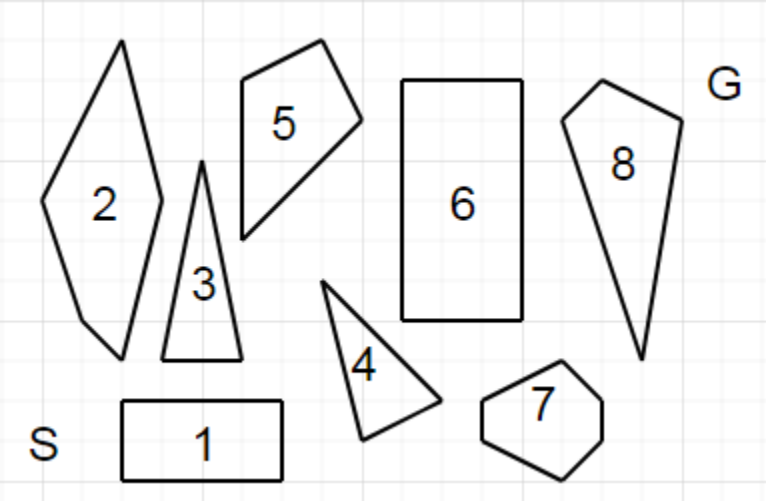
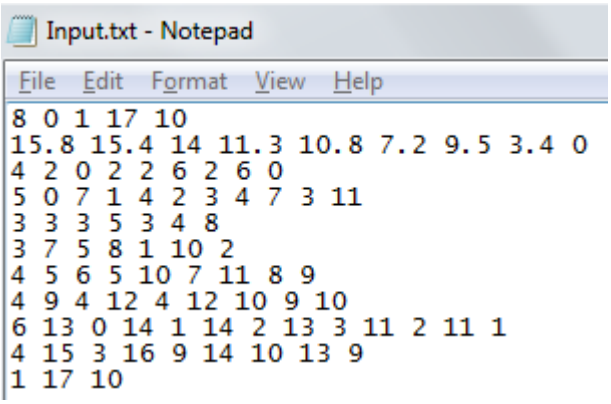
# BÁO CÁO BÀI THỰC HÀNH 4

## I. Đồ thị:




## II. Đọc file .txt đầu vào:




Dòng 1: N $S_x$ $S_y$ $G_x$ $G_y$

N: Số lượng đa giác

$S_x$ $S_y$ $G_x$ $G_y$: Tọa độ điểm bắt đầu và đích, cụ thể, (0,1) và (17,10)

Dòng 2: Heuristic, là khoảng cách từ trung điểm từng đa giác đến điểm đích được tính thủ công. Ở đây ta sẽ dùng thuật toán tìm kiếm A*

Các dòng còn lại: Đại diện cho các đa giác được xếp theo thứ tự. Số đầu tiên cho biết số đỉnh của đa giác, các số sau lần lượt là tọa độ của từng đỉnh.

```
with open("Input.txt") as file:
    line1 = file.readline().split()
    ShapeAmount = int(line1[0])
    start = (int(line1[1]), int(line1[2]), 0)
    goal = (int(line1[3]), int(line1[4]), 9)
    heuristic = [float(num) for num in file.readline().split()]
    graph = []
    for x in range(ShapeAmount + 1):
        shape = []
        line = file.readline().split()
        vertices = int(line[0])
        for i in range(vertices):
            location = (int(line[i * 2 + 1]), int(line[i * 2 + 2]), x + 1)
            shape.append(location)
        graph.append(shape)
```

Khi đọc dữ liệu vào, ngoài tọa độ (x, y), ta còn phải thêm tham số cho biết tọa độ (x, y) thuộc đa giác nào do đề bài yêu cầu.

Quy ước:

0 : Điểm bắt đầu

1 – 8 : Các đa giác theo thứ tự trong hình ở đầu file báo cáo

9 : Điểm đích

Chú ý: Ở đây, ta sẽ xem điểm đích như một đa giác có 1 đỉnh. Nên ShapeAmount = 8, khi làm vòng lặp để lưu dữ liệu vào mảng 2 chiều, ta cần cộng thêm 1, nếu không sẽ đọc thiếu dòng cuối trong file .txt

## III. Các hàm trợ giúp:

```python
def line_func(ver1, ver2):
    if (ver1[0] == ver2[0]):
        a = 1; b = 0; c = -ver1[0]
    elif (ver1[1] == ver2[1]):
        a = 0; b = 1; c = -ver1[1]
    else:
        a = ver1[1] - ver2[1]
        b = ver2[0] - ver1[0]
        c = - ver1[0] * a - ver1[1] * b
    return a, b, c
```

Hàm line_func dùng để trả về phương trình đường thẳng đi qua 2 điểm đóng vai trò tham số đầu vào ver1, ver2 (ver là viết gọn của vertice)

$$(y_1 - y_2)(x - x_1) + (x_2 - x_1)(y - y_1) = 0$$

$$\Longleftrightarrow \quad ax + by + c = 0$$

```python
def line_check(d, ver1, ver2):
    first = d[0] * ver1[0] + d[1] * ver1[1] + d[2]
    second = d[0] * ver2[0] + d[1] * ver2[1] + d[2]
    return first * second

def visible_vertice(d1, d2, d3, current, ver1, ver2, vertice):
    first = line_check(d1, vertice, ver2)
    second = line_check(d2, vertice, ver1)
    third = line_check(d3, vertice, current)
    if first >= 0 and second >= 0 and third < 0:
        return False
    return True
```

Hàm line_check dùng để cho biết vị trí tương đối của 2 điểm ver1, ver2 so với đường thẳng *d*.

