Lab02: Sử dụng Python để giải bài toán 8-puzzle sử dụng thuật toán A*.

Yêu cầu: SV tự viết code, không nên tham khảo ChatGPT hay DeepSeek

Bài toán 8-puzzle (hay còn gọi là trò chơi 8-số) là một trong những bài toán kinh điển trong Trí tuệ Nhân tạo (AI), thường được dùng để minh họa cho các thuật toán tìm kiếm (search algorithms). Bài toán bao gồm một bàn cờ là một bảng 3x3 chứa 8 ô số (từ 1 đến 8) và 1 ô trống (ký hiệu là 0 hoặc ô rỗng). Mỗi một trạng thái là một cách sắp xếp các số và ô trống trên bàn cờ. Mục tiêu của bài toán là từ một trạng thái ban đầu, di chuyển ô trống (lên, xuống, trái, phải) sao cho đạt được trạng thái đích (thường là dãy số được sắp xếp tăng dần).

Trạng thái ban đầu	Trạng thái đích
1 2 3 4 0 6 7 5 8	1 2 3 4 5 6 7 8 0

Ở đây, ô 0 là ô trống, ta có thể di chuyển các số liền kề nó để đến trạng thái đích. Phép toán hợp lệ: di chuyển ô trống lên/xuống/trái/phải nếu không vượt ra ngoài biên. Chi phí bước đi thường được tính là 1 cho mỗi nước di chuyển.

Để việc tìm kiếm nhanh hơn, ta sử dụng một trong hai heuristic h1(n): Số lượng ô không đúng vị trí và h2(n): Tổng khoảng cách Manhattan của các ô so với vị trí đích. Hàm heuristic tổng quát cho thuật toán A*: f(n) = g(n) + h(n), trong đó g(n): số bước đi từ trạng thái bắt đầu đến trạng thái hiện tại và h(n): heuristic = ước lượng số bước từ trạng thái hiện tại đến đích.

Cấu trúc dữ liệu & một số Python code cơ bản cho bài toán 8-puzzle:

Python code

```
start_state = [ # Trạng thái bắt đầu
        [5, 2, 1]
        [7, 4, 3]
        [0, 8, 6]]
goal_state = [ # Trạng thái đích
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 0]]
moves = [(-1, 0), (1, 0), (0, -1), (0, 1)] # Các di chuyển có thể: lên, xuống, trái, phải
```

```
def manhattan_distance(state): # Hàm tính heuristic (Manhattan distance)
  distance = 0
  for i in range(3):
    for j in range(3):
      value = state[i][j]
      if value != 0:
        goal_x, goal_y = divmod(value - 1, 3)
        distance += abs(goal_x - i) + abs(goal_y - j)
```

```
    return distance
def get_neighbors(state): # Sinh trạng thái mới từ 1 trạng thái
    x, y = find_zero(state)
    neighbors = []
    for dx, dy in moves:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_state = [row[:] for row in state]
            new_state[x][y], new_state[nx][ny] = new_state[nx][ny], new_state[x][y]
            neighbors.append(new_state)
    return neighbors
def a_star(start): # Thuật toán A*
    frontier = []
    heapq.heappush(frontier, (manhattan_distance(start), 0, start, []))
    visited = set()

    while frontier:
        f, g, state, path = heapq.heappop(frontier)
        if state == goal_state:
            return path + [state]

        state_key = state_to_tuple(state)
        if state_key in visited:
            continue
        visited.add(state_key)

        for neighbor in get_neighbors(state):
            if state_to_tuple(neighbor) not in visited:
                new_g = g + 1
                new_f = new_g + manhattan_distance(neighbor)
                heapq.heappush(frontier, (new_f, new_g, neighbor, path + [state]))

    return None
```

Yêu cầu:

1. Viết chương trình hoàn thiện giải quyết bài toán 8-puzzle sử dụng heuristic h1(n): Số lượng ô không đúng vị trí với trạng thái đầu và trạng thái đích đã nói trên. In ra các trạng thái đã "viếng thăm" từ trạng thái đầu đến trạng thái đích có kèm theo giá trị của các hàm f(n) và h(n) của từng trạng thái.
2. Giống câu 1 nhưng áp dụng heuristic h2(n): Tổng khoảng cách Manhattan của các ô so với vị trí đích. In ra các trạng thái đã "viếng thăm" từ trạng thái đầu đến trạng thái đích có kèm theo giá trị của các hàm f(n) và h(n) của từng trạng thái.
3. Hãy cho nhận xét 2 câu trên với 2 heuristic h1(n) và h2(n), heusristic nào gần với thực tế hơn. Khi áp dụng thuật toán A* trong cả 2 cách trên với công thức heuristic tổng quát:

f(n) = g(n) + h(n), g(n) có giá trị ra sao? Có ảnh hưởng đến độ tốc độ tìm kiếm của giải thuật A* không?