Trường Đại học Khoa học Tự Nhiên
ĐHQG.TPHCM

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

TPHCM, ngày 24 tháng 11 năm 2024

BÁO CÁO THỰC HÀNH – NHẬP MÔN TRÍ TUỆ NHÂN TẠO – LẦN 4

Họ và tên: Nguyễn Thanh Kiên.
MSSV: 22110092.

Nhắc lại kiến thức

1. Thuật toán BFS:

```
Procedure Breadth_Search

Begin
1. Khởi tạo danh sách L chứa trạng thái ban đầu;
2. While (1)
        2.1 if L rỗng then
            {
                Thông báo tìm kiếm thất bại;
                stop;
            }
        2.2 Loại trạng thái u ở đầu danh sách L;
        2.3 if u là trạng thái kết thúc then
            {
                Thông báo tìm kiếm thành công;
                stop;
            }
        2.4 Lấy các trạng thái v kề với u và thêm vào cuối danh sách L;
            for mỗi trạng thái v kề u do
                father(v) = u;
 end
```

2. Thuật toán DFS:

```
Procedure Depth_Search
Begin
1. Khởi tạo danh sách L chứa trạng thái ban đầu;
2. While (1)
        2.1 if L rỗng then
            {
                Thông báo tìm kiếm thất bại;
                stop;
            }
        2.2 Loại trạng thái u ở đầu danh sách L;
        2.3 if u là trạng thái kết thúc then
            {
                Thông báo tìm kiếm thành công;
                stop;
            }
        2.4 Lấy các trạng thái v kề với u và thêm vào đầu danh sách L;
            for mỗi trạng thái v kề u do
                father(v) = u;
end
```

3. Thuật toán UCS:

```
function Tìm_kiếm_UCS(bài_toán, ngăn_chứa) return lời giải hoặc thất bại.
ngăn_chứa ←Tạo_Hàng_Đợi_Rỗng()
ngăn_chứa ←Thêm(TẠO_NÚT(Trạng_Thái_Đầu[bài_toán]), ngăn_chứa)
loop do
        if Là_Rỗng(ngăn_chứa) then return thất bại.
        nút←Lấy_Chi_phí_Nhỏ_nhất(ngăn chứa)
        if Kiểm_tra_Câu_hỏi_đích[bài_toán] trên Trạng_thái[nút] đúng.
                then return Lời_giải(nút).
lg←Mở(nút, bài_toán)     //lg tập các nút con mới
ngăn_chứa←Thêm_Tất_cả(lg, ngăn_chứa)
```

