Bài tập thực hành nhập môn trí tuệ nhân tạo lần 1

Tên: Nguyễn Công Tiến Dũng

MSSV: 22280014

- *Note: V_i+1 = node i (V1 = node 0, V2 = node 1,…, V18 = node 17)*

- Chạy tay các thuật toán:

+ BFS: (QUEUE)

1. L = [0]
2. Node = 0, L = [1, 2, 3], father[1, 2, 3] =0
3. Node = 1, L = [2, 3, 6], father[6] =1
4. Node = 2, L = [3, 6, 5, 7], father[5, 7] = 2
5. Node = 3, L = [6, 5, 7, 4], father[4] = 3
6. Node = 6, L = [5, 7, 4]
7. Node = 5, L = [7, 4, 11], father[11] = 5
8. Node = 7, L = [4, 11, 8, 9], father[8, 9] = 7
9. Node = 4, L = [11, 8, 9, 13], father[13] = 4
10. Node = 11, L = [8, 9, 13, 12], father[12] =11
11. Node = 8, L = [9, 13, 12, 10], father[10] = 8
12. Node = 9, L = [13, 12, 10]
13. Node = 13, L = [12, 10], father[10] = 13
14. Node = 12, L = [10]
15. Node = 10, L = [14], father[14] = 10
16. Node = 14, L = [15], father[15] = 14
17. Node = 15, L = [16], father [16] = 15
18. Node = 16, L = [17], father[17] = 16
19. Node 17 (trạng thái kết thúc) → dừng.

⇨ Đường đi từ đỉnh 0 tới đỉnh 17 là:

- 0 → 2 → 7 → 8 → 10 → 14 → 15 → 16 → 17
  hoặc:
- 0 → 3 → 4 → 13 → 10 → 14 → 15 → 16 → 17

+ DFS: (STACK)

1. L = [0]
2. Node = 0, L = [3, 2, 1], father[3, 2, 1] = 0
3. Node = 3, L = [4, 2, 1], father[4] = 3
4. Node = 4, L = [13, 2, 1], father[13] = 4
5. Node = 13, L = [10, 9, 2, 1], father[10] = 13
6. Node = 10, L = [14, 9, 2, 1], father[14] = 10
7. Node = 14, L = [15, 9, 2, 1], father[15] = 14
8. Node = 15, L = [16, 9, 2, 1], father[16] = 15

9. Node = 16, L = [17, 9, 2, 1], father[17] = 16
10. Node 17 (trạng thái kết thúc) → dừng.

⇨ Đường đi từ đỉnh 0 tới đỉnh 17 là :0→3→4→13→10→14→15→16→17

+ UCS: (PRIORITY QUEUE)

1. PQ = {(0,0)}
2. PQ = {(1, 50), (3, 300), (2, 350)}
3. PQ = {(3, 300), (2,350), (6,650)}
4. PQ = {(2,350), (6,650), (4,1600)}
5. PQ = {(5,450), (6,650), (7,1250), (4,1600)}
6. PQ = {(6,650), (11,1150), (7,1250), (4,1600)}
7. PQ = {(11,1150), (7,1250), (4,1600)}
8. PQ = {(7,1250), (4,1600), (12,2100)}
9. PQ = {(9,1550), (4,1600), (8,2040), (12,2100)}
10. PQ = {(4,1600), (8,2040), (12, 2100)}
11. PQ = {(8,2040), (12,2100), (13,3000)}
12. PQ = {(12,2100), (13,3000), (10,3240)}
13. PQ = {(13,2700), (10,3240)}
14. PQ = {(10,3240)}
15. PQ = {(14,3640)}
16. PQ = {(15,4940)}
17. PQ = {(16,5710)}
18. PQ = {(17,6910)}

⇨ Đường đi ngắn nhất từ START tới GOAL là :
0 → 2 → 7 → 8 → 10 → 14 → 15 → 16 → 17 với chi phí là 6910.

