Chương 1: Tìm hiểu về các thuật toán tìm kiếm trong không gian trạng thái

* 1. Tổng quan
  2. Chiến lược tìm kiếm mù
     1. Tìm kiếm theo chiều rộng
     2. Tìm kiếm theo chiều sâu
  3. Chiến lược tìm kiếm tối ưu
     1. Thuật giải At
     2. Thuật giải Akt
     3. Thuật giải A\*

Chương 2: Xây dựng chương trình

2.1. Mô tả trò chơi

2.2. Xây dựng trò chơi

2.3. Áp dụng thuật toán A\* vào trò chơi

Chương 1: Tìm hiểu về các thuật toán tìm kiếm trong không gian trạng thái

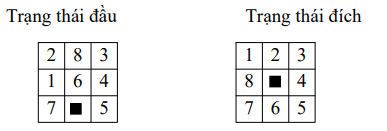
* 1. Tổng quan

Trong thực tế có rất nhiều bài toán mà chúng ta cần phải sử dụng đến phương pháp tìm kiếm như tìm đường đi tới một địa điểm trên bản đồ, tìm kiếm nước đi cho quan cờ, tìm kiếm thông tin, tìm kiếm đường đi tối ưu,... Cụ thể hơn là tìm kiếm một hay một số đối tượng nào đó trong một tập lớn các đôi tượng nhằm thỏa mãn yêu cầu nào đó của bài toán.

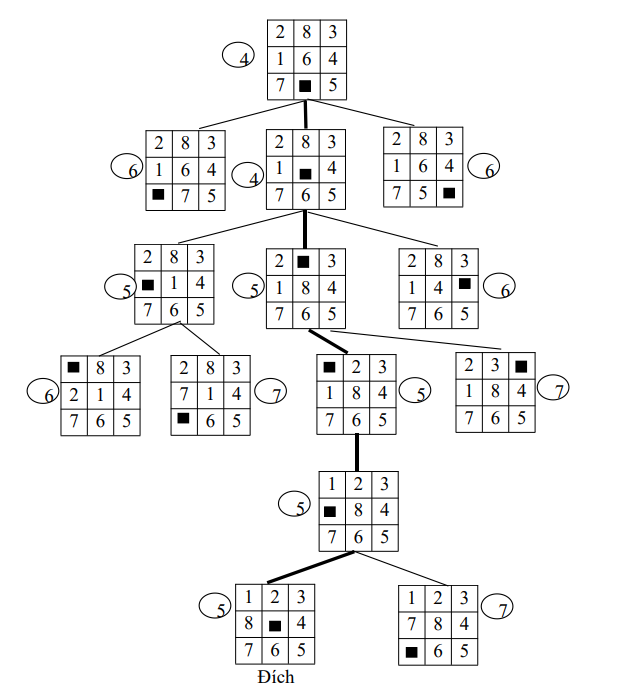
Ví dụ như trong trò chơi cờ caro, mỗi lượt đi ta có thể đánh ở nhiều vị trí khác nhau, nhưng ta cần phải tìm ra vị trí đánh sao cho cuối cùng ta có thể dành chiến thắng.

Trong lĩnh vực trí tuệ nhân tạo, vấn đề tìm kiếm lại trở nên cực kỳ quan trọng do ta phải thường xuyên đối đầu với các bài toán tìm kiếm. Để giải quyết bài toán bằng phương pháp tìm kiếm, trước hết ta phải xác định được không gian tìm kiếm (bao gồm tất cả các đối tượng mà trên đó thực hiện việc tìm kiếm). Nó có thể là không gian liên tục và nó cũng có thể là không gian các đối tượng rời rạc. Như vậy, ta sẽ xét việc biểu diễn một bài toán trong không gian trạng thái sao cho việc giải quyết bài toán này được quy về việc tìm kiếm lời giải trong không gian trạng thái.

Giải bài toán trong không gian trạng thái, trước hết phải xác định dạng mô tả trạng thái bài toán sao cho bài toán trở nên đơn giản hơn, phug hợp bản chất vật lý của bài toán (có thể sử dụng các xâu ký hiệu, vector, mảng hai chiều, cây, danh sách, ...). Mỗi trạng thái chính là mỗi hình trạng của bài toán, các tình trạng ban đầu và tình trạng cuối của bài toán gọi là trạng thái đầu và trạng thái cuối. Dưới đây là một ví dụ về trạng thái đầu và trạng thái cuối của trò chơi 8 số:



Dưới đây là các bước chuyển đổi trạng thái để biến trạng thái đầu của bài toán này về với trạng thái đích của nó:



Mỗi trạng thái là mỗi hình trạng của bài toán cho nên mỗi hình trạng của bài toán trên là mỗi một cách sắp xếp các con số. Ta có thể biểu diễn mỗi hình trạng của bài toán bằng một mảng hai chiều kích thước 3x3 trên máy tính.

Toán tử là các phép biến đổi từ trạng thái này sang trạng thái khác. Có hai cách để biểu diễn các toán tử:

* Biểu diễn như một hàm xác định trên tập các trạng thái và nhận giá trị cũng trong tập này.
* Biểu diễn dưới dạng các tập luật sản xuất.

Trong ví dụ về trò chơi 8 số, mỗi toán tử là cách chuyển ô trống, có 4 kiểu toán tử: chuyển ô trống lên trên, xuống dưới, sang trái, sang phải. Tuy nhiên, đối với một số trạng thái nào đó, một toán tử nào đó có thể không áp dụng được, chẳng hạn, nếu ô trống nằm ở cột đầu tiên thì ô trống không thể sang trái được.

Quá trình tìm kiếm lời giải của bài toán được biểu diễn trong không gian trạng thái được xem như quá trình dò tìm trên đồ thị, xuất phát từ trạng thái đầu, thông quan các toán tử dịch chuyển trạng thái, lần lượt đến với các trạng thái tiếp theo cho đến khi gặp được trạng thái đích hoặc không còn trạng thía nào để tiếp tục được nữa.

Khi áp dụng các phương pháp tìm kiếm trong không gian trạng thái người ta thường quan tâm tới các vấn đề sau:

* Kỹ thuật tìm kiếm lời giải;
* Phương pháp luận của việc tìm kiếm;
* Chiến lược tìm kiếm.

