TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VỀ TRUYỀN THÔNG

----------*----------

BÁO CÁO MÔN HỌC
TRÍ TUỆ NHÂN TẠO

Đề tài: Đường đi Robot.

Giáo viên hướng dẫn: TS. Phạm Văn Hải

Nhóm Sinh viên thực hiện:

1) Lê Anh Tuấn – 20122675 – CNTT2.04 K57
2) Phạm Quốc Khánh – 20111690 – CNTT2.03 K56
3) Yoem Rattana – 20114610 – CNTT2.02 K56
4) Đậu Văn Thắng – 20112678 – CNTT2.04 K56

Hà nội, 31/07/2014

Mục lục.

Lời nói đầu

Ai trong chúng ta cúng biết tầm quan trọng của robot trong cuộc sống của con người. Việc chọn một nền tảng là tìm đường đi cho robot là cực kỳ quan trọng. Do vậy chúng em đã chọn đề tài "tìm đường đi cho robot trong cứu nạn cứu hộ".

Bài toán tìm đường đi đã xuất hiện từ lâu, có nhiều nền tảng viết về vấn đề này. Nhưng sau khi học môn Trí tuệ Nhân Tạo chúng em đã sử dụng những kiến thức đã học được để làm bài báo cáo này. Trong báo cáo có một số thuật toán không phải đúng tuyệt đối, nhưng để đảm bảo thời gian thực hiện chương trình thì chúng em đã sử dụng thuật giải Heuristic nhằm tăng tốc độ thực hiện thuật toán.

Mặc dù chúng em đã cố gắng nhưng vẫn không tránh được sai sót, chúng em mong nhận được sự đóng góp của thầy đề báo cáo của chúng em hoàn thiện hơn. Em xin chân thành cảm ơn!

I) Lựa chọn đề tài và các vấn đề liên quan.

1) Giới thiệu về đề tài.

a) Tầm quan trọng của Robot.

Xã hội ngày càng phát triển mạnh mẽ, nhu cầu hiện đại hóa đang được chú ý. Việc mà chế tạo các con Robot có các hành vi, cảm xúc như một con người thực sự đang là điểm nóng trong thế giới hiện nay. Việc chế tạo robot là rất quan trọng nó sẽ giúp con người rất nhiều, làm nhiều việc mà con người không thể tiếp cận được môi trường.

Robot thì có rất nhiều vấn đề cần phải nghiên cứu tìm hiểu. Nhưng chúng em chỉ viết về một đề tài đó chính là tìm đường đi cho robot.

Để dễ hiểu chúng em lấy ví dụ cụ thể như sau:

- Có một trung tâm thương mại Thế giới ở Mỹ đang bị tấn công bởi nhóm khủng bố Al-Qaeda. Sau khi xảy ra vụ việc thì cả tòa nhà bốc cháy con người không thể tiếp cận tòa nhà. Trách nhiệm của robot là phải giải cứu càng nhiều người càng tốt. Do thời gian liên quan trực tiếp đến tính mạng của người bị nạn. Nên trong một thời gian nhất định Robot phải cứu nhiều nhất có thể. Hình ảnh cho vụ khủng bố đang xảy ra.

Hình 1: Toàn cảnh khủng bố ngày 11/9.

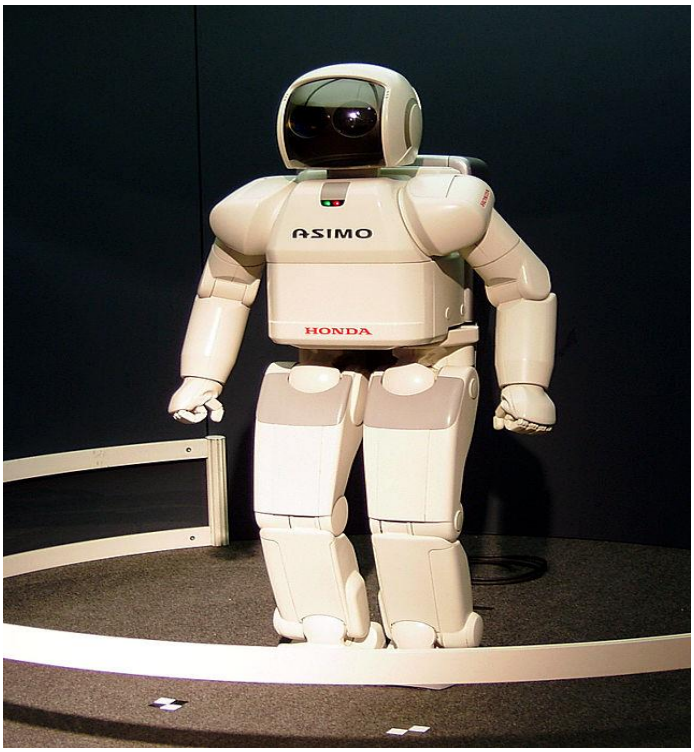
- Nhà máy hạt nhân ở Nhật bản đang gặp sự cố, đã làm rò rỉ phóng xạ ra môi trường bên ngoài rất cao. Với sự nguy hiểm của phóng xạ thì con người không thể tiếp cận được nhà máy. Để làm nguội nhà máy hạt nhân chỉ có cách duy nhất là bơm nước vào lò. Nhưng ai sẽ làm việc này, con người là không thể. Vậy nên Robot sẽ thay thế con người đi thực hiện việc bơm nước. Với một đống đổ nát trong nhà máy thì việc lập trình tìm đường là rất cần thiết.

Hình ảnh của vụ nổ nhà máy hạt nhân ở Nhật Bản.

H2: Vụ nổ nhà máy điện hạt nhân ở Nhất Bản.

b) Những thành tựu đã đạt được của Robot.

- Robot biết nói giọng của con người, có cảm xúc biết làm những việc cơ bản của một con người như, đi lại, biết rót nước, làm phát thanh viên, làm hướng dẫn viên du lịch .v.v.



H3: Robot ASIMO của Nhật Bản.

- Robot trong y tế. Làm công việc cứu người, những công việc liên quan đến độ chính xác cao, liên quan đến tính mạng của con người.

H4: Robot thực hiện ca phẫu thuật.

