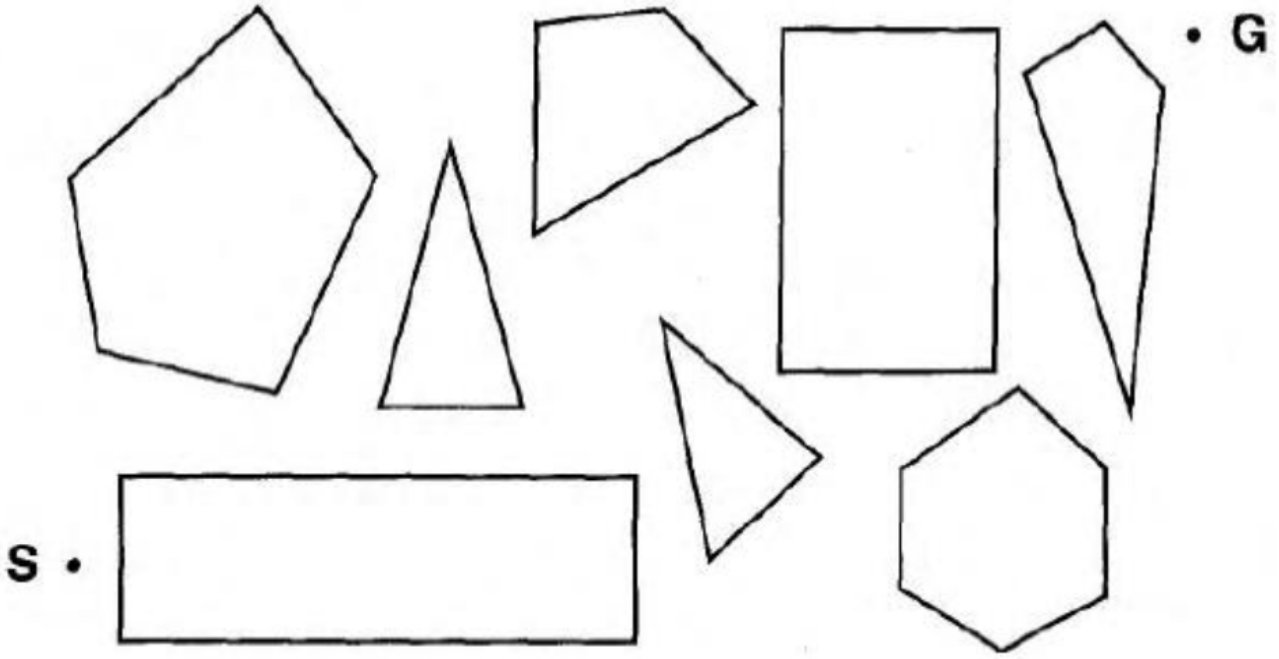
Bài tập Thực hành Nhập môn Trí tuệ Nhân tạo tuần 4

Nguyễn Lê Ngọc Duy - 20280023 - 20KDL1

Mục lục

1 Giới thiệu bài toán.

Trong tuần này chúng ta sẽ vận dụng tất cả các thuật toán tìm kiếm đã được tìm hiểu ở các tuần trước để giải quyết bài toán được đặt ra trong tuần này như sau: Tìm đường đi ngắn nhất từ điểm xuất phát là điểm S tới điểm kết thúc là điểm G trong mặt phẳng hai chiều với các chướng ngại vật là các đa giác lồi như hình 1.



Hình 1: Hình vẽ mô tả bài toán

Trước khi tiến hành giải bài toán, ta cần nhắc lại các thuật toán tìm kiếm đã được chúng ta tìm hiểu ở các tuần trước đó.

2 Các giải thuật tìm kiếm.

Ở phần này, chúng ta sẽ chỉ nhắc lại mã giả của các giải thuật tìm kiêm đã được tìm hiểu ở các tuần trước đó (xem các thuật toán từ 1 đến 5).