Tuy nhiên không phải các phương pháp này đều có thể áp dụng để giải quyết cho tất cả các bài toán phức tạp mà chỉ cho từng lớp bài toán. Việc lựa chọn chiến lược tìm kiếm cho bài toán cụ thể phụ thuộc nhiều vào các đặc trưng của bài toán.

* 1. Chiến lược tìm kiếm mù

Trong các kỹ thuật tìm kiếm này, không có một sự hướng dẫn nào cho sự tìm kiếm mà ta chỉ phát triển các trạng thái một cách hệ thống từ trạng thái ban đầu cho đến khi gặp một trạng thái đích nào đó. Có 3 kỹ thuật tìm kiếm mù cơ bản đó là tìm kiếm theo chiều rộng, tìm kiếm theo chiều sâu và tìm kiếm sâu dần.

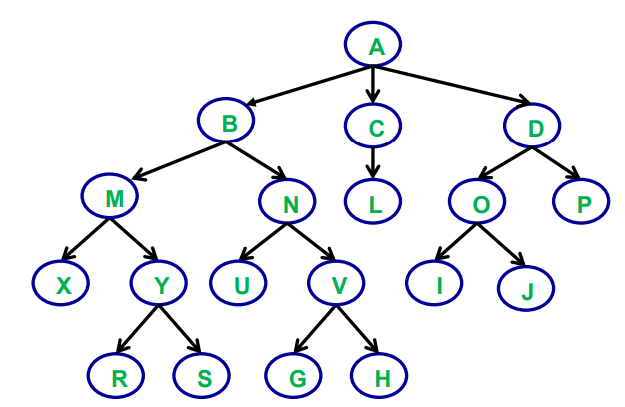
* + 1. Tìm kiếm theo chiều rộng (BFS)

Phương pháp sử dụng: từ đỉnh xuất phát, duyệt tất cả các đỉnh kề với đỉnh này, sau đó lại làm như vậy với các đỉnh vừa được duyệt. Quá trình duyệt kết thúc khi tìm thấy đỉnh kết thúc hoặc duyệt hết đồ thị mà không tìm thấy.

Sử dụng hai danh sách DONG và MO để lưu trữ các đỉnh. Trong đó cả hai danh sách DONG và MO đều hoạt động theo cơ chế hàng đợi (vào trước ra trước). Danh sách DONG chứa các đỉnh đã xét còn danh sách MO chứa các đỉnh đang xét.

Thuật giải:

Ví dụ: n là đỉnh đang xét, A(n) là các đỉnh kề với đỉnh đang xét



|  |  |  |  |
| --- | --- | --- | --- |
| n | A(n) | MO | DONG |
|  |  | A |  |
| A | B,C,D | B,C,D | A |
| B | M,N | M,N,C,D | A,B |
| M | X,Y | X,Y,N,C,D | A,B,M |
| X | rỗng | Y,N,C,D | A,B,M,X |
| Y | R,S | R,S,N,C,D | A,B,M,X,Y |
| R |  |  |  |

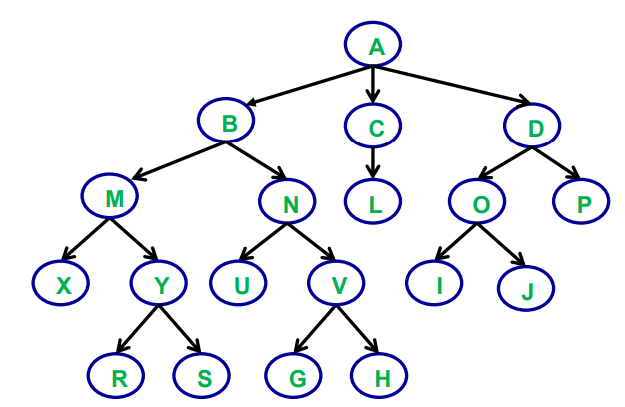
Đỉnh xét là R nên ta dừng tìm kiếm.

* + 1. Tìm kiếm theo chiều sâu (DFS)

Phương pháp sử dụng: các đỉnh trong đồ thị sẽ được duyệt theo từng mức độ sâu, cụ thể là các đỉnh của đồ thị được duyệt theo từng nhánh đến nút lá và nếu chưa tìm thấy đỉnh đích thì quay lui tới một đỉnh nào đó để sang nhánh khác.

Tìm kiếm theo chiều sâu chỉ khác ở danh sách DONG và MO để lưu trữ các đỉnh. Trong đó, danh sách DONG hoạt động theo cơ chế hàng đợi (vào trước ra trước) và danh sách MO hoạt động theo cơ chế ngăn xếp (vào sau ra trước) – Thay vì bổ sung các đỉnh vào cuối tập MO ta bổ sung các đỉnh kề vào đầu tập OPEN, nên khi bổ sung các đỉnh mới hoặc lấy đỉnh ra ta chỉ thực hiện ở một đầu.

Ví dụ: n là đỉnh xét, A(n) là đỉnh kề với đỉnh xét



|  |  |  |  |
| --- | --- | --- | --- |
| n | A(n) | MO | DONG |
|  |  | A |  |
| A | B,C,D | B,C,D | A |
| B | M,N | M,N,C,D | A,B |
| M | X,Y | X,Y,N,C,D | A,B,M |
| X | rỗng | Y,N,C,D | A,B,M,X |
| Y | R,S | R,S,N,C,D | A,B,M,X,Y |
| R |  |  |  |

Đỉnh xét là đỉnh R nên ta dừng tìm kiếm.

* 1. Chiến lược tìm kiếm tối ưu

Ưu điểm của DFS là không quan tâm tới sự mở rộng của tất cả các nhánh. Ư điểm của BFS là không bị sa vào các đường dẫn bế tắc. Tuy nhiên, trong BFS và DFS, khi chúng ta đang ở một nút, ta có thể xét bất kỳ nút kề nào là nút kế tiếp. Vì vậy, cả BFS và DFS đều tìm các đường đi một cách mù quáng mà không cần xét đến bất kỳ hàm chi phí nào. Ý tưởng của tìm kiếm tối ưu (BeFS) là sử dụng hàm đánh giá để quyết định vùng lân cận nào có triển vọng nhất và sau đó tìm đường đi trong vùng này. BeFS thuộc lớp phương pháp tìm kiếm theo kinh nghiệm hay tìm kiếm có thông tin bổ sung.

