19110482_main

December 3, 2021

1 Nhập môn Trí tuệ Nhân tạo

- Bài thực hành tuần 3
- Sinh viên: Huỳnh Thị Bảo Trân
- MSSV: 19110482

Đề bài:
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.
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.

Các thư viện được sử dụng (nếu có)

[ ]:
```
from queue import PriorityQueue
```

Bài 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.

Ý tưởng:
1. Greedy - Best - First search mở rộng nút gần đích nhất với hi vọng cách làm này sẽ dẫn đến lời giải một cách nhanh nhất.
2. Đánh giá chi phí của các nút chỉ dựa trên hàm heuristic: **f(n) = h(n)**

Cách thực hiện:

1. Tạo graph

[ ]:
```
GRAPH = {'Arad': [('Zerind', 374), ('Timisoara', 329), ('Sibiu', 253)],
        'Zerind': [('Oradea', 380), ('Arad', 366)],
        'Oradea': [('Sibiu',253)],
        'Sibiu': [('Rimniciu Vilcea', 193), ('Fagaras', 176), ('Arad', 366)],
        'Fagaras': [('Sibiu', 253), ('Bucharest', 20)],
        'Rimniciu Vilcea': [('Pitesti', 100), ('Craiova', 160), ('Sibiu', 253)],
        'Timisoara': [('Lugoj', 244), ('Arad', 366)],
        'Lugoj': [('Mehadia', 241)],
        'Mehadia': [('Lugoj', 244), ('Dorbeta', 242)],
        'Dobreta': [('Mehadia', 241), ('Craiova', 160)],
        'Pitesti': [('Craiova', 160), ('Bucharest', 20)],
        'Craiova': [('Pitesti', 100), ('Dobreta', 242), ('Rimniciu Vilcea',␣
↪193)],
```

```
        'Bucharest': [('Giurgiu', 77), ('Urziceni', 10), ('Fagaras', 176),␣
↪('Pitesti', 98)],
        'Giurgiu': [('Bucharest', 20)],
        'Urziceni': [('Vaslui', 199), ('Hirsova', 0), ('Bucharest', 20)],
        'Vaslui': [('Lasi', 226), ('Urziceni', 80)],
        'Lasi': [('Neamt', 234), ('Vaslui', 199)],
        'Neamt': [('Lasi', 226)],
        'Hirsova': [('Eforie', 161), ('Urziceni', 10)],
        'Eforie': [('Hirsova', 0)]
}
```

2. Cài đặt hàm

[ ]:
```
def GBFS(graph, start, end):
    queue = []
    visited = []
    if start not in visited:
        print(start)
        visited.append(start)
    queue = queue + [x for x in graph[start] if x[0][0] not in visited]
    queue.sort(key = lambda x:x[1])
    if queue[0][0] == end:
        print(queue[0][0])
    else:
        processing = queue[0]
        queue.remove(processing)
        GBFS(graph, processing[0], end)
```

3. Kết quả thực hiện được

[ ]:
```
GBFS(GRAPH, start = 'Arad', end = 'Hirsova')
```

```
Arad
Sibiu
Fagaras
Bucharest
Urziceni
Hirsova
```

Bài 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.

Ý tưởng:
1. 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)**
2. Hàm h(n) được gọi là chấp nhận được nếu với mọi trạng thái n, h(n) độ dài đường đi ngắn nhất thực tế từ u tới trạng thái đích.

Cách thực hiện: Thuật giải A* sử dụng 2 tập hợp sau đây:
1. **OPEN:** tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến nên 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.
2. **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.

1. Tạo Graph

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

2. Cài đặt hàm

[ ]:
```
def a_star(source, destination):
        straight_line = {
            'Arad': 366,
            'Zerind': 374,
            'Oradea': 380,
            'Sibiu': 253,
            'Fagaras': 176,
            'Rimniciu Vilcea': 193,
            'Timisoara': 329,
            'Lugoj': 244,
            'Mehadia': 241,
            'Dobreta': 242,
            'Pitesti': 100,
            'Craiova': 160,
            'Bucharest': 0,
            'Giurgiu': 77,
```

```
            'Urziceni': 80,
            'Vaslui': 199,
            'Lasi': 226,
            'Neamt': 234,
            'Hirsova': 151,
            'Eforie': 161
        }
        p_q, visited = PriorityQueue(), {}
        p_q.put((straight_line[source], 0, source, [source]))

        visited[source] = straight_line[source]
        while not p_q.empty():
            (heuristic, cost, vertex, path) = p_q.get()
            if vertex == destination:
                return heuristic, cost, path

            for next_node in GRAPH[vertex].keys():
                current_cost = cost + GRAPH[vertex][next_node]
                heuristic = current_cost + straight_line[next_node]
                if not next_node in visited or visited[next_node] >= heuristic:
                    visited[next_node] = heuristic
                    p_q.put((heuristic, current_cost, next_node, path +␣
↪[next_node]))
```

3. Xuất kết quả

[ ]:
```
def main():
    source = 'Arad'
    end = 'Bucharest'
    if source not in GRAPH or end not in GRAPH:
        print('CITY DOES NOT EXIST.')
    else:
        heuristic, cost, optimal_path = a_star(source, end)
        print('total min cost:', cost)
        print('Route:')
        print(' -> '.join(city for city in optimal_path))

main()
```

```
total min cost: 418
Route:
Arad -> Sibiu -> Rimniciu Vilcea -> Pitesti -> Bucharest
```