

## NHẬP MÔN TRÍ TUỆ NHÂN TẠO

Tên :TRẦN KIM KHANH

Mssv:21110318

### THỰC HÀNH 4

Giải thích từng dòng code:

Class Point sẽ xử lý các điểm sẽ:

x và y là tọa độ điểm

hàm :Xác định 2 điểm có nằm cùng phía với đường line hay không

```
Tabnine | Edit | Test | Explain | Document | Ask
def rel(self, other, line):
    return line.d(self) * line.d(other) >= 0
```

Hàm :xác định điểm hiện tại có nhìn thấy điểm other qua một đường thẳng line hay không

Nếu 1 trong d1 , d2 ,d3 không thỏa thì trả về true

```
Tabnine | Edit | Test | Explain | Document | Ask
def can_see(self, other, line):
    l1 = self.line_to(line.p1)
    l2 = self.line_to(line.p2)
    d3 = line.d(self) * line.d(other) < 0
    d1 = other.rel(line.p2, l1)
    d2 = other.rel(line.p1, l2)
    return not(d1 and d2 and d3)
```

Hàm : tạo đoạn thẳng và tính khoảng cách giữa 2 điểm

```

5  Tabnine | Edit | Test | Explain | Document | Ask
   def line_to(self, other):
6     return Edge(self, other)
7
8  Tabnine | Edit | Test | Explain | Document | Ask
   def heuristic(self, other):
9     return euclid_distance(self, other)
10

```

Class Edge tạo các cạnh lưu điểm đầu và cuối của cạnh

```

49
50  class Edge(object):
   Tabnine | Edit | Test | Explain | Document | Ask
51     def __init__(self, point1, point2):
52         self.p1 = point1
53         self.p2 = point2
54
   Tabnine | Edit | Test | Explain | Document | Ask

```

hàm : Tìm điểm liền kề

```

   Tabnine | Edit | Test | Explain | Document | Ask
   def get_adjacent(self, point):
       if point == self.p1:
           return self.p2
       if point == self.p2:
           return self.p1

```

Hàm : tính khoảng cách hướng (signed distance) từ một điểm đến một đoạn thẳng.

```

   Tabnine | Edit | Test | Explain | Document | Ask
   def d(self, point):
       vect_a = Point(self.p2.x - self.p1.x, self.p2.y - self.p1.y)
       vect_n = Point(-vect_a.y, vect_a.x)
       return vect_n.x * (point.x - self.p1.x) + vect_n.y * (point.y - self.p1.y)

```

Class Graph:

**self.graph:** Lưu trữ các điểm trong đồ thị, mỗi điểm là khóa, giá trị là tập hợp các điểm kề với điểm đó.

**self.edges:** Lưu trữ tất cả các cạnh trong đồ thị, mỗi cạnh là một đối tượng Edge.

**self.polygons:** Lưu trữ các đa giác, mỗi đa giác có một ID duy nhất (là khóa), giá trị là tập hợp các cạnh của đa giác đó.

**pid:** Biến lưu ID của đa giác, giúp phân biệt các đa giác khác nhau trong đồ thị.

```
99 class Graph:
100
101     def can_see(self, start):
102         see_list = list()
103         cant_see_list = list()
104
105         for polygon in self.polygons:
106             for edge in self.polygons[polygon]:
107                 for point in self.get_adjacent_points(edge):
108                     # Kiểm tra điểm xuất phát
109                     if start == point:
110                         cant_see_list.append(point)
111                     # Kiểm tra nếu điểm start có trong các điểm của đa giác:
112                     if start in self.get_polygon_points(polygon):
113                         for poly_point in self.get_polygon_points(polygon):
114                             # nếu không phải là điểm kề thì thêm vào tập không nhìn thấy
115                             if poly_point not in self.get_adjacent_points(start):
116                                 cant_see_list.append(poly_point)
117                     if point not in cant_see_list:
118                         # kiểm tra xem
119                         if start.can_see(point, edge):
120                             if point not in see_list:
121                                 see_list.append(point)
122                             elif point in see_list:
123                                 see_list.remove(point)
124                                 cant_see_list.append(point)
125                         else:
126                             cant_see_list.append(point)
127
128         return see_list
```

Hàm `get_adjacent_points` trả về một danh sách các điểm kề với điểm `point` trong đồ thị, thông qua việc duyệt qua tất cả các cạnh và lấy các điểm kề từ mỗi cạnh.

```
Tabnine | Edit | Test | Explain | Document | Ask
def get_adjacent_points(self, point):
    return list(filter(None.__ne__, [edge.get_adjacent(point) for edge in self.edges]))
Tabnine | Edit | Test | Explain | Document | Ask
```