Thuật toán 1 Thuật toán Depth First Search - DFS

Đầu vào: đồ thị G, node xuất phát *start*, node kết thúc *end*.
Đầu ra: đường đi từ *start* đến *end*
begin
Khởi tạo danh sách L chứa trạng thái ban đầu.
while true **do**
  if L rỗng **then**
    Thông báo không có đường đi.
    break
  end if
end while
Loại trạng thái u ở đầu danh sách L.
if u là trạng thái kết thúc **then**
  return đường đi từ *start* đến u.
end if
Lấy các trạng thái v kề với u và thêm vào đầu danh sách L.
for mỗi trạng thái v kề với u **do**
  $father(v) = u$
end for
end

Thuật toán 2 Thuật toán Breath First Search - BFS

Đầu vào: đồ thị G, node xuất phát *start*, node kết thúc *end*.
Đầu ra: đường đi từ *start* đến *end*
begin
Khởi tạo danh sách L chứa trạng thái ban đầu.
while true **do**
  if L rỗng **then**
    Thông báo không có đường đi.
    break
  end if
end while
Loại trạng thái u ở đầu danh sách L.
if u là trạng thái kết thúc **then**
  return đường đi từ *start* đến u.
end if
Lấy các trạng thái v kề với u và thêm vào cuối danh sách L.
for mỗi trạng thái v kề với u **do**
  $father(v) = u$;
end for
end

Thuật toán 3 Thuật toán Uniform Cost Search - UCS

Đầu vào: đồ thị G, node xuất phát *start*, node kết thúc *end*.
Đầu ra: đường đi từ *start* đến *end* và chi phí của đường đi
begin
Tạo hàng đợi ưu tiên rỗng $NganChua$.
Thêm nút *start* vào $NganChua$.
loop
  if $NganChua$ rỗng **then**
    return không tồn tại đường đi.
  else
    Nut = TimChiPhiNhoNhat($NganChua$)
    if KiemTraCauHoiDich($BaiToan$) trên TrangThai(Nut) đúng **then**
      return LoiGiai(Nut)
    end if
  end if
end loop
$LoiGiai$ = Mo(Nut, $BaiToan$)
$NganChua$ = ThemTatCa($LoiGiai$, $NganChua$)
end

Thuật toán 4 Thuật toán Greedy Best First Search (GBFS)

Đầu vào: Bài toán.
Đầu ra: Lời giải hoặc thông báo: Không tồn tại lời giải.

```
insert(state = initial_state, priority = 0) into search.queue;
while search.queue not empty do
  current_queue.entry = pop item from front of search.queue
  current_state = current_queue.entry.state;
  current_heuristic = current_queue.entry.heuristic;
  starting_counter = counter from current_queue.entry;
  applicable_actions = array of actions applicable in current_state;
  for all index ?i in applicable_actions >= starting_counter do
    current_action = applicable_actions[?i];
    successor_state = current_state.apply(current_action);
    if successor_state is goal state then
      return solution_path;
    end if
    successor_heuristic = heuristic value of successor_state;
    if successor_heuristic < current_heuristic then
      insert(current_state, current_heuristic, ?i + 1) to search.queue;
      insert(successor_state, successor_heuristic, 0) to search.queue;
      break for;
    else
      insert(successor_state, successor_heuristic, 0) to search.queue;
    end if
  end for
end while
```

Thuật toán 5 Thuật toán A*

Đầu vào: Bài toán.
Đầu ra: Lời giải hoặc thông báo: Không tồn tại lời giải.

OPEN $= T_0$
$g(T_0) = 0, h(T_0) = 0, f(T_0) = 0$
CLOSE $= \emptyset$
loop
  if OPEN rỗng **then**
    Bài toán vô nghiệm.
    exit
  else
    Lấy $T_{\max}$ ra khỏi OPEN
    Đưa $T_{\max}$ vào CLOSE
    if $T_{\max}$ là trạng thái đích **then**
      Lời giải là $T_{\max}$
      exit
    else
      Tạo danh sách tất cả các trạng thái kế tiếp T_K của $T_{\max}$
      for each T_K **do**
        $g(T_K) = g(T_{\max}) + \text{cost}(T_{\max}, T_K)$
        if tồn tại $T_{K'}$ trong OPEN trùng với T_K **then**
          if $g(T_K) < g(T_{K'})$ **then**
            $g(T_{K'}) = g(T_K)$
            Tính lại $f(T_{K'})$
            FATHER$(T_{K'}) = T_{\max}$
          end if
        end if
        if tồn tại $T_{K'}$ trong CLOSE trùng với T_K **then**
          if $g(T_K) < g(T_{K'})$ **then**
            $g(T_{K'}) = g(T_K)$
            Tính lại $f(T_{K'})$
            FATHER$(T_{K'}) = T_{\max}$
          end if
        end if
        Lan truyền sự thay đổi của f và g cho tất cả các trạng thái kế tiếp của T_i (ở tất cả các cấp) đã được lưu trữ trong CLOSE và OPEN.
        if T_K chưa xuất hiện trong cả OPEN và CLOSE **then**
          Thêm T_K vào OPEN
          $f(T_K) = g(T_K) + h(T_K)$
        end if
      end for
    end if
  end if
end loop