Thuật giải tìm kiếm tối ưu:

(1). Đặt OPEN chứa trạng thái khởi đầu T0.

(2). Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, ta thực hiện:

(a). Chọn trạng thái tốt nhất (Tmax) trong OPEN (và xóa Tmax khỏi OPEN).

(b). Nếu Tmax là trạng thái kết thúc thì thoát.

(c). Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện: Tính f(Tk); thêm Tk vào OPEN;

Thông thường trong các trong các phương án tìm kiếm theo kiểu BeFS, chi phí f của một trạng thái được tính dựa theo hai giá trị mà ta gọi là g và h. Trong đó, h là một ước lượng về chi phí từ trạng thái hiện hành cho đến trạng thái đích (thông tin tương lai), còn g là chiều dài quãng đường đã đi từ trạng thái đầu cho đến trạng thái hiện tại (thông tin quá khứ). Khi đó, hàm ước lượng tổng chi phí f(n) được tính theo công thức: f(n) = g(n) + h(n).

Một số phiên bản của BeFS có thể kể đến là: At, Akt và A\*.

* + 1. Thuật giải At

Thuật giải At là một phương pháp tìm kiếm theo kiểu BeFS với chi phí của đỉnh là giá trị hàm g (tổng chiều dài thực sự cảu đường đi từ đỉnh bắt đầu đến đỉnh hiện tại).

Phương pháp:

* Tìm đường đi có chi phí nhỏ nhất từ một đỉnh cho trước tới một đỉnh.
* Sử dụng hai danh sách MO và DONG.
* Ta sẽ chọn đỉnh có chi phí nhỏ nhất trong tập MO để đem ra xét tiếp. Xét đến khi nào xuất hiện đỉnh cần tìm thì dừng.
* Chi phí f(n) = g(n).
* Lưu ý: Nếu xuất hiện hai đỉnh cùng min (chúng có cùng chi phí nhỏ nhất) thì ta chọn ngẫu nhiên một đỉnh để xét.

Thuật giải:

Void AT ()

{

Open 🡨 { T0 }, g( T0) = 0, Closed = Ø

while Open ≠[](https://thuthuatnhanh.com/cach-viet-dau-khac-trong-excel/) Ø do

{

n 🡨 getNew (Open) //lấy đỉnh n sao cho g(n) 🡪 min

if ( n = TG ) then return True

else

{

for each m ∈ A(n) do

if ( m ∉ Open) and (m ∉ Closed) then

{

g(m) = g(n) + cost(m, n)

Open = Open ∪ {m}

}

else g(m) = min {g(m), gnew (m)}

Closed = Closed ∪ {n}

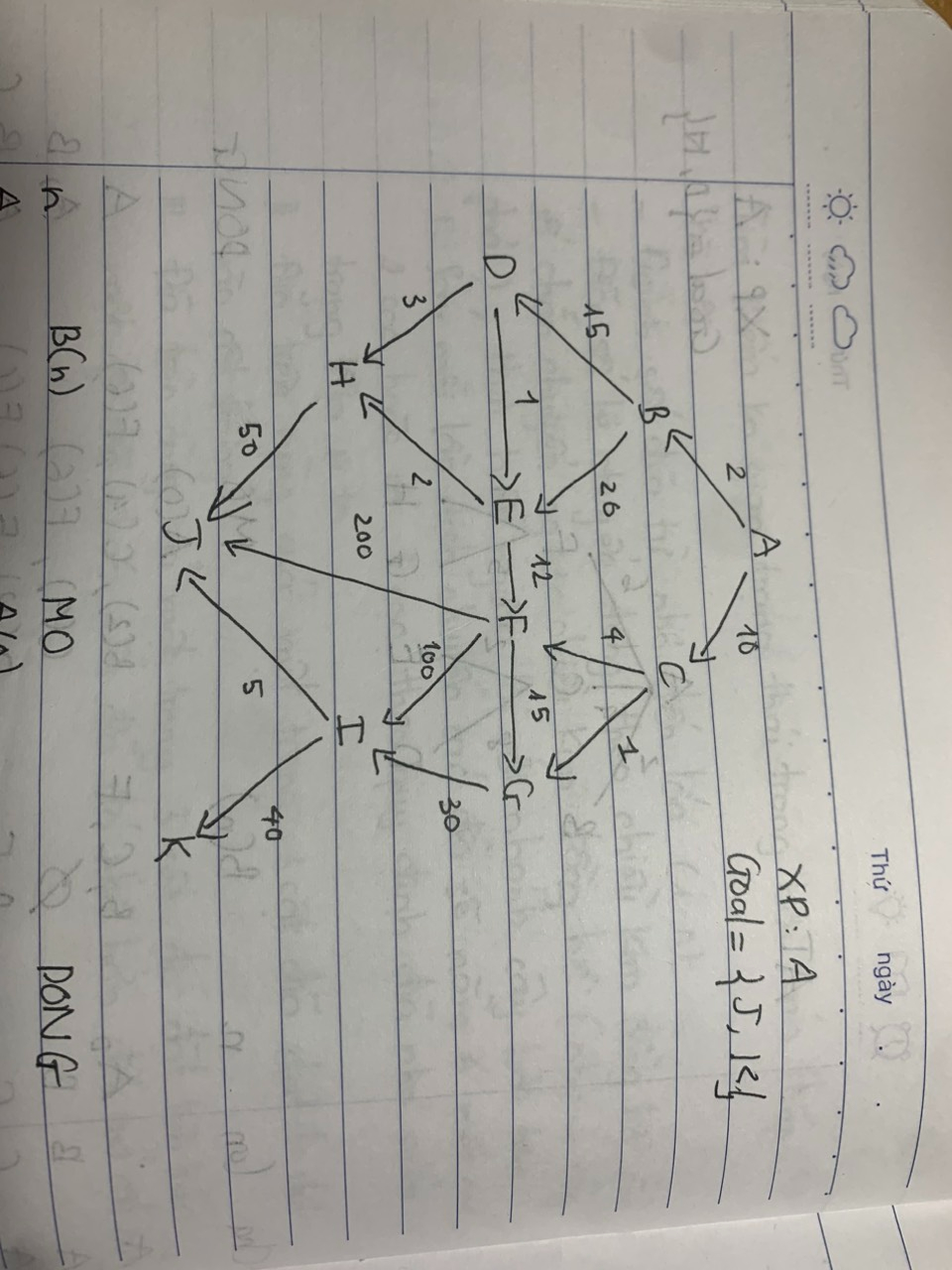
}

return False;