- Robot trong an ninh quốc phòng. Ngày càng hiện đại trên các chiến trường không cần sự có mặt của con người nữa. Tất cả đều được điều khiển bằng robot tự động. Vd. Máy bay không người lái, tên lửa hành trình, hệ thống lá chắn tên lửa .v.v.

H5: Hình ảnh máy bay không người lái đang phóng tên lửa hành trình.

2) Mô tả bài toán.

Như những ví dụ ở trên chúng ta đã hiểu được một phần quan trọng của robot. Nhưng quan trọng là robot chỉ là một cỗ máy, chúng ta phải làm thế nào mà Robot hiểu được và làm được những việc như thế.

Đó chính là mô hình bài toán vào máy tính.

Ta có một bản đồ cụ thể như bản đồ một thành phố, một tòa nhà, bản đồ qua vệ tinh, định vị GPS.

Và Robot được biểu diễn là một điểm trên Bản đồ cụ thể nào đó.

Điểm mà robot cần đến, hay đi qua cũng là một điểm trên bản đồ, và nhiệm vụ của chúng ta là phải tìm đường đi cho robot đến được đích để thực hiện một công việc nào đó.

Vd:

- Robot trong vụ khủng bố ngày 11/9 ở Mỹ.
  Bản đồ đó chính là bản đồ thành phố, và bản đồ của tòa nhà trung tâm thương mại thế giới. Điểm cần đến của robot là các phòng trong tào nhà và cứu những người trong đó ra khỏi tòa nhà.
- Robot trong vụ nổ nhà máy hạt nhân ở Nhật Bản.
  Bản đồ là toàn bộ nhà máy, và công việc cần làm là tìm đường đi đến nhà máy, phòng điều khiển và khởi động hệ thống làm nguội.

3) Biểu diễn bài toán vào máy tính.

Khi chúng ta có bản đồ và điểm xuất phát, kết thúc thì làm sao để máy tính hiểu được điều đó, hay nói cách khác là lập trình sao cho Robot nhận thức được điều đó.

a) Biểu diễn bản đồ và robot.

Chúng ta biểu diễn bản đồ bằng lưới các ô vuông(hay ma trận MxN). Trên mỗi ô vuông có các thông tin cụ thể mà chúng ta đưa vào cho robot.

Việc chia bản đồ thành các ô vuông giúp chúng ta kiểm soát được vị trí, tìm kiếm robot một cách dễ dàng hơn. Cũng giống như chia Trái đất thành các kinh độ và vĩ độ vậy. Thực chất cũng chỉ là quy ước thôi.

Robot chắc chắn sẽ thuộc một ô vuông nào đó trên bản đồ. Nên ta chỉ cần đánh dấu ô đó bằng một kí hiệu đặc biệt để thể hiện ở đó đang là vị trí của robot.

Vd: Trong tòa nhà thì chúng ta biểu diễn trong máy tính như sau:
Ma trận là toàn bộ tòa nhà, một ô vuông chính là một căn phòng. Và robot chắc chắn sẽ thuộc một phòng nào đó.

b) Tìm đường đi cho robot.

Tùy thuộc vào hoàn cảnh mà chúng ta có những nhiệm vụ riêng cho robot do đó đường đi sẽ có sự khác biệt. Để dễ hiểu chúng ta sẽ tìm hiểu qua ví dụ.

Vd1: Trong tòa nhà đang cháy thì nhiệm vụ của robot là phải đi cứu được nhiều người nhất trong đó. Bài toán này quy về tìm đường đi dài nhất trong ma trận.

Vd1: Robot đang ở trung tâm thành phố cần phải đến để giải cứu tào nhà cách đó rất xa. Bài toán quy về tìm đường đi ngắn nhất trên ma trận.

Từ ví dụ trên chúng ta có 2 bài toán cụ thể như sau:

- Tìm đường đi dài nhất trong đồ thị.
- Tìm đường đi ngắn nhất trong đồ thị. Và có thêm các yêu cầu như lặp lại các đỉnh, hay lại không lặp lại.

4) Không gian tìm kiếm.

Ma trận chỉ là cách mà chúng ta biểu diễn dữ liệu cho máy tính hiểu, còn cách tìm kiếm thì chúng ta sẽ dùng các thuật toán tìm kiếm, duyệt trên đồ thị.

<u>Biểu diễn ma trận:</u>

Ta có ma trận bản đồ A[M][N]. Trong đó A[i][j] = 0, nếu tọa độ (i,j) có thể đi được và ngược lại thì có chướng ngại vật.

Mỗi ô trong ma trận có thể đi sang được 4 ô có chung cạnh với nó và ô đang đến không phải là ô đang chứa chướng ngại vật.

Cây tìm kiếm:

Gốc của cây tìm kiếm chính là vị trí xuất phát của robot.

Hai đỉnh (u,v) có đường đi trực tiếp nếu hai đỉnh đó kề nhau và đều không chứa chướng ngại vật trong ma trận.

Trọng số trên mỗi đường đi trực tiếp đều bằng 1.

Như vậy cây tìm kiếm có một nút không có quá 4 nút con. Việc tìm đường đi trong ma trận sẽ đưa về bài toán tìm kiếm đường đi trên cây tìm kiếm.

II) Thuật toán.

1) Tìm đường đi ngắn nhất trong đồ thị.

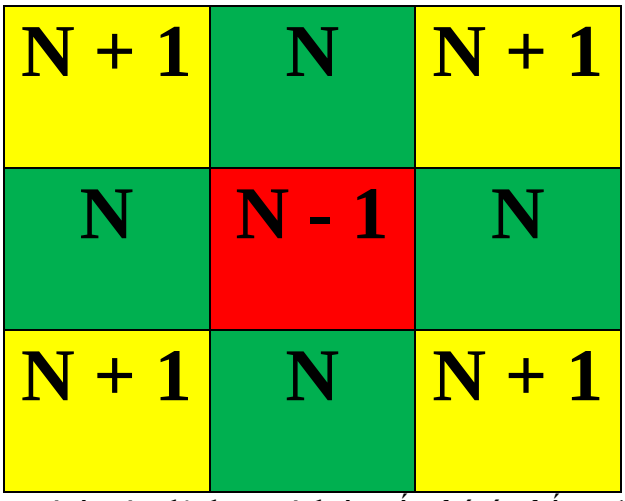
Nhắc đến việc tìm đường đi ngắn nhất trên đồ thị thì không ai chúng ta xa lạ nữa. Chúng ta đã làm quen vơi nhiều thuật toán để giải bài toán này kinh điển nhất là thuật toán Dijkstra nổi tiếng.