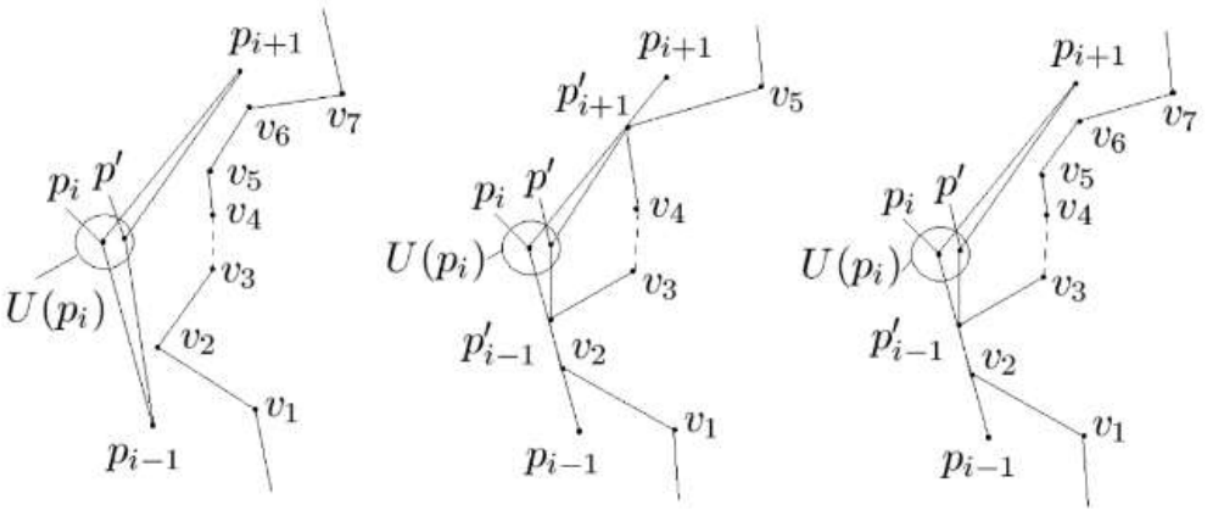
3 Giải bài toán.

3.1 Trả lời câu hỏi 1a.

Ta có thể định nghĩa lại bài toán như sau: Cho một không gian $W = \mathbb{R}^2$ và một tập hợp hữu hạn các đa giác lồi (cũng chính là các chướng ngại vật O), mục tiêu của chúng ta là di chuyển tác nhân (được biểu diễn bởi một điểm) từ vị trí ban đầu (x_s, y_s) tới vị trí kết thúc (x_g, y_g) sao cho đường đi được tạo thành là ngắn nhất có thể và đường đi đó phải đi qua những khoảng trống, định nghĩa bởi không gian $F = W - O$, hay nói cách khác, tác nhân không thể di chuyển qua các đa giác lồi O.
Khi đó tập không gian trạng thái của chúng ta chính là tất cả các vị trí $(x, y) \in \mathbb{R}^2$ nằm trong không gian F. Do số lượng các chướng ngại vật là hữu hạn, cho nên ta chỉ ra được sự tồn tại của không gian F này. Trong không gian $\mathbb{R}^2$, số lượng các điểm (x, y) nằm trong này là vô hạn, do đó số lượng các điểm trong không gian F cũng là vô hạn (tuy nhiên số lượng các điểm này không nhiều hơn số lượng các điểm trong không gian $\mathbb{R}^2$).
Có rất nhiều đường đi từ vị trí ban đầu (x_s, y_s) tới vị trí kết thúc (x_g, y_g), nhưng chỉ có một đường đi tối ưu được hình thành bởi tập hợp các đỉnh thuộc các đa giác chướng ngại vật và 2 cạnh nối 2 đầu của dường gấp khúc này đến lần lượt đỉnh đầu và đỉnh cuối của đường đi tối ưu.