}

}

Ví dụ:



* + 1. Thuật toán Akt

Thuật giải Akt trong quá trình tìm đường đi chỉ xét đến các đỉnh và giá của chúng. Nghĩa là việc tìm đỉnh triển vọng chỉ phụ thuộc vào hàm g(n) – thông tin quá khứ. Tuy nhiên, thuật giải này không còn phù hợp khi gặp phải những bài toán phức tạp (độ phức tạp cấp hàm mũ) do ta phải xét một lượng nút lớn. Để khắc phục nhược điểm này, người ta sử dụng thêm các thông tin bổ sung xuất phát từ bản thân bài toán để tìm ra các đỉnh có triển vọng, tức là đường đi tối ưu sẽ tập trung xung quanh đường đi tốt nhất nếu sử dụng các thông tin đặc tả về bài toán.

Theo thuật giả này, chi phí của đỉnh được xác định: f(n) = g(n) + h(n):

* Đỉnh n được chọn nếu f(n) min.
* g(n) là chi phí đi từ đỉnh trước đó đến đỉnh hiện tại.
* h(n) là hàm thỏa mãn yêu cầu loại bỏ bớt các đỉnh không liên quan và tìm ra các đỉnh có triển vọng (tùy thuộc vào từng bài toán)
* Một số phương pháp xây dựng hàm h(n):
  + Dựa vào xác suất của đỉnh trên đường đi tối ưu.
  + Dựa vào khoảng cách, sự sai khác của trạng thái đang xét với trạng thái đích hoặc các thông tin liên quan đến trạng thái đích.

Thuật giải:

Void AKT ()

{

Open 🡨 { T0 }, g( T0) = 0

Tính h ( T0), f ( T0) = g (T0) +h (T0)

while Open ≠[](https://thuthuatnhanh.com/cach-viet-dau-khac-trong-excel/) Ø do

{

n 🡨 getNew (Open) //lấy đỉnh n sao cho g(n) 🡪 min

if ( n = TG ) then return True

else

{

for each m ∈ A(n) do

{

g(m) = g(n) + cost(m, n)

Tính h (m), f (m) = g (m) + h (m)

Open = Open ∪ {m}

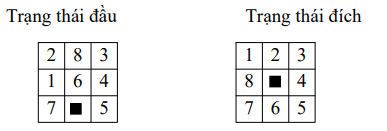
}

}

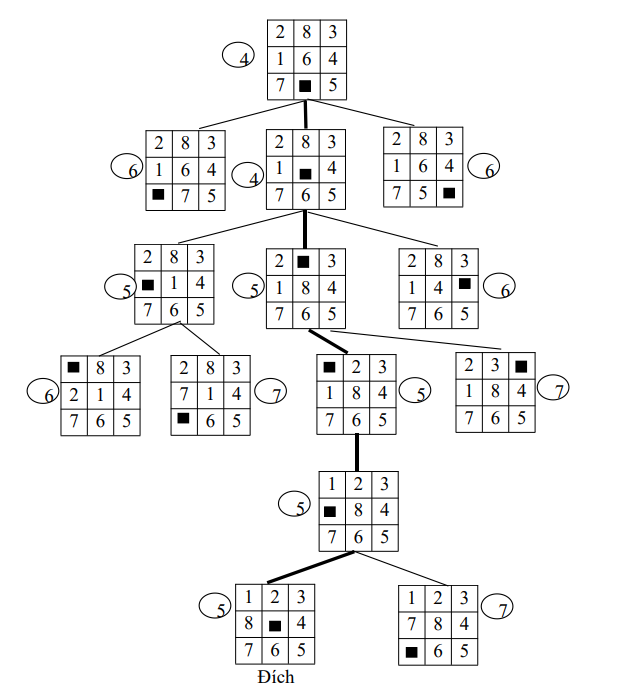
return False;

}

Ví dụ:



* Chọn hàm h(n) là số các con số không nằm đúng vị trí của nó so với trạng thái đích.
* Chọn hàm g(n) là số lần dịch chuyển ô trống từ trạng thái đầu đến trạng thái n.



* + 1. Thuật toán A\*

Thuật toán tìm kiếm A\* là một trong những kỹ thuật tốt nhất và phổ biến nhất được sử dụng để tìm kiếm đường đi và duyệt đồ thị. Không giống như các kỹ thuật duyệt khác, A\* có “bộ não”. Điều đó có nghĩa là nó thực sự là một thuật toán thông minh tách biệt nó với các thuật toán thông thường khác. Nhiều trò chơi và bản đồ dựa trên nền tảng web sử dụng thuật toán này để tìm đường đi ngắn nhất rất hiệu quả.

Thuật toán A\* là mở rộng của Akt bằng cách bổ sung giả quyết trường hợp khi mở một nút mà nút này đã có sẵn trong tập MO hoặc DONG.

Ngoài lưu trữ các giá trị cơ bản như f, h, g. Nó còn lưu trữ:

* Trạng thái cha của trạng thái Ti. Trong trường hợp có nhiều trạng thái dẫn đến Ti (nhiều cha) thì chọn Cha(Ti) sao cho chi phí đi từ trạng thái khởi đầu đến Ti là thấp nhất: g(Ti)min = (g(Tcha) + cost(Tcha, Ti)min).
* Danh sách các trạng thái kế tiếp của Ti: danh sách này lưu trữ các trạng thái kế tiếp Tk của Ti sao cho chi phí đến Tk thông qua Ti từ trạng thái ban đầu là thấp nhất.