Hàm `can_see(self, start)` kiểm tra tất cả các điểm có thể nhìn thấy từ điểm `start`

```

class Graph:
    Tabnine | Edit | Test | Explain | Document | Ask
    def __init__(self, polygons):
        self.graph = defaultdict(set)
        self.edges = set()
        self.polygons = defaultdict(set)
        pid = 0

        for polygon in polygons:
            if len(polygon) == 2:
                polygon.pop()
            if polygon[0] == polygon[-1]:
                self.add_point(polygon[0])
            else:
                for i, point in enumerate(polygon):
                    neighbor_point = polygon[(i + 1) % len(polygon)]
                    edge = Edge(point, neighbor_point)
                    if len(polygon) > 2:
                        point.polygon_id = pid
                        neighbor_point.polygon_id = pid
                        self.polygons[pid].add(edge)
                    self.add_edge(edge)
                if len(polygon) > 2:
                    pid += 1

```

hàm lấy các điểm trong đa giác

```

Tabnine | Edit | Test | Explain | Document | Ask
def get_polygon_points(self, index):
    point_set = set()
    for edge in self.polygons[index]:
        point_set.add(edge.p1)
        point_set.add(edge.p2)
    return point_set

```

Thuật toán A\* hoặc greedy tùy vào hàm func sử dụng

.

Tabnine | Edit | Test | Explain | Document | Ask

```
def search(graph, start, goal, func):
    closed = set()
    queue = PriorityQueue()
    queue.put((0 + func(graph, start), start))
    if start not in closed:
        closed.add(start)
    while not queue.empty():
        cost, node = queue.get()
        if node == goal:
            return node
        for i in graph.can_see(node):
            new_cost = node.g + euclid_distance(node, i)
            if i not in closed or new_cost < i.g:
                closed.add(i)
                i.g = new_cost
                i.pre = node
                new_cost = func(graph, i)
                queue.put((new_cost, i))

    return node
```

Thuật toán bfs:

```
19
    Tabnine | Edit | Test | Explain | Document | Ask
20 def bfs(graph, start, goal):
21     queue = Queue()
22     queue.put(start)
23     visited = set()
24     visited.add(start)
25     while not queue.empty():
26         node = queue.get()
27         if node == goal:
28             return node
29         for neighbor in graph.can_see(node):
30             if neighbor not in visited:
31                 neighbor.pre = node
32                 visited.add(neighbor)
33                 queue.put(neighbor)
34     return None
35
```

Thuật toán DFS:

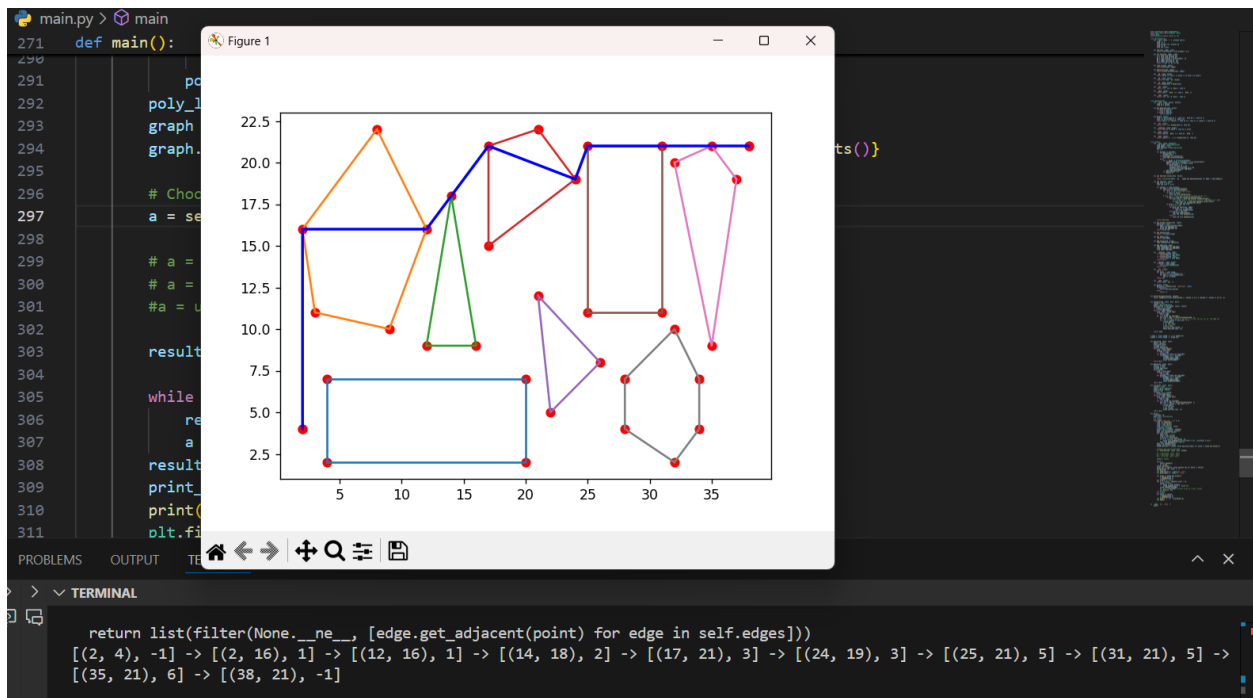
```
Tabnine | Edit | Test | Explain | Document | Ask
36  def dfs(graph, start, goal):
37      stack = [start]
38      visited = set()
39      visited.add(start)
40      while stack:
41          node = stack.pop()
42          if node == goal:
43              return node
44          for neighbor in graph.can_see(node):
45              if neighbor not in visited:
46                  neighbor.pre = node
47                  visited.add(neighbor)
48                  stack.append(neighbor)
49      return None
```

