NHẬP MÔN TRÍ TUỆ NHÂN TẠO

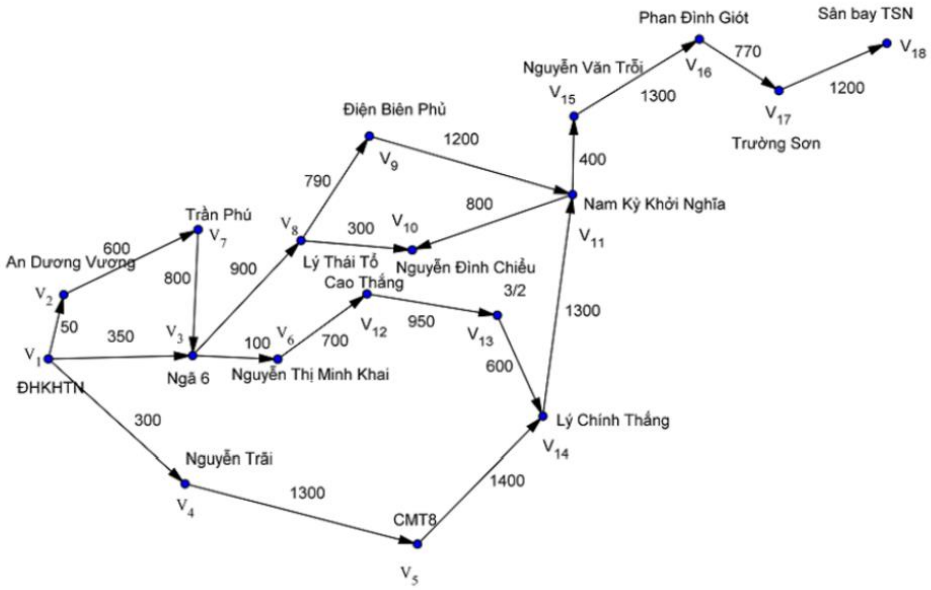
Tên: Trần kim Khanh

Mssv : 21110318

Bài tập thực hành Tuần 1

III. NỘI DUNG THỰC HÀNH:

Cho đồ thị như hình vẽ bên dưới Tìm đường đi ngắn nhất từ trường Đại học Khoa học



Ghi ra các đỉnh kề dựa vào file input trong pdf Đỉnh sẽ tính từ đỉnh 0:

0: [1, 2, 3]

1: [6]

2: [5, 7]

3: [4]

4: [13]

5: [11]

6: [2]

7: [8, 9]

8: [10]

10: [9, 14]

11: [12]

12: [13]

13: [10]

14: [15]

15: [16]

16: [17]

Chạy tay thuật toán BFS :

```
Procedure Breadth_Search

Begin
 1. Khởi tạo danh sách L chứa trạng thái ban đầu;
 2. While (1)
        2.1 if L rỗng then
            {
                Thông báo tìm kiếm thất bại;
                stop;
            }
        2.2 Loại trạng thái u ở đầu danh sách L;
        2.3 if u là trạng thái kết thúc then
            {
                Thông báo tìm kiếm thành công;
                stop;
            }
        2.4 Lấy các trạng thái v kề với u và thêm vào cuối danh sách L;
            for mỗi trạng thái v kề u do
                father(v) = u;
    end
```

Dựa vào đó áp dụng chạy tay trên mã giả :

Node = 0, L = [1, 2, 3], father = [1, 2, 3] = 0

Node = 1, L = [2, 3, 6], father = [6] = 1

Node = 2, L = [3, 6, 5, 7], father = [5, 7] = 2

Node = 3, L = [6, 5, 7, 4], father = [4] = 3

Node = 6, L = [5, 7, 4]

Node = 5, L = [7, 4, 11], father = [11] = 5

Node = 7, L = [4, 11, 8, 9], father = [8, 9] = 7

Node = 4, L = [11, 8, 9, 13], father = [13] = 4

Node = 11, L = [8, 9, 13, 12], father = [12] = 11

Node = 8, L = [9, 13, 12, 10], father = [10] = 8

Node = 9, L = [13, 12, 10]

Node = 13, L = [12, 10]

Node = 12, L = [10]

Node = 10, L = [14], father = [14] = 10

Node = 14, L = [15], father = [15] = 14

Node = 15, L = [16], father = [16] = 15

Node = 16, L = [17], father = [17] = 16

Node = 17 , đã tới được đích nên dừng

Đường đi tìm thấy: [0, 2, 7, 8, 10, 14, 15, 16, 17]

```
Procedure Depth_Search
Begin
1. Khởi tạo danh sách L chứa trạng thái ban đầu;
2. While (1)
        2.1 if L rỗng then
            {
                Thông báo tìm kiếm thất bại;
                stop;
            }
        2.2 Loại trạng thái u ở đầu danh sách L;
        2.3 if u là trạng thái kết thúc then
            {
                Thông báo tìm kiếm thành công;
                stop;
            }
        2.4 Lấy các trạng thái v kề với u và thêm vào đầu danh sách L;
            for mỗi trạng thái v kề u do
                father(v) = u;
end
```

Chạy thay thuật toán DFS dựa vào mã giả :

Node = 0, L = [3, 2, 1], father = [1, 2, 3] = 0

Node = 3, L = [4, 2, 1], father = [4] = 3

Node = 4, L = [13, 2, 1], father = [13] = 4

Node = 13, L = [10, 2, 1], father = [10] = 13

Node = 10, L = [14, 9, 2, 1], father = [9, 14] = 10

Node = 14, L = [15, 9, 2, 1], father = [15] = 14

Node = 15, L = [16, 9, 2, 1], father = [16] = 15

Node = 16, L = [17, 9, 2, 1], father = [17] = 16

Node = 17 đã tới đích nên dừng

Đường đi tìm thấy: [0, 3, 4, 13, 10, 14, 15, 16, 17]

Thuật toán UCS:

```
function Tìm_kiếm_UCS(bài_toán, ngăn_chứa) return lời giải hoặc thất bại.
ngăn_chứa ←Tạo_Hàng_Đợi_Rỗng()
ngăn_chứa ←Thêm(TẠO_NÚT(Trạng_Thái_Đầu[bài_toán]), ngăn_chứa)
loop do
        if Là_Rỗng(ngăn_chứa) then return thất bại.
        nút←Lấy_Chi_phí_Nhỏ_nhất(ngăn chứa)
        if Kiểm_tra_Câu_hỏi_đích[bài_toán] trên Trạng_thái[nút] đúng.
                then return Lời_giải(nút).
lg←Mở(nút, bài_toán)     //lg tập các nút con mới
ngăn_chứa←Thêm_Tất_cả(lg, ngăn_chứa)
```

Ma trận trọng số chuyển sang ma các list:

0: [(1, 50), (2, 350),

(3, 300)], 1: [(6, 600)]

2: [(5, 100), (7, 900)],

3: [(4, 1300)]

4: [(13, 1400)

5: [(11, 700)]

6: [(2, 800)]

7: [(8, 790),(9, 300)]

8: [(10, 1200)]

10: [(9, 800), (14, 400)]

11: [(12, 950)]

12: [(13, 600)]

13: [(10, 1300)]

14: [(15, 1300)]

15: [(16, 770)]

16: [(17, 1200)]

Chạy tay :

Đỉnh đang xét 0

Node = 0, PQ = [(50, 1), (300, 3), (350, 2)], father = [1, 2, 3] = 0

Đỉnh đang xét 1

Node = 1, PQ = [(300, 3), (350, 2), (650, 6)], father = [6] = 1

Đỉnh đang xét 3

Node = 3, PQ = [(350, 2), (650, 6), (1600, 4)], father = [4] = 3

Đỉnh đang xét 2

Node = 2, PQ = [(450, 5), (650, 6), (1250, 7), (1600, 4)], father = [5, 7] = 2

Đỉnh đang xét 5

Node = 5, PQ = [(650, 6), (1150, 11), (1250, 7), (1600, 4)], father = [11] = 5

Đỉnh đang xét 6

Node = 6, PQ = [(1150, 11), (1250, 7), (1600, 4)]

