

BÁO CÁO BÀI THỰC HÀNH 3

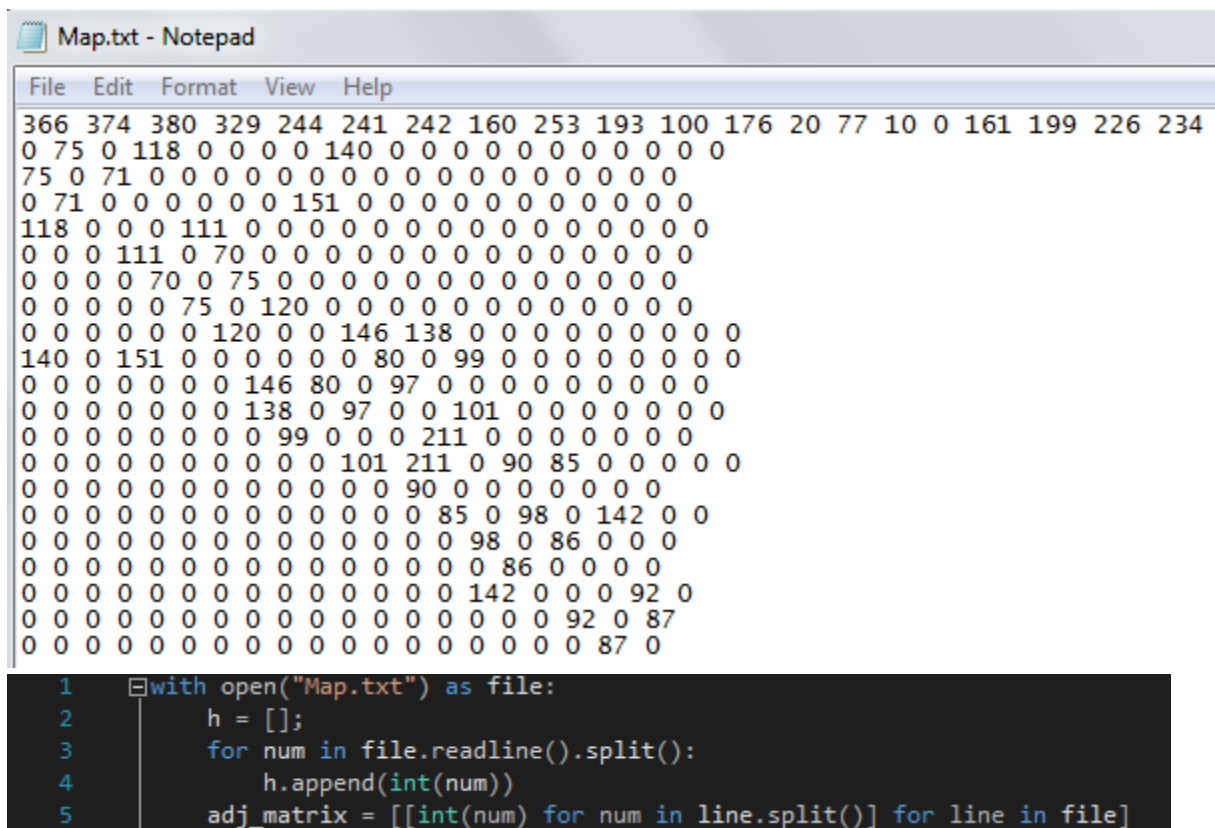
I. Lưu ý về cách thức làm bài:

Do người làm không có khả năng làm theo cách được hướng dẫn trong file hướng dẫn nên đã làm theo một cách khác.

Theo đó, ta có một file .txt đầu vào chứa thông tin cần thiết.

Hàm tìm kiếm cũng khác với hàm được hướng dẫn.

