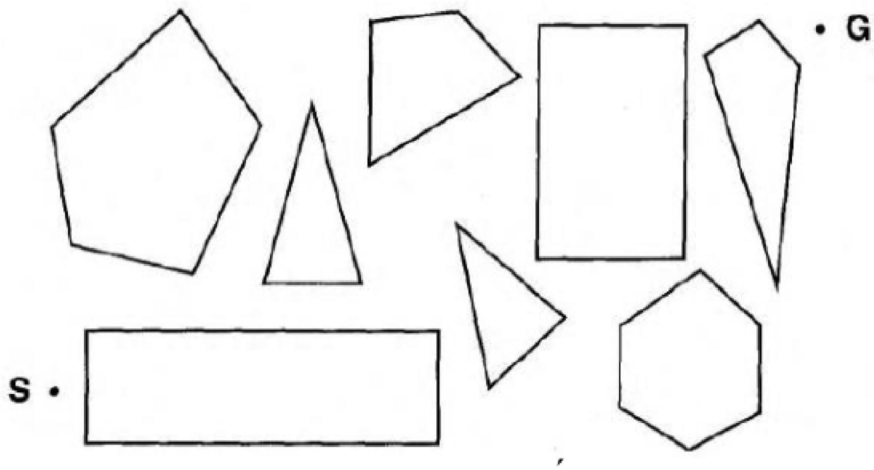
BÁO CÁO THỰC HÀNH NHẬP MÔN TRÍ TUỆ NHÂN TẠO

Tuần 4

Bài toán. Xét bài toán tìm đường đi ngắn nhất từ điểm S tới điểm G trong một mặt phẳng có các chướng ngại vật là những đa giác lồi như hình.



(a) Giả sử không gian trạng thái chứa tất cả các vị trí (x, y) nằm trong mặt phẳng. Có bao nhiêu trạng thái ở đây? Có bao nhiêu đường đi từ đỉnh xuất phát tới đỉnh đích?

(b) Giải thích ngắn gọn vì sao đường đi ngắn nhất từ một đỉnh của đa giác tới một đỉnh khác trong mặt phẳng nhất định phải bao gồm các đoạn thẳng nối một số đỉnh của các đa giác? Hãy định nghĩa lại không gian trạng thái. Không gian trạng thái này sẽ lớn bao nhiêu?

(c) Định nghĩa các hàm cần thiết để thực thi bài toán tìm kiếm, bao gồm hàm successor nhận một đỉnh làm đầu vào và trả về tập đỉnh có thể đi đến được từ đỉnh đó trong vòng 1 bước.

(d) Áp dụng một thuật toán tìm kiếm để giải bài toán.

1 Dữ liệu đầu vào

input.txt

```
8   0   200 480 0
5   60  0   0   60  20  100 100 120 140 40
3   180 20  140 120 220 120
4   260 0   220 20  220 80  300 20
4   320 0   320 100 380 100 380 0
4   440 0   400 0   440 120 460 20
4   20  160 20  220 240 220 240 160
3   260 100 300 180 340 140
6   400 120 360 160 360 200 400 220 440 200 440 160
```

Trong đó

- Dòng 1: N S_x S_y G_x G_y
  - N là số đa giác trong mặt phẳng ($0 \leq N \leq 100$)
  - (S_x, S_y) là toạ độ đỉnh xuất phát, (G_x, G_y) là toạ độ đỉnh đích.
- N dòng tiếp theo: M X_1 Y_1 ... X_M Y_M
  - M là số đỉnh của đa giác ($3 \leq M \leq 10$)
  - (X_i, Y_i) là toạ độ thực của đỉnh thứ i trong đa giác.

2 Xử lý dữ liệu đầu vào

Đọc dữ liệu từ file ở trên và trả về

- **number_polygons**: số đa giác.
- **start_point**: điểm bắt đầu.
- **goal_point**: điểm đích.
- **polygons**: danh sách các điểm theo từng đa giác.
- **number_vertices**: danh sách số điểm của từng đa giác.

```python
def handle_input(name_file):
    with open(name_file, 'r') as f:
        number_polygons, start_x, start_y, goal_x, goal_y = [float(num) for num in f.readline().split('\t')]
        number_polygons = int(number_polygons)
        start_point = (start_x, start_y, 'S')
        goal_point = (goal_x, goal_y, 'G')
        polygons = []
        number_vertices = []
        for i in range(number_polygons):
            polygon_values = [float(num) for num in f.readline().split('\t')]
            number_vertices.append(int(polygon_values[0]))
            polygon = []
            for j in range(1, len(polygon_values) - 1, 2):
                vertice = (polygon_values[j], polygon_values[j+1], i+1)
                polygon.append(vertice)
            polygons.append(polygon)
        f.close()
    return number_polygons, start_point, goal_point, polygons, number_vertices
```

Giả sử không gian trạng thái chứa tất cả các vị trí (x,y) nằm trong mặt phẳng, thì có tất cả 35 trạng thái trong dữ liệu này. Có vô số đường đi từ đỉnh xuất phát tới đỉnh đích trong mặt phẳng này.