thuật toán USC :

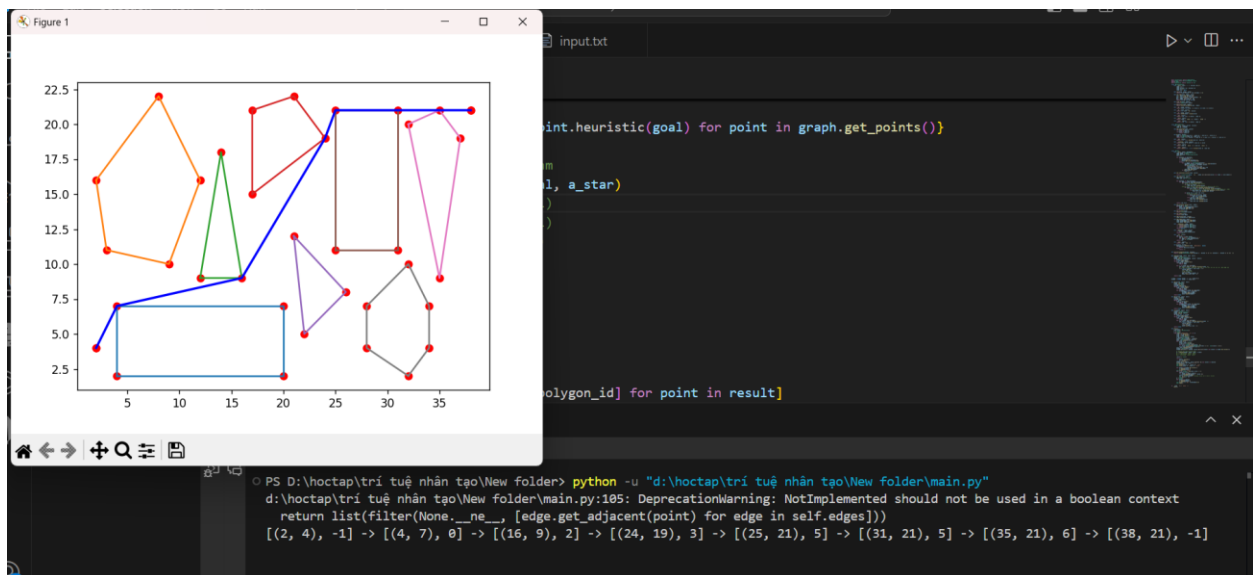
```
Tabnine | Edit | Test | Explain | Document | Ask
251  def ucs(graph, start, goal):
252      closed = set()
253      queue = PriorityQueue()
254      queue.put((0, start))
255      start.g = 0
256      if start not in closed:
257          closed.add(start)
258      while not queue.empty():
259          cost, node = queue.get()
260          if node == goal:
261              return node
262          for i in graph.can_see(node):
263              new_cost = node.g + euclid_distance(node, i)
264              if i not in closed or new_cost < i.g:
265                  closed.add(i)
266                  i.g = new_cost
267                  i.pre = node
268                  queue.put((new_cost, i))
269      return None
```

Kết quả của từng thuật toán:

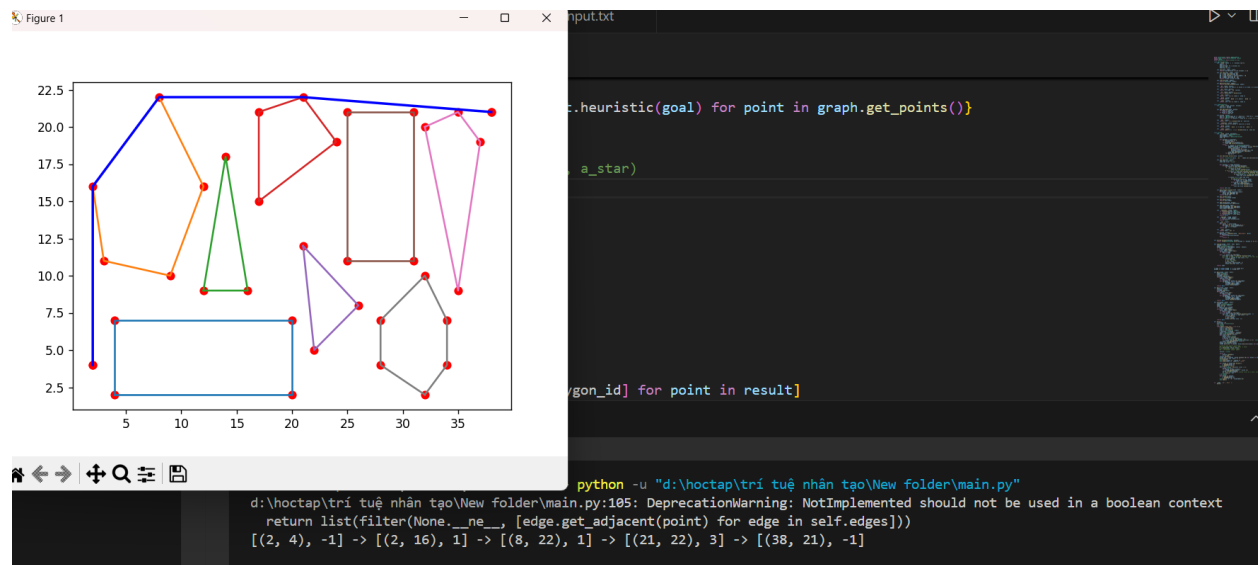
## 1 Tìm kiếm tham lam: Greedy Search:



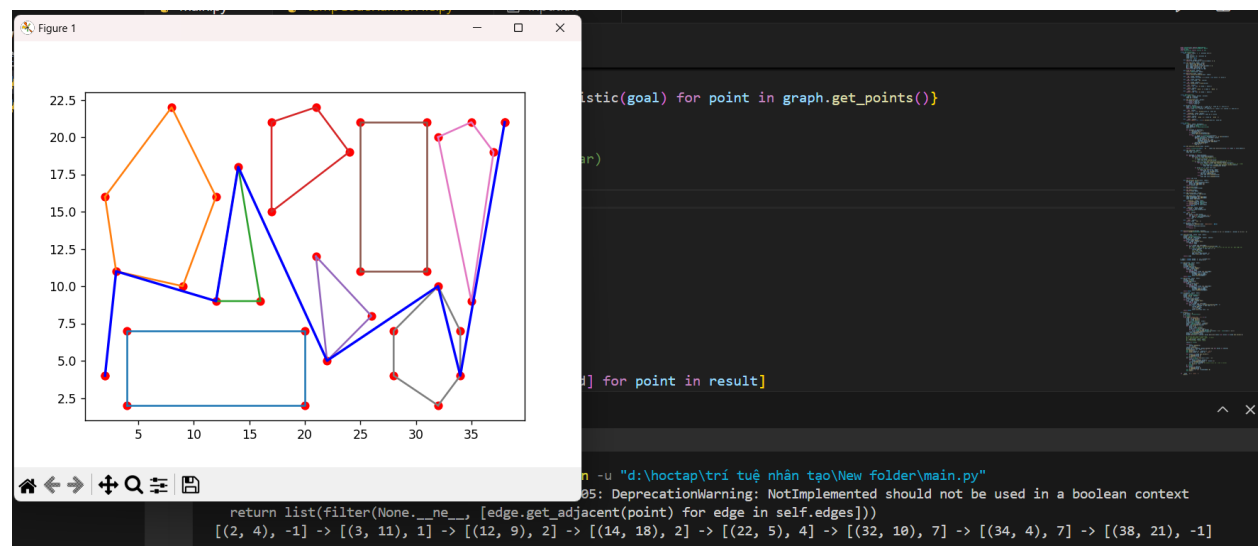
## 2 thuật toán A\*



### 3 thuật toán BFS:



### 4 thuật toán DFS:



### Thuật toán USC:



