Bài tập Thực hành Nhập môn Trí tuệ Nhân tạo tuần 3

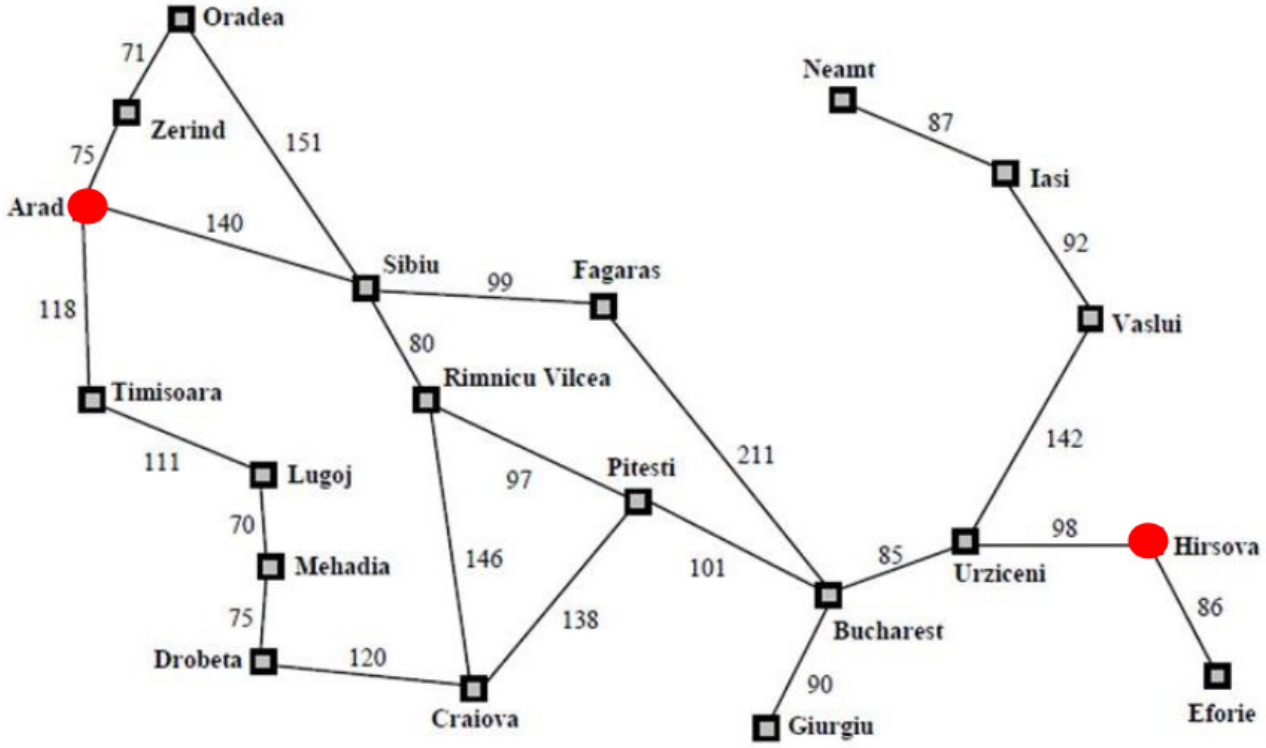
Nguyễn Lê Ngọc Duy - 20280023 - 20KDL1

Mục lục

1 Bài toán tìm đường đi ngắn nhất

Bài toán tìm đường đi ngắn nhất có thể được xem là bài toán kinh điển của lĩnh vực Khoa học máy tính, Lý thuyết đồ thị nói chung và ngành Trí tuệ nhân tạo nói riêng. Bài toán được phát biểu như sau: tìm một đường đi giữa hai đỉnh sao cho tổng các trọng số của các cạnh tạo nên đường đi đó là nhỏ nhất.
Có nhiều thuật toán được hình thành để giải quyết bài toán này, bao gồm thuật toán Dijkstra (1959) dùng trong trường hợp đồ thị có các trọng số không âm, hay thuật toán Breadth First Search dùng khi đồ thị không có trọng số.
Trong bài báo cáo này, chúng ta sẽ tập trung vào hai thuật toán: Greedy Best First Search (GBFS) và A-star Search (A*). Hai thuật toán này được sử dụng cho đồ thị có trọng số không âm. Sau đó, chúng ta sẽ sử dụng hai thuật toán này để giải quyết bài toán tìm đường đi ngắn nhất trên đồ thị có hướng ở hình 1.
Để thuận tiện, ta sẽ đánh dấu thứ tự các đỉnh như bảng sau:

Arad: 1	Bucharest: 2	Craiova: 3	Drobeta: 4
Eforie: 5	Fagaras: 6	Giurgiu: 7	Hirsova: 8
Iasi: 9	Lugoj: 10	Mehadia: 11	Neamt: 12
Oradea: 13	Pitesti: 14	Rimnicu Vilcea: 15	Sibiu: 16
Timisora: 17	Urziceni: 18	Vaslui: 19	Zerind: 20



Hình 1: Đồ thị sử dụng để minh hoạ bài toán tìm đường đi ngắn nhất

2 Giải thuật Heuristic

Heuristic trong tiếng Hy lạp cổ nghĩa là **tìm kiếm** hay **khám phá**. Heuristic là các kỹ thuật dựa trên kinh nghiệm để giải quyết vấn đề, học hỏi hay khám phá nhằm đưa ra một giải pháp gần như là tối ưu. Các phương pháp heuristic được sử dụng nhằm tăng nhanh quá trình tìm kiếm với các phương án, giải pháp hợp lý (gần như là tối ưu) để giảm bớt việc nhận thức vấn đề khi đưa ra quyết định.
Giải thuật heuristic là một sự mở rộng khái niệm thuật toán, thể hiện cách giải bài toán với các đặc tính sau:

- Thường tìm được lời giải tốt (nhưng chưa chắc là tốt nhất).
- Thường dễ dàng và nhanh chóng đưa ra kết quả hơn so với một số thuật giải tối ưu, do đó có chi phí thấp hơn.
- Thường có vẻ khá tự nhiên, gần gũi với suy nghĩ và hành động của con người.
- Có nhiều phương pháp để xây dựng một thuật giải Heuristic, trong đó người ta thường đưa vào một số nguyên lý cơ sở sau:
  - **Nguyên lý vét cạn thông minh**: Trong một bài toán tìm kiếm nào đó, khi không gian tìm kiếm lớn, ta thường tìm cách giới hạn lại không gian tìm kiếm hoặc thực hiện một kiểu dò tìm đặc biệt dựa vào đặc thù của bài toán để nhanh chóng tìm ra trạng thái mục tiêu.
  - **Nguyên lý tham lam (Greedy)**: Lấy tiêu chuẩn tối ưu (trên phạm vi toàn cục) của bài toán để làm tiêu chuẩn lựa chọn hành động cho phạm vi cục bộ của từng bước (hay từng giai đoạn) trong quá trình tìm kiếm lời giải.
  - **Nguyên lý thứ tự**: Thực hiện hành động dựa trên một cấu trúc thứ tự hợp lý của không gian khảo sát nhằm nhanh chóng đạt được một lời giải tốt.
  - **Hàm Heuristic**: Trong việc xây dựng các thuật giải Heuristic, người ta thường dùng các *Hàm Heuristic*. Đó là các hàm đánh giá thô, giá trị của hàm phụ thuộc vào trạng thái hiện tại của bài toán ở mỗi bước giải. Nhờ giá trị này, ta có thể chọn được cách hành động tương đối hợp lý trong từng bước của thuật giải.

