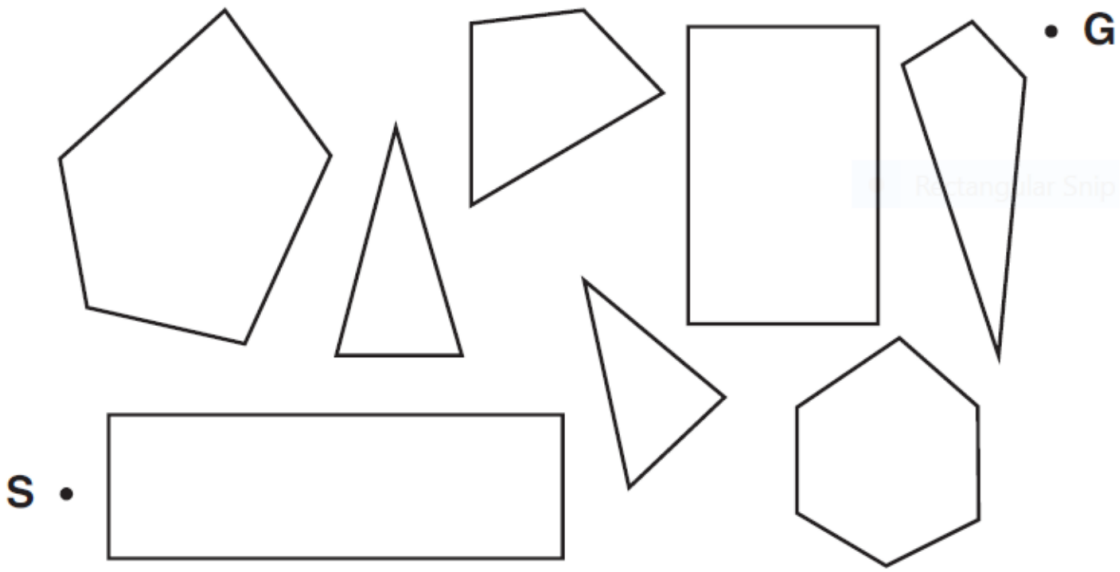
BTTH - NM TTNT - TUẦN 4

21280099 - Nguyễn Công Hoài Nam

Ngày 27 tháng 11 năm 2023

I. Bài toán

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



Hình 1: Minh họa

II. Cài đặt và thực thi chương trình

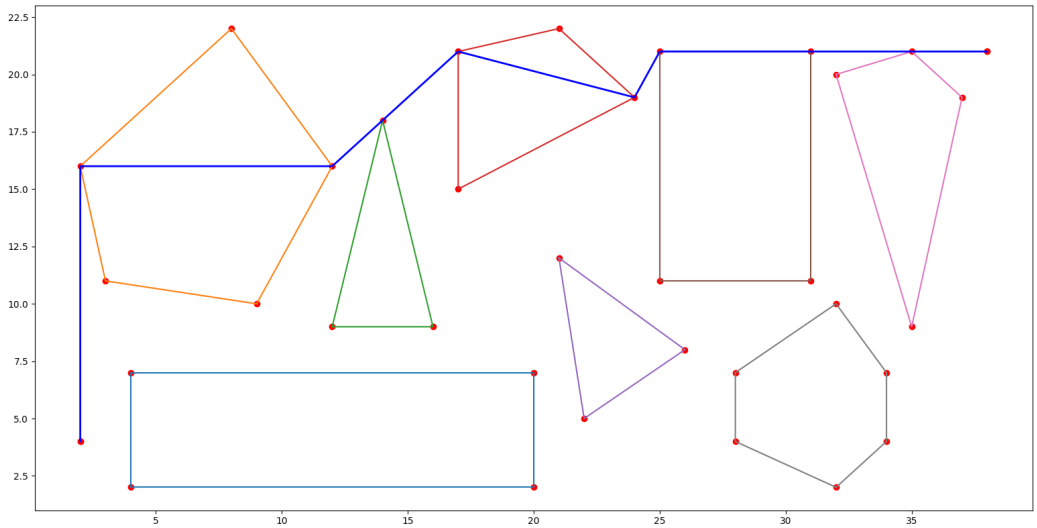
1. Kiểm tra lỗi

Chương trình lỗi tại phương thức **can_see** của lớp **Graph**

```
    def can_see(self, start):
        see_list = list()
        cant_see_list = list()

        for polygon in self.polygons:
            for edge in self.polygons[polygon]:
                for point in self.get_points():
                    if start == point:
                        cant_see_list.append(point)
                    if start in self.get_polygon_points(polygon):
                        for poly_point in self.get_polygon_points(polygon):
                            if poly_point not in self.get_adjacent_points(start):
                                cant_see_list.append(poly_point)
                    if point not in cant_see_list:
                        if start.can_see(point,edge):
                        ### error code
                                if point not in see_list:
                                    see_list.append(point)
                            elif point in see_list:
```

```
                                see_list.remove(point)
                                cant_see_list.append(point)
                            else:
                                cant_see_list.append(point)
                        ###
        return see_list
```

