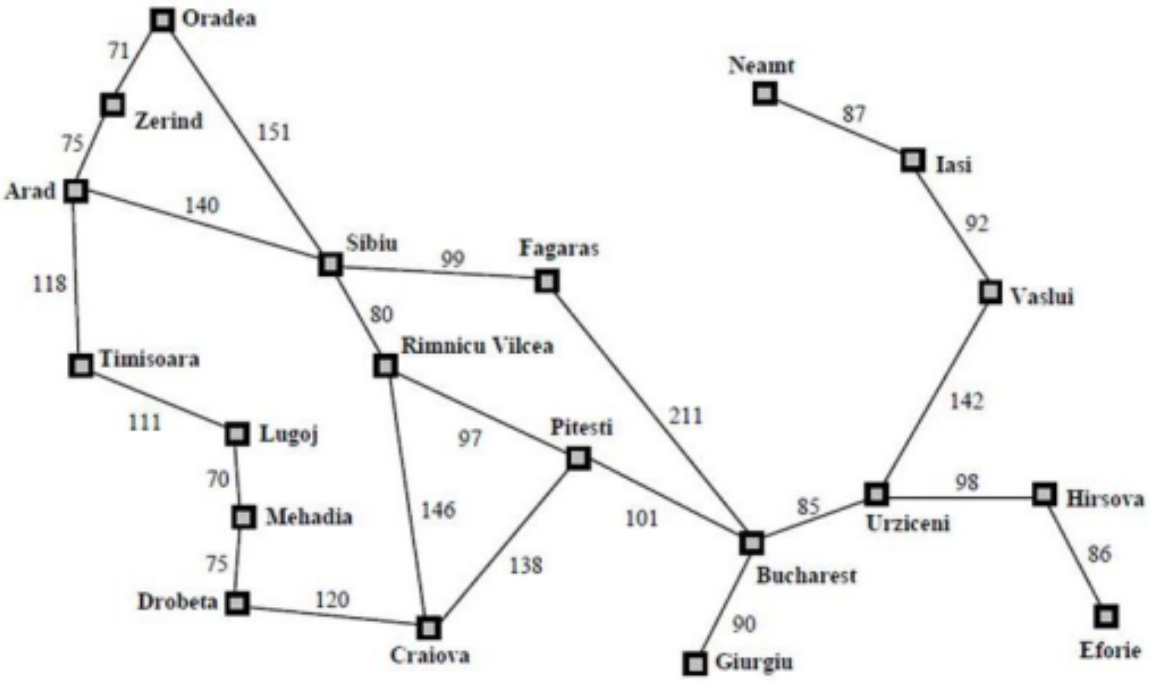
# Report AI

Nguyễn Quốc Bảo

18110053

Lab03

**Cho dữ liệu graph**



# 1. Cài đặt thuật toán Greedy – Best – First search để tìm đường đi từ Arad tới Hirsova như hình với h(n) được xác định như sau:

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

GBFS mở rộng nút gần đích nhất với hi vọng cách làm này sẽ dẫn đến lời giải một cách nhanh nhất. Đánh giá chi phí của các nút chỉ dựa trên hàm heuristic:

$$f(n) = h(n)$$

1. Đọc file

```
def read_txt(file):
    vertices = int(file.readline())
    start, goal = [int(num) for num in file.readline().split(' ')]
    name_ct = [item for item in file.readline()[:-1].split('\t')]
    heuristic = [int(num) for num in file.readline().split('\t')]
    # heuristic = [(heuristic[i],weight[i]) for i in range(size)]
    matrix = [[int(num) for num in line.split('\t')] for line in file]
    return vertices,start,goal,name_ct,heuristic,matrix



def convert_graph(vertices,name_ct,heuristic,matrix):
    graph = Graph(vertices,name_ct,heuristic,matrix)
    for i in range(vertices):
        for j in range(vertices):
            if (matrix[i][j] != 0):
                graph.add_edge(i,j)
    return graph
```

Ta đọc file và chuyển dữ liệu thành graph như hình trên.

