Trường Đại học Phenikaa

Khoa Công nghệ thông tin

Nhập môn trí tuệ nhân tạo

Bài tập lớn: Xây dựng chương trình tìm đường đi trong mê cung

---

Nhóm: 08

Họ và tên	Mã SV
Nguyễn Hoàng Linh Phương	21010562
Nguyễn Minh Duy	22010511

Lớp tín chỉ: Nhập môn trí tuệ nhân tạo - N01

Giảng viên hướng dẫn: TS. Trần Đức Minh

Năm học: 2023-2024

Hà Nội 10/2023

Danh sách hình vẽ

1 Trò mê cung 8
2 Khoảng cách Chebyshev 9
3 Khoảng cách Euclidean 10
4 Khoảng cách Manhattan 10
5 Ví dụ thực tế ước lượng 11
6 Kết quả mê cung Python 26x26(A* bên trái - Nhánh_cận bên phải) . . 19
7 Kết quả mê cung Python 20x20(A* bên trái - Nhánh_cận bên phải) . . 19
8 Kết quả mê cung Python 35x35(A* bên trái - Nhánh_cận bên phải) . . 20
9 Kết quả mê cung C++ 38x38(A* bên trái - Nhánh_cận bên phải) . . . 21
10 Kết quả mê cung_2 C++ 38x38 (A* bên trái - Nhánh_cận bên phải) . 21

List of Algorithms

1 Thuật toán A* 12
2 Thuật toán Nhánh- cận 14

Mục lục

1 BẢNG PHÂN CHIA CÔNG VIỆC **5**

2 MỞ ĐẦU **6**

3 LỊCH SỬ HÌNH THÀNH VÀ PHÁT TRIỂN **7**
3.1 Thuật toán A* 7
3.2 Thuật toán Nhánh - cận 7

4 MÔ TẢ BÀI TOÁN **8**

5 LÝ THUYẾT **9**
5.1 Hàm đánh giá 9
5.2 Trình bày thuật toán 12
5.2.1 Thuật toán A* 12
5.2.2 Thuật toán Nhánh-cận (Branch and Boun) 13

6 TRIỂN KHAI THUẬT TOÁN **16**
6.1 Ngôn ngữ Python 16
6.1.1 Thuật toán A* 16
6.1.2 Thuật toán Nhánh-cận 16
6.2 Ngôn ngữ C++ 16
6.2.1 Thuật toán A* 16
6.2.2 Thuật toán Nhánh-cận 17

7 KẾT QUẢ VÀ THẢO LUẬN **19**
7.1 Ngôn ngữ Python 19
7.2 C++ 21

1 BẢNG PHÂN CHIA CÔNG VIỆC

Nhiệm vụ	Họ và tên
Tìm hiểu thuật toán A* và nhánh - cân	Nguyễn Hoàng Linh Phương
Viết code Python	
Viết báo cáo	
Tìm hiểu thuật toán A* và nhánh - cân	Nguyễn Minh Duy
Viết code C++	
Viết báo cáo	

2 MỞ ĐẦU

Trong lĩnh vực trí tuệ nhân tạo, có một bài toán kinh điển là tìm đường đi ngắn nhất trong mê cung. Bài toán này là một bài toán thú vị và được ứng dụng trong nhiều lĩnh vực. Không chỉ là một trò chơi, tìm đường ngắn nhất trong mê cung cũng là một bài toán tối ưu hóa cho robot hoặc trong hệ thống giao thông,....

Trong bài tập lớn này, chúng tôi sẽ trình bày chương trình giải quyết bài toán này bằng một số phương pháp với điểm bắt đầu và kết thúc đã cho. Có rất nhiều thuật toán có thể áp dụng để giải bài toán này như Dijkstra, A*, DFS, BFS,.... Tuy nhiên trong báo cáo này, chúng tôi thực hiện so sánh hai thuật toán là A* và nhánh - cận.

Trong báo cáo này, chúng tôi thực hiện triển khai hai thuật toán A* và nhánh - cận trên cả hai ngôn ngữ là Python và C++, rồi đem so sánh kết quả, thời gian và bộ nhớ hai thuật toán sử dụng trên cả hai ngôn ngữ, từ đó để đưa ra được đánh giá khách quan với cả hai thuật toán trong bài tìm đường ngắn nhất của mê cung.

3 LỊCH SỬ HÌNH THÀNH VÀ PHÁT TRIỂN

3.1 Thuật toán A*

Năm 1964, Nils Nilsson phát minh ra một phương pháp tiếp cận dựa trên khám phá để tăng tốc độ của thuật toán Dijkstra. Thuật toán này được gọi là A1. [4]

Năm 1967 Bertram Raphael đã cải thiện đáng kể thuật toán này, nhưng không thể hiển thị tối ưu. Ông gọi thuật toán này là A2.

Năm 1968 Peter Hart, Nils Nilsson, và Bertram Raphael đã nói đến thuật toán A trong bài báo của họ, khi sử dụng thuật toán này với một đánh giá heuristic thích hợp sẽ thu được hoạt động tối ưu, do đó mà có tên A*.