Lý do sai: khoảng trắng của các câu lệnh không hợp lý và thiếu điều kiện lọc lại **see_list** dẫn tới **see_list** sai nên các thuật toán tìm kiếm cũng sai theo



Hình 2: Thuật toán Greedy chạy bị lỗi do can_see sai

Sửa lại: nếu node có cả **see_list** và **cant_see_list** thì remove khỏi **see_list** và khoảng trắng hợp lý

```
    def can_see(self, start):
        see_list = list()
        cant_see_list = list()

        for polygon in self.polygons:
            for edge in self.polygons[polygon]:
                for point in self.get_points():
                    if start == point:
                        cant_see_list.append(point)
                    if start in self.get_polygon_points(polygon):
                        for poly_point in self.get_polygon_points(polygon):
                            if poly_point not in self.get_adjacent_points(start):
                                cant_see_list.append(poly_point)
                    if point not in cant_see_list:
                        if start.can_see(point,edge):
                            if point not in see_list:
                                see_list.append(point)
                        elif point in see_list:
                            see_list.remove(point)
                            cant_see_list.append(point)
                        else:
                            cant_see_list.append(point)
                    if point in see_list and point in cant_see_list:
                        see_list.remove(point)

        return see_list
```

Sau khi sửa lại thì các thuật toán đã chạy đúng (minh họa ở phần sau) Ở hàm search, đoạn cuối nên **return None** (khi tìm kiếm thất bại, thay vì **return node** cuối)

III. Cài đặt thêm thuật toán DFS, BFS, UCS

1. BFS

```
def BFS_search(graph, start, goal):
    closed = set()
    queue = Queue()
    queue.put(start)
    closed.add(start)

    while not queue.empty():
        node = queue.get()

        if node == goal:
            return node

        for i in graph.can_see(node):
            if i not in closed:
                closed.add(i)
                i.pre = node
                queue.put(i)
    return None
```

2. DFS

```
def DFS_search(graph, start, goal):
    closed = set()
    stack = []
    start.pre = None
    stack.append(start)
    closed.add(start)

    while stack:
        node = stack.pop()

        if node == goal:
            return node

        for i in graph.can_see(node):
            if i not in closed and i not in stack:
                closed.add(i)
                i.pre = node
                stack.append(i)

    return None
```

3. UCS

Thêm func cho UCS (dùng chung hàm với Astar, Greedy)

```
ucs = lambda graph, i: i.g
```

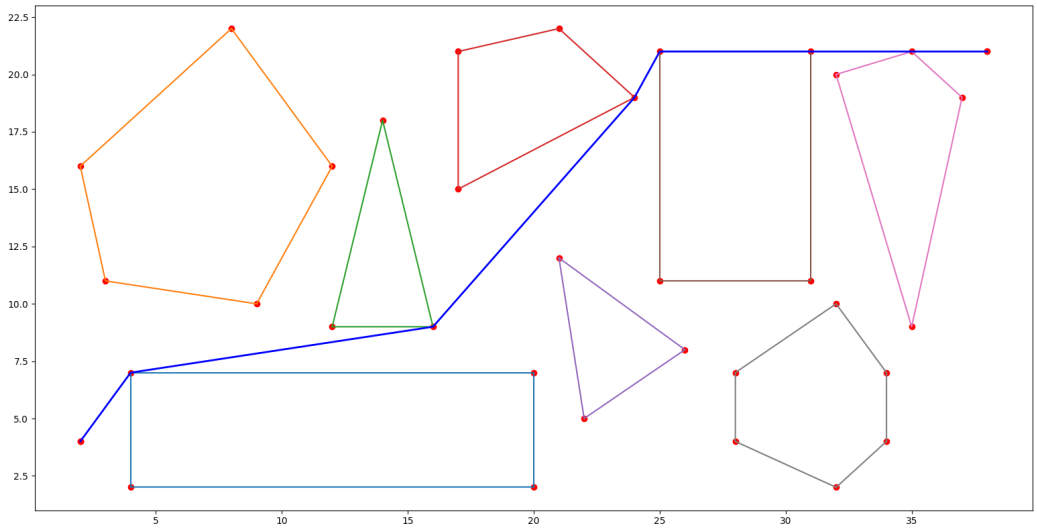
Hoặc xây dựng một hàm riêng biệt

```
def UCS_search(graph, start, goal):
    closed = set()
    PQ = PriorityQueue()
    PQ.put((0,start))
    closed.add(start)

    while not PQ.empty():
        cost, node = PQ.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
                PQ.put((new_cost, i))

    return None
```

