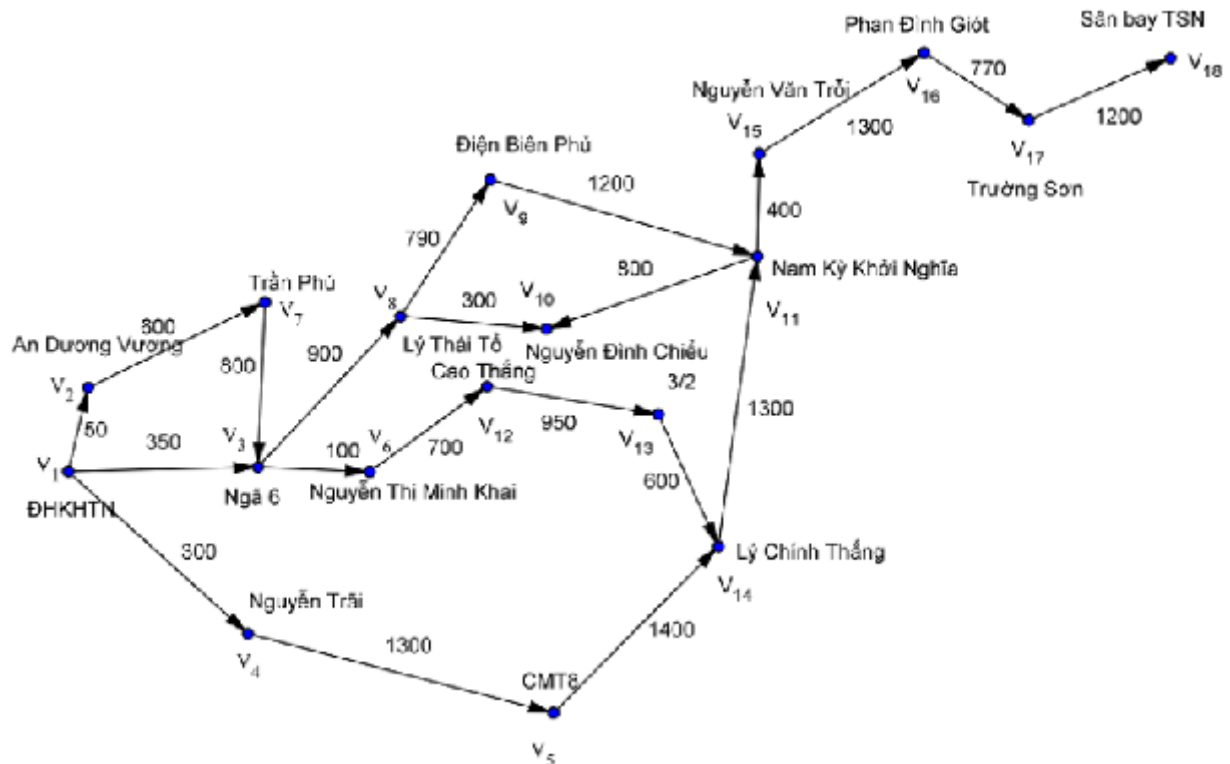
Nguyễn Lê Ngọc Duy

Mục lục

1	Giới thiệu	2
2	Giới thiệu các thuật toán	3
2.1	Thuật toán BFS	3
2.2	Thuật toán Depth First Search - DFS	4
2.3	Thuật toán UCS	5
3	Cài đặt các thuật toán	6
3.1	Ngôn ngữ lập trình	6
3.2	Cài đặt bằng phương pháp lập trình hàm	6
3.2.1	Dữ liệu đầu vào	6
3.2.2	Các hàm phụ trợ	8
3.2.3	Cài đặt thuật toán BFS	8
3.2.4	Cài đặt thuật toán DFS	9
3.2.5	Cài đặt thuật toán UCS	10
3.3	Xây dựng lớp Graph riêng	11
4	Nhận xét	13

1 Giới thiệu

Bài tập tuần này sử dụng các thuật toán duyệt đồ thị như Breadth-first search (BFS), Depth-first search (DFS) và Uniform Cost Search (UCS) để giải quyết bài toán tìm đường đi ngắn nhất từ ĐHKHTN (đỉnh V_1) đến Sân bay TSN (đỉnh V_{18}) trên đồ thị sau:



Hình 1: Đồ thị cho bài toán

2 Giới thiệu các thuật toán

2.1 Thuật toán BFS

Dưới đây là mô tả cho thuật toán BFS

Thuật toán 1 Thuật toán Breadth First Search - BFS

Đầu vào: đồ thị G , node xuất phát $start$, node kết thúc end .