3 Best First Search

Best First Search là một nhánh thuộc lớp các bài toán tìm kiếm. Theo đó, **Best First Search** ưu tiên duyệt đồ thị theo trình tự các trạng thái tuân theo một quy luật nào đó.
Judea Pearl (nhà khoa học máy tính sinh năm 1936, người phát triển lý thuyết mạng Bayesian) miêu tả thuật toán **Best First Search** bằng việc ước lượng khả năng phù hợp của node n bằng một hàm đánh giá $f(n)$ tổng quát dựa trên miêu tả của node n, miêu tả của node mục tiêu, tri thức ban đầu có được, và quan trọng nhất, chính là tri thức bổ sung cho bài toán được đặt ra.
Lời giải phù hợp có thể được cài đặt bằng cách sử dụng một hàng đợi ưu tiên (priority queue).
Hai trường hợp đặc biệt của giải thuật **Best First Search** chính là **Greedy Best First Search** và **A* Search**.

4 Thuật toán Greedy Best First Search (GBFS)

Thuật toán sử dụng hàm đánh giá $f(n)$ chính là hàm Heuristic $h(n)$. Hàm Heuristic $h(n)$ đánh giá chi phí ước lượng để đi từ node hiện tại n để node đích (mục tiêu). Thuật toán GBFS sẽ ưu tiên xét node "có vẻ"gần với node đích (mục tiêu) nhất.
Mã giả cho thuật toán được thể hiện ở bên dưới:

Thuật toán 1 Thuật toán Greedy Best First Search (GBFS)

Đầu vào: Bài toán.
Đầu ra: Lời giải hoặc thông báo: Không tồn tại lời giải.

```
1: insert(state = initial_state, priority = 0) into search.queue;
2: while search.queue not empty do
3:   current_queue.entry = pop item from front of search.queue
4:   current_state = current_queue.entry.state;
5:   current_heuristic = current_queue.entry.heuristic;
6:   starting_counter = counter from current_queue.entry;
7:   applicable_actions = array of actions applicable in current_state;
8:   for all index ?i in applicable_actions >= starting_counter do
9:     current_action = applicable_actions[?i];
10:    successor_state = current_state.apply(current_action);
11:    if successor_state is goal state then
12:      return solution_path;
13:    end if
14:    successor_heuristic = heuristic value of successor_state;
15:    if successor_heuristic < current_heuristic then
16:      insert(current_state, current_heuristic, ?i + 1) to search.queue;
17:      insert(successor_state, successor_heuristic, 0) to search.queue;
18:      break for;
19:    else
20:      insert(successor_state, successor_heuristic, 0) to search.queue;
21:    end if
22:  end for
23: end while
```

Ví dụ, đối với đồ thị ở hình 1, giả sử ta có hàm Heuristic $h(n)$ được xác định như ở bảng sau:

h(Arad) = 366	h(Bucharest) = 20	h(Craiova) = 160	h(Drobeta) = 242
h(Eforie) = 161	h(Fagaras) = 176	h(Giurgiu) = 77	h(Hirsova) = 0
h(Iasi) = 226	h(Lugoj) = 244	h(Mehadia) = 241	h(Neamt) = 234
h(Oradea) = 380	h(Pitesti) = 100	h(Rimnicu Vilcea) = 193	h(Sibiu) = 253
h(Timisoara) = 329	h(Urziceni) = 10	h(Vaslui) = 199	h(Zerind) = 374

Áp dụng thuật toán GBFS, ta có lời giải cho bài toán tìm đường đi ngắn nhất từ Arad (1) đến Hirsova (8) được thể hiện như bảng sau:

Lần lặp	**Đỉnh**	**Hàng đợi ưu tiên (theo thứ tự hàm $h(n)$)**	**Tổng chi phí $f(n)$**
0	$\emptyset$	(1, 366)*	0
1	1	(16, 253)*, (17, 329), (1, 366), (20, 374)	$0 + 140 = 140$
2	16	(6, 176)*, (15, 193), (1, 366), (15, 380)	$140 + 99 = 239$
3	6	(2, 20)*, (16, 253)	$239 + 211 = 450$
4	2	(14, 10)*, (7, 77), (6, 176)	$450 + 85 = 535$
5	14	(8, 0)*, (2, 20), (19, 199)	$535 + 98 = 633$