- **Kiểm tra tính đúng đắn: sau cài đặt và thực thi trương trình.**

Output: khi chạy thì code không báo lỗi vẫn trả ra output có kết quả đúng với so với những gì đã giải bằng tay.

```
Kết quả sử dụng thuật toán BFS:
 [0, 2, 7, 8, 10, 14, 15, 16, 17]
Kết quả sử dụng thuật toán DFS:
 [0, 3, 4, 13, 10, 14, 15, 16, 17]
Kết quả sử dụng thuật toán UCS:
 [0, 2, 7, 8, 10, 14, 15, 16, 17] với tổng chi phí là:  6910
```

+ Trong triển khai BFS: cần sửa lại một số chỗ cho hợp logic

1. Thay thế lưu trữ visited list bằng set() để tối ưu việc truy cập và tìm kiếm.
2. Loại bỏ “visited.append(start)” sau khi “frontier.put(start)” và sau điều kiện “if node not in visited”. Vì nếu node được thêm vào visited

khi kiểm tra node không thuộc visited ngay thì sẽ khó kiểm tra node đã được visited và cập nhật parent, mà chỉ cần them vào visited sau khi xét “visited.append(current_node)” là đủ
Output sau khi sửa lại:

```
# Thuật toán BFS
def BFS(self, start, end):
    if not self.adj_list:
        self.convert_to_adj_list()

    # Khởi tạo queue và visited
    frontier = Queue()
    frontier.put(start)
    visited = set()

    parent = dict()
    parent[start] = None

    path_found = False

    while True:
        if frontier.empty():
            raise Exception("No way Exception")

        current_node = frontier.get()
        visited.add(current_node)

        if current_node == end:
            path_found = True
            break

        for node in self.adj_list[current_node]:
            if node not in visited:
                frontier.put(node)
                parent[node] = current_node

    path = []
    if path_found:
        while end is not None:
            path.append(end)
            end = parent[end]
        path.reverse()

    return path
```

+ Trong triển khai DFS: ta cũng sửa lỗi tương tự như BFS

1. Thay thế lưu trữ visited list bằng set() để tối ưu việc truy cập và tìm kiếm.
2. Loại bỏ “visited.append(start)” sau khi “frontier.put(start)” và sau điều kiện “if node not in visited”.
   Output sau khi sửa lại code:

```python
# Thuật toán DFS
def DFS(self, start, end):
    if not self.adj_list:
        self.convert_to_adj_list()

    frontier = []
    visited = set()

    frontier.append(start)

    parent = dict()
    parent[start] = None

    path_found = False

    while True:
        if frontier == []:
            raise Exception("No way Exception")

        current_node = frontier.pop()
        visited.add(current_node)

        if current_node == end:
            path_found = True
            break

        for node in self.adj_list[current_node]:
            if node not in visited:
                frontier.append(node)
                parent[node] = current_node

    path = []
    if path_found:
        while end is not None:
            path.append(end)
            end = parent[end]
        path.reverse()

    return path
```

+ Trong triển khai UCS: cần cập nhật cost nếu có cost nhỏ hơn của node đã xét trước đó hoặc node chưa được xét thì cho phép cập nhật cost để tìm chi phí nhỏ nhất.

Output sau khi sửa lại code:

```python
# Thuật toán UCS
def UCS(self, start, end):
    if not self.adj_list:
        self.convert_to_weighted_adj_list()

    visited = set()
    frontier = PriorityQueue()

    parent = {start: None}
    frontier.put((0, start))
    cost_so_far = {start: 0} #thêm dictionary để lưu trữ chi phí
    path_found = False

    while True:
        if frontier.empty():
            raise Exception("No way Exception")

        current_w, current_node = frontier.get()
        visited.add(current_node)

        if current_node == end:
            path_found = True
            break

        for nodei in self.adj_list[current_node]:
            node, weight = nodei
            new_cost = current_w + weight

            if node not in cost_so_far or new_cost < cost_so_far[node]:
                cost_so_far[node] = new_cost
                frontier.put((new_cost, node))
                parent[node] = current_node

    path = []
    if path_found:
        while end is not None:
            path.append(end)
            end = parent[end]
        path.reverse()

    return cost_so_far[path[-1]], path
```

- **Nhận xét:**

Sau khi sửa và điều chỉnh lại code thì thuật toán thì giải thuật BFS vẫn chỉ in được 1 trường hợp trong khi đó giải tay có thể cho ra các trường hợp có đường đi ngắn nhất cụ thể ở đây là 2 đường. Còn các giải thuật như DFS và UCS vẫn cho ra kết quả tương tự như giải tay. Kết quả cho ra đã phù hợp với thuật toán cần triển khai.

Output sau khi sửa code:

```
Kết quả sử dụng thuật toán BFS:
 [0, 3, 4, 13, 10, 14, 15, 16, 17]
Kết quả sử dụng thuật toán DFS:
 [0, 3, 4, 13, 10, 14, 15, 16, 17]
Kết quả sử dụng thuật toán UCS:
 [0, 2, 7, 8, 10, 14, 15, 16, 17] với tổng chi phí là:  6910
```