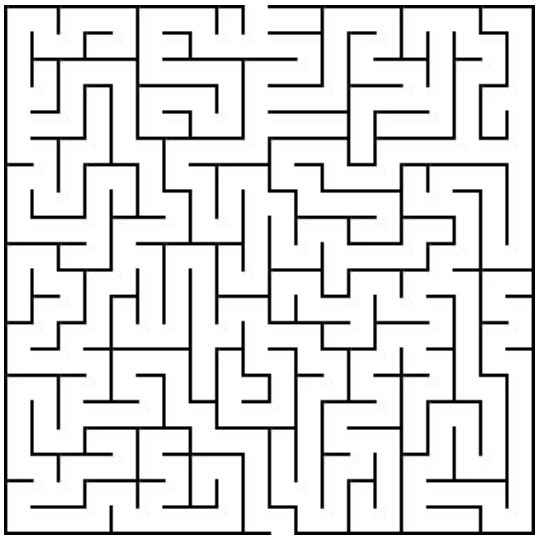
3.2 Thuật toán Nhánh - cận

AH Land và AG Doig đề xuất lần đầu tiên vào năm 1960, cho lập trình rời rạc. [7] Một số bài toán nổi tiếng được sử dụng nhánh và cận:[2]

- Bài toán Flowshop hoán vị (PFSP) (IIgnall & Schrage, 1965)
- metaheuristic (Nagar, Heragu, & Haddock, 1996)
- Bài toán người bán hàng du lịch (Wolsey, 1998)
- Phương pháp hàm đánh giá (Haouari & Ladhari, 2003)
- Bài toán lựa chọn nhà cung cấp với chiết khấu (Goossens, Maas, Spieksma, & van de Klundert, 2007)
- Vào 2018, Li và cộng sự đư ể khái niệm về một hàm được sử dụng để đánh giá và giới hạn các giải pháp tiềm năng khi giải quyết bài toán tối ưu, thường là bài toán tối ưu hóa tổ hợp. Thuật toán này tính đến cả cấu trúc đồ thị và trọng số của các đỉnh cần lược bỏ.

4 MÔ TẢ BÀI TOÁN

Bài toán tìm đường đi ngắn nhất trong mê cung được mô tả là một mê cung với các đỉnh hoặc ô có các thuộc tính như là, khoảng cách, trạng thái (đã duyệt, chưa duyệt) và cách thức di chuyển. Mục tiêu của bài toán là từ điểm xúât phát tìm đường đi ngắn nhất tới điểm đích cho trước.



Hình 1: Trò mê cung

Bài toán này có một số khó khăn thường gặp phải như kích thước mê cung, tài nguyên, thời gian. Khi kích thước mê cung quá lớn và phức tạp thì thuật toán tìm đường đi càng cần hiệu quả và tối ưu hơn, tuy nhiên nó càng phức tạp thì sẽ cần nhiều tài nguyên hoặc thời gian để tìm đường đi. Nếu ưu tiên về mặt thời gian thì sẽ tốn nhiều tài nguyên và ngược lại, ưu tiên tiết kiệm tài nguyên thì lại cần thời gian lâu hơn. Ngoài ra trong thực tế, bài toán tìm đường đi ngắn nhát trong mê cung còn gặp phải khó khăn rất lớn trong việc thay đổi trạng thái của các ô, vì trong thực tế không phải lúc nào trạng thái cũng được đảm bảo giữ nguyên. Ví dụ như bài toán được áp dụng trong giao thông thì trạng thái của đèn báo hoặc mật độ xe cộ sẽ luôn thay đổi, bởi vậy yêu cầu thuật toán phải nhanh nhạy và cập nhật liên tục, đây là một thách thức lớn trong bài toán này.

Bài tập này yêu cầu chúng tôi tìm đường trong mê cung hình vuông với kích thước cạnh nhỏ nhất là bằng 20.

5 LÝ THUYẾT

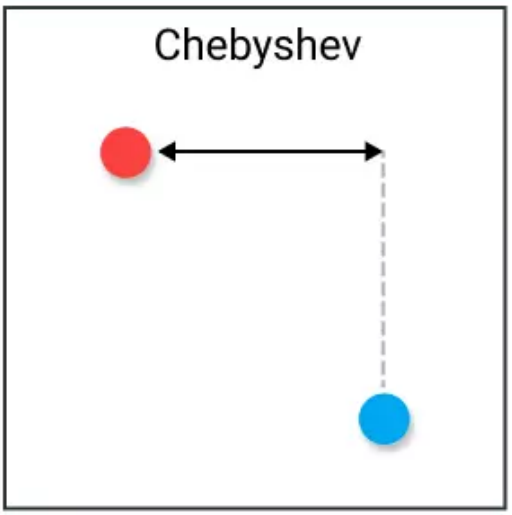
5.1 Hàm đánh giá

Hàm đánh giá [6] dùng để đo lường và đánh giá mức độ tiềm năng tốt nhất cho node trong quá trình tìm kiếm tốt nhất. Mục tiêu hàm đánh giá là ước tính tổng đường đi từ xuất phát đến đích. Hàm đánh giá đóng vai trò quan trọng trong việc tìm đường đi ngắn nhất, nếu hàm đánh giá đủ tốt thì thuật toán có thể tìm được đường đi ngắn nhất, nếu hàm đánh giá không tốt thì đường đi không được tối ưu.

Tùy vào bài toán sẽ sử dụng hàm đánh giá khác nhau như Manhattan, Chebyshev, Euclidean, Jaccard Index, Hamming Distance,.... Với bào toán tìm đường ngắn nhất trong mê cung này, chúng tôi sử dụng các hàm đánh giá cho điểm trên mặt phẳng, tiêu biểu là 3 hàm đánh giá là Manhattan, Chebyshev và Euclidean.

Lấy $(x_0, y_0), (x_1, y_1)$ là 2 điểm trên mặt phẳng cần sử dụng hàm đánh giá.

Chebyshev Khoảng cách Chebyshev tính độ lệch lớn nhất của 2 vector theo trục tọa độ.