II. Đọc file .txt đầu vào:



```
Map.txt - Notepad
File Edit Format View Help
366 374 380 329 244 241 242 160 253 193 100 176 20 77 10 0 161 199 226 234
0 75 0 118 0 0 0 0 140 0 0 0 0 0 0 0 0 0 0 0
75 0 71 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 71 0 0 0 0 0 0 151 0 0 0 0 0 0 0 0 0 0 0
118 0 0 0 111 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 111 0 70 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 70 0 75 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 75 0 120 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 120 0 0 146 138 0 0 0 0 0 0 0 0
140 0 151 0 0 0 0 0 80 0 99 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 146 80 0 97 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 138 0 97 0 0 101 0 0 0 0 0 0
0 0 0 0 0 0 0 0 99 0 0 0 211 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 101 211 0 90 85 0 0 0 0
0 0 0 0 0 0 0 0 0 0 90 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 85 0 98 0 142 0 0
0 0 0 0 0 0 0 0 0 0 98 0 86 0 0 0
0 0 0 0 0 0 0 0 0 0 0 86 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 142 0 0 92 0
0 0 0 0 0 0 0 0 0 0 0 0 0 92 0 87
0 0 0 0 0 0 0 0 0 0 0 0 0 0 87 0

1 with open("Map.txt") as file:
2     h = [];
3     for num in file.readline().split():
4         h.append(int(num))
5     adj_matrix = [[int(num) for num in line.split()] for line in file]
```

Hàng đầu: Hàm heuristic

Các hàng còn lại: Ma trận, với i và j chạy từ 0 đến 19, $A_{ij} = x > 0$ nghĩa là có đường đi từ node i đến node j với chi phí là x . Ngược lại, $A_{ij} = 0$

Tên địa điểm và số tương ứng:

Arad 0	Craiova 7	Urziceni 14
Zerind 1	Sibiu 8	Hirsova 15
Oradea 2	Rimnicu Vilcea 9	Eforie 16
Timisoara 3	Pitesti 10	Vaslui 17
Lugoj 4	Fagaras 11	Iasi 18
Mehadia 5	Bucharest 12	Nearmt 19
Drobeta 6	Giurgiu 13	

III. Thuật toán tìm kiếm A*:

```
7 from queue import PriorityQueue
8 def aStar(start, end, heu, graph):
9     frontier = PriorityQueue()
10    frontier.put([0, [0, [start]]])
11    explored = []
12    while True:
13        if frontier.empty():
14            raise Exception("No way found")
15        f_value, cost_and_path = frontier.get()
16        move_cost, path = cost_and_path
17        #print(type(path), "Current path:", path)
18        #print(type(move_cost), "Current cost:", move_cost)
19        #print(type(f_value), "Current cost + heuristic:", f_value, '\n')
20        current = path[-1]
21        explored.append(current)
22        if current == end:
23            name = ["Arad", "Zerind", "Oradea", "Timisoara", "Lugoj", "Mehadia", "Drobeta",
24                  "Craiova", "Sibiu", "Rimnicu Vilcea", "Pitesti", "Fagaras", "Bucharest",
25                  "Giurgiu", "Urziceni", "Hirsova", "Eforie", "Vaslui", "Iasi", "Neamt"]
26            for i in range(len(path)):
27                change = path[i]
28                path[i] = name[change]
29            print("Solution for A* search:\n", path, "\nCost:", move_cost)
30            return
31        i = -1
32        for cost in graph[current]:
33            i = i + 1
34            if cost > 0:
35                if i not in explored:
36                    new_path = list(path)
37                    new_path.append(i)
38                    frontier.put([move_cost + cost + heu[i], [move_cost + cost, new_path]])
```

“Uncomment các dòng comment đã thấy rõ, *path*, *move_cost*, *f_value* trong mỗi lần lặp”

Hàng đợi ưu tiên *frontier* chứa các phần tử có dạng $[a, [b, [c]]]$.

c: List chứa các số, đại diện cho đường đi

b: Chi phí để đi được c

a: Tổng của b và heuristic của phần tử cuối cùng của c. Giá trị a quyết định vị trí của phần tử trong hàng đợi ưu tiên. Ví dụ, với đường đi c là $[0, 1, 2]$, chi phí b là 80, heuristic của phần tử cuối cùng của c là 120, ta sẽ có giá trị $a = 80 + 120 = 200$. Vậy ta có $[200, [80, [0, 1, 2]]]$

Khi in *path*, tức kết quả của bài toán, ta sẽ cần chuyển có số nguyên trong *path* thành các tên tương ứng như đã nêu ở đầu file báo cáo.