3.2 Trả lời câu hỏi 1b.

Trước khi trả lời câu hỏi này, ta cần định nghĩa đường đi ngắn nhất Eucildean: Đường đi ngắn nhất từ một đỉnh p nằm trên đa giác đến đỉnh q bất kì nào khác được biểu diễn bởi $\rho = \langle p, p_1, p_2, \cdots, p_k, q \rangle$, trong đó mỗi đỉnh $p_1, p_2, \cdots, p_k$ là một đỉnh của một đa giác. (*)
Ta cần chứng minh rằng nhận định trên (cũng chính là ý của câu hỏi 1b) là dúng. Để chứng minh nhận định này, ta sẽ dùng phương pháp phản chứng. Đầu tiên, ta giả sử rằng có ít nhất một đỉnh p_i nào đó trong đường đi ngắn nhất *không* là đỉnh của đa giác. Ta xét các đỉnh từng đôi một không thẳng hàng, nghĩa là 3 đỉnh bất kì luôn tạo thành một tam giác.
Ta chia thành 3 trường hợp như hình 2:



Hình 2: Minh hoạ ba trường hợp cần chứng minh

- **Trường hợp 1: Cả hai cạnh $p_{i-1}p_i$ và p_ip_{i+1} đều không là tiếp tuyến của đa giác hay không tiếp xúc với bất cứ đỉnh nào của đa giác.**

Dựa vào hình vẽ, tồn tại một vùng nhỏ U quanh p_i sao cho tồn tại một điểm $p_i' \in U$ thoả mãn

$$d(p_{i-1}, p_i') + d(p_i', p_{i+1}) < d(p_{i-1}, p_i) + d(p_i, p_{i+1}) \quad (1)$$

Suy ra nếu ta thay p_i bởi p_i', ta có thể thu dược một đường đi ngắn hơn, mâu thuẫn với giả sử ρ là đường đi ngắn nhất.

- **Trường hợp 2: Cả hai cạnh $p_{i-1}p_i$ và p_ip_{i+1} đều nằm trên cạnh của đa giác hay tiếp xúc với bất cứ đỉnh nào của đa giác.**
  Xét 2 đỉnh p_{i-1}' và p_{i+1}' gần nhất sao cho $p_{i-1}'p_i$ và p_ip_{i+1}' là tiếp tuyến của đa giác. Tương tự **Trường hợp 1**, ta có thể tìm được một điểm p_i' sao cho

  $$d(p_{i-1}, p_i') + d(p_i', p_{i+1}) < d(p_{i-1}, p_i) + d(p_i, p_{i+1}) \quad (2)$$

  Điều này mâu thuẫn với giả sử ρ là đường đi ngắn nhất.

- **Trường hợp 3: Trong hai cạnh $p_{i-1}p_i$ và p_ip_{i+1} chỉ có một cạnh là tiếp tuyến của đa giác.**
  Trường hợp này là sự kết hợp của hai trường hợp trên, do đó hoàn toàn tương tự, ta cũng thu được điều mâu thuẫn với giả sử.

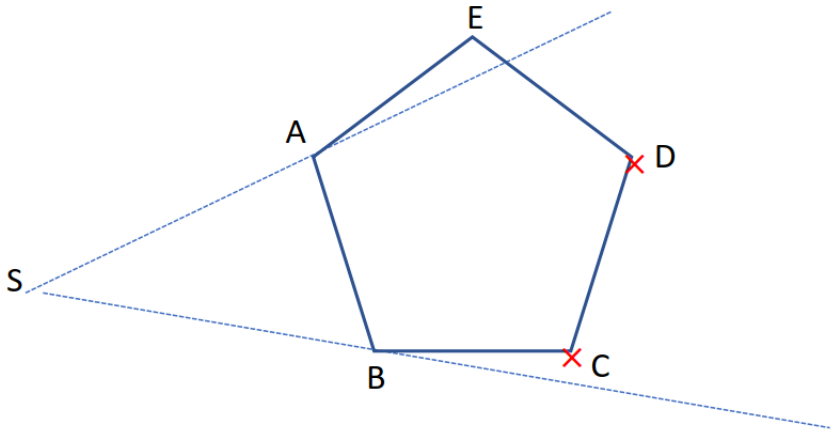
Như vậy, ta đã chứng minh được rằng giả sử ρ là đường đi ngắn nhất là sai. Do đó, ta đã chứng minh được (*) đúng.
Từ nhận xét đó, ta nhận thấy rằng để có thể đi đến điểm cuối (bằng cách đi qua các đỉnh của các đa giác) bằng đường đi ngắn nhất, ta chỉ cần xem xét các đỉnh của các đa giác, từ đó có thể yên tâm bỏ qua các điểm (x, y) khác trong không gian F. Nói cách khác, ta có thể định nghĩa lại không gian trạng thái như sau: $S = \{(x, y) \in \mathbb{R}^2 | (x, y)$ là một đỉnh của $O\}$. Độ lớn của không gian này chính là số lượng các đỉnh của tập đa giác chướng ngại O, và không gian này hữu hạn cũng như là nhỏ hơn rất nhiều so với không gian F.