Thuật giải:

Void AStar ()

{

Open = { T0 }, Closed = Ø

g( T0) = 0, tính h ( T0), f ( T0) = g (T0) +h (T0)

while Open ≠[](https://thuthuatnhanh.com/cach-viet-dau-khac-trong-excel/) Ø

{

n 🡨 getNew (Open) //lấy đỉnh n sao cho g(n) 🡪 min

if ( n = TG ) then return path T 🡪 TG

else

{

for each m ∈ A(n) do

if (m ∉ Open + Closed)

{

Tính h (m), g (m),

f (m) = g (m) + h (m)

Open = Open ∪ {m}

}

else

{

g (m) = min { gold (m), gnew (m)}

Cập nhật lại Open

}

}

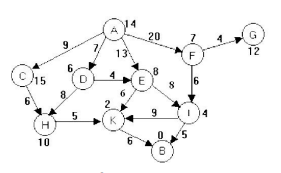
Closed = Closed ∪ {n}

}

return False;

}

Ví dụ: n là đỉnh xét, A(n) là đỉnh kề với đỉnh đang xét



Trạng thái ban đầu là: T0 = A; trạng thái đích là B, cách số chi cạnh cung là chi phí đường đi, các số ghi cạnh đỉnh là giá trị hàm h

Ban đầu MO = {A, g(A)=0, f(A)=14}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n | A(n) | MO | DONG |  |  |
|  |  | A{g(A)=0,f(A)=14} |  |  |  |
| A(0) | C,D,E,F | C{g(C)=9,f(C)=g(C)+h(C)=9+15=24,cha(D)=A}  D{g(D)=7,f(D)=7+6=13,cha(D)=A}  E{g(E)=13,f(E)=13+8=21,cha(E)=A}  F{g(F)=20,f(F)=20+7=27,cha(F)=A} | A{g(A)=0,f(A)=14} |  |  |
| D(7) | E,H | En{g(En)=4+7=11}<E->Lấy En  E{g(E)=11,f(E)=11+8=19,cha(E)=D}  H{g(H)=7+8=15,f(H)=25,cha(H)=D}  C{g(C)=9,f(C)=g(C)+h(C)=9+15=24,cha(D)=A}  F{g(F)=20,f(F)=20+7=27,cha(F)=A} | A{g(A)=0,f(A)=14}  D{g(D)=7,f(D)=7+6=13,cha(D)=A} |  |  |
| E(11) | K,I | K{g(K)=17,f(K)=19,cha(K)=E}  I{g(I)=19,f(I)=23,cha(I)=E}  H{g(H)=7+8=15,f(H)=25,cha(H)=D}  C{g(C)=9,f(C)=g(C)+h(C)=9+15=24,cha(D)=A}  F{g(F)=20,f(F)=20+7=27,cha(F)=A} | A{g(A)=0,f(A)=14}  D{g(D)=7,f(D)=7+6=13,cha(D)=A}  E{g(E)=11,f(E)=11+8=19,cha(E)=D} |  |  |
| K(17) | B | B{g(B)=23,f(B)=23,cha(B)=K}  I{g(I)=19,f(I)=23,cha(I)=E}  H{g(H)=7+8=15,f(H)=25,cha(H)=D}  C{g(C)=9,f(C)=g(C)+h(C)=9+15=24,cha(D)=A}  F{g(F)=20,f(F)=20+7=27,cha(F)=A} | A{g(A)=0,f(A)=14}  D{g(D)=7,f(D)=7+6=13,cha(D)=A}  E{g(E)=11,f(E)=11+8=19,cha(E)=D}  K{g(K)=17,f(K)=19,cha(K)=E} |  |  |
| B(23) |  |  |  |  |  |

B -> K -> E -> D -> A.

Đường đi: A -> D -> E -> K -> B.

Lưu ý: Đường đi tìm được có thể không phải là tốt nhất.

Chương 2: Xây dựng chương trình

2.1. Mô tả trò chơi

Pacman là một trò chơi kinh điển trở thành biểu tượng của văn hóa đại chúng từ những năm 80.

Trong trò chơi người chơi sẽ điều khiển Pacman trong một mê cung và ăn các chấm pac. Nếu người chơi ăn hết chấm pac thì sẽ được qua màn. Có 4 đối thủ là Blinky, Pinky, Inky và Clyde đi tự do trong mê cung và sẽ cố gắng để bắt Pacman. Nếu bị bắt Pacman sẽ bị mất mạng. Trong mê cung có các chấm tròn to (big pac) hay còn gọi là viên sức mạnh, khi ăn được các chấm này Pacman sẽ có khả năng ăn kẻ địch trong một khoảng thời gian ngắn. Khi đó kẻ địch sẽ chuyển sang màu lam, di chuyển chậm lại và chạy trốn khỏi Pacman. Khi một kẻ địch bị ăn, đôi mắt của chúng sẽ di chuyển về vị trí hồi sinh ở trung tâm màn hình, sau đó chung sẽ được hồi sinh lại như ban đầu.

Pacman: người chơi sẽ sử dụng cái phím mũi tên để giúp pacman chuyển hướng trong mê cung, giúp pacman tránh khỏi sự truy đuổi của 4 con ma và ăn hết các chấm pac. Mỗi khi bị giết bởi các con ma, Pacman sẽ bị trừ đi một mạng và khi không còn mạng nào trò chơi sẽ kết thúc.

Ghosts: 4 con ma sẽ có các trạng thái: đuổi bắt pacman, ngừng đuổi bắt và chạy khỏi pacman – khi panman ăn được viên sức mạnh. Khi bị pacman giết, nó sẽ chuyển thành hình đôi mắt và di chuyển về nơi hồi sinh và sẽ được hồi sinh sau một khoảng thời gian sau đó lại tiếp tục đuổi bắt pacman. Mỗi con ma sẽ được cài đặt các cách di chuyển khác nhau.