Nhìn chung, hàm tìm kiếm A* có cách hoạt động khá giống với Uniform-cost Search với sự bổ sung heuristic. Nhờ vậy, việc mở rộng node sẽ tập trung hơn đến node kết thúc, những node đi không hướng đến đích đến sẽ có giá trị heuristic cao, do đó sẽ bị đẩy xuống cuối hàng đợi và chúng ta gần như sẽ không lấy ra các node này được.

Thuật toán tìm kiếm Uniform-cost Search và giải thích:

```
65 from queue import PriorityQueue
66 def ucs(graph, start, end):
67     frontier = PriorityQueue()
68     frontier.put([0, [start]]) # (priority, node)
69     explored = []
70     while True:
71         if frontier.empty():
72             raise Exception("No way found")
73         current_cost, path = frontier.get()
74         current_node = path[-1]
75         #print(type(current_cost), "Current cost =", current_cost)
76         #print(type(path), "Current path =", path)
77         #print(type(current_node), "Current node =", current_node, '\n')
78         explored.append(current_node)
79         if current_node == end:
80             print("Solution for UCS:", path, "- Cost:", current_cost)
81             return
82         i = -1
83         for cost in graph[current_node]:
84             i = i + 1
85             if (cost > 0):
86                 if i not in explored:
87                     new_path = list(path)
88                     new_path.append(i)
89                     frontier.put([current_cost + cost, new_path])
```

frontier là hàng đợi ưu tiên (PriorityQueue) chứa "List chứa 1 số nguyên là chi phí đường đi (cost) và CON ĐƯỜNG dưới dạng 1 list".

Ví dụ: [150, [0,2,8]] nghĩa là để đi được đoạn đường từ node 0 đến node 2 đến node 8, chi phí sẽ là 150. Số nguyên đại diện cho chi phí này là yếu tố quyết định vị trí của phần tử trong hàng đợi ưu tiên, số nhỏ hơn xếp trước số lớn hơn.

Trong mỗi lần lặp, *frontier.get()* sẽ trả ra chi phí và con đường, được gán đồng thời vào *current_cost* và *path*. Tương tự BDS và DFS, đặt *current_node = path[-1]*. Nếu *current_node == end*, tức node kết thúc, ta sẽ in ra kết quả là *path*.

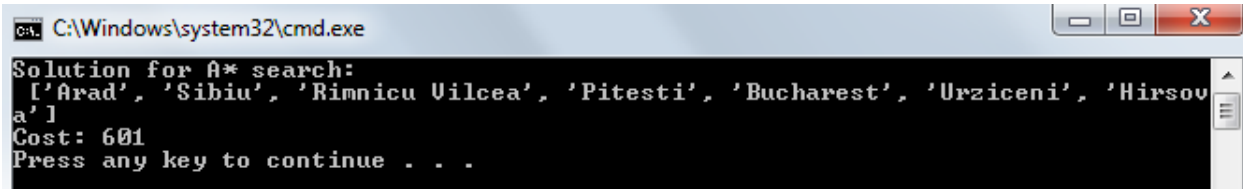
Graph mà ta truyền vào khi này sẽ chứa chi phí khi đi từ node này đến node khác, với 0 nghĩa là không có đường đi. Do đó biến *node* ở BFS và

DFS sẽ được thay bằng *cost* để đúng ý nghĩa. Điều kiện *cost == 0* khi này được thay bằng *cost > 0*. Khi điều kiện này thỏa, ta tiến hành tạo list *new_path* với biến đếm *i* như BFS và DFS, nhưng khi này ta sẽ thêm *new_path* vào *frontier* chung với *current_cost + cost* trong một list, với *current_cost + cost* là chi phí của *new_path*.

IV. Chạy Code:

```
34 aStar(0, 15, h, adj_matrix)
```

Nhắc lại, 0 đại diện cho Arad, 15 đại diện cho Hirsova.



```
C:\Windows\system32\cmd.exe
Solution for A* search:
['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest', 'Urziceni', 'Hirsova']
Cost: 601
Press any key to continue . . .
```