2. Xây dựng Class Graph

Với thông số chuyền vào là (số cạnh, tên thành phố, ma trận có chứa khoảng cách đường đi)

```python
class Graph :

    def __init__(self,vertices,name_ct,heuristic,matrix):
        self.heuristic = heuristic
        self.name_ct = name_ct
        self.graph = defaultdict(list)
        self.matrix = matrix
        self.vertices = vertices

    # add edge for graph
    def add_edge(self,src,dest):
        self.graph[src].append((dest,self.name_ct[dest],self.matrix[src][dest],self.heuristic[dest]))

    # Show infomation graph
    def show_graph(self):
        for i in range(self.vertices):
            print((i, self.name_ct[i]),  ':', self.graph[i])
```

Từ đó, ta có thể dễ dàng tạo các cạnh từ điểm đầu đến điểm kết thúc với tên thành phố và giá trị của nó.

```
(0, 'Arad') : [(15, 'Sibiu', 140, 253), (16, 'Timisoara', 118, 329), (19, 'Zerind', 75, 374)]
(1, 'Bucharest') : [(5, 'Fagaras', 211, 176), (6, 'Giurgiu', 90, 77), (13, 'Pitesti', 101, 100), (17, 'Urziceni', 85, 10)]
(2, 'Craiova') : [(3, 'Drobeta', 120, 242), (13, 'Pitesti', 138, 100), (14, 'Rimnicu Vilcea', 146, 193)]
(3, 'Drobeta') : [(2, 'Craiova', 120, 160), (10, 'Mehadia', 75, 241)]
(4, 'Eforie') : [(7, 'Hirsova', 86, 0)]
(5, 'Fagaras') : [(1, 'Bucharest', 211, 20), (15, 'Sibiu', 99, 253)]
(6, 'Giurgiu') : [(1, 'Bucharest', 90, 20)]
(7, 'Hirsova') : [(4, 'Eforie', 86, 161), (17, 'Urziceni', 98, 10)]
(8, 'Iasi') : [(11, 'Neamt', 87, 234), (18, 'Vaslui', 92, 199)]
(9, 'Lugoj') : [(10, 'Mehadia', 70, 241), (16, 'Timisoara', 111, 329)]
(10, 'Mehadia') : [(3, 'Drobeta', 75, 242), (9, 'Lugoj', 70, 244)]
(11, 'Neamt') : [(8, 'Iasi', 87, 226)]
(12, 'Oradea') : [(15, 'Sibiu', 151, 253), (19, 'Zerind', 71, 374)]
(13, 'Pitesti') : [(1, 'Bucharest', 101, 20), (2, 'Craiova', 138, 160), (14, 'Rimnicu Vilcea', 97, 193)]
(14, 'Rimnicu Vilcea') : [(2, 'Craiova', 146, 160), (13, 'Pitesti', 97, 100), (15, 'Sibiu', 80, 253)]
(15, 'Sibiu') : [(0, 'Arad', 140, 366), (5, 'Fagaras', 99, 176), (12, 'Oradea', 151, 380), (14, 'Rimnicu Vilcea', 80, 193)]
(16, 'Timisoara') : [(0, 'Arad', 118, 366), (9, 'Lugoj', 111, 244)]
(17, 'Urziceni') : [(1, 'Bucharest', 85, 20), (7, 'Hirsova', 98, 0), (18, 'Vaslui', 142, 199)]
(18, 'Vaslui') : [(8, 'Iasi', 92, 226), (17, 'Urziceni', 142, 10)]
(19, 'Zerind') : [(0, 'Arad', 75, 366), (12, 'Oradea', 71, 380)]
```

3. Thuật toán Geedy Best First Search

```python
# Geedy best first search
def GBFS(self,start,goal):
    frontier = PriorityQueue()
    root = (start,self.heuristic[start])
    frontier.put(root)
    explored = []
    father = {}
    path = [goal]
    while True:
        if frontier.empty():
            raise Exception("No way Exception")
        current_node, current_h = frontier.get()
        explored.append(current_node)
        if current_node == goal:
            key = goal
            while key in father.keys():
                value = father.pop(key)
                path.append(value)
                key = value
                if key == start:
                    break
            path.reverse()
            return path
        if current_node not in self.graph:
            continue
        for Node in self.graph[current_node]:
            node ,heuris = Node[0], Node[-1]
            if node not in explored:
                frontier.put((node, heuris))
                if node not in father.keys():
                    father[node] = current_node
```

Ta tạo hàng đợi ưu tiên (PriortyQueue) để lấy phần tử với heuristic như bảng trên thấp nhất khi xét những phần tử đi qua được. Và explored là tập chứa những trạng thái đã xét đến. Khi trạng thái mới trùng với trạng thái đã xét sẽ lướt qua. Bên cạnh đó khi xét những trạng thái mới đồng thời lưu lại trạng thái cũ (father) của trạng thái mới. Lặp lại như thế cho đến khi tìm được trạng thái kết thúc. Ta sẽ xét ngược lại những father đã lưu trước đó lưu vào path và trả ra đường đi (path) đạo ngược.

## 2. Cài đặt thuật toán A* để tìm đường đi ngắn nhất từ Arad tới Hirsova như hình với hàm h(n) được xác định như trong bài tập 1 và g(n) là khoảng cách giữa 2 thành phố.