4. Thuật toán 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
```

5. Thuật toán A*

Procedure Astar-Search

Begin

1. Đặt OPEN chỉ chứa T_0. Đặt $g(T_0) = 0$, $h(T_0) = 0$ và $f(T_0) = 0$. Đặt CLOSE là tập rỗng.
2. Lặp lại các bước cho đến khi gặp điều kiện dừng

2.a. Nếu OPEN rỗng: bài toán vô nghiệm, thoát.

2.b. Ngược lại, chọn T_{max} trong OPEN sao cho $f(T_{max})$ là nhỏ nhất

2.b.1. Lấy T_{max} ra khỏi OPEN và đưa T_{max} vào CLOSE.

2.b.2. Nếu T_{max} là T_G (trạng thái đích) thì thoát và thông báo lời giải là T_{max}

2.b.3. Nếu T_{max} không phải là T_G. Tạo ra danh sách tất cả các trạng thái kế tiếp của T_{max}. Gọi một trạng thái này T_k. Với mỗi T_k, làm các bước sau:

2.b.3.1. Tính $g(T_k) = g(T_{max}) + cost(T_{max}, T_k)$

2.b.3.2. Nếu tồn tại $T_{k'}$ trong OPEN trùng với T_k.

Nếu $g(T_k) < g(T_{k'})$ thì

Đặt $g(T_{k'}) = g(T_k)$

Tính lại $f(T_{k'})$

Đặt $Cha(T_{k'}) = T_{max}$

2.b.3.3. Nếu tồn tại $T_{k'}$ trong CLOSE trùng với T_k

Nếu $g(T_k) < g(T_{k'})$ thì

Đặt $g(T_{k'}) = g(T_k)$

Tính lại $f(T_{k'})$

Đặt $Cha(T_{k'}) = T_{max}$

Lan truyền sự thay đổi giá trị g, f cho tất cả các trạng thái tiếp theo của T_i (ở tất cả các cấp) đã được lưu trữ trong CLOSE và OPEN.

2.b.3.4. Nếu T_k chưa xuất hiện trong cả OPEN lẫn CLOSE thì

Thêm T_k vào OPEN

Tính: $f(T_k) = g(T_k) + h(T_k)$

Nhận xét tổng quan:

1. **Bài toán:**

Bài toán được định nghĩa như sau: Trong không gian $W = R^2$, cho một tập hợp hữu hạn các đa giác lồi O (chướng ngại vật), cần tìm đường đi ngắn nhất từ điểm xuất phát (xs, ys) đến điểm đích (xg, yg) sao cho đường đi chỉ nằm trong không gian khả dụng F = W – O, hay nói cách khác, , tức là không cắt qua các đa giác OO.

Không gian trạng thái là tập hợp các điểm $(x, y) \in R^2$ nằm trong không gian F. Mặc dù số lượng các điểm trong F là vô hạn, đường đi tối ưu luôn được tạo thành bởi các đỉnh của các đa giác chướng ngại vật và các đoạn thẳng nối điểm đầu và điểm cuối qua không gian F.

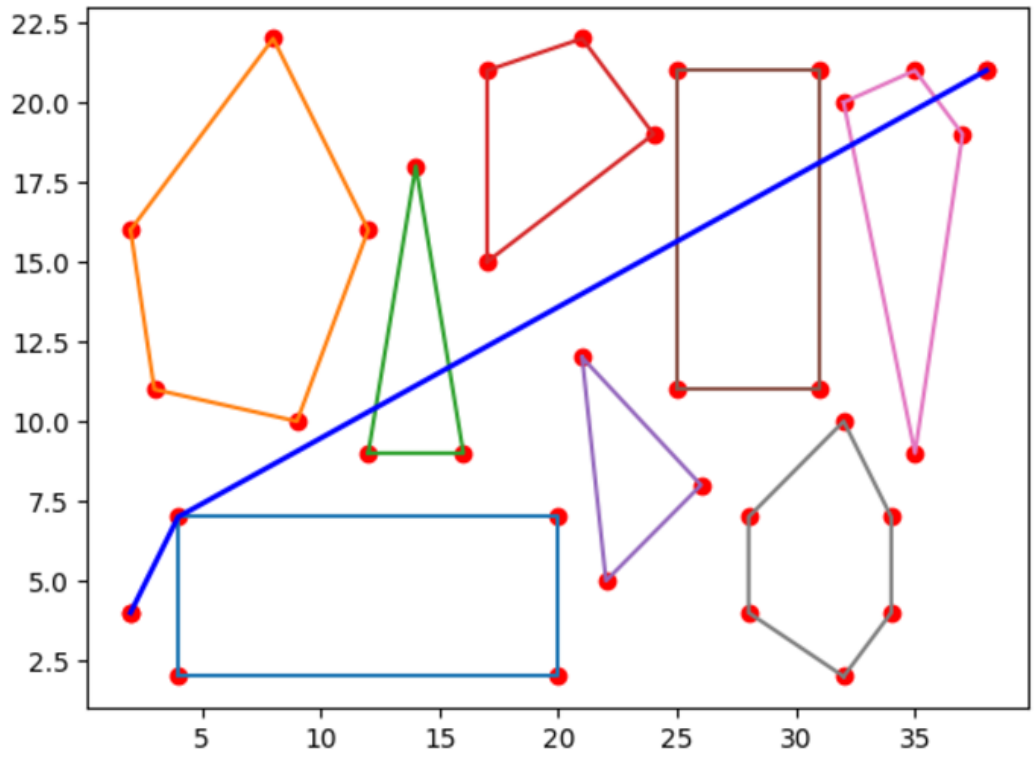
2. **Đường đi ngắn nhất Euclid.**

Ta có nhận định sau về đường đi ngắn nhất Euclid đã được chứng minh: Đường đi ngắn nhất từ một đỉnh p nằm trên đa giác đến đỉnh q bất kì nào khác được biểu biễn bởi p = <p, p1, p2, … ,pk ,q>, trong đó mỗi đỉnh p1, p2, …, pk là một đỉnh của một đa giác.

Từ nhận định đó, ta nhận thấy rằng để có thể đi đến điểm cuối bằng cách đi qua đỉnh của các đa giác bằng đường đi Euclid, ta chỉ xem xét các đỉnh của đa giác mà không cần phải xem xét các điểm (x, y) khác nằm trong không gian F. Vậy, không gian trạng thái là $S = \{(x,y) \in R^2 \mid (x,y)$ là một đỉnh của o$\}$. Độ lớn của không gian này là số lượng đỉnh, và không gian hữu hạn này nhỏ hơn rất nhiều so với không gian F.