Vậy đường đi ngắn nhất từ Arad đến Hirsova thông qua thuật toán GBFS là:

$$\text{Arad} \xrightarrow{140} \text{Sibiu} \xrightarrow{99} \text{Fagaras} \xrightarrow{211} \text{Bucharest} \xrightarrow{85} \text{Urziceni} \xrightarrow{98} \text{Hirsova}$$

5 Thuật toán A-star (A*)

A* là một giải thuật tìm kiếm trong đồ thị, tìm đường đi từ một đỉnh hiện tại đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách. Hàm ước lượng này có dạng $f(n) = g(n) + h(n)$, trong đó:

- $g(n)$ là chi phí đi từ node gốc đến node đó.
- $h(n)$ là chi phí ước lượng từ node đó đến node đích.

Mã giả cho thuật toán A* được mô tả như ở trang 8.

Áp dụng thuật toán A* kết hợp với hàm Heuristic $h(n)$ đã cho ở mục 4.1, ta có lời giải cho bài toán tìm đường đi ngắn nhất từ Arad (1) đến Hirsova (8) được thể hiện như sau: (Mỗi node trong tập OPEN và CLOSE được thể hiện bởi 5 giá trị: số thứ tự của thành phố, giá trị hàm $g(n)$, giá trị hàm $h(n)$ giá trị hàm $h(n)$ và số thứ tự của node cha tương ứng).

Sau khi kết thúc thuật toán, ta thu được đường đi ngắn nhất từ Arad đến Hirsova theo thuật toán A* là

$$\text{Arad} \xrightarrow{140} \text{Sibiu} \xrightarrow{80} \text{Rimincu Vilcea} \xrightarrow{97} \text{Pitesti} \xrightarrow{101} \text{Bucharest} \xrightarrow{85} \text{Urziceni} \xrightarrow{98} \text{Hirsova}$$

Lần lặp	OPEN	CLOSE
0	(1, 0, 0, 0, ∅)*	∅
1	(16, 140, 253, 393, 1)* (17, 118, 329, 447, 1) (20, 75, 374, 449, 1)	(1, 0, 0, 0, ∅)
2	(17, 118, 329, 447, 1) (20, 75, 374, 449, 1) (15, 220, 193, 413, 16)* (6, 239, 176, 415, 16) (1, 280, 366, 646, 16) (13, 291, 380, 671, 16)	(1, 0, 0, 0, ∅) (16, 140, 253, 393, 1)
3	(17, 118, 329, 447, 1) (20, 75, 374, 449, 1) (6, 239, 176, 415, 16)* (1, 280, 366, 646, 16) (13, 291, 380, 671, 16) (14, 317, 100, 417, 15) (3, 366, 160, 526, 15) (16, 300, 253, 553, 15)	(1, 0, 0, 0, ∅) (16, 140, 253, 393, 1) (15, 220, 193, 413, 16)
4	(17, 118, 329, 447, 1) (20, 75, 374, 449, 1) (1, 280, 366, 646, 16) (13, 291, 380, 671, 16) (14, 317, 100, 417, 15)* (3, 366, 160, 526, 15) (16, 300, 253, 553, 15) (2, 450, 0, 450, 6)	(1, 0, 0, 0, ∅) (16, 140, 253, 393, 1) (15, 220, 193, 413, 16)
5	(17, 118, 329, 447, 1) (20, 75, 374, 449, 1) (1, 280, 366, 646, 16) (13, 291, 380, 671, 16) (3, 366, 160, 526, 15) (16, 300, 253, 553, 15) (2, 418, 0, 418, 6)* (15, 414, 193, 604, 14)	(1, 0, 0, 0, ∅) (16, 140, 253, 393, 1) (15, 220, 193, 413, 16) (14, 317, 100, 417, 15)
6	(17, 118, 329, 447, 1)* (20, 75, 374, 449, 1) (1, 280, 366, 646, 16) (13, 291, 380, 671, 16) (3, 366, 160, 526, 15) (16, 300, 253, 553, 15) (15, 414, 193, 604, 14) (6, 629, 176, 805, 2) (7, 508, 77, 585, 2) (18, 503, 10, 513, 2)	(1, 0, 0, 0, ∅) (16, 140, 253, 393, 1) (15, 220, 193, 413, 16) (14, 317, 100, 417, 15) (2, 418, 0, 418, 6)