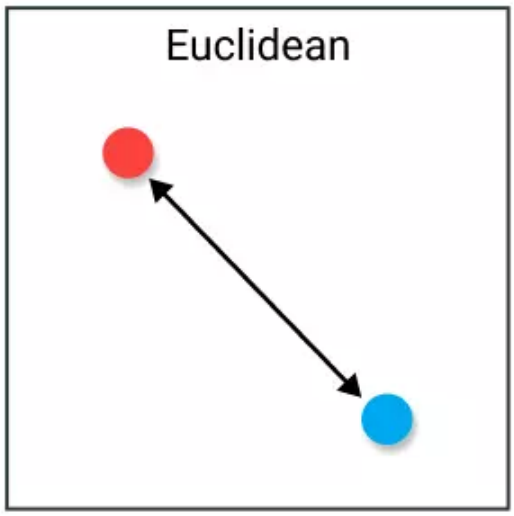
[1]

Hình 2: Khoảng cách Chebyshev

$$D(x, y) = max(|x_0 - x_1|, |y_0 - y_1|) \tag{1}$$

Đặc điểm của hàm này cho phép tính được số bước đi tối thiểu từ ô này sang ô khác nên có thể có lợi trong các trò chơi cho phép di chuyển 8 hướng (đi ngang, dọc và đi chéo). Trong thực tế phương pháp này thường được dùng trong logicstic. Nó không hoàn toàn phù hợp với bài toán mê cung di chuyển ngang dọc này của chúng tôi.

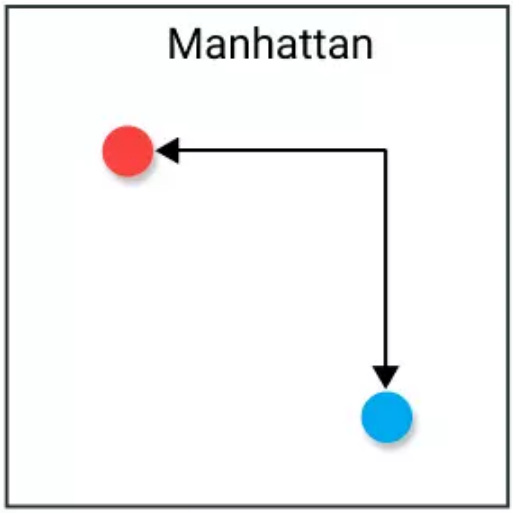
Euclidean Hàm đánh giá Euclidean tính khoảng cách ngắn nhất giữa 2 điểm, tương tự như đường chim bay trên thực tế.



Hình 3: Khoảng cách Euclidean

$$D(x, y) = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2} \tag{2}$$

Manhattan Hàm đánh giá khoảng cách Manhattan, tính khoảng cách theo hướng ngang và dọc trên mê cung. Hàm đánh giá này hoạt động tốt với các đường di chuyển ngang, dọc hơn.



Hình 4: Khoảng cách Manhattan

$$D(x, y) = |x_0 - x_1| + |y_0 - y_1| \tag{3}$$

Vì bài toán tìm đường đi ngắn nhất trong mê cung của chúng tôi di chuyển theo hướng ngang, dọc, nên chúng tôi sử dụng hàm đánh giá Manhattan để tối ưu nhất cho bài toán.

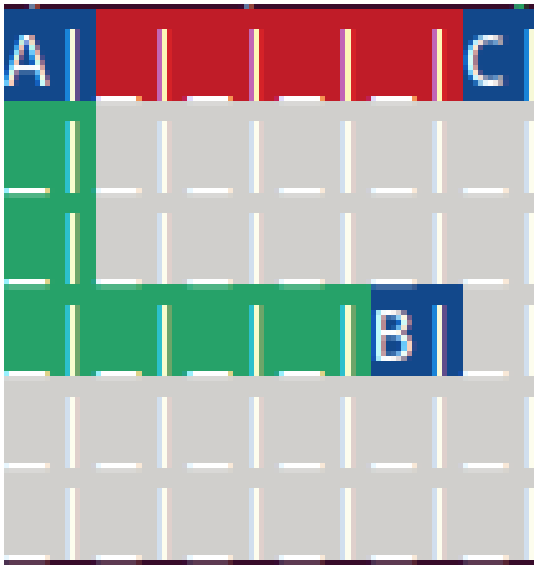
Ước lượng ưu thế: Manhattan Một ước lượng chấp nhận được h(u) có giá trị không bao giờ lớn hơn thực tế. Cả 3 hàm đánh giá Chebyshev, Manahttan và Euclidean đều là ước lượng chấp nhận được cho bài toán mê cung di chuyển theo hướng ngang và dọc này.

Giả sử với h_a(u) và h_b(u) là 2 ước lượng chấp nhận được, nếu h_b(u) $\geqslant$ h_a(u) với mọi u thì h_b(u) là ước lượng có ưu thế tốt hơn. Với 2 tọa độ trùng nhau về giá trị 1 trục (x_0 = x_1 và y_0 = y_1) thì giá trị của 3 hàm cho ra kết quả bằng nhau. Với 2 điểm bất kỳ không trùng nhau về tọa đồ trục nào (x_0 $\neq$ x_1 và y_0 $\neq$ y_1) thì ta tạo được một tam giác vuông với góc vuông ở đỉnh thứ ba. Từ đó áp dụng vào 3 hàm đánh giá Chebyshev, Manahttan và Euclidean, từ đó có thể quy đổi như sau:

- giá trị hàm Chebyshev: giá trị cạnh góc vuông lớn nhất
- giá trị Euclidean: giá trị cạnh huyền
- giá trị Manahttan: tổng giá trị 2 cạnh góc vuông

Theo bất đẳng thức ba cạnh trong tam giác trong mặt phẳng, tổng hai cạnh luôn lớn hơn cạnh còn lại, như vậy có thể thấy tổng giá trị 2 cạnh góc vuông luôn lớn hơn hoặc bằng các cạnh còn lại. Vì vậy cho nên với mọi giá trị của tọa độ (x,y) thì giá trị hàm Manahttan luôn lớn hơn hoặc bằng giá trị của 2 hàm còn lại. Bởi vậy hàm đánh giá Manhattan là ước lượng có ưu thế tốt nhất.