Nhưng trong môn Trí Tuệ Nhân Tạo chúng ta sẽ làm quen với 2 phương pháp hoàn toàn mới là: Thuật toán tìm kiếm loang trên đồ thị, thuật toán tìm kiếm đường đi ngắn nhất bằng Heuritic.

a) Thuật toán "loang" trên đồ thị.

Ý tưởng: Tại bước thứ N thì chúng ta sẽ tìm kiếm các đỉnh cách điểm đầu đúng N bước đi. Việc tìm kiếm thông qua các bước đi của bước thứ N-1. Tức là kiểm tra những đỉnh kề với đỉnh thứ N-1 rằng nếu đỉnh thứ N chưa được đi hay đi rồi nhưng số bước lớn hơn N thì xác lập cho đỉnh đó. Cứ tìm kiếm cho đến khi tìm đúng ô cần tìm thì dừng lại.

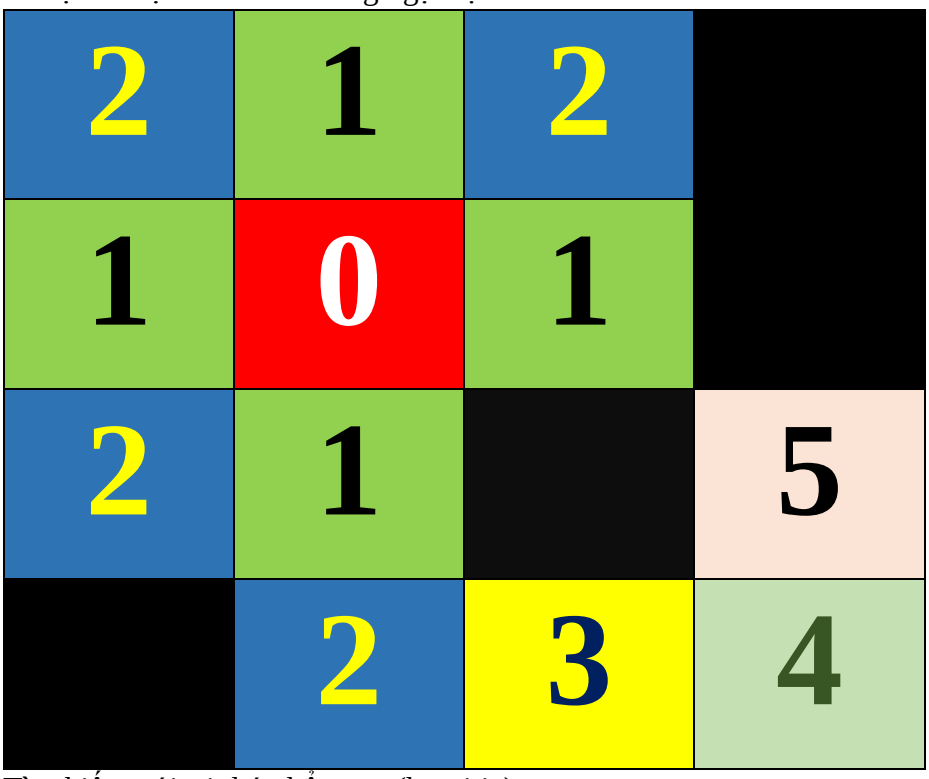
Ví dụ ma trận 3x3 như sau:



Từ ô màu đỏ đang cách ô xuất phát ít nhất N-1 bước đi. Sau đó ta sẽ xét các ô xung quanh ô đó(màu xanh) và nếu các ô đó chưa đi thì các ô đó cách ít nhất ô xuất phát đúng N bước đi. Tức là đi đến ô thứ N-1 mất N-1 bước đi và đi từ

ô N-1 đến ô thứ N mất 1 bước đi nữa vậy tổng sẽ mất N – 1 + 1 = N bước đi. Và tương tự cho các ô còn lại.

Ví dụ ma trận 4x4 có chướng ngại vật là ô màu đen.



b) Tìm kiếm với tri thức bổ sung.(heuritic)

Các chiến lược tìm kiếm cơ bản (uninformed search strategies) chỉ sử dụng các thông tin chứa trong định nghĩa của bài toán nên không phù hợp với nhiều bài toán thực tế(do đòi hỏi chi phí quá
cao về thời gian và bộ nhớ).

Các chiến lược tìm kiếm với tri thức bổ sung (informed search strategies) sử dụng các tri thức cụ thể của bài toán→Quá trình tìm kiếm hiệu quả hơn.

- Giải thuật Best-first search

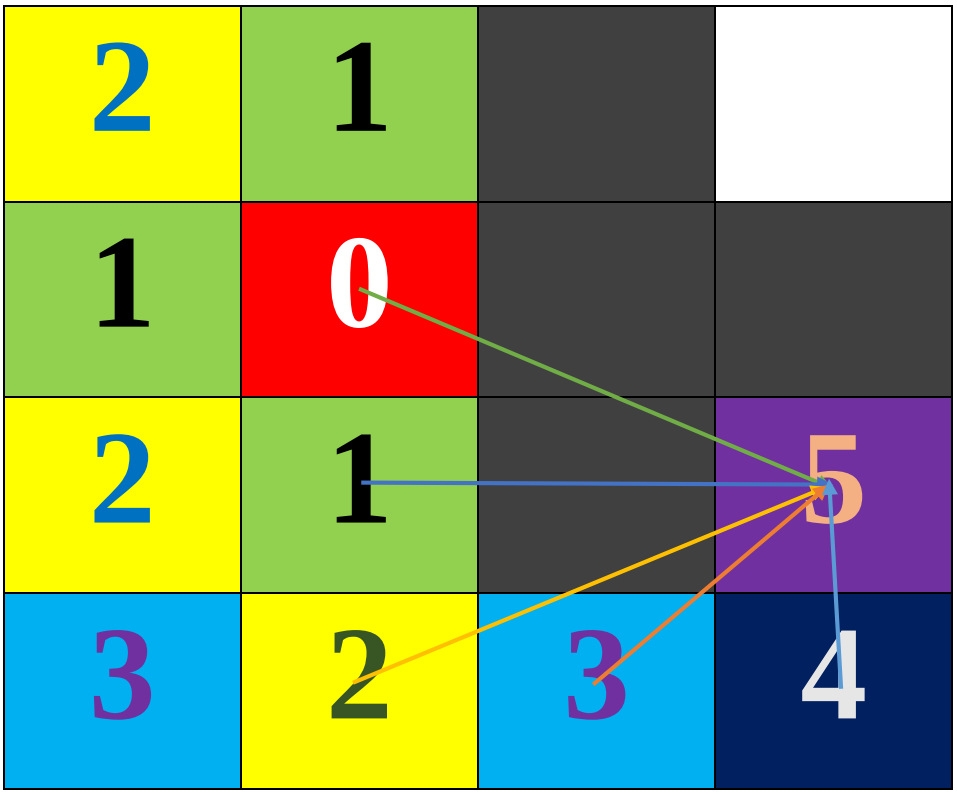
Ý tưởng: Sử dụng một hàm đánh giá f(n) cho mỗi nút của cây tìm kiếm. Để đánh giá mức độ phù hợp của từng đỉnh, trong quá trình tìm kiếm sẽ lựa chọn các nút có mức độ phù hợp cao nhất.