3. **Đoạn code đã cho đã thiếu hàm**

Xét hình ảnh xuất ra sau khi chạy đoạn code trong file bài tập



Rõ ràng bằng trực quan, ta thấy được rằng đoạn mã đã có vấn đề khi đường đi đã đi xuyên qua các đỉnh của đa giác. Vì vậy, ta cần phải sửa, thay đổi, thêm những hàm cần thiết để giải quyết bài toán.

Ta cần xác định xem, từ một đỉnh A, ta có thể nhìn thấy các đỉnh B nào và có thể đi đến được các đỉnh C nào với B và C là các đỉnh bất kỳ trên đồ thị. Trước hết, ta phải xác định những điểm nhìn thấy được.

★ **Nhắc lại:**

– Cho 2 điểm $A(x_1, y_1), B(x_2, y_2)$. Phương trình đường thẳng tạo bởi đoạn thẳng AB có dạng

$$d : \frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1}$$

– Xét vị trí tương đối của $C(x_3, y_3)$ và $D(x_4, y_4)$ đối với đường thẳng d:

✓ $d(C) * d(D) > 0 \Rightarrow C, D$ nằm cùng phía.

✓ $d(C) * d(D) < 0 \Rightarrow C, D$ nằm khác phía.

★ **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.

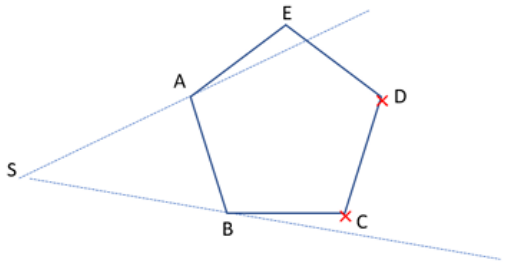
✓ Xét tất cả các đỉnh Q còn lại với d_1, d_2, d_3

✧ Nếu $d_1(Q) * d_1(B) > 0$ và $d_2(Q) * d_2(A) > 0$ 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.

Xét bài toán tìm những điểm thấy được từ một đỉnh cho trước. Ta thực hiện như sau: Từ đỉnh ta đang xét và các cạnh của đa giác, ta xét lần lượt các cạnh, giả sử cạnh là AB. Những điểm nằm trong cung ASB sẽ là những điểm không nhìn thấy được, và những điểm còn lại sẽ nhìn thấy được. Vì vậy, ta có thể xây dựng thuật toán cho các điểm nhìn thấy được từ một đỉnh cho trước đến các đa giác lân cận.

★ **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 $\Rightarrow$ những điểm không bị loại là đỉnh nhìn thấy được.



Tuy vậy, ta lại phát sinh thêm một vấn đề. Thuật toán mà ta xây dựng chỉ cho ta tìm được những điểm nhìn thấy hay có thể đi qua được. Tuy vậy, thực tế rằng không phải tất cả những điểm nhìn thấy được đều có thể đi qua được. Vì vậy, 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.

Xây dựng thuật toán:

Sau 5 thuật toán tìm kiếm, ta xây dựng thêm 2 thuật toán nhìn thấy và đi được.

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, ..., O_n.
E = tập hợp tất cả các cạnh của các đa giác O_1, O_2, ..., O_n.
V_see = ∅
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 ∈ V \ {S, A, B} do
        if (Q và B nằm cùng phía với d_1) ∧ (Q và A nằm cùng phía với d_2) ∧ (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 ∪ {Q}
        end if
    end for
end for
```