2.2. Các bước xây dựng trò chơi

Nhóm đã thực hiện các bước sau để xây dựng trò chơi

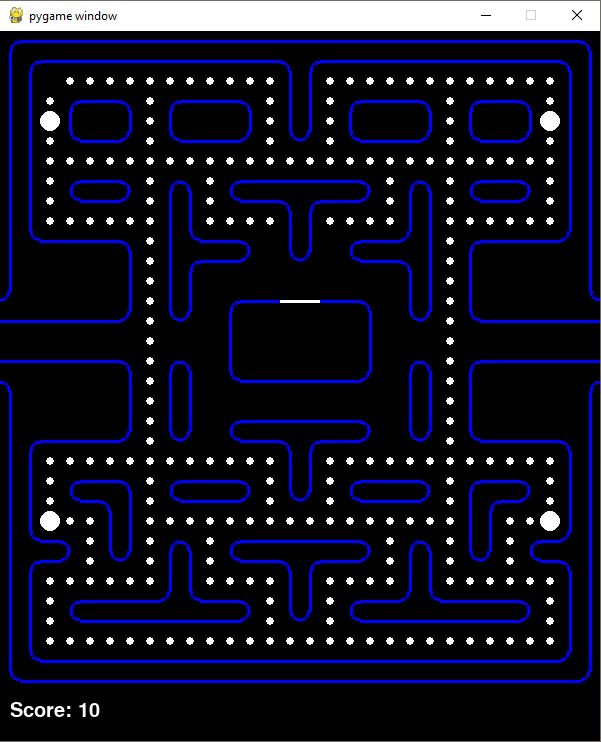
* Bước 1: Tạo mê cung và các chấm pac.
* Bước 2: Tạo pacman bao gồm các thuộc tính và chuyển động cho pacman.
* Bước 3: Tạo ra các ghost bao gồm các thuộc tính cho ghost và cách di chuyển của ghost.

Bước 1: Tạo mê cung và các chấm pac

Sử dụng một mảng 2 chiều có kích thước 30x32 để tạo mê cung. Trong đó:

* Giá trị 0: là các khoảng trống màu đen trong mê cung;
* Giá trị 1: là các chấm pac;
* Giá trị 2: là các chấm big – pac;
* Giá trị 3,4,5,6,7,8: lần lượt là các giá trị đại diện cho các cạnh của bức tường;
* Giá trị 9: là các cánh cổng hồi sinh;

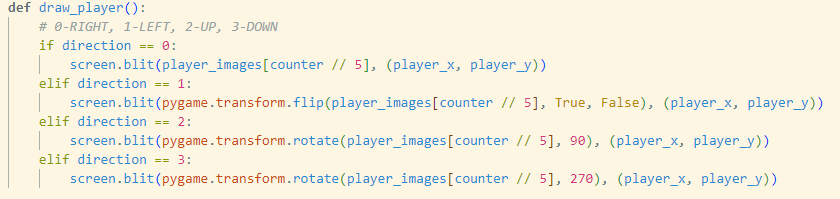


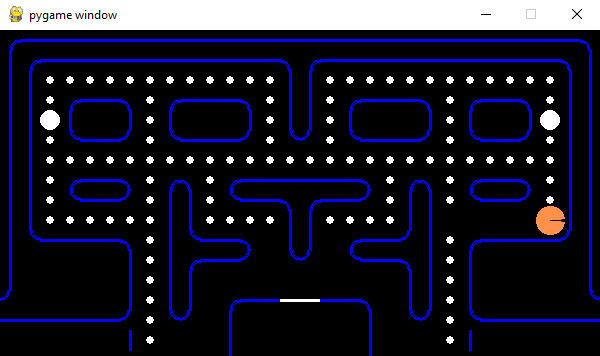


Bước 2: Tạo Pacman, các thuộc tính và chuyển động cho Pacman.

Tạo Pacman:

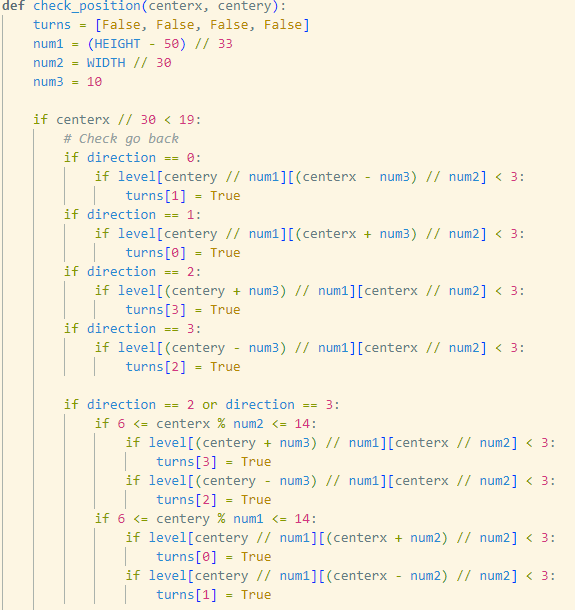
* Pacman sẽ có các hình ảnh khác nhau khi người chơi chuyển hướng pacman trong mê cung;
* Các hình ảnh đó tương ứng với các dirrection 0, 1, 2 và 3.





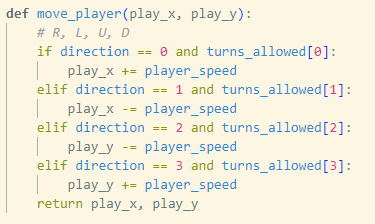
Kiểm tra vị trí di chuyển của Pacman:

* Pacman chỉ được phép chuyển hướng hoặc quay lại ở các ô có giá trị cho phép (0, 1 và 2);
* Do đó, mỗi lần di chuyển các vị trí mà pacman đang hướng đến luôn phải được kiểm tra trước xem vị trí đó có thể đi hay không để pacman có thể tránh đi vào các bức tường.



Tạo chuyển động cho Pacman:

* Để chuyển hướng Pacman trong mê cung, người chơi sử dụng phím mũi tên và trước khi chuyển hướng đến mũi tên được nhấn, vị trí của nó sẽ được kiểm tra.