Lần lặp	OPEN	CLOSE
7	(20, 75, 374, 449, 1)* (1, 280, 366, 565, 17) (13, 291, 380, 671, 16) (3, 366, 160, 526, 15) (16, 300, 253, 553, 15) (15, 414, 193, 604, 14) (6, 629, 176, 805, 2) (7, 508, 77, 585, 2) (18, 503, 10, 513, 2) (10, 299, 244, 543, 17)	(1, 0, 0, 0, ∅) (16, 140, 253, 393, 1) (15, 220, 193, 413, 16) (14, 317, 100, 417, 15) (2, 418, 0, 418, 6) (17, 118, 329, 447, 1)
8	(1, 280, 366, 524, 20) (13, 291, 380, 671, 16) (3, 366, 160, 526, 15) (16, 300, 253, 553, 15) (15, 414, 193, 604, 14) (6, 629, 176, 805, 2) (7, 508, 77, 585, 2) (18, 503, 10, 513, 2)* (10, 299, 244, 543, 17) (13, 146, 380, 526, 20)	(1, 0, 0, 0, ∅) (16, 140, 253, 393, 1) (15, 220, 193, 413, 16) (14, 317, 100, 417, 15) (2, 418, 0, 418, 6)
9	(1, 280, 366, 524, 20) (13, 291, 380, 671, 16) (3, 366, 160, 526, 15) (16, 300, 253, 553, 15) (15, 414, 193, 604, 14) (6, 629, 176, 805, 2) (7, 508, 77, 585, 2) (10, 299, 244, 543, 17) (13, 146, 380, 526, 20) (2, 588, 20, 608, 18) (8, 601, 0, 601, 18) (dừng)	(1, 0, 0, 0, ∅) (16, 140, 253, 393, 1) (15, 220, 193, 413, 16) (14, 317, 100, 417, 15) (2, 418, 0, 418, 6) (18, 503, 10, 513, 2)

Thuật toán 2 Thuật toán A*

Đầu vào: Bài toán.
Đầu ra: Lời giải hoặc thông báo: Không tồn tại lời giải.

OPEN $= T_0$
$g(T_0) = 0, h(T_0) = 0, f(T_0) = 0$
CLOSE $= \emptyset$
loop
  if OPEN rỗng **then**
    Bài toán vô nghiệm.
    exit
  else
    Lấy $T_{\max}$ ra khỏi OPEN
    Đưa $T_{\max}$ vào CLOSE
    if $T_{\max}$ là trạng thái đích **then**
      Lời giải là $T_{\max}$
      exit
    else
      Tạo danh sách tất cả các trạng thái kế tiếp T_K của $T_{\max}$
      for each T_K **do**
        $g(T_K) = g(T_{\max}) + \text{cost}(T_{\max}, T_K)$
        if tồn tại $T_{K'}$ trong OPEN trùng với T_K **then**
          if $g(T_K) < g(T_{K'})$ **then**
            $g(T_{K'}) = g(T_K)$
            Tính lại $f(T_{K'})$
            FATHER$(T_{K'}) = T_{\max}$
          end if
        end if
        if tồn tại $T_{K'}$ trong CLOSE trùng với T_K **then**
          if $g(T_K) < g(T_{K'})$ **then**
            $g(T_{K'}) = g(T_K)$
            Tính lại $f(T_{K'})$
            FATHER$(T_{K'}) = T_{\max}$
          end if
        end if
        Lan truyền sự thay đổi của f và g cho tất cả các trạng thái kế tiếp của T_i (ở tất cả các cấp) đã được lưu trữ trong CLOSE và OPEN.
        if T_K chưa xuất hiện trong cả OPEN và CLOSE **then**
          Thêm T_K vào OPEN
          $f(T_K) = g(T_K) + h(T_K)$
        end if
      end for
    end if
  end if
end loop