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.

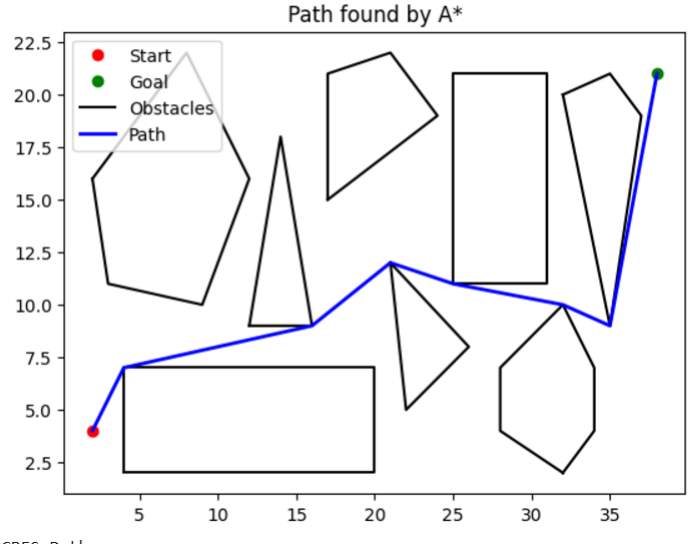
```
V_see = tập hợp các đỉnh nhìn thấy được từ S.
V_pass = ∅
for each đỉnh Q ∈ V_see do
    if Q không nằm trên đường chéo của đa giác lân cận then
        Q là đỉnh có thể đi qua được.
        V_pass = V_pass ∪ {Q}
    end if
end for
```

Kết hợp 7 thuật toán, ta sẽ tìm được đường đi đúng cho bài toán với từng thuật toán tìm kiếm.

Kết quả sau khi chạy:

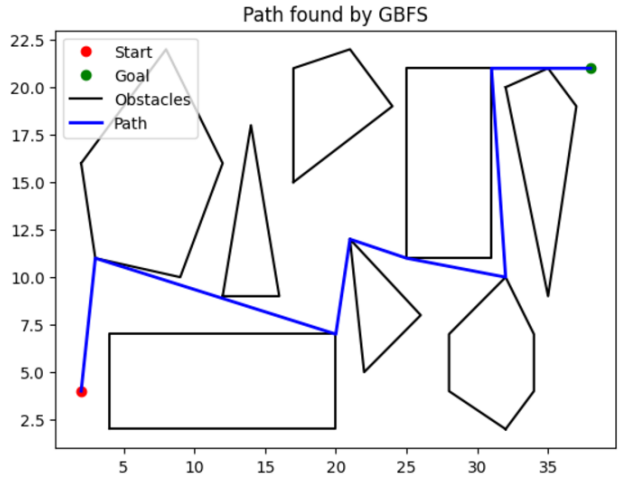
A*

```
A* Path:
[[(2, 4), -1], [(4, 7), 0], [(16, 9), 2], [(21, 12), 4], [(25, 11), 5], [(32, 10), 7], [(35, 9), 6], [(38, 21), -1]]
```



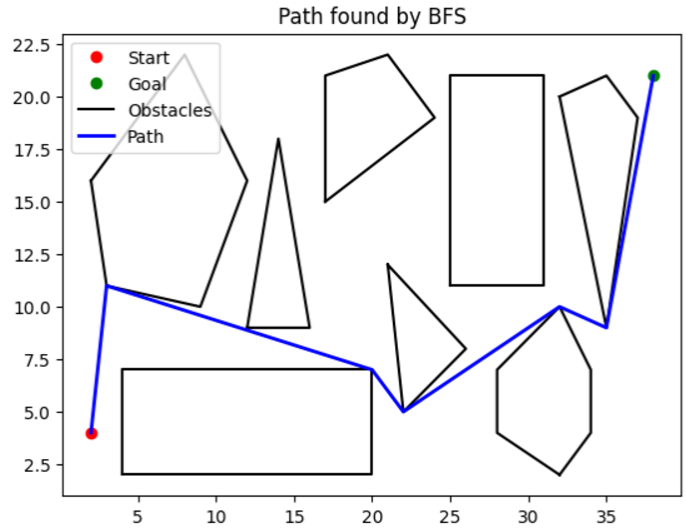
GBFS:

```
GBFS Path:
[[(2, 4), -1], [(3, 11), 1], [(20, 7), 0], [(21, 12), 4], [(25, 11), 5], [(32, 10), 7], [(31, 21), 5], [(35, 21), 6], [(38, 21), -1]]
```



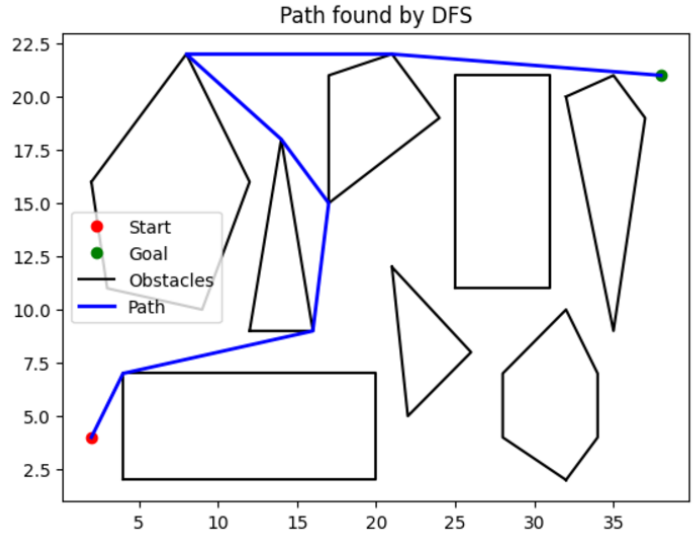
BFS:

```
BFS Path:
[[(2, 4), -1], [(3, 11), 1], [(20, 7), 0], [(22, 5), 4], [(32, 10), 7], [(35, 9), 6], [(38, 21), -1]]
```