Bước 3: Tạo các Ghost, các thuộc tính và cách di chuyển cho các Ghost

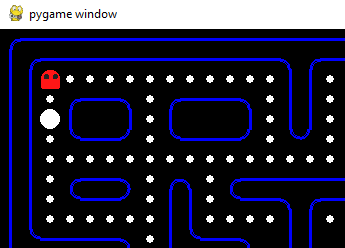
Tạo ghost và tọa độ mặc định của ghost trên map:

* Thao tác tạo ghost tương tự với Pacman nhưng ghost chỉ cần 1 ảnh để hiển thị trong suốt quá trình chuyển hướng.



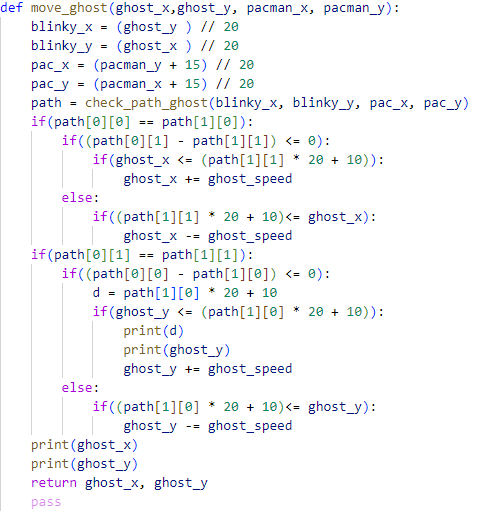
* Thiết lập tọa độ mặc định cho ghost





Tạo cách di chuyển cho các ghost:

* Ghost sẽ di chuyển theo đường dẫn được lựa chọn trong hàm check\_path\_ghost;
* Các câu lệnh điều kiện kiểm tra quãng đường đi được và điều hướng cho ghost khi cần.



2.3. Áp dụng thuật toán A\* vào trò chơi

2.3.1. Ý tưởng xây dựng thuật toán:

Mỗi một ô trên map có các giá trị là 0, 1 và 2 đều có thể di chuyển được. Do đó, ta thiết lập các ô này làm các node. Vì ghost và Pacman luôn luôn di chuyển trên các node do đó ghost có thể dễ dàng tìm thấy được Pacman khi nó di chuyển.

Một thuật giải A\* cơ bản cần có các chỉ số: nút xét, các nút kề với nút đang xét, danh sách mở lưu trữ các các nút cho các lần xét tiếp theo và danh sách đóng lưu trữ các nút đã được xét. Ngoài ra, thuật giả A\* còn sử dụng hàm f = g + h để đánh giá các nút. Trong đó, g là chi phí đường đi từ nút ban đầu cho tới nút đang xét còn f là chi phí ước lượng tính từ nút đang xét tới nút đích.

Để áp dụng thuật giải này vào trong trò chơi, ta sử dụng khoảng cách Manhattan cho hàm ước lượng h, đây là một dạng khoảng cách giữa hai điểm trong không gian Euclid với hệ tọa độ Descartes. Đại lượng này được tính bằng tổng chiều dài của hình chiếu đường của đường thẳng nối hai điểm này trong hệ trục tọa độ Descartes. Khoảng cách Manhattan giữa hai điểm P1(x1, y1) và P2(x2, y2) là:

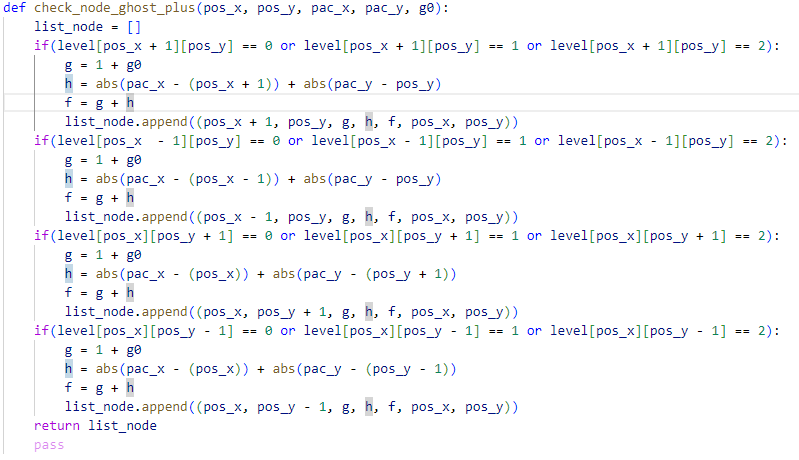
|x1 – x2| + |y1 – y2|

Ngoài sử dụng khoảng cách Manhattan cho hàm heuristic h ta còn có thể sử dụng khoảng cách chéo hay khoảng cách Euclide cho hàm này. Nhưng trong trò chơi này nhóm em sử dụng khoảng cách Manhattan cho hàm h vì hàm này sẽ tối ưu nhất cho những bài toán chỉ được phép di chuyển theo 4 hướng (trái, phải, trên, dưới).

2.3.2. Áp dụng thuật toán

Các bước triển khai theo thứ tự trong thuật toán A\* đó là: Từ nút bắt đầu, đưa ra các nút kề với nút đầu, đưa các nút kề vào một danh sách, đưa nút đang xét vào danh sách đóng ,xem xét và tìm ra nút tối ưu trong danh sách, xét nút tối ưu, đưa ra các nút kề với nút tối ưu, đưa các nút kề vào danh sách, đưa nút đang xét vào danh sách đóng, chọn nút tối ưu trong danh sách, ... Quá trình này tiếp tục cho đến khi nút tối ưu được tìm thấy là nút kết thúc.