3.3 Trả lời câu hỏi 1c.

Để có thể giải quyết bài toán, trước hết ta cần xác định xem các đỉnh nào có thể nhìn thấy và đi đến được từ một điểm bất kì nào đó trên đồ thị. Để làm được điều đó, trước tiên ta cần xác định những điểm có thể *nhìn thấy* được.
Trước tiên, ta nhắc lại định nghĩa về phương trình đường thẳng. Cho 2 điểm $A(x_1, y_1)$ và $B(x_2, y_2)$. Khi đó phương trình đường thẳng tạo bởi đoạn thẳng AB có dạng: $(d) : \dfrac{x - x_1}{x_2 - x_1} = \dfrac{y - y_1}{y_2 - y_1}$.
Xét hai điểm $C(x_3, y_3)$ và $D(x_4, y_4)$. Khi đó, ta có thể xác định được vị trí tương đối của hai điểm này đối với đường thẳng (d) như sau:

- $d(C)d(D) \geq 0 \Rightarrow C$ và D nằm cùng phía đối với đường thẳng (d).
- $d(C)d(D) \leq 0 \Rightarrow C$ và D nằm khác phía đối với đường thẳng (d).

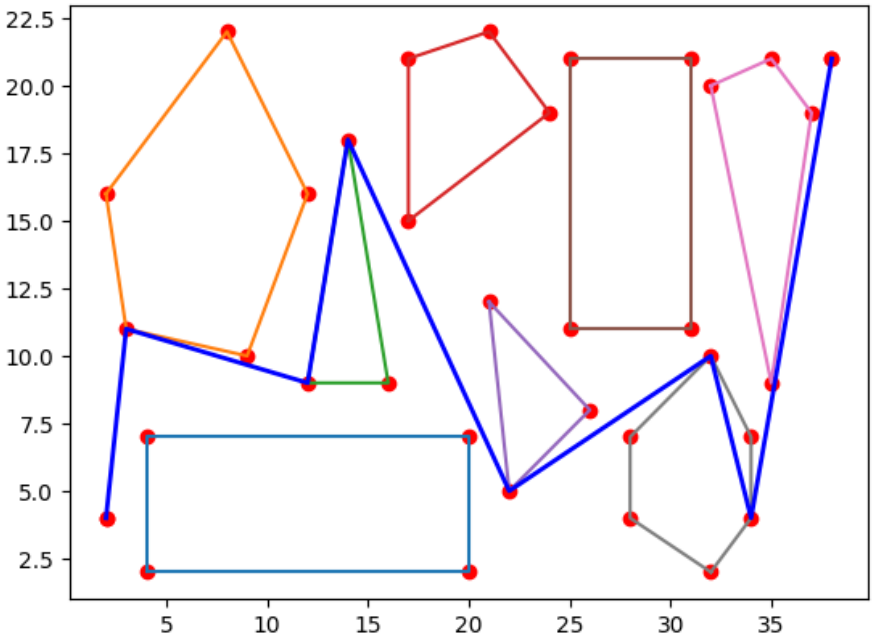
Bây giờ ta xét bài toán tìm những điểm thấy được từ một điểm cho trước. Ta sẽ dùng ý tưởng tìm các điểm *nhìn thấy* được từ một đỉnh đến một đa giác như sau: Từ đỉnh ta đang xét (giả sử là đỉnh S) và các cạnh của đa giác (giả sử các cạnh đó lần lượt là AB, BC, CD, DE, EA), ta xét lần lượt các cạnh, giả sử là cạnh AB. Những điểm nào nằm trong cung ASB sẽ là những điểm không nhìn thấy được, những điểm đó được đánh dấu x đỏ như hình 3, các điểm còn lại là các điểm thấy được.
Như vậy, từ ý tưởng trên, ta có thể tìm được tất cả các điểm *nhìn thấy* được từ một đỉnh cho trước đến các đa giác lân cận như thuật toán 6 ở bên dưới.