Hàm đánh giá f(n) là hàm heuristic h(n). Với hàm h(n) hàm heuristic h(n) đánh giá chi phí để đi từ nút hiện tại n đến nút đích (mục tiêu).

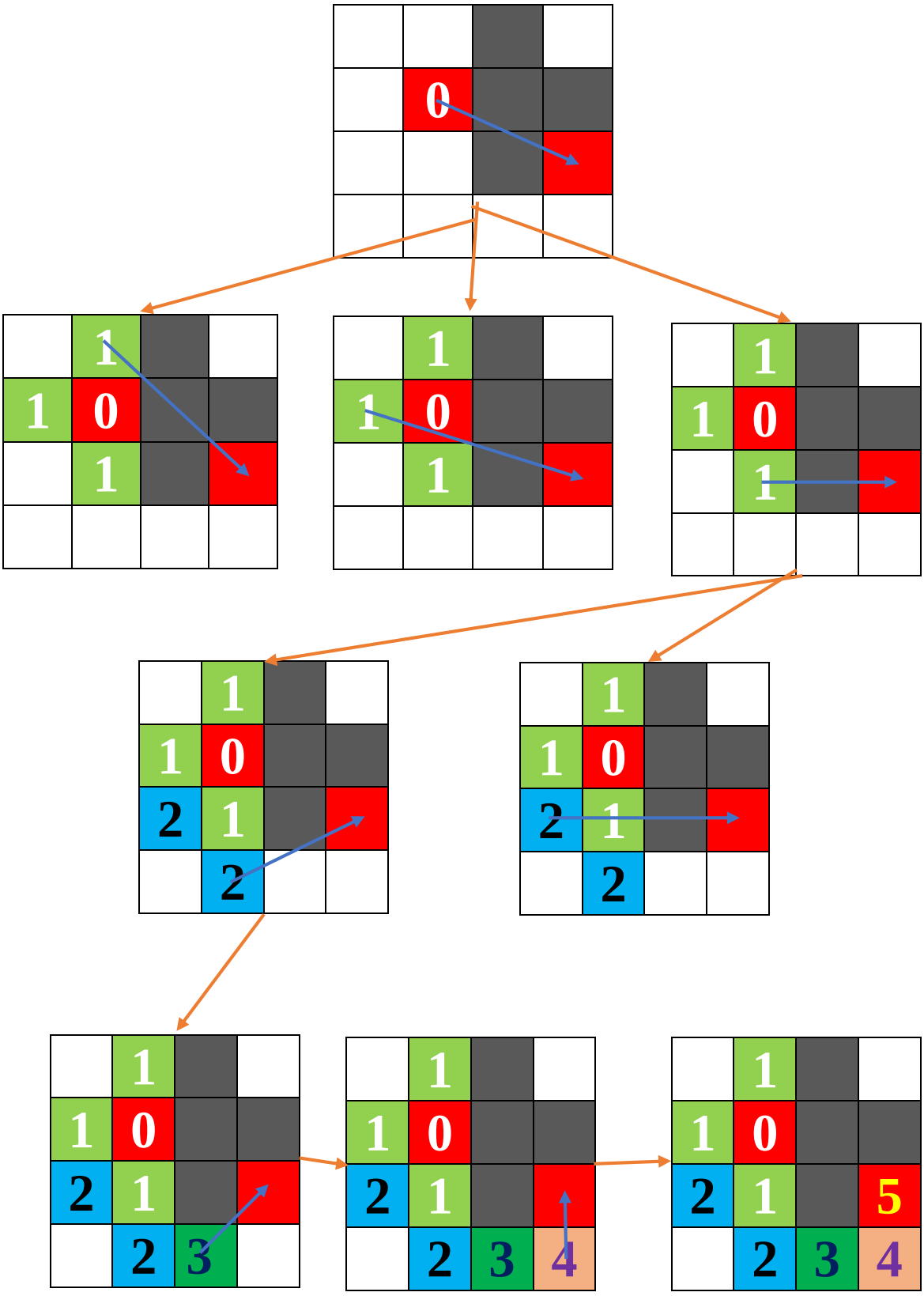
Trong bài toán này h(n) = Ước lượng khoảng cách đường thẳng(đường chim bay) đến nút hiện tại.

Khoảng cách từ điểm hiện tại (x1,y1) đến nút đích (x2,y2) là h(n) = (x2-x1)*(x2-x1) + (y2-y1)*(y2-y1).

Cụ thể trong hình.



Ví dụ: Chẳng hạn chúng ta có ma trận 4x4, những ô màu đen là những chướng ngại vật, 2 ô màu đỏ là điểm xuất phát và điểm kết thúc. Chúng ta cần tìm đường đi giữa 2 ô màu đỏ đó.


0
1
1 0
1
1
1 0
1
1
1 0
1
1
1 0
2 1
2
1
1 0
2 1
2
1
1 0
2 1
2 3
1
1 0
2 1
2 3 4
1
1 0
2 1
2 3 5

Thuật toán tìm kiếm có tri thức không có tính hoàn chỉnh, và không tìm thấy đáp án tối ưu. Để cải thiện cách tìm kiếm ta sử dụng một hàm h(n) tốt.