3 Ý tưởng bài toán

Tìm đường đi ngắn nhất từ điểm xuất phát tới đỉnh đích bằng cách đi qua các đoạn thẳng nối một số đỉnh của các đa giác. Ta nhận thấy rằng, đường đi ngắn nhất cần tìm là một đường gấp khúc, trên lý thuyết sẽ có đồ dài ngắn hơn so với những đường đi dạng đường cong. Và đường gấp khúc này sẽ tối ưu hơn tức ngắn hơn nếu nó đi qua các đỉnh của các đa giác vì theo bất đẳng thức tam giác thì đó là các đoạn ngắn nhất.
Ta cần định nghĩa lại không gian trạng thái. Không gian trạng thái bây giờ sẽ chứa các vị trí (x,y) nằm trong mặt phẳng và danh sách các vị trí có thể đi đến đước tương ứng của từng vị trí (x,y). Không gian trạng thái này có độ lớn tối đa là 35^2 trạng thái.

4 Thực thi bài toán

4.1 Khởi tạo class Graph

```
class Graph:
    def __init__(self, number_polygons, polygons, number_vertices):
        self.number_polygons = number_polygons
        self.graph = defaultdict(list)
        self.weights = defaultdict(list)
        self.polygons = polygons
        self.number_vertices = number_vertices
```

Trong đó

- **number_polygons**: số đa giác.
- **graph**: danh sách các điểm có thể đi đến được theo từng điểm.
- **weights**: danh sách khoảng cách đến các điểm có thể đi đến được theo từng điểm.
- **polygons**: danh sách các điểm theo từng đa giác.
- **number_vertices**: danh sách số đỉnh của các đa giác.

4.2 Các hàm và phương thức hỗ trợ thực thi bài toán

```python
def is_different_side(edge, point_1, point_2):
    x1 = edge[0][0]
    y1 = edge[0][1]
    x2 = edge[1][0]
    y2 = edge[1][1]
    d1 = (point_1[0] - x1)*(y2 - y1) - (point_1[1] - y1)*(x2 - x1)
    d2 = (point_2[0] - x1)*(y2 - y1) - (point_2[1] - y1)*(x2 - x1)
    return d1*d2 < 0

def distance(point1, point2):
    return np.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)
```

- Hàm **is_different_side** là hàm kiểm tra 2 điểm có nằm khác phía so với một đường thẳng trong mặt phẳng hay không.
- Hàm **distance** là hàm trả về khoảng cách giữa 2 điểm trong mặt phẳng.

```python
    def get_all_edges(self):
        edges = []
        for i in range(self.number_polygons):
            pol = self.polygons[i]
            num_ver = self.number_vertices[i]
            for j in range(num_ver):
                edge = [pol[j % num_ver], pol[(j + 1) % num_ver]]
                edges.append(edge)
        return edges
```

- Phương thức **get_all_edges** trả về danh sách các cạnh của các đa giác trong mặt phẳng.

```python
    def get_all_points(self, start_point, goal_point):
        points = [start_point, goal_point]
        for polygon in self.polygons:
            for i in range(len(polygon)):
                points.append(polygon[i])
        return points
```

- Phương thức **get_all_points** trả về danh sách các điểm có trong mặt phẳng.

```python
    def get_adjacent_point(self):
        adjacent_points = defaultdict(list)
        edges = self.get_all_edges()
        for edge in edges:
            point1 = edge[0]
            point2 = edge[1]
            adjacent_points[point1].append(point2)
            adjacent_points[point2].append(point1)
        return adjacent_points
```

- Phương thức **get_adjacent_points** trả về danh sách các đỉnh kề của các đỉnh đa giác.

```python
    def get_weight(self):
        for cur_point in self.graph.keys():
            for point in self.graph[cur_point]:
                self.weights[cur_point].append(round(distance(cur_point, point), 2))
```

- Phương thức **weight** trả về danh sách khoảng cách đến các điểm có thể đi đến được theo từng điểm.