Hình 3: Minh hoạ ý tưởng tìm các điểm *nhìn thấy* được từ một đỉnh đến một đa giác

3.4 Hiệu chỉnh thuật toán.

Một vấn đề của thuật toán trên chính là thuật toán trên chỉ giúp ta tìm được những điểm *nhìn thấy* hay có thể đi qua được. Trên thực tế, không phải điểm nào *nhìn thấy* được là có thể *đi qua* được. Một trường hợp như thế được minh hoạ như hình 4. Ta có thể thấy được đỉnh có toạ độ $(31, 11)$ *nhìn thấy* được đỉnh có toạ độ $(34, 3)$, tuy nhiên ta không thể đi từ đỉnh có toạ độ $(31, 11)$ đến đỉnh có toạ độ $(34, 3)$ vì hai đỉnh này là đường chéo của lục giác.



Hình 4: Một đường đi không hợp lệ

Để có thể giải quyết vấn đề trên, sau khi ta xác định được các điểm nhìn thấy được, ta sẽ loại bỏ đi những đỉnh tạo với đỉnh đang xét một đường chéo của đa giác lân cận, từ đó có thuật toán 7. Kết hợp

Thuật toán 6 Thuật toán tìm tất cả các điểm *nhìn thấy* được

Đầu vào: Đỉnh S và các đa giác lân cận $O_1, O_2, \ldots, O_n$.
Đầu ra: Tập hợp V_{see} các đỉnh nhìn thấy được.

$V =$ tập hợp tất cả các đỉnh của các đa giác $O_1, O_2, \ldots, O_n$.
$E =$ tập hợp tất cả các cạnh của các đa giác $O_1, O_2, \ldots, O_n$.
$V_{see} = \emptyset$
for each cạnh AB trong E **do**
  $d_1 =$ đường thẳng đi qua S và A.
  $d_2 =$ đường thẳng đi qua S và B.
  $d_3 =$ đường thẳng đi qua A và B.
  for each đỉnh $Q \in V \setminus \{S, A, B\}$ **do**
    if (Q và B nằm cùng phía với d_1) $\wedge$ (Q và A nằm cùng phía với d_2) $\wedge$ (Q và S nằm khác phía với d_3) **then**
      Q là đỉnh không nhìn thấy được từ P.
    else
      Q là đỉnh nhìn thấy được từ P.
      $V_{see} = V_{see} \cup \{Q\}$
    end if
  end for
end for

với các thuật toán tìm kiếm ở trên, ta có thể tiến hành giải quyết bài toán đã cho. Để minh hoạ, ta sẽ giải quyết bài toán trên với đỉnh đầu tiên là đỉnh có toạ độ $(2, 4)$ và đỉnh kết thúc có toạ độ $(38, 21)$.

Thuật toán 7 Thuật toán tìm tất cả các điểm *đi qua* được

Đầu vào: Đỉnh S và các đa giác lân cận $O_1, O_2, \ldots, O_n$.
Đầu ra: Tập hợp V_{pass} các đỉnh có thể đi qua được.

$V_{see} =$ tập hợp các đỉnh nhìn thấy được từ S.
$V_{pass} = \emptyset$
for each đỉnh $Q \in V_{see}$ **do**
  if Q không nằm trên đường chéo của đa giác lân cận **then**
    Q là đỉnh có thể đi qua được.
    $V_{pass} = V_{pass} \cup \{Q\}$
  end if
end for

4 Cài đặt chương trình.

4.1 Ghi chú.

Chương trình sử dụng ngôn ngữ `Python` phiên bản 3.9.13, với file đầu vào là `input.txt` và mã nguồn được lưu trong file `script.py`.

4.2 Định dạng file đầu vào.