Thuật toán đánh giá 1 nút dựa trên chi phí đi từ nút gốc đến nút đó (g(n)) cộng với chi phí từ nút đó đến nút đích (h(n)).

$$f(n) = g(n) + h(n)$$

Hàm h(n) được gọi là chấp nhận được nếu với mọi trạng thái n, h(n) $\leqslant$ độ dài đường đi ngắn nhất thực tế từ u tới trạng thái đích.

Thuật giải A * sử dụng 2 tập hợp sau đây:

- OPEN: tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến $\Longrightarrow$ OPEN là 1 hàng đợi ưu tiên (priority queue) mà trong đó, phần tử có độ ưu tiên cao nhất là phần tử tốt nhất.

- CLOSE: tập chứa các trạng thái đã được xét đến. Ta cần lưu trữ những trạng thái này trong bộ nhớ để đề phòng khi một trạng thái mới được tạo ra lại trùng với 1 trạng thái mà ta đã xét đến trước đó. Trong trường hợp không gian tìm kiếm có dạng cây thì không cần dùng tập này.

Khi xét đến một trạng thái $T_i$ bên cạnh việc lưu trữ 3 giá trị cơ bản $g(T_i)$, $h(T_i)$, $f(T_i)$ để phản ánh độ tốt của trạng thái đó, A * còn lưu trữ thêm 2 thông số sau:

✓ Trạng thái cha của trạng thái $T_i$ ($Father(T_i)$). Trong trường hợp có nhiều trạng thái dẫn đến trạng thái $T_i$ thì chọn ($Father(T_i)$) sao cho chi phí đi từ trạng thái khởi đầu đến $T_i$ là thấp nhất, nghĩa là $g(T_i) = g(T_{father}) + cost(T_{father}, T_i)$ là thấp nhất.

✓ Danh sách các trạng thái kế tiếp của $T_i$ : danh sách này lưu trữ các trạng thái kế tiếp $T_k$ của $T_i$ sao cho chi phí đến $T_k$ thông qua $T_i$ từ trạng thái ban đầu là thấp nhất $\Longrightarrow$ danh sách này được tính từ thuộc tính Cha của các trạng thái được lưu trữ.

Xây dựng lại đường đi từ $T_0$ tới $T_G$ : ta lần ngược theo thuộc tính Cha(father) của các trạng thái đã được lưu trữ trong CLOSE cho đến khi đạt đến $T_0$.

1. hàm tính cost

```python
def min_cost(cost_a, cost_b):
    if (cost_a < cost_b):
        return cost_a
    return cost_b
```

   Chỉ lấy giá trị thấp hơn để $g(T_i)$ thấp nhất

2. Thuật Toán A* Search

```python
# A* search
def aStart(self,start,goal):
    frontier = PriorityQueue()
    f = 0; g = 0; h = 0
    root = (f,g,h,start)
    frontier.put(root)
    explored = []
    father = {}
    path = [goal]
    while True:
        if frontier.empty():
            raise Exception("No way Exception")
        current_f,current_h,current_g,current_node = frontier.get()
        explored.append(current_node)
        if current_node == goal:
            key = goal
            while key in father.keys():
                value = father.pop(key)
                path.append(value)
                key = value
                if key == start:
                    break
            path.reverse()
            return path
        if current_node not in self.graph:
            continue
        for Node in self.graph[current_node]:
            node,grid,heuris = Node[0],Node[-2],Node[-1]
            grid = current_g + min_cost(current_g,grid)
            f = grid + heuris
            if node not in explored:
                frontier.put((f, grid, heuris, node))
                if node not in father.keys():
                    father[node] = current_node
```

## 3. Hàm main

```python
def main():
    file_1 = open("Input.txt","r")
    vertices, start,goal,name_ct,heuristic,matrix = read_txt(file_1)
    print('Vertices: ',vertices,'\n','Start: ' ,start,'Goal: ' ,goal,'\n')
    graph = convert_graph(vertices,name_ct,heuristic,matrix)
    # graph.show_graph()
    print('\n')
    print('path GBFS:')
    path_GBFS = graph.GBFS(0,7)
    print(" --> ".join(name_ct[i] for i in path_GBFS))
    print('\n')
    print('path A*:')
    path_aStart = graph.aStart(0,7)
    print(" --> ".join(name_ct[i] for i in path_aStart))
```

## 4. Kết quả

```
Vertices:  20
 Start:  0 Goal:  7



path GBFS:
Arad --> Sibiu --> Fagaras --> Bucharest --> Urziceni --> Hirsova


path A*:
Arad --> Sibiu --> Rimnicu Vilcea --> Pitesti --> Bucharest --> Urziceni --> Hirsova
```