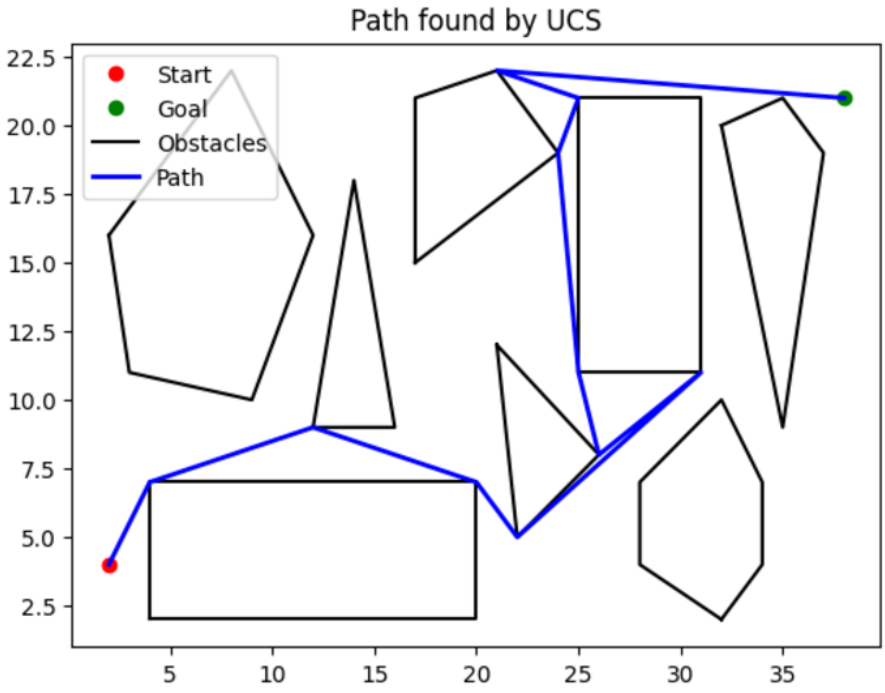


DFS:

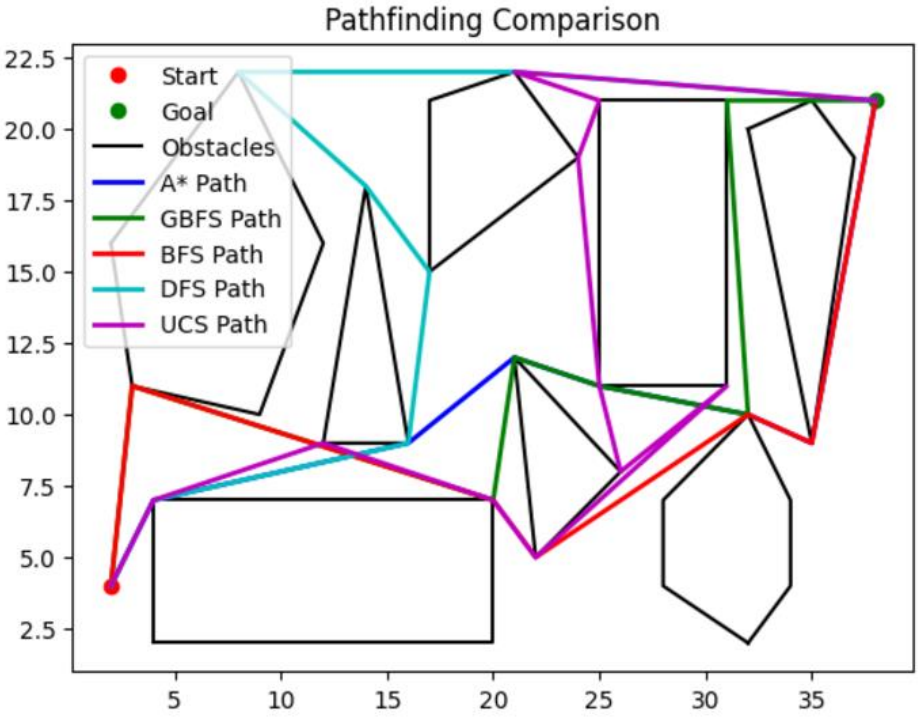
```
DFS Path:
[[(2, 4), -1], [(4, 7), 0], [(16, 9), 2], [(17, 15), 3], [(14, 18), 2], [(8, 22), 1], [(21, 22), 3], [(38, 21), -1]]
```



UCS:



Biểu đồ hiển thị 5 đường đi trong 1 hình:



Nhận xét:

1. Thuật toán A* cho đường đi tối ưu nhất trong 5 đường đi, vì A* cân bằng chi phí di chuyển và hàm h(n) để tạo ra đường đi tối ưu nhất có thể.
2. UCS cho đường đi xấu hơn hay không tốt hơn BFS và DFS, vì đường đi của nó dài hơn và ngoằn ngoèo hơn.
3. GBFS cho đường đi tốt hơn UCS và GBFS có đường đi ngắn hơn nhiều so với đường đi của BFS và DFS.

Người báo cáo

Nguyễn Thanh Kiên