Một ví dụ thực tế là, với tọa độ A(x_0, y_0) = (0,0), B(x_1, y_1) = (3,4), C(x_2, y_2) = (0,5) thì ta có thể tính như sau:



Hình 5: Ví dụ thực tế ước lượng

Với quãng đường từ B đến A, ước lượng:

- Chebyshev: $max(|3-0|, |4-0|) = 4$
- Euclidean: $\sqrt{(3-0)^2 + (4-0)^2} = 5$
- Manahttan: $|3-0| + |4-0| = 7$

Với quãng đường từ C đến A, ước lượng:

- Chebyshev: $max(|0-0|, |5-0|) = 5$
- Euclidean: $\sqrt{(0-0)^2 + (5-0)^2} = 5$
- Manahttan: $|0-0| + |5-0| = 5$

Với ví dụ trên có thể thấy khi 2 điểm trùng nhau về 1 tọa độ điểm thì kết quả của 3 hàm đánh giá là như nhau, nhưng với 2 điểm không trùng thì giá trị hàm Manhattan luôn lớn hơn 2 hàm còn lại. Ngoài ra từ ví dụ ta có thể thấy, nếu sử dụng hàm đánh giá Euclidean thì cho thấy ước lượng quãng đường từ 2 điểm A và B đến C là như nhau. Tuy nhiên Manahttan lại cho thấy quãng đường từ C là tốt hơn. Và thực tế vì di chuyển theo 4 toán tử: trên, dưới, trái , phải, nên rõ ràng hàm Manahttan đánh giá chính xác hơn.

5.2 Trình bày thuật toán

5.2.1 Thuật toán A*

Thuật toán A* [3] là một thuật toán tìm kiếm trong đồ thị. Thuật toán này tìm đường đi từ nút khởi đầu đến nút đích cho trước với hàm đánh giá thích hợp. Vì thuật toán A* sử dụng hàm đánh giá để chọn đường thích hợp và lựa chọn đường tốt nhất theo chiều rộng nên A* luôn tìm được đường ngắn nhất nếu như có tồn tại. A* hoạt động theo phương thức kết hợp tối ưu đường đã đi qua từ điểm xuất phát và ước tính đường đi còn lại (từ điểm hiện tại đến đích). Từ đó có thể tìm được tổng đường đi ngắn nhất từ điểm xuất phát đến đích.

Dữ liệu đầu vào gồm có: ma trận gồm các node hoặc ô kèm trọng số (với bài toán tìm đường mê cung, trọng số tương tự ô có thể đi qua hay bị chặn); Gồm có điểm xuất phát và điểm đích; Gồm có hàm đánh giá (với bài toán này, hàm đánh giá chúng tôi sử dụng là hàm tính khoảng cách giữa các node).

Algorithm 1 Thuật toán A*

Input: Mê cung, điểm bắt đầu, điểm kết thúc
Output: Lời giải tốt nhất tìm được
begin
  $open \leftarrow$ Khởi tạo danh sách chờ
  $father \leftarrow$ Khởi tạo danh sách cha
  $open \leftarrow S$
  while $open \neq \varnothing$ **do**
    $X \leftarrow open[0]$
    Loại $open[0]$ khỏi $open$
    if $X \in GD$ **then**
      return *Đường đi ngắn nhất trong mê cung*
    end
    else
      $g(Y_i) \leftarrow g(X) + k(X, Y_i)$
      $f(Y_i) \leftarrow g(Y_i) + h(Y_i)$
      $Y_i \leftarrow open$
      $father(Y_i) \leftarrow X$
    end
    Xếp các node trong $open$ theo thứ tự tốt nhất đến xấu nhất theo giá trị f
  end
  return *Không thấy đường đi ngắn nhất trong mê cung*
end
0

Trong đó:

- open: danh sách chờ
- S: điểm bắt đầu
- GD: điểm kết thúc
- X: điểm đang xét
- Y: được sinh ra từ X
- $father(Y_i)$: lưu lại đỉnh Y_i
- $g(u)$: đường từ gốc đến u
- $h(u)$: đường ước lượng từ u đến đích
- $k(u)$: đường thực tế từ điểm u-1 đến u
- $f(u)$: tổng đường ước lượng đường từ đầu đến đích qua u

Với thuật toán A* có thể thấy nếu không gian trạng thái là hữu hạn và có giải pháp khử lặp thì thuật toán này sẽ luôn tìm được đường đi ngắn nhất (nếu có) tuy nhiên lại không đảm bảo được sự tối ưu. Về độ phức tạp thì được đánh giá bậc của hàm mũ: số lượng các node được xét là hàm mũ độ dài đường đi của lời giải. Độ phức tạp về không gian bộ nhớ, nó lưu trữ toàn bộ các node trong bộ nhớ. Khi hàm h là hàm chấp nhận đuọc thì thuật toán A* luôn luôn tìm được lời giải tối ưu.

Nếu hàm đánh giá h có tính chất không bao giờ đánh giá cao hơn chi phí nhỏ nhất thực sự của việc đi tới đích, thì bản thân A* tối ưu nếu sử dụng một tập đóng ("Tập hợp đóng"lưu giữ tất cả các nút cuối cùng của p (các nút mà các đường đi mới đã được mở rộng tại đó) để tránh việc lặp lại các chu trình (việc này cho ra thuật toán tìm kiếm theo đồ thị)). Phát biểu một cách hình thức, với mọi nút x, y trong đó y là nút tiếp theo của x:

$$h(x) + g(x) \leq g(y) + h(y) \tag{4}$$

5.2.2 Thuật toán Nhánh-cận (Branch and Boun)

Thuật toán nhánh-cận[5] cũng là một trong các thuật toán tìm đường đi ngắn nhất. Tuy nhiên khác với A* sử dụng "best first search", nhánh-cận sử dụng kết hợp kỹ thuật tìm kiếm leo đồi và hàm đánh giá tương tự A*:

$$f(x) = g(x) + h(x) \tag{5}$$

Nó tập trung vào việc cố gắng tìm ra một giải pháp tốt nhất trong "lân cận"của giải pháp hiện tại. Nhánh- cận có phương pháp tương tự A*, khác biệt là A* tìm kiếm theo chiều rộng và nhánh-cận theo chiều sâu. Cũng sử dụng hàng đợi, danh sách được xếp theo $f(x)$ tốt nhất đến kém nhất.