Dữ liệu của file đầu vào gồm (xem hình 5):

- Dòng thứ nhất gồm 5 số nguyên N, S_x, S_y, G_x, G_y cách nhau bởi khỏng trắng, trong đó:
  - N là số đa giác nằm 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 kết thúc.
- N dòng tiếp theo gồm các số nguyên $M, X_1, Y_1, \cdots, X_M, Y_M$ cách nhau bởi khỏng trắng, trong đó:
  - M là số đỉnh của đa giác.
  - (X_i, Y_i) là toạ độ thực của đỉnh thứ i trong đa giác.

```
1  8 2 4 38 21
2  4 4 2 4 7 20 7 20 2
3  5 2 16 3 11 9 10 12 16 8 22
4  3 12 9 16 9 14 18
5  4 17 15 17 21 21 22 24 19
6  3 21 12 22 5 26 8
7  4 25 11 31 11 31 21 25 21
8  4 32 20 35 21 37 19 35 9
9  6 32 2 28 4 28 7 32 10 34 7 34 4
```

Hình 5: Định dạng file đầu vào `input.txt`.

4.3 Các lớp và hàm.

4.3.1 Lớp `Point`.

Lớp `Point` dùng để định nghĩa một điểm trong không gian trạng thái của bài toán, gồm một số hàm qaun trọng sau:

- `__init__(self, x, y, polygon_id=-1)`: Khởi tạo những thuộc tính của một điểm, bao gồm hoành độ, tung độ và đa giác mà đỉnh đó thuộc về, nếu không nằm trong đa giác nào, `polygon_id = -1`. Ngoài ra, hàm còn khởi tạo thêm chi phí ban đầu `g = 0` và thuộc tính `pre` lưu lại thông tin `Point` trước đó trong đường đi tìm kiếm.

- `rel(self, other, line)`: Xét vị trí tương đối giữa điểm hiện tại với điểm `other` so với đường thẳng `line`.
- `can_see(self, other, line)`: Kiểm tra xem từ điểm hiện tại có thể nhìn thấy được điểm `other` qua đường thẳng `line` hay không.
- `line_to(self, other)`: Trả về một đối tượng cạnh `Edge` (xem mục **4.3.2**) nối từ điểm hiện tại đến điểm `other`.
- `heuristic(self, other)`: Trả về kết quả hàm `euclid_distance(self, other)` (xem mục **4.3.4**).

4.3.2 Lớp `Edge`.

Lớp `Edge` dùng để định nghĩa một cạnh trong không gian trạng thái của bài toán, trong đó có hai hàm quan trọng sau:

- `__init__(self, point1, point2)`: Khởi tạo những thuộc tính của một cạnh, bao gồm hai đỉnh `Point` là `point1` và `point2`.
- `get_adjacent(self, point)`: Trả về đỉnh kề với đỉnh hiện tại (hai đỉnh được gọi là kề nhau nếu chúng nối với nhau bởi một cạnh).

4.3.3 Lớp `Graph`.

Lớp `Graph` dùng để định nghĩa một đồ thị trong không gian trạng thái của bài toán. Đây cũng là lớp quan trọng nhất của chương trình, trong lớp này chúng ta quan tâm đến các hàm sau:

- `__init__(self, polygons)`: Khởi tạo đồ thị `self.graph` của bài toán, trong đó `polygons` là một danh sách các đa giác, mỗi đa giác là một danh sách các điểm `Point` được đọc vào từ file `input.txt`. Ngoài ra, hàm này còn lưu tập hợp các cạnh `self.edges` và các đa giác `self.polygons` của đồ thị.
- `add_point(self, point)`: Thêm đỉnh `point` vào đồ thị.
- `add_edge(self, edge)`: Thêm cạnh `edge` vào đồ thị.
- `get_points(self)`: Trả về danh sách của đồ thị.
- `get_edges(self)`: Trả về tập hợp các cạnh của đồ thị.
- `get_polygons(self)`: Trả về tập hợp các đa giác của đồ thị.
- `get_adjacent_points(self, point)`: Trả về danh sách các đỉnh kề với đỉnh `point`.
- `can_see(self, start)`: Trả về danh sách các điểm *nhìn thấy* được từ đỉnh `start`.
- `can_go(self, start)`: Trả về tập hợp các điểm có thể di chuyển đến từ đỉnh `start`.
- `h(self, point)`: Trả về giá trị hàm heuristic nếu cần sử dụng, ngược lại trả về -1.