IV. Kết quả thực thi

```
[(2, 4), -1] -> [(4, 7), 0] -> [(16, 9), 2] -> [(24, 19), 3] -> [(25, 21), 5] -> [(31, 21), 5] -> [(35, 21), 6] -> [(38, 21), -1]
```

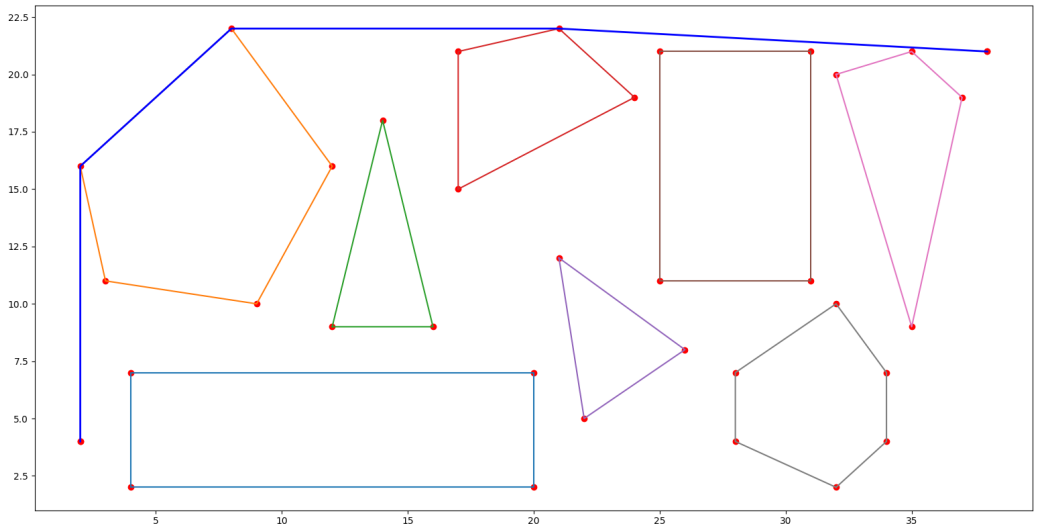
Hình 3: Console thuật toán Astar



Hình 4: Thuật toán Astar

```
[(2, 4), -1] -> [(2, 16), 1] -> [(8, 22), 1] -> [(21, 22), 3] -> [(38, 21), -1]
```

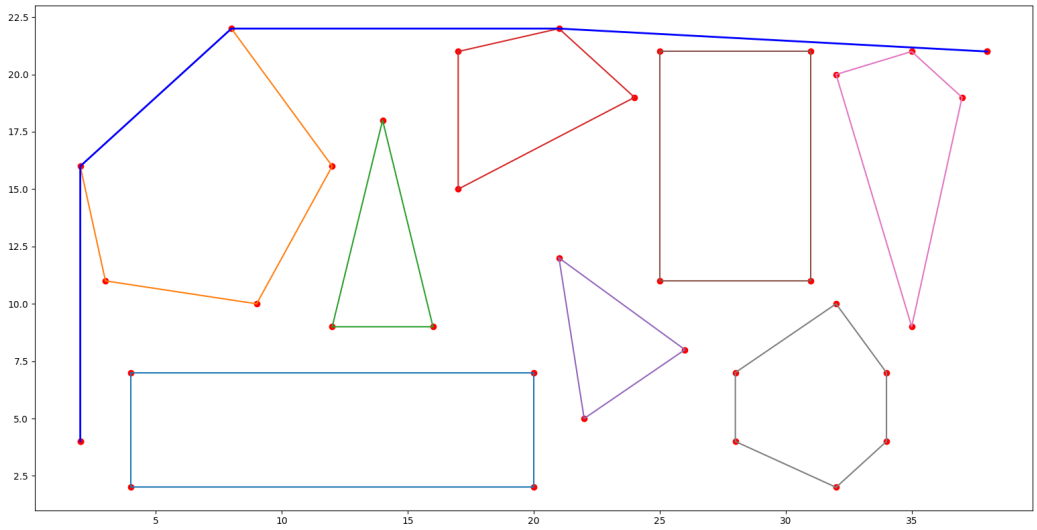
Hình 5: Console thuật toán Greedy



Hình 6: Thuật toán Greedy

```
[(2, 4), -1] -> [(2, 16), 1] -> [(8, 22), 1] -> [(21, 22), 3] -> [(38, 21), -1]
```