Algorithm 2 Thuật toán Nhánh- cận

Input: Mê cung, điểm bắt đầu, điểm kết thúc
Output: Lời giải tốt nhất tìm được
begin
    $open \leftarrow$ Khởi tạo danh sách chờ
    $father \leftarrow$ Khởi tạo danh sách cha
    $cost \leftarrow +\infty$ Khởi tạo biến cost lưu đường đi ngắn nhất
    $open \leftarrow S$
    while $open \neq \varnothing$ **do**
        $X \leftarrow open[0]$
        Loại $open[0]$ khỏi $open$
        if $X \in GD$ **then**
            if $g(X) \leq cost$ **then**
                $cost \leftarrow g(X)$
                continue
            end
        end
        else
            if $f(X) > cost$ **then**
                continue
            end
            else
                $g(Y_i) \leftarrow g(X) + k(X, Y_i)$
                $f(Y_i) \leftarrow g(Y) + h(Y_i)$
                $temp \leftarrow Y_i$
                $father(Y_i) \leftarrow X$
            end
            Xếp các node trong *temp* theo thứ tự tốt nhất đến xấu nhất theo giá trị f
        end
        Đưa toàn bộ danh sách *temp* vào đầu danh sách *open*
    end
    return *Không thấy đường đi ngắn nhất trong mê cung*
end

Trong đó:

- *cost*: đường đi ngắn nhất
- *temp*: danh sách các node được sinh ra từ X
- *open*: danh sách chờ
- S: điểm bắt đầu
- GD: điểm kết thúc
- X: điểm đang xét
- Y: được sinh ra từ X
- $father(Y_i)$: lưu lại đỉnh Y_i
- $g(u)$: đường từ gốc đến u

- $h(u)$: đường ước lượng từ u đến đích
- $k(u)$: đường thực tế từ điểm u-1 đến u
- $f(u)$: tổng đường ước lượng đường từ đầu đến đích qua u

6 TRIỂN KHAI THUẬT TOÁN

6.1 Ngôn ngữ Python

link github: https://github.com/linhphuong06/nhap_mon

Sử dụng các thư viện trong python như "heapq"để sắp xếp hàng đợi, "time"đo thời gian chạy chương trình, "psutil"tính toán bộ nhớ sử dụng cho chường trình, "random"để tạo mê cung và tạo các điểm bắt đầu kết thúc ngẫu nhiên và "gc"sử dụng xóa, thu gọn bộ nhớ đệm.

Vì bải toán yêu cầu mê cung hình vuông với cạnh nhỏ nhất bằng 20, mà bộ nhớ trên máy tính cá nhân của chúng tôi có hạn nên chúng tôi sử dụng thư viện "random"để hỗ trợ chọn ngẫu nhiên cạnh mê cung dải từ 20 đến 50. Với mê cung, chúng tôi sử dụng $np.random.choice([0,1], size = (num, num), p = [0.7, 0.3])$ để tạo ô có thể đi qua và ô bị chặn, với 0 là ô có thể đi qua và 1 là ô bị chặn, num là số một cạnh của mê cung, mảng p chứa xác suất xuất hiện của các giá trị 0, 1.

Trong chương trình của chúng tôi, hàm đánh giá được sử dụng là hàm khoảng cách Manhattan 3. Vì trong bài toán mê cung này, đường đi của bài toán chỉ đi ngang và dọc nên chúng tôi xét 4 điểm hàng xóm có tọa độ tính theo node được xét $(0,-1), (-1,0), (1,0), (0,1)$.

6.1.1 Thuật toán A*

Khởi tạo các danh sách để lưu trữ như mô tả ở thuật toán A* 1. Bên cạnh đó, mỗi node được lưu trữ vào danh sách chờ *open* sẽ được biểu diễn với 4 đại lượng: giá trị các hàm $f()$, $g()$, $h()$ và cuối cùng là tọa độ của node đó. Node S bắt đầu được gán giá trị các hàm $f(S) = 0$, $g(S) = 0$ và $h(S) = manhattan(S, E)$ (giá trị hàm $h(S)$ chính bằng khoảng cách Manhattan tính từ điểm xuất phát (S) đến điểm kết thúc (E)). Đầu tiên đẩy node S vào hàng chờ *open*, tuy nhiên vì mọi dữ liệu đưa vào đều là random nên để tránh trường hợp node S và E trùng nhau, khiến bài toán rơi vào vòng lặp vô hạn, chúng tôi tiến hành kiểm tra 2 điểm trước, nếu trùng nhau thì số bước ngắn nhất bằng 0.

6.1.2 Thuật toán Nhánh-cận

Với thuật toán Nhánh-cận, đầu tiên chúng tôi gán giá trị cho biến $cost \leftarrow float('inf')$, giá trị dương vô cùng. Tiếp theo tương tự như cài đặt của A*, tuy nhiên với nhánh-cận, thông tin 1 node chỉ chứa giá trị hàm $f()$ và tọa độ node. Trong thuật toán này, chi phí của mỗi đường đi bằng đúng giá trị hàm $g()$, nó được lưu trong từ điển. Mỗi khóa trong từ điển có giá trị bằng đường đi từ đầu đến node đó.

6.2 Ngôn ngữ C++

link github: https://github.com/haruhamy/BTL

6.2.1 Thuật toán A*

- Mảng hai chiều h làm ma trận ban đầu.
- Biến n để lưu số hàng số cột, tự động sinh ra các tường bằng hàm sinh số ngẫu nhiên, lưu tọa độ đỉnh bắt đầu A vào hai biến s và t, tọa độ đỉnh kết thúc B vào hai biến u và v.