Hàm visible_vertice dùng để xét xem từ đỉnh current đang xét, ta có thể nhìn thấy được đỉnh vertice không. Điều kiện kiểm tra được lấy từ file hướng dẫn.

## IV. Ý tưởng bài toán:

```python
import math
from queue import PriorityQueue
def find_way_aStar(start, goal, graph, heu):
    frontier = PriorityQueue()
    frontier.put([0, [0, [start]]])
    explored = []
    while True:
        if frontier.empty():
            raise Exception("No way found")
        f_value, cost_and_path = frontier.get()
        move_cost, path = cost_and_path
        #print(type(path), "Current path:", path)
        #print(type(move_cost), "Current cost:", move_cost)
        #print(type(f_value), "Current cost + heuristic:", f_value, '\n')
        current = path[-1]
        explored.append(current)
        if current == goal:
            solution = ""
            for node in path[0:-1]:
                solution += str(node) + " --> "
            solution += str(path[-1])
            print("Solution:", solution)
            return
```

*Uncomment các dòng comment đã thấy rõ, path, move_cost, f_value trong mỗi lần lặp"*

Do ta sẽ áp dụng thuật toán A*, nên ta sẽ dùng hàng đợi ưu tiên. Hàng đợi ưu tiên *frontier* chứa các phần tử có dang [a, [b, [c]]].

- c: List chứa đỉnh đại diện cho đường đi
- b: Chi phí để đi được c, là tổng độ dài đã đi được theo Pytago
- a: Tổng của b và heuristic của phần tử cuối cùng của c. Giá trị a quyết định vị trí của phần tử trong hàng đợi ưu tiên. Ví dụ, với đường đi c là [(0,1,0), (2,2,1)], chi phí b là sqrt(5), heuristic của phần từ cuối cùng của c là 15.8 (khoảng cách từ trung điểm đa
  giác 1 đến điểm đích), ta sẽ có giá trị a = sqrt(5) + 15.8 = 18. Vậy ta có [18, [sqrt(5), [(0,1,0), (2,2,1)]]].

Đỉnh đang xét *current* là phần tử cuối của *path*. Ta thêm current vào hàng đợi và bắt đầu xét.

Nếu *current* là điểm đích, ta in ra *path* là kết quả bài toán.

```
        invisible = set()
        for shape in graph:
            for i in range(len(shape)):
                if i == len(shape) - 1:
                    ver1 = shape[i]
                    ver2 = shape[0]
                else:
                    ver1 = shape[i]
                    ver2 = shape[i + 1]
                d1 = line_func(current, ver1)
                d2 = line_func(current, ver2)
                d3 = line_func(ver1, ver2)
                for shape2 in graph:
                    for vertice in shape2:
                        if vertice not in explored:
                            if not visible_vertice(d1, d2, d3, current, ver1, ver2, vertice):
                                invisible.add(vertice)
        count = -1
        for shape in graph:
            count = count + 1
            for vertice in shape:
                if vertice not in explored and vertice not in invisible:
                    new_path = list(path)
                    new_path.append(vertice)
                    cost = math.sqrt((current[0] - vertice[0]) ** 2 + (current[1] - vertice[1]) ** 2)
                    frontier.put([move_cost + cost + heu[count], [move_cost + cost, new_path]])
```

Nếu *current* không phải điểm đích ta tiến hành mở rộng

*current*. Vòng lặp 1:

Tiến hành duyệt qua từng cạnh của từng đa giác, tương ứng mỗi cạnh ta có 2 đỉnh ver1, ver2. Ta khởi tạo một vòng lặp bên trong để duyệt qua tất cả các đỉnh của tất cả đa giác mà không có trong *explored* (để tránh trường hợp đi từ A đến B xong lại đi ngược từ B về A) để xem đỉnh nào không nhìn thấy được. Theo đó, ta thêm các đỉnh không nhìn thấy được vào tập hợp *invisible*.

Vòng lặp 2:

Ta duyệt qua tất cả các đỉnh của tất cả đa giác. Đỉnh nào không nằm trong *explored* và *invisible*, ta sẽ thêm vào *new_path* rồi thêm *new_path* cùng với các tham số cần thiết vào hàng đợi ưu tiên.

Trong đó, cost là khoảng cách Pytago từ đỉnh đang xét *current* với đỉnh *vertice* có được từ vòng lặp. Biến đếm *count* để cho biết ta đang ở đa giác nào, nhờ đó mà ta lấy được giá trị Heuristic.

## V. Chạy Code:

```
93      find_way_aStar(start, goal, graph, heuristic)
Solution: (0, 1, 0) --> (2, 2, 1) --> (5, 3, 3) --> (7, 5, 4) --> (9, 4, 6) --> (12, 4, 6) --> (13, 9, 8) --> (14, 10, 8) --> (17, 10, 9)
Cost: 22.97607381588003
Press any key to continue . . .
```