Đỉnh đang xét 11

Node = 11, PQ = [(1250, 7), (1600, 4), (2100, 12)], father = [12] = 11

Đỉnh đang xét 7

Node = 7, PQ = [(1550, 9), (1600, 4), (2040, 8), (2100, 12)], father = [8, 9] = 7

Đỉnh đang xét 9

Node = 9, PQ = [(1600, 4), (2040, 8), (2100, 12)]

Đỉnh đang xét 4

Node = 4, PQ = [(2040, 8), (2100, 12), (3000, 13)], father = [13] = 4

Đỉnh đang xét 8

Node = 8, PQ = [(2100, 12), (3000, 13), (3240, 10)], father = [10] = 8

Đỉnh đang xét 12

Node = 12, PQ = [(3000, 13), (3240, 10)]

Đỉnh đang xét 13

Node = 13, PQ = [(3240, 10)]

Đỉnh đang xét 10

Node = 10, PQ = [(3640, 14)], father = [14] = 10

Đỉnh đang xét 14

Node = 14, PQ = [(4940, 15)], father = [15] = 14

Đỉnh đang xét 15

Node = 15, PQ = [(5710, 16)], father = [16] = 15

Đỉnh đang xét 16

Node = 16, PQ = [(6910, 17)], father = [17] = 16

Node = 17 đã tới đích

Đường đi tìm thấy: (6910, [0, 2, 7, 8, 10, 14, 15, 16, 17])

Kiểmtra tính đúng đắn của các thuật toán đã cho sẵn code như trên. Nếu chưa đúng thì em sửa lại như thế nào cho phù hợp ?

- Lỗi chương trình : Nhớ phải dùng try vs catch để bắt các lỗi khi không tìm thấy đường đi để tránh chương trình bị dừng
- Thuật toán BFS giải thuật đúng , nhưng phải chú ý xét một số diều kiện đầu vào .
- Thuật toán DFS trong mã giả thì thêm ở đầu và lấy ở đâu , nhưng ở trong code thì thêm vào cuối lấy ở cuối (việc này giúp chúng ta thêm và xóa dễ hơn , giảm độ phức tạp) , và cx chưa kiểm tra điều kiện đầu vào ,
- Thuật toán UCS thuật toán đúng , chưa kiểm tra điều kiện đầu vào và nên khai bào current_weight = 0 trước

Chúng ta sẽ thêm điều kiện

if not graph:

raise Exception("No path found") ở các hàm trên và in ra chữ No path found thay vì No way expection

⇨ Kết quả thuật toán và giải tay trong trường hợp input trong bài tập đều tìm ra đường đi tới đích là giống nhau .

Và đây là đoạn code hiện thực lại class Graph

graph.py > Graph > __init__

```python
from queue import Queue, PriorityQueue
from collections import defaultdict


class Graph:
    def __init__(self):
        self.adjancy_matrix = None # ma trận kề
        self.adjancy_list = None # danh sách kề
        self.start_node = None # node bắt đầu
        self.end_node = None # node kết thúc
        self.has_weight = False # ma trận có trọng số hay không
        self.size = 0 # kích thước của ma trận

    def read_file_text(self, file_name="input.txt"):
        file = open(file_name, "r")
        self.size = int(file.readline())
        self.start_node, self.end_node = [int(num) for num in file.readline().split(' ')]
        # ma trận
        self.adjancy_matrix = [[int(num) for num in file.readline().split(' ')] for num in range(self.size)]
        return self.size, self.start_node, self.end_node, self.adjancy_matrix