Thuật toán A*: Tránh việc xét các nhánh tìm kiếm đã xác định là có chi phí quá cao.

Sử dụng hàm đánh giá f(n) = h(n) + g(n).

Với h(n) là hàm heuristic ước lượng giá trị từ đỉnh N đến đích.

g(n) là giá trị thực đi từ đỉnh đầu đến đỉnh hiện tại.

f(n) chi phí ước lượng từ đỉnh đầu đến đỉnh cuối và đi qua nút hiện tại. Nhưng trong thuật toán này giá trị mỗi ô vuông đều bằng 1 nên sử dụng 2 thuật toán là như nhau.

Nhận xét giải thuật A*.

Nếu không gian các trạng thái là hữu hạnvà có giải pháp để tránh việc xét(lặp) lại các trạng thái thì giải pháp để tránh việc xét (lặp) lại các trạng thái, thì giải thuật A* là hoàn chỉnh (tìm được lời giải) – nhưng không đảm bảo là tối ưu.

Nếu không gian các trạng thái là hữu hạn và không có giải pháp để tránh việc xét (lặp) lại các trạng thái, thì giải thuật A* là không hoàn chỉnh (không đảm bảo tìm được lời giải).

Nếu không gian các trạng thái là vô hạn, thì giải thuật A*
là không hoàn chỉnh (không đảm bảo tìm được lời giải).

2) Tìm đường đi dài nhất trong đồ thị.

a) Tìm đường đi dài nhất trong đồ thị và không lặp lại.

Qua một thời gian tìm tòi nghiên cứu các thuật toán, tham khảo nhiều tài liệu nhiều nguồn thông tin mà không thể có một thuật toán tối ưu để giải bài toán này. Nếu không gian tìm kiếm bé thì chúng ta có thể sử dụng thuật toán quay lui, tham lam để giải quyết. Nhưng chúng ta phải áp dụng thuật toán để giải các bài toán lớn hơn rất nhiều, số đỉnh có thể lớn hơn 1000. Vì vậy để giải bài toán này thì chỉ còn cách sử dụng heuritic vào việc tìm kiếm, thời gian sử dụng cũng tương đối nhanh, kết quả chấp nhận được.

Để giải quyết bài toán cho đơn giản thì mình quy ước như sau điểm xuất phát là tọa độ (0,0) và kết thúc là điểm kết thúc là (size – 1, size – 1).

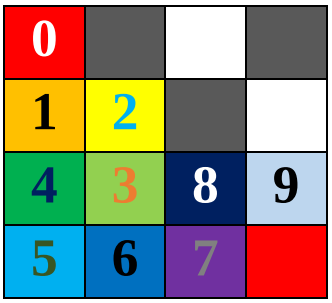
Ta suy nghĩ đơn giản như sau:

Ta đi càng gần điểm xuất phát, càng xa điểm kết thúc thì chắc chắn một điều rằng những đường đi như vậy sẽ đi được tối đa số bước đi sẽ đạt được.

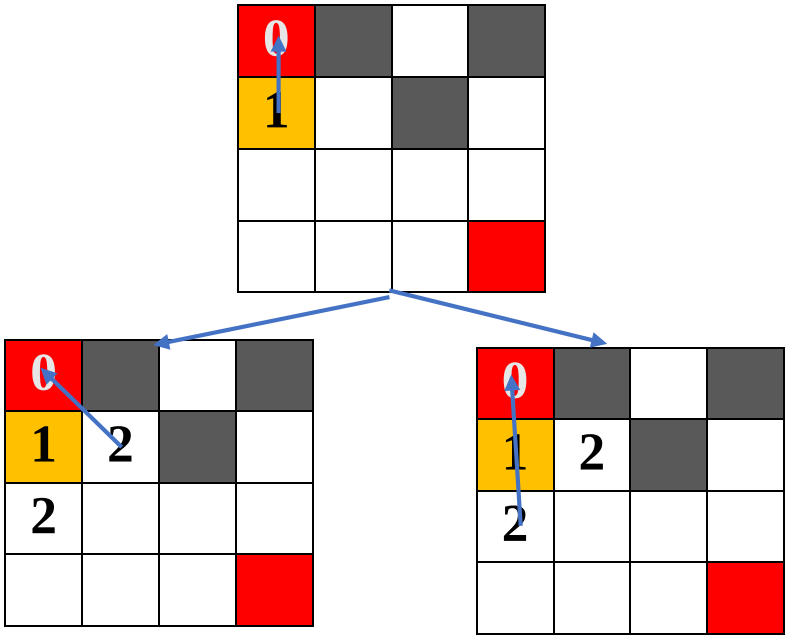
Hàm lượng giá như sau: h(n) =min(x*x + y*y). Tức là càng gần điểm xuất phát thì càng tốt.