- Mảng hai chiều visited làm ma trận để check đỉnh đang xét đã được duyệt chưa, khởi tạo mảng visited tất cả là false (chưa được thăm).
- Khởi tạo biến z = 0 để check
- Khởi tạo vector open làm danh sách lưu các đỉnh được sinh ra.
- Sử dụng các pair để lưu tọa độ đỉnh, hàm g(x) và f(x).
- Sử dụng hàm đánh giá là hàm tính khoảng cách manhattan.

  → Đẩy điểm A vào open, đánh dấu là true, tức là đã được thăm

  → Lặp while cho đến khi nào open rỗng

  → Lấy ra đỉnh đầu vector

  → Nếu tìm thấy, tức điểm lấy ra trùng với u và v, kết thúc vòng lặp, đồng thời cho z = 1

  → Với mỗi đỉnh sinh ra bởi nó, sắp xếp theo hàm f(x) từ tốt nhất đến xấu nhất, rồi push_back trở lại vector open.

  → Sử dụng một map father để lưu đỉnh cha của các đỉnh đã được duyệt

  → Kết thúc vòng while, nếu z vẫn bằng 0, tức là tìm kiếm thất bại, ngược lại thì in ra danh sách value trong map với key là đỉnh kết thúc.

6.2.2 Thuật toán Nhánh-cận

- Một hàm manhattan_distance tính khoảng cách manhattan giữa hai đỉnh bất kì thuộc ma trận
- Mảng 2 chiều visited (bool) đánh dấu các đỉnh đã được duyệt qua đường đi, đánh dấu tất cả các đỉnh ban đầu là false (chưa được duyệt)
- Mảng 2 chiều bando (char) in ra đường đi theo dạng trực quan hóa
- Biến cost (int) = 1e6 (vô cùng lớn), biến check z = 9
- Vector open các pair lưu danh sách đỉnh, hàm g(x) và f(x) của đỉnh
- Một map father có key là các pair và value là các vector, lưu đỉnh cha của đỉnh đang xét
- Ban đầu đẩy đỉnh A, hàm g(x) = 0 và f(x) = 0 + khoảng cách manhattan từ đỉnh A đến đỉnh B vào vector open
- Lặp đến khi nào open rỗng
- Lấy phần tử đầu vector open, đánh dấu visited của đỉnh đó là true
- Xóa phần tử vừa lấy ra đi
- Nếu phần tử đó bằng đỉnh kết thúc, đánh dấu z = 1, đồng thời nếu g(x) <= cost thì gán cost = g(x), tiếp tục vòng lặp và bỏ qua phần bên dưới

- Nếu phần tử không phải đỉnh kết thúc, tạo một vector tên là tmp,

  ← Nếu hàm f(x) của phần tử đó lớn hơn cost, bỏ qua tất cả phần bên dưới mà chạy tiếp vòng lặp while, n

  ← Ngược lại, duyệt 4 đỉnh kề với đỉnh đó, nếu 4 đỉnh đó không phải tường, chưa được duyệt và vẫn nằm trong ma trận thì hàm g(y) của đỉnh kề sẽ bằng hàm g(x) của đỉnh đang xét cộng thêm 1, đồng thời hàm f(y) = f(x) + khoảng cách manhattan giữa y và đỉnh kết thúc, sau đó sắp xếp tất cả theo thứ tự tăng dần hàm f(y) và đẩy vào vector tmp, đánh dấu key có giá trị là đỉnh kề của đỉnh đang xét có value là đỉnh đang xét bằng map father ban đầu khởi tạo.

- Cuối cùng là in ra:

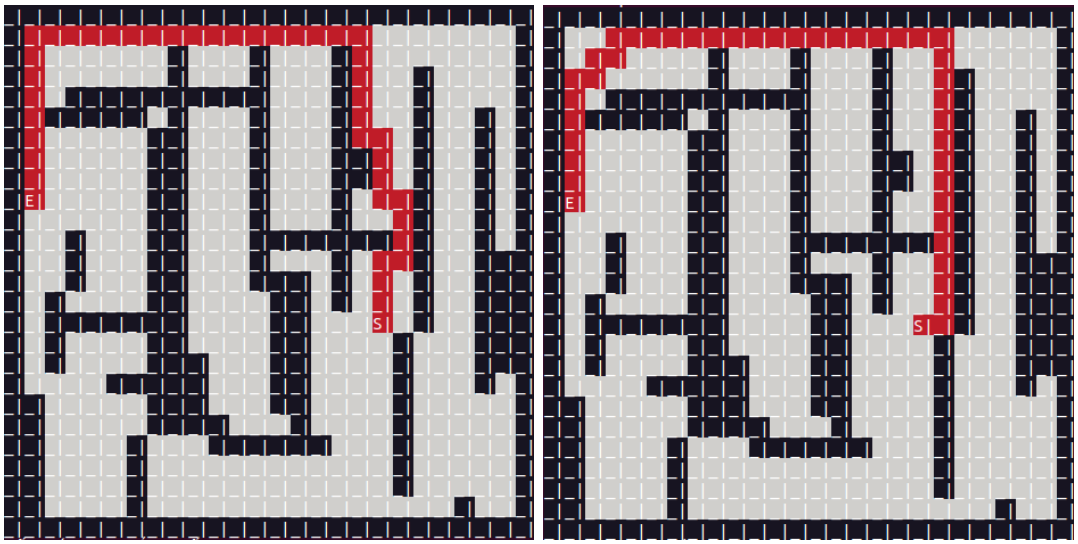
  ← Nếu z = 0 nghĩa là không tồn tại đường đi, không có đỉnh xét nào bằng đỉnh kết thúc

  ← Nếu z ≠ 0 lật ngược đường đi đã lưu lại và in ra

7 KẾT QUẢ VÀ THẢO LUẬN

7.1 Ngôn ngữ Python

Một ví dụ về mê cung với kích thước 26x26 được chúng tôi tạo ra tên python:



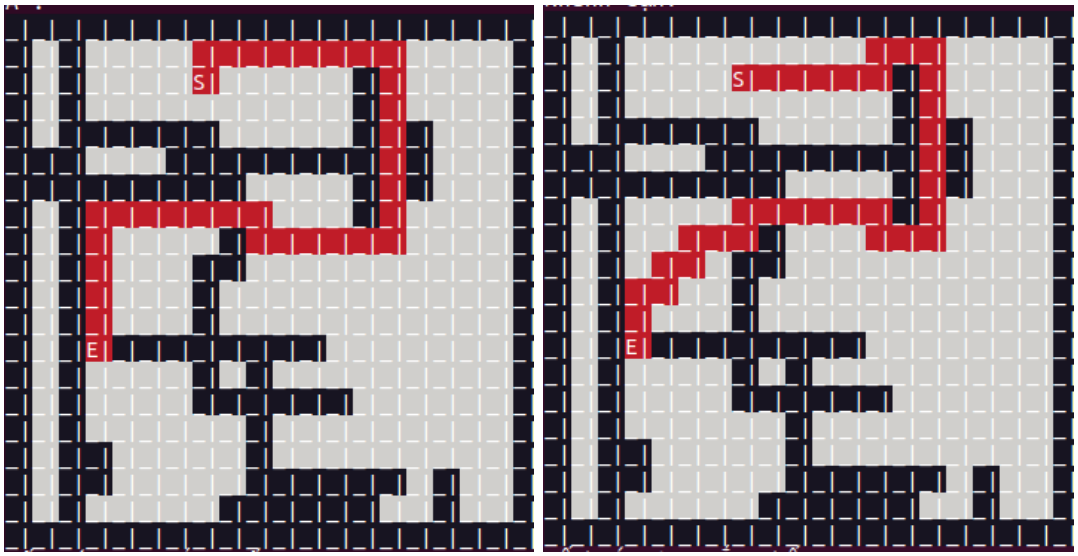
Hình 6: Kết quả mê cung Python 26x26(A* bên trái - Nhánh_cận bên phải)

Với kết quả trên hình 6 chúng tôi thu được kết quả đánh giá như sau:

Đánh giá	A*	Nhánh- cận
Số bước đi	41	41
Thời gian xử lý (giây)	0.00678	0.09081
Bộ nhớ sử dụng (KB)	5303533568	5303754752

Như vậy có thể thấy ở ví dụ hình 6, cả 2 thuật toán cùng đưa ra được đường ngắn nhất với 41 bước đi, lượng bộ nhớ của A* ít hơn và thời gian xử lý của A* nhanh hơn hẳn so với Nhánh- cận.

Một ví dụ về mê cung 20x20:



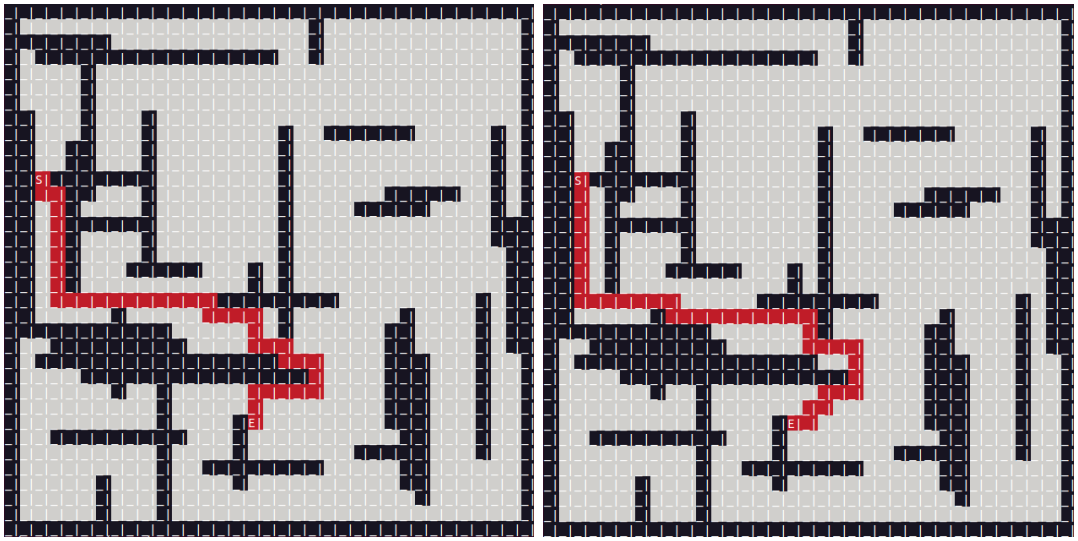
Hình 7: Kết quả mê cung Python 20x20(A* bên trái - Nhánh_cận bên phải)

Với kết quả trên hình 7 chúng tôi thu được kết quả đánh giá như sau:

Đánh giá	A*	Nhánh- cận
Số bước đi	32	32
Thời gian xử lý (giây)	0.00652	0.02544
Bộ nhớ sử dụng (KB)	5474463744	5474287616

Như có thể thấy kết quả của hình 7, số bước đi ngắn nhất tìm được của 2 thuật toán đều là 32, thời gian xử lý của A* nhanh hơn Nhánh- cận, nhưng Nhánh-cận tiêu tốn ít bộ nhớ hơn.

Một ví dụ khác về kết quả thu được với mê cung 40x40:



Hình 8: Kết quả mê cung Python 35x35(A* bên trái - Nhánh_cận bên phải)

Với kết quả trên hình 8 chúng tôi thu được kết quả đánh giá như sau:

Đánh giá	A*	Nhánh- cận
Số bước đi	38	38
Thời gian xử lý (giây)	0.0048	0.45735
Bộ nhớ sử dụng (KB)	6305951744	6321172480

Như có thể thấy kết quả của hình 8, số bước đi ngắn nhất tìm được của 2 thuật toán đều là 38, thời gian xử lý của A* nhanh hơn Nhánh- cận, A* cũng tốn ít bộ nhớ hơn.