Đầu ra: đường đi từ $start$ đến end

begin

```
1: Khởi tạo danh sách  $L$  chứa trạng thái ban đầu.  
2: while true do  
3:   if  $L$  rỗng then  
4:     Thông báo không có đường đi.  
5:     break  
6:   end if  
7: end while  
8: Loại trạng thái  $u$  ở đầu danh sách  $L$ .  
9: if  $u$  là trạng thái kết thúc then  
10:  return đường đi từ  $start$  đến  $u$ .  
11: end if  
12: Lấy các trạng thái  $v$  kề với  $u$  và thêm vào cuối danh sách  $L$ .  
13: for mỗi trạng thái  $v$  kề với  $u$  do  
14:    $father(v) = u$ ;  
15: end for  
end
```

Áp dụng thuật toán BFS, ta giải quyết bài toán ở trên bằng bảng như sau:

Lần lặp	Đỉnh	Hàng đợi
1	V_1 (node bắt đầu)	V_2, V_3, V_4
2	V_2	V_3, V_4, V_7
3	V_3	V_4, V_7, V_6, V_8
4	V_4	V_7, V_6, V_8, V_5
5	V_7	V_6, V_8, V_5
6	V_6	V_8, V_5, V_{12}
7	V_8	V_5, V_{12}, V_9, V_{10}
8	V_5	$V_{12}, V_9, V_{10}, V_{14}$
9	V_{12}	$V_9, V_{10}, V_{14}, V_{13}$
10	V_9	$V_{10}, V_{14}, V_{13}, V_{11}$
11	V_{10}	V_{14}, V_{13}, V_{11}
12	V_{14}	V_{13}, V_{11}
13	V_{13}	V_{11}
14	V_{11}	V_{15}
15	V_{15}	V_{16}
16	V_{16}	V_{17}
17	V_{17}	V_{18}
18	V_{18} (node kết thúc)	\emptyset

Ta thu được đường đi theo thuật toán BFS là:

$$V_1 \rightarrow V_4 \rightarrow V_5 \rightarrow V_{14} \rightarrow V_{11} \rightarrow V_{15} \rightarrow V_{16} \rightarrow V_{17} \rightarrow V_{18}$$

2.2 Thuật toán Depth First Search - DFS

Dưới đây là mô tả cho thuật toán DFS

Thuật toán 2 Thuật toán Depth First Search - DFS

Đầu vào: đồ thị G , node xuất phát $start$, node kết thúc end .

Đầu ra: đường đi từ $start$ đến end

begin

```

1: Khởi tạo danh sách  $L$  chứa trạng thái ban đầu.
2: while true do
3:   if  $L$  rỗng then
4:     Thông báo không có đường đi.
5:     break
6:   end if
7: end while
8: Loại trạng thái  $u$  ở đầu danh sách  $L$ .
9: if  $u$  là trạng thái kết thúc then
10:  return đường đi từ  $start$  đến  $u$ .
11: end if
12: Lấy các trạng thái  $v$  kề với  $u$  và thêm vào đầu danh sách  $L$ .
13: for mỗi trạng thái  $v$  kề với  $u$  do
14:    $father(v) = u$ 
15: end for
end

```

Áp dụng thuật toán DFS, ta giải quyết bài toán trên bằng bảng như sau:

Lần lặp	Đỉnh	Ngăn xếp
1	V_1 (node xuất phát)	V_2, V_3, V_4
2	V_2	V_7, V_3, V_4
3	V_7	V_3, V_4
4	V_3	V_6, V_8, V_4
5	V_6	V_{12}, V_8, V_4
6	V_{12}	V_{13}, V_8, V_4
7	V_{13}	V_{14}, V_8, V_4
8	V_{14}	V_{11}, V_8, V_4
9	V_{11}	V_{15}, V_8, V_4
10	V_{15}	V_{16}, V_8, V_4
11	V_{16}	V_{17}, V_8, V_4
12	V_{17}	V_{18}, V_8, V_4
13	V_{18} (node kết thúc)	V_8, V_4