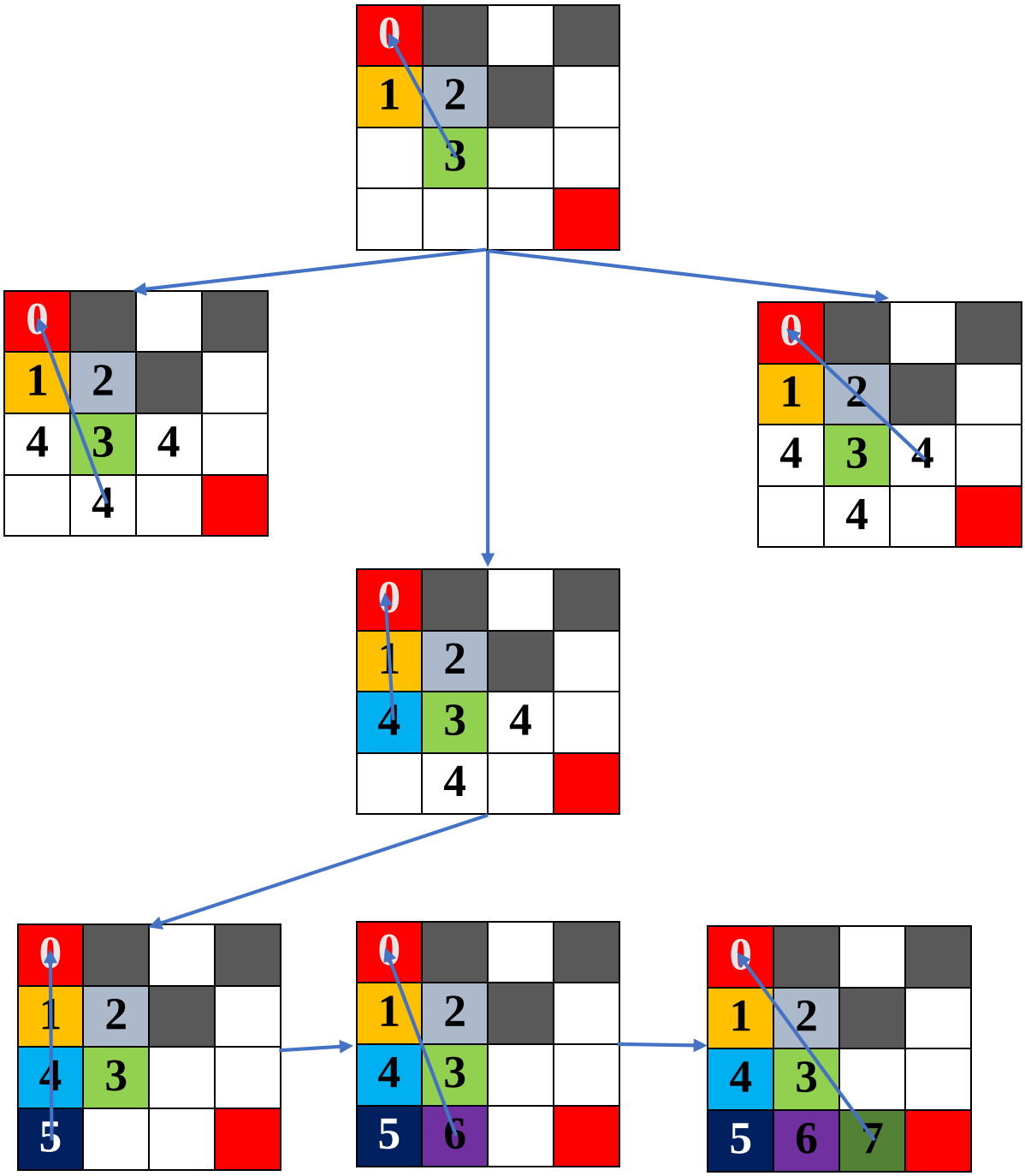
Chúng ta sẽ xét các điểm thành vòng tròn. Cứ theo bán kính từ trong ra ngoài, cho đến lúc gặp được đích thì dừng lại.

Tại mỗi nút đi chúng ta sẽ tìm nút đi tối ưu nhất để thực hiện. Cụ thể hơn qua ví dụ sau:



Các bước như sau:





b) Thuật toán tìm đường đi dài nhất trong đồ thị và có lặp lại với số lần ít nhất.

Giả sử ta đã có đường đi dài nhất không lặp lại, thì chắc chắn rằng đường đi dài nhất có lặp lại sẽ luôn chứa đường đi không lặp lại.

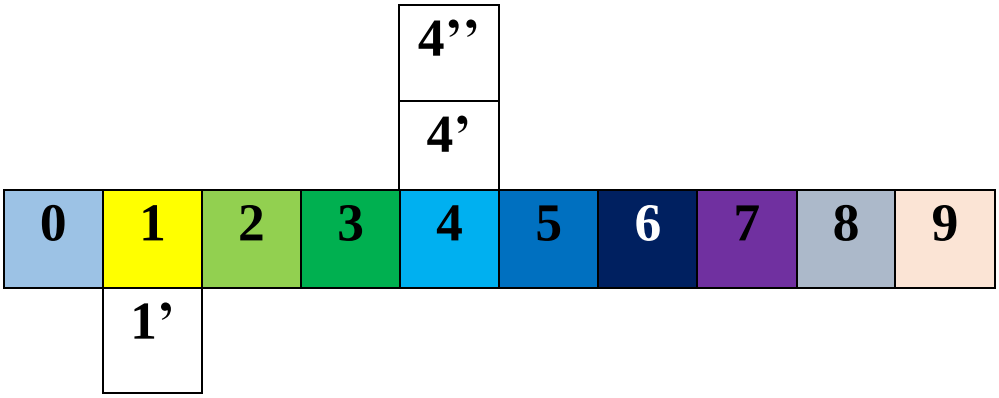
Giả thuyết rằng, đường đi dài nhất ta đã tìm ra là S (không tồn tại S’ sao cho S’>S).

Đường đi dài nhất có lặp lại là G và S thuộc G, và F = G/S, F là những phần tử thuộc G và không thuộc S.

Ta cần chứng minh rằng F là một nhánh trực thuộc S, nghĩa là S và F liên thông với nhau.

Chứng minh:

Đúng vậy, nếu như G, S tồn tại, mọi phần tử trong G liên thông với nhau. Nếu S, F không liên thông với nhau => Trong G được chia ít nhất thành 2 miền không gian liên thông, trái với giả thuyết là G liên thông. Suy ra S,F liên thông, và F sẽ thuộc một nhánh của S.



Để tìm kiếm đường đi dài nhất có lặp lại, chúng ta sử dụng hoàn toàn thuật toán tìm đường đi dài nhất không lặp lại và bổ sung thêm các bước đi đến đường cụt chính là F. Tức là chúng ta bổ sung F vào S suy ra được G.

c) Đường đi có vật cản di chuyển.

Bài toán được hình dung như sau: Trong quá trình di chuyển với bản đồ biết trước chúng ta thêm vào bản đồ những chướng ngại vật và lập tức Robot phải nhận thức đó là chướng ngại và xử lý để tìm ra đường đi tối ưu nhất phù hợp với từng hoàn cảnh.

Để xử lý bài toán không cố định thì chúng ta phải lưu giá trị đã đi từ trước và xem điểm vừa đi qua là vật cản cố định. Đưa vào những vật cản di chuyển xem như là vật cản cố định. Cập nhật bản đồ mới tiếp tục áp dụng các thuật toán ở trên với bản đồ vừa cập nhật xong.