Tùy vào mê cung và vị trí điểm bắt đầu, kết thúc sẽ cho thấy thuật toán nào tốt hơn. Tuy nhiên nhìn chung, A* là thuật toán tìm kiếm theo chiều rộng, bởi vậy khi tìm thấy node đánh giá tốt hơn sẽ luôn được ưu tiên hàng đầu (đẩy vào đầu danh sách). Nhánh- cận thì tìm kiếm theo chiều sâu, nhưng thay vì đẩy tất cả vào hàng đợi rồi xếp theo thứ tự ưu tiên như A*, thì các node cùng cha sẽ được xếp theo thứ tự ưu tiên, rồi đẩy toàn bộ vào đầu danh sách chờ. Bởi vậy theo các ví dụ có thể thấy, đường đi của A* có xu hướng đi đường "zích zắc"hơn là Nhánh- cận, Nhánh- cận vẫn có xu hướng đi theo đường thẳng hơn A*.

7.2 C++

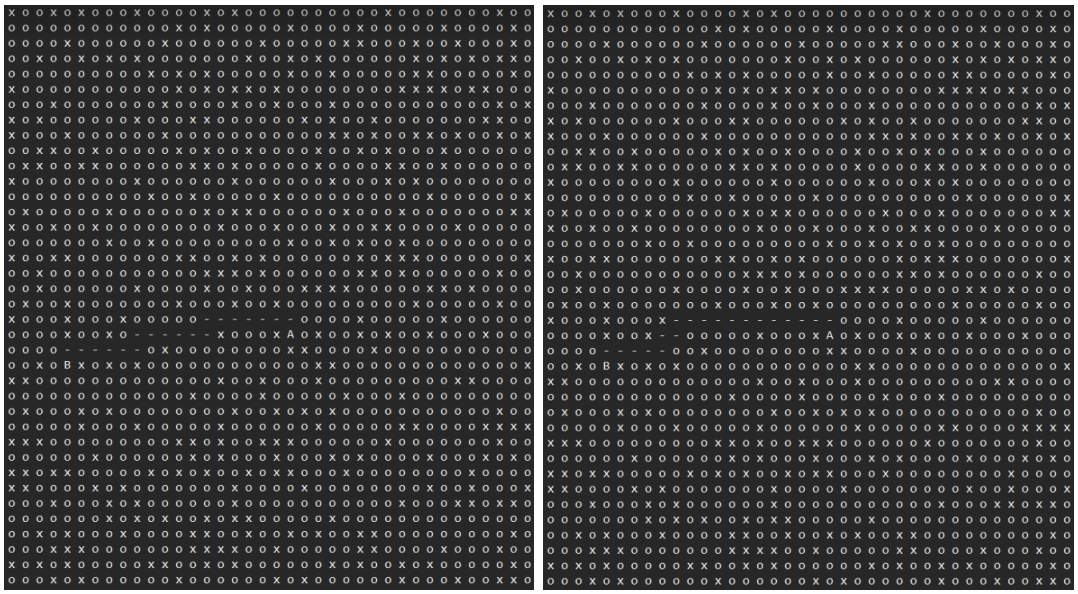
Một ví dụ về mê cung với kích thước 38x38 được chúng tôi tạo ra tên C++:

Hình 9: Kết quả mê cung C++ 38x38(A* bên trái - Nhánh_cận bên phải)

Với kết quả trên hình 10 chúng tôi thu được kết quả đánh giá như sau:

Đánh giá	A*	Nhánh- cận
Số bước đi	42	42
Thời gian xử lý (giây)	0.78	1.32
Số nút xét	216	346

Một ví dụ về mê cung với kích thước 38x38 được chúng tôi tạo ra tên C++:



Hình 10: Kết quả mê cung_2 C++ 38x38 (A* bên trái - Nhánh_cận bên phải)

Với kết quả trên hình 10 chúng tôi thu được kết quả đánh giá như sau:

Đánh giá	A*	Nhánh- cận
Số bước đi	19	19
Thời gian xử lý (giây)	0.842	1.42
Số nút xét	84	131

So sánh hai thuật toán A* và nhánh cận trên C++, sử dụng hàm đánh giá là hàm tính khoảng cách manhattan Về số nút phải xét:

- Thuật toán A* có số nút phải xét ít hơn so với thuật toán nhánh cận
- Về thời gian thực thi: Đối với ma trận có n cỡ 90 - 100, thời gian thực thi trung bình của A* là 1,3s trong khi ở nhánh cận là 2,7s.
- Về không gian lưu trữ: Cả hai đều tốn bộ nhớ là O(n) để lưu ma trận, mảng đánh dấu, tuy nhiên ở nhánh cận tốn thêm bộ nhớ để lưu giá trị biến cost

Tài liệu

[1] Blink. Distance measure trong machine learning. https://viblo.asia/p/distance-measure-trong-machine-learning-ByEZkopYZQ0, 24/6/2021.

[2] Marcelo Seido Nagano Caio Paziani Tomazella. *A comprehensive review of Branch-and-Bound algorithms: Guidelines and directions for further research on the flow-shop scheduling problem.* Elsevier, 15 November 2020.

[3] Lê Minh Hoàng. *Sách Giải thuật và Lập trình.* Đại học Sư Phạm Hà Nôi, 2002.

[4] và Bertram Raphael Peter Hart, Nils Nilsson. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions on Systems Science and Cybernetics SSC4, 1968.

[5] Kevin Wayne Robert Sedgewick. *Algorithms.* 2014. Fourth edition.

[6] Pulkit Sharma. Understanding distance metrics used in machine learning. https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/. Updated On August 8th, 2023.

[7] Ellen Zhuang. Spatial branch and bound method, 1/04/2022.