6 Cài đặt bằng Python

Trong phần này chúng ta sẽ đi vào chi tiết cách cài đặt chương trình Python để giải quyết bài toán được đặt ở phần 1. File chương trình có tên là `program.py`. Chương trình được viết bằng ngôn ngữ `Python` và sử dụng thư viện `matplotlib` để vẽ giao diện. Chương trình được chia thành 2 phần chính: các hàm và hàm `main`.

6.1 Cấu trúc file đầu vào

Có ba file đầu vào cho chương trình. File đầu tiên là file `cities.txt`. Mỗi dòng trong file gồm 3 thành phần phân cách nhau bởi khoảng trắng. Thành phần đầu tiên là tên thành phố, 2 thành phần còn lại chính là toạ độ của thành phố đó (hình 2a).
File thứ hai là `citiesGraph.txt` gồm 3 thành phần phân cách nhau bởi khoảng trắng. Thành phần đầu tiên là tên thành phố, thành phần thứ hai là tên thành phố kề với thành phố đầu tiên, thành phần thứ ba là khoảng cách giữa hai thành phố đó (hình 2b).
File cuối cùng là `heuristic.txt` gồm 2 thành phần phân cách nhau bởi khoảng trắng. Thành phần đầu tiên là tên thành phố, thành phần thứ hai là hàm Heuristic khoảng cách từ thành phố đó đến thành phố đích (hình 2c).

```
Arad 29 192
Bucharest 268 55
Craiova 163 22
Dobreta 91 32
Eforie 420 28
Fagaras 208 157
Giurgiu 264 8
Hirsova 396 74
Iasi 347 204
Lugoj 91 98
Mehadia 93 65
Neamt 290 229
Oradea 62 258
Pitesti 220 88
Rimnicu_Vilcea 147 124
Sibiu 126 164
Timisoara 32 124
Urziceni 333 74
Vaslui 376 153
Zerind 44 225
```

(a) `cities.txt`

```
Arad Sibiu 140
Arad Timisoara 118
Arad Zerind 75
Bucharest Fagaras 211
Bucharest Giurgiu 90
Bucharest Pitesti 101
Bucharest Urziceni 85
Craiova Dobreta 120
Craiova Pitesti 138
Craiova Rimnicu_Vilcea 146
Dobreta Mehadia 75
Eforie Hirsova 86
Fagaras Sibiu 99
Hirsova Urziceni 98
Iasi Neamt 87
Iasi Vaslui 92
Lugoj Mehadia 70
Lugoj Timisoara 111
Oradea Zerind 71
Oradea Sibiu 151
Sibiu Rimnicu_Vilcea 80
Urziceni Vaslui 142
Rimnicu_Vilcea Pitesti 97
```

(b) `citiesGraph.txt`

```
Arad 366
Bucharest 20
Craiova 160
Dobreta 242
Eforie 161
Fagaras 176
Giurgiu 77
Hirsova 0
Iasi 226
Lugoj 244
Mehadia 241
Neamt 234
Oradea 380
Pitesti 100
Rimnicu_Vilcea 193
Sibiu 253
Timisoara 329
Urziceni 10
Vaslui 199
Zerind 374
```

(c) `heuristic.txt`

Hình 2: Các file đầu vào

6.2 Các hàm

- `getHeuristics()`: đọc dữ liệu từ file `heuristics.txt` và lưu vào `dictionary` tên là `heuristics` với `key` là tên thành phố và `value` là hàm Heuristic của thành phố đó.
- `getCity()`: đọc dữ liệu từ file `cities.txt` và lưu vào 2 `dictionary` tên là `city` và `cityCode`:

  - Đối với `city`, `key` là tên thành phố và `value` là một `list` gồm toạ độ của thành phố đó.
  - Đối với `cityCode`, `key` là số thứ tự cùa hàng chứa tên thành phố đó và `value` là tên thành phố đó.