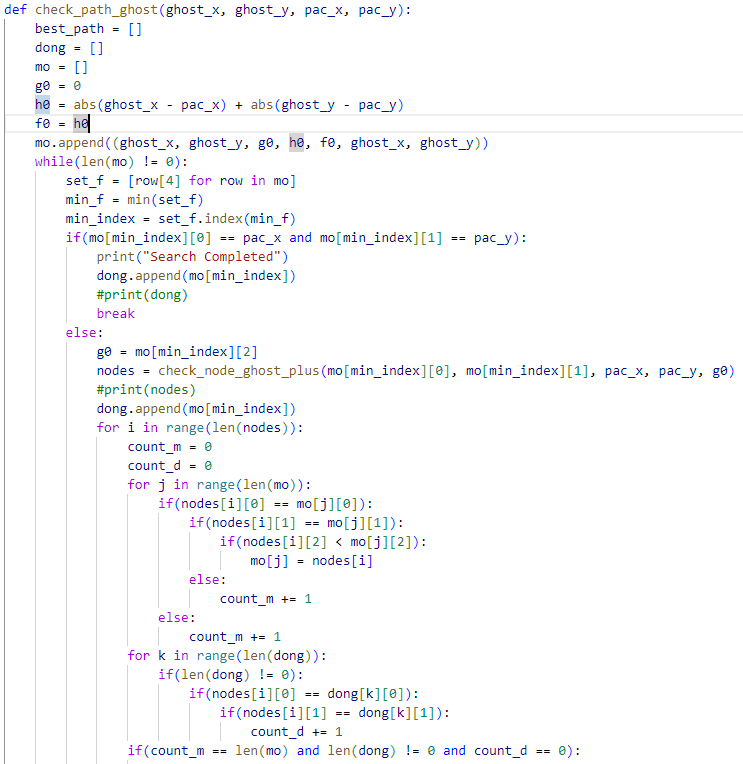
Bước đầu tiên cần phải làm trong thuật toán này đó là: tìm ra các nút kề với nút đang xét (hay đưa ra các node kề với node đang xét). Để thực hiện điều này ta tạo một hàm xem xét các nút, hàm này được đặt tên là: check\_ghost\_node\_plus. Với các tham số đầu vào của hàm là: vị trí hàng hiện tại, vị trí cột hiện tại, vị trí hàng đích, vị trí cột đích, chi phí tại vị trí hiện tại tương ứng với các tham số lần lượt là (pos\_x, pos\_y, pac\_x, pac\_y, g0). Giá trị trả về của hàm là danh sách các node kề với node hiện tại:

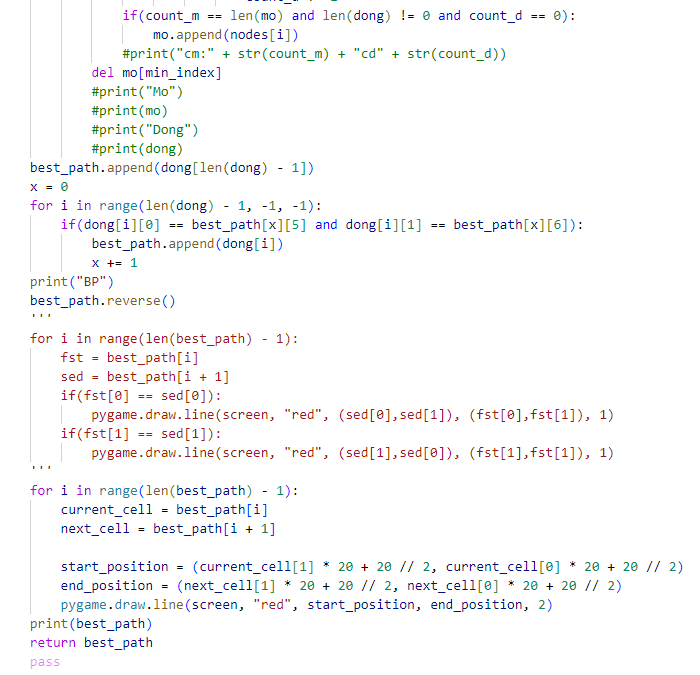


Trong hàm này, ta lần lượt kiểm tra 4 vị trí xung quanh node hiện tại để phát hiện các node kề (các node có giá trị 0, 1 và 2). Ngoài ra, các giá trị g, h, f tại nút đó cũng được tính toán theo. Cuối cùng các node được thêm vào list\_node đã được khai báo ban đầu và trả về khi hoàn thành xong hàm này. Mỗi một chỉ số trong list\_node sẽ lưu trữ các giá trị (vị trí hàng hiện tại, vị trí cột hiện tại, g, h, f, vị trí hàng của nút cha, vị trí cột của nút cha).

Bước 2 sau khi đã có các node kề, ta phải lựa chọn các các node cho đến khi nào node được chọn là node đích và lúc đó ta sẽ tìm được đường đi. Vậy nên ta sẽ tạo một hàm đảm nhiệm nhiệm vụ tìm đường dẫn từ node hiện tại tới node đích. Hàm này sẽ có tên là: check\_path\_ghost. Đầu vào của hàm bao gồm vị trí hàng hiện tại, vị trí cột hiện tại, vị trí hàng đích, vị trí cột đích tương ứng với các tham số lần lượt là (ghost\_x, ghost\_y, pac\_x, pac\_y). Tạo các giá trị g, h, f cho nút xét và đưa nó vào danh sách mở. Sau đó là lần lượt các thao tác đưa các node vào tập mở lấy các node ra khỏi tập mở, thêm các node vao tập đóng cho đến khi tìm được node đích, chúng ta thêm node đích vào tập đóng và thoát khỏi vòng lặp while.

Lúc này tập đóng sẽ chứa các node đường dẫn từ node đầu cho đến node đích. Chúng ta sử dụng một danh sách nữa có tên là best\_path để lưu trữ và truy ngược đường dẫn. Lúc này giá trị trả về của hàm check\_path\_ghost này là best\_path có lưu trữ đường dẫn tối ưu từ node đầu cho tới node cuối.





Bước cuối cùng là sử dụng hàm move\_ghost đã được giới thiệu phía trên để đưa ghost đi theo đường dẫn đã tìm được. Đầu vào của hàm nào bao gồm các giá trị (tọa độ trục hoành của ghost, tọa độ trục tung của ghost, tọa độ trục hoành của pacman, tọa độ trục tung của pacman). Đầu ra của hàm là trả về các giá trị đã cập nhật cho vị trí của ghost và vị trí của pacman. Hàm này sẽ đảm nhiệm vai trò chuyển đổi các vị trí tung độ, hoành độ của ghost và pacman thành các chỉ số hàng và cột để thuận tiện cho việc tính toán.