III) Cài đặt thuật toán.

1) Thuật toán tìm đường đi ngắn nhất trong đồ thị.

a) Biểu diễn dữ liệu.

Trong bản đồ tìm kiếm chúng ta lưu trữ chúng dưới dạng ma trận, mỗi ô có một tạo độ riêng, tương ứng với một điểm trên bản đồ.

Định nghĩa như sau:

Hx[i], Hy[i] tương ứng là độ dịch chuyển so với vị trí hiện tại. Trong này do robot chỉ di chuyển được 4 hướng nên mảng chỉ có 4 phần tử. Và các phần tử tương ứng của từng mảng là:

Hx[]={0,1,0,-1};
Hy[]={1,0,-1,0};

Các phần tử trong ma trận sẽ tương ứng với một điểm trên bản đồ trên thực địa. Nếu như điểm đó có thể di chuyển qua thì A[i][j] = 0, ngược lại nếu như A[i][j] = 1 thì có vật cản tại đó và không thể di chuyển.

Việc mình tìm đường đi chính là tìm dãy các phần tử kề nhau, mỗi phần tử đều = 0. Điểm cuối của đường đi chính là tạo độ mà chúng ta cần đến.

b) Các bước cài đặt và giao diện người dùng.

B1: Cho người dùng nhập dữ liệu vào;

Gồm nhập chướng ngại vật, nhập tạo độ điểm đầu, tọa độ điểm cuối. Do hỗ trợ giao diện cho người dùng thay vì việc nhập bình thường thì chúng ta chỉ cần Click vào màn hình thì sẽ xuất hiện lên đó.

B2: Xử lý thuật toán. Tìm kiếm các đường đi có thể.

B3: Đưa ra thông báo, có tồn tại đường đi hay không, nếu có thì đi như thế nào.

c) Xử lý thuật toán.

- Tìm kiếm đường đi ngắn nhất sử dụng thuật toán "loang".

  Ban đầu khởi tạo giá trị cho tất cả các phần tử của mảng B = 1000(vô cùng). Mảng B[i][j] là khoảng cách ngắn nhất từ điểm đầu đến điểm có tọa độ [i][j].

  Giá trị khởi tạo B[tạo độ x ban đầu][tạo độ y ban đầu] = 0, vì nó đi đến chính nó thì cần số bước đi ít nhất là 0.

  Sau đó loang ra từ đỉnh ban đầu ra các đỉnh lân cận và xác lập số đường đi ngắn nhất cho các đỉnh đó, và cũng tiếp tục như thế cho các đỉnh lân cận còn lại.

```
public void dijkstra_1() throws InterruptedException{
        int dem = 0, d = 0;
        int [][] dd = new int[600][3];

        Thread th = new Thread();

        Robot_Tim_duong_stack = new Stack();
        int x1,y1,x2,y2;
        for(int i=0;i<24;i++)
           for(int j=0;j<24;j++)
           {
              b[i][j] = 1000;
```

```
    }
x1 = Robot_Tim_duong_x1;
y1 = Robot_Tim_duong_y1;
a[x1][y1] = 1;
b[x1][y1] = 0;
Robot_Tim_duong_stack.push(x1,y1);
while(Robot_Tim_duong_stack.check(dem)){
    x1 = Robot_Tim_duong_stack.a[0][dem];
    y1 = Robot_Tim_duong_stack.a[1][dem];
    if(b[Robot_Tim_duong_x2][Robot_Tim_duong_y2]!=1000)
        break;
    for(int i=0;i<4;i++){
        x2 = x1 + hx[i];
        y2 = y1 + hy[i];
        if(check(x2,y2) && (b[x1][y1]+1<b[x2][y2]) && a[x2][y2]!=1)
        {
            delta[x2][y2] = i;
            Robot_Tim_duong_stack.push(x2, y2);
            b[x2][y2] = b[x1][y1]+1;
        }
    }
    dem++;
}

if(b[Robot_Tim_duong_x2][Robot_Tim_duong_y2] != 1000)
{
    dem = b[Robot_Tim_duong_x2][Robot_Tim_duong_y2];
    x1 = Robot_Tim_duong_x2;
    y1 = Robot_Tim_duong_y2;
    dd[dem][0] = x1;
    dd[dem][1] = y1;
    for(int i=0;i<dem;i++){
        x2 = hx[delta[x1][y1]];
        y2 = hy[delta[x1][y1]];
        x1 = x1 - x2;
        y1 = y1 - y2;
        dd[dem-i-1][0] = x1;
        dd[dem-i-1][1] = y1;
    }
    dd[0][0] = Robot_Tim_duong_x1;
    dd[0][1] = Robot_Tim_duong_y1;
    th.start();
    for(int i=1;i<dem;i++)
```

```
            {
                u[dd[i][0]][dd[i][1]].setO();
            }
            th.stop();
        }

        else
            System.out.print("khong co duong di =
"+b[Robot_Tim_duong_x2][Robot_Tim_duong_y2]);
    }
```

- Thuật toán Heuritic trong việc tìm kiếm đường đi nhỏ nhất.

  Với ý tưởng như trên ta sử dụng thêm thuật toán quay lui để tìm trạng thái tốt nhất. Trong hàm đệ quy chúng sử dụng hàm heuristic kết hợp với kỹ thuật quay lui, đánh dấu các bước đã đi qua và không lặp lại các bước đã đi qua.