- `createGraph()`: đọc dữ liệu từ file `citiesGraph.txt` và lưu vào `dictionary` tên là `graph` với `key` là tên thành phố và `value` là một `list` gồm tên thành phố kề với nó và khoảng cách giữa hai thành phố đó.
- `drawMap(city, gbfs, astar, graph)`: mô phỏng đồ thị của bài toán cũng như lời giải cho hai thuật toán GBFS và A*.
- `GBFS(startNode, heuristics, graph, goalNode)`: thực thi thuật toán Greedy Best First Search với node bắt đầu `startNode`, giá trị hàm Heuristics `heuristics`, đồ thị `graph` và node đích `endNode`.
- `Astar(startNode, heuristics, graph, goalNode)`: thực thi thuật toán A-star với node bắt đầu `startNode`, giá trị hàm Heuristics `heuristics`, đồ thị `graph` và node đích `endNode`.

6.3 Hàm main

Nhiệm vụ của hàm main là mô phỏng lại bài toán tìm đường đi ngắn nhất đã được miêu trả ở mục 1. Chương trình sẽ dừng lại khi người dùng nhập giá trị của đỉnh bắt đầu hoặc đỉnh kết thúc là 0:

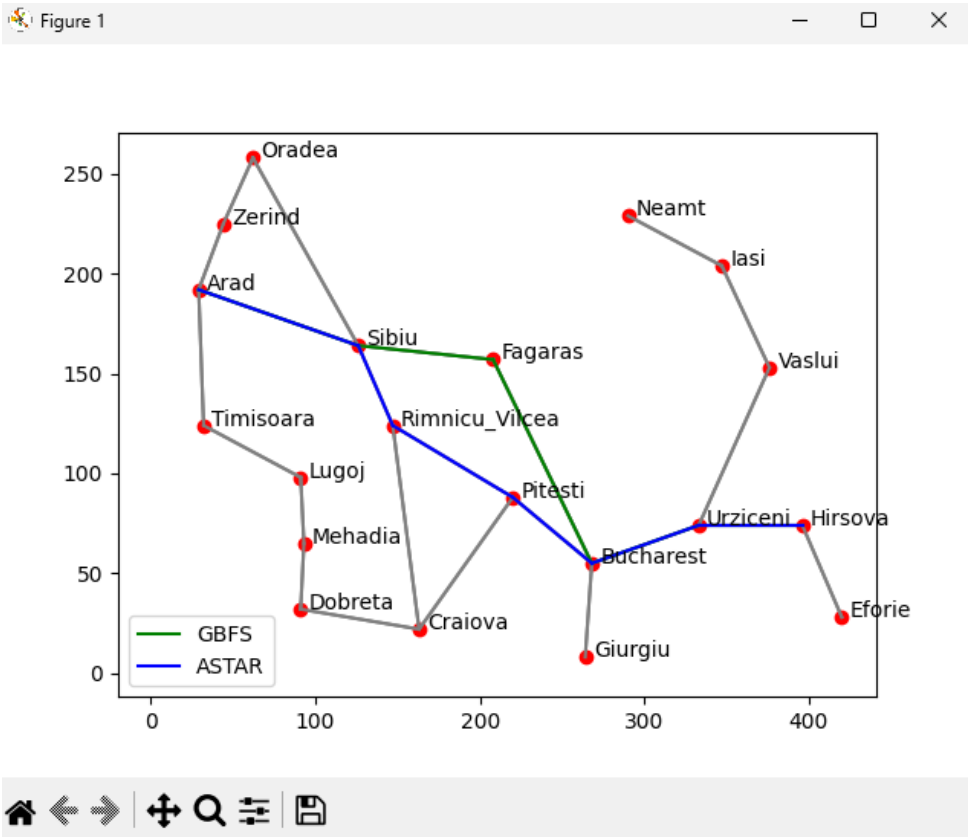
```
if __name__ =="__main__":
   heuristic = getHeuristics()
   graph = createGraph()
   city, citiesCode = getCity()

   print(heuristic)

   for i, j in citiesCode.items():
        print(i, j)

   while True:
        inputCode1 = int(input("Nhap dinh bat dau: "))
        inputCode2 = int(input("Nhap dinh ket thuc: "))

        if inputCode1 ==0 or inputCode2 ==0:
             break

        startCity = citiesCode[inputCode1]
        endCity = citiesCode[inputCode2]

        gbfs = GBFS(startCity, heuristic, graph, endCity)
        astar = Astar(startCity, heuristic, graph, endCity)
        print("GBFS => ", gbfs)
        print("ASTAR =>", astar)

        drawMap(city, gbfs, astar, graph)
```