4.3.4 Các hàm phụ trợ.

Ngoài các lớp ở trên, để giải quyết bài toán, chúng ta còn cần các hàm sau:

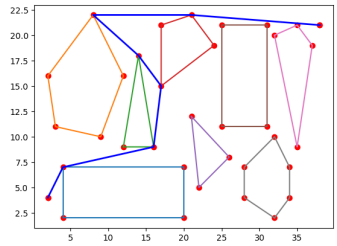
- `euclid_distance(point1, point2)`: Trả về khoảng cách Euclide giữa hai điểm `point1` và `point2`. Khoảng cách Euclide giữa hai điểm $A(x_A, y_A)$ và $B(x_B, y_B)$ được tính bằng công thức:

$$d_{AB} = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

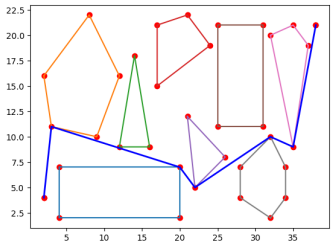
- `bfs(graph, start, goal)`: Trả về đường đi ngắn nhất từ điểm bắt đầu `start` đến điểm kết thúc `goal` trong đồ thị `graph` bằng thuật toán BFS.
- `dfs(graph, start, goal)`: Trả về đường đi ngắn nhất từ điểm bắt đầu `start` đến điểm kết thúc `goal` trong đồ thị `graph` bằng thuật toán DFS.
- `ucs(graph, start, goal)`: Trả về đường đi ngắn nhất từ điểm bắt đầu `start` đến điểm kết thúc `goal` trong đồ thị `graph` bằng thuật toán UCS.
- `search(graph, start, goal, func)`: Trả về đường đi ngắn nhất từ điểm bắt đầu `start` đến điểm kết thúc `goal` trong đồ thị `graph`, trong đó `func` là một trong hai hàm lambda sau:
  - `greedy = lambda graph, i: graph.h(i)`: Ứng với thuật toán GBFS.
  - `a_star = lambda graph, i: i.g + graph.h(i)`: Ứng với thuật toán A*.

5 Kết quả thu được.

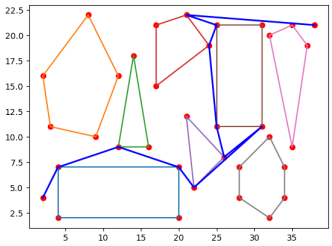
Lời giải cho bài toán ban đầu được thể hiện trong hình 6.



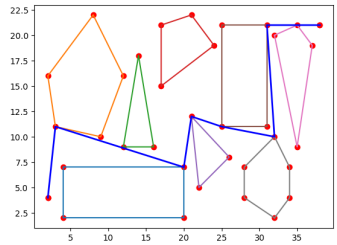
(a) Thuật toán DFS



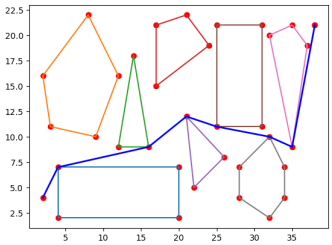
(b) Thuật toán BFS



(c) Thuật toán UCS



(d) Thuật toán GBFS



(e) Thuật toán A*

Hình 6: Lời giải bài toán ban đầu

6 Nhận xét.

- Dễ dàng nhận thấy thuật toán A* cho đường đi tốt nhất trong 5 thuật toán tìm kiếm, lý do là bởi thuật toán A* cân bằng giữa chi phí di chuyển và hàm heuristic để tạo ra một đường đi tối ưu nhất có thể.
- Thuật toán UCS cho đường đi không tốt hơn thuật toán BFS và thuật toán DFS. Ta có thể thấy đường đi của thuật toán UCS dài hơn đường đi của thuật toán BFS và thuật toán DFS.
- Thuật toán GBFS cho đường đi tốt hơn thuật toán UCS. Ta có thể thấy đường đi của thuật toán GBFS ngắn hơn nhiều đường đi của thuật toán BFS và thuật toán DFS.
- Mặc dù vậy, cả 5 thuật toán trên vẫn chưa đảm bảo được đường đi tối ưu nhất có thể. Trên thực tế, người ta thường sử dụng một số thuật toán khác để giải quyết bài toán tìm đường đi vượt qua chướng ngại này như là RRT, APF,...