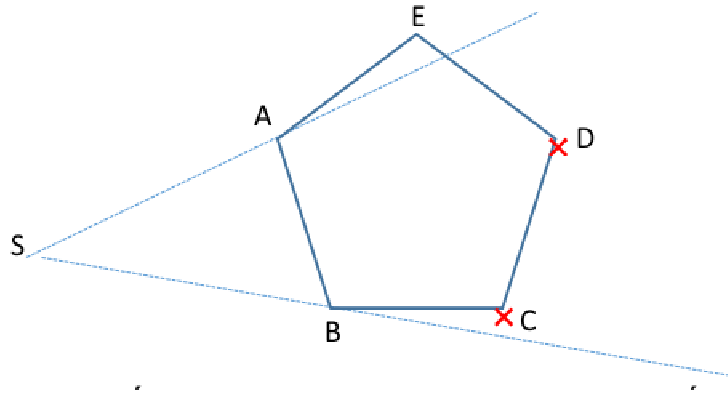
4.3 Hàm successor

Hàm **successor** là hàm nhận vào danh sách các điểm và trả về tập điểm có thể đi đến được từ từng điểm.

```python
    def successor(self, start_point, goal_point):
        edges = self.get_all_edges()
        points = self.get_all_points(start_point, goal_point)
        adjacent_points = self.get_adjacent_point()
        viewless_points = defaultdict(list)
        for cur_point in points:
            for edge in edges:
                for point in points:
                    if (point == cur_point) or (point in edge):
                        continue
                    if (point[2] == cur_point[2]) and (point not in adjacent_points[cur_point]):
                        if point not in viewless_points[cur_point]:
                            viewless_points[cur_point].append(point)
                    else:
                        edge1 = (cur_point, edge[0])
                        edge2 = (cur_point, edge[1])
                        if (is_different_side(edge1, edge[1], point) == False\
                            and is_different_side(edge2, edge[0], point) == False\
                            and is_different_side(edge, cur_point, point) == True):
                            if point in self.graph[cur_point]:
                                self.graph[cur_point].remove(point)
                            if point not in viewless_points[cur_point]:
                                viewless_points[cur_point].append(point)
                        else:
                            if point not in viewless_points[cur_point]:
                                if point not in self.graph[cur_point]:
                                    self.graph[cur_point].append(point)
```

4.3.1 Ý tưởng

Từ S (hay là từ một đỉnh đang xét bất kì) và các cạnh AB của các đa giác, những đỉnh nằm trong cung ASB sẽ bị loại ⇔ những điểm không bị loại là đỉnh nhìn thấy được.



⇔ E là đỉnh nhìn thấy, C và D là 2 đỉnh không nhìn thấy.

4.3.2 Các bước thực hiện

- Gọi P là một đỉnh đang xét, V là tập cạnh của tất cả các đa giác.
- Với mỗi cạnh đa giác, gọi là AB, trong tập V:
  - Tạo d_1 từ P và A, d_2 từ P và B, d_3 từ A và B.
  - Xét tất cả các đỉnh Q còn lại với d_1 , d_2 , d_3.
    - Nếu $d_1(Q) * d_1(B) \geq 0$ và $d_2(Q) * d_2(A) \geq 0$ và $d_3(Q) * d_3(P) < 0$ thì Q là đỉnh không nhìn thấy được từ P.
    - Ngược lại, nhìn thấy được từ P.

4.4 Tìm đường đi ngắn nhất

Sử dụng thuật toán **Uniform-cost search** với chi phí là khoảng cách giữa các đỉnh trong mặt phẳng để tìm ra đường đi ngắn nhất từ điểm xuất phát tới điểm đích.

```python
    def find_shortest_path(self, start_point, goal_point):
        frontier = PriorityQueue()
        frontier.put((0, [start_point]))
        explored = set()
        while frontier:
            if frontier.empty():
                raise Exception("No way Exception")
            current_w, path = frontier.get()
            current_point = path[len(path)-1]
            if current_point not in explored:
                explored.add(current_point)
                if current_point == goal_point:
                    return round(current_w, 2), path
            for i in range(len(self.graph[current_point])):
                point = self.graph[current_point][i]
                weight = self.weights[current_point][i]
                if point not in explored:
                    temp = path[:]
                    temp.append(point)
                    frontier.put((weight + current_w, temp))
```

5 Kết quả thực thi

Kết quả chạy code

```
The shortest path from (0.0, 200.0, 'S')  to  (480.0, 0.0, 'G') :
Distance:  564.68
(0.0, 200.0, 'S')  --> (20.0, 160.0, 6)  --> (100.0, 120.0, 1)  --> (180.0, 20.0, 2)  --> (260.0, 0.0, 3)
 --> (320.0, 0.0, 4)  --> (380.0, 0.0, 4)  --> (400.0, 0.0, 5)  --> (440.0, 0.0, 5)  --> (480.0, 0.0, 'G')
```

Kết quả mô phỏng