6.4 Kết quả

Sau khi chạy chương trình trên, ta thu được kết quả đường đi ngắn nhất từ Arad đến Hirsova như hình 3 và được mô phỏng như hình 4:

```
19 Vaslui
20 Zerind
Nhap dinh bat dau: 1
Nhap dinh ket thuc: 8
GBFS =>  ['Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Urziceni', 'Hirsova']
ASTAR => ['Arad', 'Sibiu', 'Rimnicu_Vilcea', 'Pitesti', 'Bucharest', 'Urziceni', 'Hirsova']
```

Hình 3: Kết quả đường đi ngắn nhất từ Arad đến Hirsova bằng Terminal



Hình 4: Mô phỏng đường đi ngắn nhất từ Arad đến Hirsova bằng Matplotlib

7 Nhận xét

- Hai thuật toán sử dụng hai hàm đánh giá khác nhau: đối với GBFS là $f(n) = h(n)$ còn đối với A* là $f(n) = g(n) + h(n)$.
- Thuật toán GBFS là thuật toán **không hoàn thiện** vì thuật toán này không đảm bảo ta sẽ tìm được dường đi từ node bắt đầu đến node đích. Trái lại, thuật toán A* là thuật toán **hoàn thiện** vì thuật toán này luôn đảm bảo đường đi từ node bắt đầu đến node kết thúc là có thể tìm được.
- Thuật toán GBFS tốn ít bộ nhớ hơn, vì các node sau khi được duyệt sẽ luôn được đẩy ra sau mỗi bước lặp. Trong khi đó, thuật toán A* cần phải lưu lại vết của các node đã được duyệt để có thể truy vết lại đường đi ngắn nhất.
- Thuật toán GBFS là thuật toán **không tối ưu**, nghĩa là đường đi tìm được có thể không phải là đường đi ngắn nhất. Ngược lại, thuật toán A* là thuật toán **tối ưu** vì ta luôn tìm được đường đi ngắn nhất từ node bắt đầu đến node kết thúc.
- Cả hai thuật toán đều có độ phức tạp thời gian là $\mathcal{O}(b^m)$ với b là số lượng node con (tối đa) của mỗi node và m là chiều sâu tối đa của cây tìm kiếm.
- Trên thực tế, người ta thường dùng một thuật toán khác để tìm đường đi ngắn nhất, ví dụ như thuật toán Iterative Deepening A* - IDA* với độ phức tạp $\mathcal{O}(d)$ với d là độ sâu của cây.

8 Tham khảo

1. Stuart J. Russell, Peter Norvig (2010), Aritificial Intelligence: A Modern Approach, 3rd Edition, Prentice Hall, ISBN-13: 978-0-13-604259-4, ISBN-10: 0-13-604259-7 (https://zoo.cs.yale.edu/classes/cs470/materials/aima2010.pdf)
2. Greedy Best-First Search - GBFS (2019), https://www.geeksforgeeks.org/greedy-best-first-search-gbfs/
3. A* Search Algorithm (2019), https://www.geeksforgeeks.org/a-search-algorithm/
4. Iterative Deepening A* - IDA* (https://en.wikipedia.org/wiki/Iterative_deepening_A*)