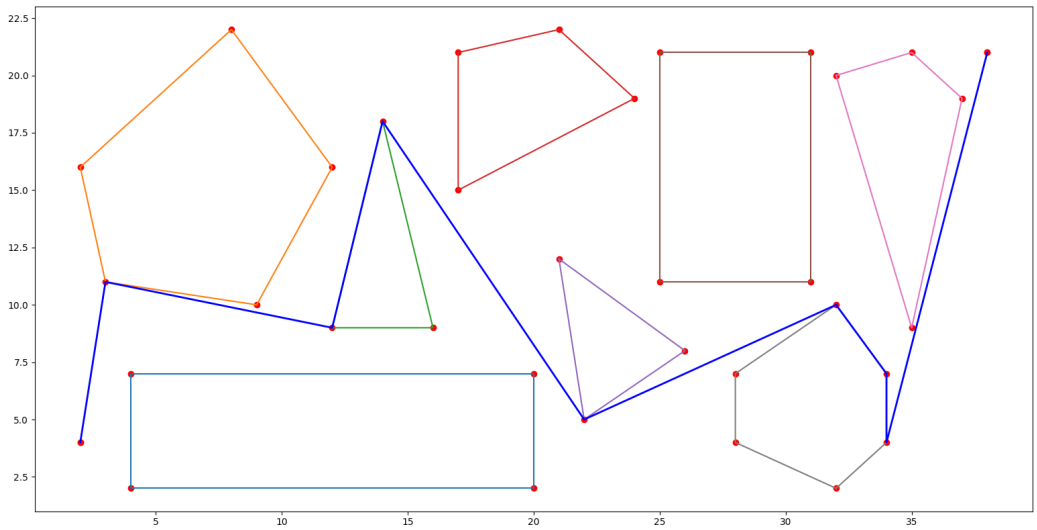
Hình 7: Console thuật toán BFS



Hình 8: Thuật toán BFS

```
[(2, 4), -1] -> [(3, 11), 1] -> [(12, 9), 2] -> [(14, 18), 2] -> [(22, 5), 4] -> [(32, 10), 7] -> [(34, 7), 7] -> [(34, 4), 7] -> [(38, 21), -1]
```

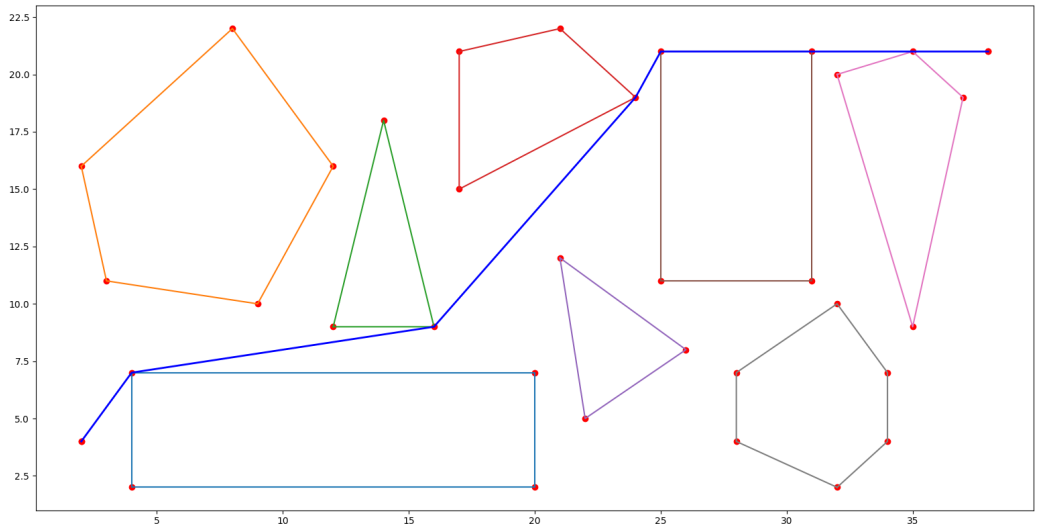
Hình 9: Console thuật toán DFS



Hình 10: Thuật toán DFS

```
[(2, 4), -1] -> [(4, 7), 0] -> [(16, 9), 2] -> [(24, 19), 3] -> [(25, 21), 5] -> [(31, 21), 5] -> [(35, 21), 6] -> [(38, 21), -1]
```

Hình 11: Console thuật toán UCS



Hình 12: Thuật toán UCS

Tất cả thuật toán đều cho ra kết quả đúng.