Ta thu được đường đi theo thuật toán DFS là:

$$V_1 \rightarrow V_3 \rightarrow V_6 \rightarrow V_{12} \rightarrow V_{13} \rightarrow V_{14} \rightarrow V_{11} \rightarrow V_{15} \rightarrow V_{16} \rightarrow V_{17} \rightarrow V_{18}$$

2.3 Thuật toán UCS

Dưới đây là mô tả cho thuật toán UCS

Thuật toán 3 Thuật toán Uniform Cost Search - UCS

Đầu vào: đồ thị G , node xuất phát $start$, node kết thúc end .

Đầu ra: đường đi từ $start$ đến end và chi phí của đường đi

begin

```

1: Tạo hàng đợi ưu tiên rỗng  $NganChua$ .
2: Thêm nút  $start$  vào  $NganChua$ .
3: loop
4:   if  $NganChua$  rỗng then
5:     return không tồn tại đường đi.
6:   else
7:      $Nut = \text{TimChiPhiNhoNhat}(NganChua)$ 
8:     if  $\text{KiemTraCauHoiDich}(BaiToan)$  trên  $\text{TrangThai}(Nut)$  đúng then
9:       return  $\text{LoiGiai}(Nut)$ 
10:    end if
11:  end if
12: end loop
13:  $\text{LoiGiai} = \text{Mo}(Nut, BaiToan)$ 
14:  $NganChua = \text{ThemTatCa}(\text{LoiGiai}, NganChua)$ 
end

```

Áp dụng thuật toán UCS, ta giải quyết bài toán trên bằng bảng như sau:

Lần lặp	Đỉnh	Hàng đợi ưu tiên (theo thứ tự tổng chi phí)	Tổng chi phí thấp nhất
1	V_1 (đỉnh bắt đầu)	$(V_2; 50)^*, (V_4; 300), (V_3; 350)$	50
2	V_2	$(V_4; 300)^*, (V_3; 350), (V_7; 650)$	300
3	V_4	$(V_3; 350)^*, (V_7; 650), (V_5; 1600)$	350
4	V_3	$(V_6; 450)^*, (V_7; 650), (V_5; 1600), (V_8; 1250)$	450
5	V_6	$(V_7; 650)^*, (V_{12}; 1150), (V_8; 1250), (V_5; 1600)$	650
6	V_7	$(V_{12}; 1150)^*, (V_8; 1250), (V_5; 1600)$	1150
7	V_{12}	$(V_8; 1250)^*, (V_5; 1600), (V_{13}; 2100)$	1250
8	V_8	$(V_{10}; 1550)^*, (V_5; 1600), (V_9; 2040), (V_{13}; 2100)$	1550
9	V_{10}	$(V_5; 1600)^*, (V_9; 2040), (V_{13}; 2100)$	1600
10	V_5	$(V_9; 2040)^*, (V_{13}; 2100), (V_{14}; 3000)$	2040
11	V_9	$(V_{13}; 2100)^*, (V_{14}; 3000), (V_{11}; 3240)$	2100
12	V_{13}	$(V_{14}; 2700)^*, (V_{11}; 3240)$	2700
13	V_{14}	$(V_{11}; 3240)^*$	3240
14	V_{11}	$(V_{15}; 3640)^*$	3640
15	V_{15}	$(V_{16}; 4940)^*$	4940
16	V_{16}	$(V_{17}; 5710)^*$	5710
17	V_{17}	$(V_{18}; 6910)^*$	6910
18	V_{18} (đỉnh kết thúc)	\emptyset	6910

Ta thu được đường đi theo thuật toán UCS (với tổng chi phí nhỏ nhất là 6910) như sau:

$$V_1 \xrightarrow{350} V_3 \xrightarrow{900} V_8 \xrightarrow{790} V_9 \xrightarrow{1200} V_{11} \xrightarrow{400} V_{15} \xrightarrow{1300} V_{16} \xrightarrow{770} V_{17} \xrightarrow{1200} V_{18}$$

3 Cài đặt các thuật toán

3.1 Ngôn ngữ lập trình

Ngôn ngữ lập trình dùng để cài đặt là ngôn ngữ lập trình Python, phiên bản 3.9.13.

3.2 Cài đặt bằng phương pháp lập trình hàm

3.2.1 Dữ liệu đầu vào

Dữ liệu đầu vào được lưu trữ trong 2 file `input.txt` và `input_ucs.txt` với định dạng như sau:

```

1 18
2 0 17
3 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
5 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
6 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
7 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
8 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
9 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
10 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
11 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
13 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0
14 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
15 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
16 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
17 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

Hình 2: Định dạng file đầu vào (không bao gồm trọng số) dùng cho thuật toán BFS và DFS

```

1 18
2 0 17
3 0 50 350 300 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 600 0 0 0 0 0 0 0 0 0 0
5 0 0 0 0 0 100 0 900 0 0 0 0 0 0 0 0 0
6 0 0 0 0 1300 0 0 0 0 0 0 0 0 0 0 0 0
7 0 0 0 0 0 0 0 0 0 0 0 0 0 1400 0 0 0
8 0 0 0 0 0 0 0 0 0 0 0 700 0 0 0 0 0
9 0 0 800 0 0 0 0 0 0 0 0 0 0 0 0 0 0
10 0 0 0 0 0 0 0 0 790 300 0 0 0 0 0 0 0
11 0 0 0 0 0 0 0 0 0 0 1200 0 0 0 0 0 0
12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
13 0 0 0 0 0 0 0 0 0 800 0 0 0 0 400 0 0
14 0 0 0 0 0 0 0 0 0 0 0 0 950 0 0 0 0
15 0 0 0 0 0 0 0 0 0 0 0 0 0 600 0 0 0
16 0 0 0 0 0 0 0 0 0 0 1300 0 0 0 0 0 0
17 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1300 0 0
18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 770 0
19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1200
20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

Hình 3: Định dạng file đầu vào (bao gồm trọng số) dùng cho thuật toán UCS

3.2.2 Các hàm phụ trợ

Các hàm phụ trợ được định nghĩa trong file `utilities.py` như sau:

```

1 # Doc du lieu tu file text
2 def read_txt(file):
3     size = int(file.readline())
4     start, goal = [int(num) for num in file.readline().split(' ')]
5     matrix = [[int(num) for num in file.readline().split(' ')] for num in range(size)
6               ]
7     return size, start, goal, matrix
8
9 # Chuyen ma tran thanh danh sach ke
10 def convert_graph(a):
11     adjList = defaultdict(list)
12     for i in range(len(a)):
13         for j in range(len(a[i])):
14             if a[i][j] == 1:
15                 adjList[i].append(j)
16     return adjList
17
18 def convert_graph_weight(a):
19     adjList = defaultdict(list)
20     for i in range(len(a)):
21         for j in range(len(a[i])):
22             if a[i][j] != 0:
23                 adjList[i].append((j, a[i][j]))
24     return adjList

```

3.2.3 Cài đặt thuật toán BFS

Thuật toán BFS ban đầu được cài đặt bằng hàm `bfs` trong file `bfs.py` sau:

```

1 def bfs(graph, start, end):
2     visited = []
3     frontier = Queue()
4
5     # Them node start vao frontier va visited
6     frontier.put(start)
7     visited.append(start)
8
9     # Node start không có node cha
10    parent = dict()
11    parent[start] = None
12
13    path_found = False
14
15    while True:
16        if frontier.empty():
17            raise Exception("No path found")
18        current_node = frontier.get()
19        visited.append(current_node)
20
21        # Kiểm tra current_node có là node end không
22        if current_node == end:
23            path_found = True
24            break
25
26        for node in graph[current_node]:
27            if node not in visited:
28                frontier.put(node)

```



```

29         parent[node] = current_node
30         visited.append(node)
31
32     # Xây dựng đường đi
33     path = []
34     if path_found:
35         path.append(end)
36         while parent[end] is not None:
37             path.append(parent[end])
38             end = parent[end]
39         path.reverse()
40
41     return path

```

Kết quả sử dụng thuật toán BFS:

[0, 2, 7, 8, 10, 14, 15, 16, 17]

Hình 4: Kết quả của hàm bfs

3.2.4 Cài đặt thuật toán DFS

Thuật toán DFS ban đầu được cài đặt bằng hàm `dfs` trong file `dfs.py` sau:

```

1 def dfs(graph, start, end):
2     visited = []
3     frontier = []
4
5     # Thêm node start vào frontier và visited
6     frontier.append(start)
7     visited.append(start)
8
9     # start không có node cha
10    parent = dict()
11    parent[start] = None
12
13    path_found = False
14    while True:
15        if frontier == []:
16            raise Exception("No path found")
17        current_node = frontier.pop()
18        visited.append(current_node)
19
20        # Kiểm tra current_node có là node end không
21        if current_node == end:
22            path_found = True
23            break
24
25        for node in graph[current_node]:
26            if node not in visited:
27                frontier.append(node)
28                parent[node] = current_node
29                visited.append(node)
30
31    # Xây dựng đường đi
32    path = []
33    if path_found:
34        path.append(end)

```

```

35     while parent[end] is not None:
36         path.append(parent[end])
37         end = parent[end]
38     path.reverse()
39
40     return path

```

Kết quả sử dụng thuật toán DFS:
[0, 3, 4, 13, 10, 14, 15, 16, 17]

Hình 5: Kết quả của hàm dfs

3.2.5 Cài đặt thuật toán UCS

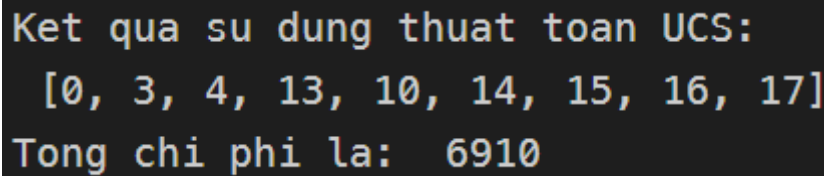
Thuật toán UCS ban đầu được cài đặt bằng hàm ucs trong file ucs.py sau:

```

1 def ucs(graph, start, end):
2     visited = []
3     frontier = PriorityQueue()
4
5     # Thêm node start vào frontier và visited
6     frontier.put((0, start))
7     visited.append(start)
8
9     # Node start không có node cha
10    parent = dict()
11    parent[start] = None
12
13    path_found = False
14    while True:
15        if frontier.empty():
16            raise Exception("No path found")
17        current_weight, current_node = frontier.get()
18        visited.append(current_node)
19
20        # Kiểm tra current_node có là node end không
21        if current_node == end:
22            path_found = True
23            break
24
25        for node_i in graph[current_node]:
26            node, weight = node_i
27            if node not in visited:
28                frontier.put((current_weight + weight, node))
29                parent[node] = current_node
30                visited.append(node)
31
32    # Xây dựng đường đi
33    path = []
34    if path_found:
35        path.append(end)
36        while parent[end] is not None:
37            path.append(parent[end])
38            end = parent[end]
39        path.reverse()
40

```

```
41 return current_weight, path
```



Ket qua su dung thuat toan UCS:
[0, 3, 4, 13, 10, 14, 15, 16, 17]
Tong chi phi la: 6910

Hình 6: Kết quả của hàm ucs

3.3 Xây dựng lớp Graph riêng

Để thuận tiện hơn, thay vì cài đặt riêng các hàm rời rạc như trên, ta có thể xây dựng một lớp Graph và cài đặt tất cả các hàm vào trong lớp này để thuận tiện cho việc khởi tạo đối tượng và chạy các thuật toán. Lớp Graph được cài đặt trong file graph.py sau:

```
1 from queue import Queue, PriorityQueue
2 from collections import defaultdict
3
4
5 class Graph:
6     def __init__(self):
7         self.adjacency_matrix = None
8         self.adjacency_list = None
9         self.start_node = None
10        self.end_node = None
11        self.has_weight = False
12        self.size = 0
13
14        def read_file_text(self, file_name="input.txt"):
15            file = open(file_name, "r")
16            self.size = int(file.readline())
17            self.start_node, self.end_node = [int(num) for num in file.readline().split('
18            ')]
19            self.adjacency_matrix = [[int(num) for num in file.readline().split(' ')] for
20            num in range(self.size)]
21            return self.size, self.start_node, self.end_node, self.adjacency_matrix
22
23        def write_file_text(self, file_name="output.txt"):
24            file = open(file_name, "w")
25            pass
26
27        def convert_to_list(self, weight=None):
28            self.adjacency_list = defaultdict(list)
29            for i in range(len(self.adjacency_matrix)):
30                for j in range(len(self.adjacency_matrix[i])):
31                    if self.adjacency_matrix[i][j] == 1 and weight is None:
32                        self.adjacency_list[i].append(j)
33                    if self.adjacency_matrix[i][j] != 0 and weight is not None:
34                        self.adjacency_list[i].append((j, self.adjacency_matrix[i][j]))
35                    self.has_weight = True
36            return self.adjacency_list
37
38        def dfs(self, start_node, end_node):
39            visited = []
40            frontier = []
```

```

39         frontier.append(start_node)
40         visited.append(start_node)
41
42         parent = dict()
43         parent[start_node] = None
44
45         path_found = False
46         while True:
47             if frontier == []:
48                 raise Exception("No path found")
49             current_node = frontier.pop()
50             visited.append(current_node)
51
52             if current_node == end_node:
53                 path_found = True
54                 break
55
56             for node in self.adjacency_list[current_node]:
57                 if node not in visited:
58                     frontier.append(node)
59                     parent[node] = current_node
60                     visited.append(node)
61
62         path = []
63         if path_found:
64             path.append(end_node)
65             while parent[end_node] is not None:
66                 path.append(parent[end_node])
67                 end_node = parent[end_node]
68             path.reverse()
69
70         return path
71
72     def bfs(self, start_node, end_node):
73         visited = []
74         frontier = Queue()
75
76         frontier.put(start_node)
77         visited.append(start_node)
78
79         parent = dict()
80         parent[start_node] = None
81
82         path_found = False
83
84         while True:
85             if frontier.empty():
86                 raise Exception("No path found")
87             current_node = frontier.get()
88             visited.append(current_node)
89
90             if current_node == end_node:
91                 path_found = True
92                 break
93
94             for node in self.adjacency_list[current_node]:
95                 if node not in visited:
96                     frontier.put(node)
97                     parent[node] = current_node
98                     visited.append(node)

```

```

100
101     path = []
102     if path_found:
103         path.append(end_node)
104         while parent[end_node] is not None:
105             path.append(parent[end_node])
106             end_node = parent[end_node]
107         path.reverse()
108
109     return path
110
111 def ucs(self, start_node, end_node):
112     visited = []
113     frontier = PriorityQueue()
114
115     frontier.put((0, start_node))
116     visited.append(start_node)
117
118     parent = dict()
119     parent[start_node] = None
120
121     path_found = False
122
123     while True:
124         if frontier.empty():
125             raise Exception("No path found")
126         current_weight, current_node = frontier.get()
127         visited.append(current_node)
128
129         if current_node == end_node:
130             path_found = True
131             break
132
133         for node_i in self.adjacency_list[current_node]:
134             node, weight = node_i
135             if node not in visited:
136                 frontier.put((current_weight + weight, node))
137                 parent[node] = current_node
138                 visited.append(node)
139
140     path = []
141     if path_found:
142         path.append(end_node)
143         while parent[end_node] is not None:
144             path.append(parent[end_node])
145             end_node = parent[end_node]
146         path.reverse()
147
148     return path, current_weight

```

4 Nhận xét

- Kết quả đường đi của các thuật toán tìm kiếm đường đi trên đồ thị vô hướng không có trọng số và có trọng số khi chạy tay đều không giống với kết quả sau khi code.
- Trọng số thấp nhất của thuật toán UCS sau khi chạy trên máy giống với kết quả khi chạy tay.
- Thuật toán BFS ưu tiên duyệt node theo chiều rộng (theo thứ tự từ trái sang phải) nên đường đi của nó sẽ ngắn hơn so với DFS.

```
BFS path:  [0, 2, 7, 8, 10, 14, 15, 16, 17]  
DFS path:  [0, 3, 4, 13, 10, 14, 15, 16, 17]  
UCS path:  [0, 2, 7, 8, 10, 14, 15, 16, 17]  
with min weight:  6910
```

Hình 7: Kết quả các thuật toán sau khi cài đặt class Graph

- Thuật toán UCS ưu tiên duyệt node theo chiều sâu nhưng ưu tiên duyệt node có trọng số thấp nhất nên đường đi của nó sẽ có tổng chi phí thấp hơn nhiều so với BFS.