```

```python
def convert_to_list(self, weight=None):
    self.adjancy_list = defaultdict(list)
    for i in range(len(self.adjancy_matrix)):
        for j in range(len(self.adjancy_matrix[i])):
            # Đồ thị không trọng số
            if self.adjancy_matrix[i][j] == 1 and weight is None:
                self.adjancy_list[i].append(j)
            # Đồ thị có trọng số
            if self.adjancy_matrix[i][j] != 0 and weight is not None:
                self.adjancy_list[i].append((j, self.adjancy_matrix[i][j]))
                self.has_weight = True
    return self.adjancy_list
```

```python
def bfs(self, start_node, end_node):
    if not self.adjancy_list:  #! Kiểm tra xem adjacency_list có rỗng không
        raise Exception("No path found")
    visited = []
    frontier = Queue()
    frontier.put(start_node)
    visited.append(start_node)
    parent = dict()
    parent[start_node] = None
    path_found = False
    while True:
        if frontier.empty():
            raise Exception("No path found")
        current_node = frontier.get()
        visited.append(current_node)

        if current_node == end_node:
            path_found = True
            break
        for node in self.adjancy_list[current_node]:
            if node not in visited:
                frontier.put(node)
                parent[node] = current_node
                visited.append(node)
```

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

    return path
```

```python
    def dfs(self, start_node, end_node):
        if not self.adjancy_list:  #! Kiểm tra xem adjacency_list có rỗng không
            raise Exception("No path found")
        visited = []
        frontier = []

        frontier.append(start_node)
        visited.append(start_node)

        parent = dict()
        parent[start_node] = None

        path_found = False
        while True:
            if frontier == []:
                raise Exception("No path found")
            current_node = frontier.pop()
            visited.append(current_node)

            if current_node == end_node:
                path_found = True
                break


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

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

            return path

    def ucs(self, start_node, end_node):
        if not self.adjancy_list:  #! Kiểm tra xem adjacency_list có rỗng không
            raise Exception("No path found")
        visited = []
        frontier = PriorityQueue()

        frontier.put((0, start_node))
        visited.append(start_node)

        parent = dict()
        parent[start_node] = None
        current_weight = 0 #! nên khởi tạo ra trước
        path_found = False

        while True:
            if frontier.empty():
                raise Exception("No path found")
            current_weight, current_node = frontier.get()
            visited.append(current_node)
```

```python
            if current_node == end_node:
                path_found = True
                break

            for node_i in self.adjancy_list[current_node]:
                node, weight = node_i
                if node not in visited:
                    frontier.put((current_weight + weight, node))
                    parent[node] = current_node
                    visited.append(node)

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

        return path, current_weight
```

main.py > ...

```python
if __name__ == '__main__':
    try:
        # Tạo đồ thị từ tệp đầu vào
        graph_new_1 = Graph()
        graph_new_1.read_file_text("input.txt")
        adjList_1 = graph_new_1.convert_to_list()
        # Thực hiện BFS
        bfs_1_path = graph_new_1.bfs(graph_new_1.start_node, graph_new_1.end_node)
        print("BFS path: ", bfs_1_path)
        # Thực hiện DFS
        dfs_1_path = graph_new_1.dfs(graph_new_1.start_node, graph_new_1.end_node)
        print("DFS path: ", dfs_1_path)
        # Tạo đồ thị từ tệp UCS
        graph_new_2 = Graph()
        graph_new_2.read_file_text("input_ucs.txt")
        adjList_2 = graph_new_2.convert_to_list(weight=True)
        # Thực hiện UCS
        ucs_2_path, ucs_2_min_weight = graph_new_2.ucs(graph_new_2.start_node, graph_new_2.end_node)
        print("UCS path: ", ucs_2_path, "\nwith min weight: ", ucs_2_min_weight)

    except FileNotFoundError:
        print("Lỗi: Không tìm thấy tệp đầu vào.")
    except Exception as e:
        print(f"Lỗi xảy ra: {e}")
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

```
PS D:\hoctap\trí tuệ nhân tạo\BTTH1> python -u "d:\hoctap\trí tuệ nhân tạo\BTTH1\main.py"
BFS path:  [0, 2, 7, 8, 10, 14, 15, 16, 17]
DFS path:  [0, 3, 4, 13, 10, 14, 15, 16, 17]
UCS path:  [0, 2, 7, 8, 10, 14, 15, 16, 17]
with min weight:  6910
PS D:\hoctap\trí tuệ nhân tạo\BTTH1>
```

Giống với đáp án khi chạy tay