  Đây là thuật toán A*. Khởi tạo các giá trị và gọi hàm đệ quy.

```
public void A(){
    int x1, y1, gt, x2, y2, min, vt;
    kiemtra = false;
    giaithuatA[0][0] = Robot_Tim_duong_x1;
    giaithuatA[0][1] = Robot_Tim_duong_y1;
    a[Robot_Tim_duong_x1][Robot_Tim_duong_y1] = 1;
    for(int i=0;i<24;i++)
       for(int j=0;j<24;j++)
       {
          b[i][j] = a[i][j];
       }
    back_tracking(Robot_Tim_duong_x1, Robot_Tim_duong_y1, 1);
  }
```

  Đây là hàm đệ quy, sử dụng đánh dấu các bước đã đi qua lần trước đó.

```
public void back_tracking(int x,int y, int k){
     int x1, y1, gt, x2, y2, vt = -1, min = 10000;
     if(kiemtra==false)
     {
        x1 = x;
        y1 = y;
        if(x1==Robot_Tim_duong_x2 && y1 == Robot_Tim_duong_y2){
          output(k);
        }
        else
        {
        x1 = x;
        y1 = y;
```

```
            for(int i=0;i<4;i++){
               x2 = x1 + hx[i];
               y2 = y1 + hy[i];
               if(check(x2, y2) && b[x2][y2]!=1){
                  if((x2 - Robot_Tim_duong_x2)*(x2 -
Robot_Tim_duong_x2)+(y2 - Robot_Tim_duong_y2)*(y2 -
Robot_Tim_duong_y2)<min)
                  {
                     min = (x2 - Robot_Tim_duong_x2)*(x2 -
Robot_Tim_duong_x2)+(y2 - Robot_Tim_duong_y2)*(y2 -
Robot_Tim_duong_y2);
                     vt = i;
                  }
               }
            }
            if(vt!=-1){
               giaithuatA[k][0] = x1+hx[vt];
               giaithuatA[k][1] = y1+hy[vt];
               b[x1+hx[vt]][y1+hy[vt]] = 1;
               back_tracking(x1+hx[vt], y1+hy[vt], k+1);//Đệ quy
               back_tracking(x1, y1, k);
               giaithuatA[k][0] = 0;
               giaithuatA[k][1] = 0;
               b[x1+hx[vt]][y1+hy[vt]] = 0;
            }
         }
      }
   }
```

Và hàm in dữ liệu ra màn hình.

```
public void output(int k){
      kiemtra = true;
      for(int i=1;i<k-1;i++)
      {
         u[giaithuatA[i][0]][giaithuatA[i][1]].setO();
      }
   }
```

2) Tìm đường đi dài nhất trong đồ thị.

a) Không lặp lại.

Ta sử dụng mảng B[M][N] để đánh dấu các ô đã đi qua, tránh việc lặp lại không cần thiết.

Sử dụng phép tính khoảng cách đường chim bay đến vị trí xuất phát (0,0) ta sẽ tìm ra được đường đi tối ưu tại đó, và đánh dấu đỉnh đó đã đi qua.

Công thức heuristic được áp dụng như sau: h(n) = x*x+y*y;
F(n) = min(h(n)); càng gần vị trí xuất phát thì càng tốt.

Trong thuật toán có sử dụng quay lui để tránh các bước đi vào "ngõ cụt", và tìm đường để đến đích.

Việc cài đặt vào chương trình như sau:

```
public void A1(){
    int x1, y1, gt, x2, y2, min, vt;
    kiemtra = false;
    giaithuatA[0][0] = robot_lau_nha_x1;
    giaithuatA[0][1] = robot_lau_nha_y1;
    a[robot_lau_nha_x1][robot_lau_nha_y1] = 1;
    for(int i=0;i<24;i++)
       for(int j=0;j<24;j++)
       {
          b[i][j] = a[i][j];
       }
    back_tracking1(robot_lau_nha_x1, robot_lau_nha_y1, 1);
  }

  public void output1(int k){
     kiemtra = true;
     for(int i=1;i<k-1;i++)
     {
        u[giaithuatA[i][0]][giaithuatA[i][1]].setO();
     }
  }

  public void back_tracking1(int x,int y, int k){
     int x1, y1, gt, x2, y2, vt = -1, min = 10000;
     //System.out.print("de quy\n");
     if(kiemtra){

     }
     else
     {
        x1 = x;
        y1 = y;
        if(x1==robot_lau_nha_x2 && y1 == robot_lau_nha_y2){
          output1(k);
        }
        else
        {
        x1 = x;
```

```
            y1 = y;
            for(int i=0;i<4;i++){
               x2 = x1 + hx[i];
               y2 = y1 + hy[i];
               if(check(x2, y2) && b[x2][y2]!=1){
                  if((x2)*(x2)+(y2)*(y2)<min)
                  {
                     min = (x2)*(x2)+(y2)*(y2);
                     vt = i;
                  }
               }
             }
            if(vt!=-1){
               giaithuatA[k][0] = x1+hx[vt];
               giaithuatA[k][1] = y1+hy[vt];
               b[x1+hx[vt]][y1+hy[vt]] = 1;
               back_tracking1(x1+hx[vt], y1+hy[vt], k+1);
               back_tracking1(x1, y1, k);
               giaithuatA[k][0] = 0;
               giaithuatA[k][1] = 0;
               b[x1+hx[vt]][y1+hy[vt]] = 0;
            }
          }
        }
    }
```

b) Đường đi dài nhất có lặp lại.

Thuật toán cơ bản giống bài toán đường đi không lặp lại, nhưng do chúng ta bổ sung việc đi sâu vào những "ngõ cụt". Nên trong chương trình có thêm hàm DFS để tìm kiếm những ô lân cận có thể đi.

```
public void vetcan(int x,int y){
      duong_di_satck.push(x,y);
      int x1,y1, x2, y2;
      x1 = x;
      y1 = y;
      for(int i=0;i<4;i++){
         x2 = x1+hx[i];
         y2 = y1+hy[i];
         if(check(x2, y2) && b[x2][y2]==0){
            b[x2][y2] = 1;
            duong_di_satck.push(x2,y2);
            vetcan(x2, y2);
            duong_di_satck.push(x2, y2);
         }
```

```
        }
    }
```

Hàm này vừa tìm kiếm nước đi và đánh dấu để tránh lặp lại.

c) Đường đi có vật cản di chuyển.

Bài toán dựa hoàn toàn trên 2 thuật toán tìm đường đi dài nhất không lặp lại, và không lặp lại. Để thực hiện việc dừng chương trình và nhập dữ liệu người dùng phải click Sleep->pause, sau đó click vào từng ô còn trống ô đó lập tức biến thành chướng ngại vật. Sauk hi hoàn thành việc nhập dữ liệu người dùng phải nhấn Sleep -> play để tiếp tục đường đi. Ngay sau đó chúng ta sẽ cập nhật bản đồ mới vào robot và tiến hành xử lý dữ liệu và đưa ra kết quả tốt nhất.

IV) Mô tả sản phẩm.

1) Tìm đường đi ngắn nhất trong đồ thị.

a) Dùng thuật toán "loang".

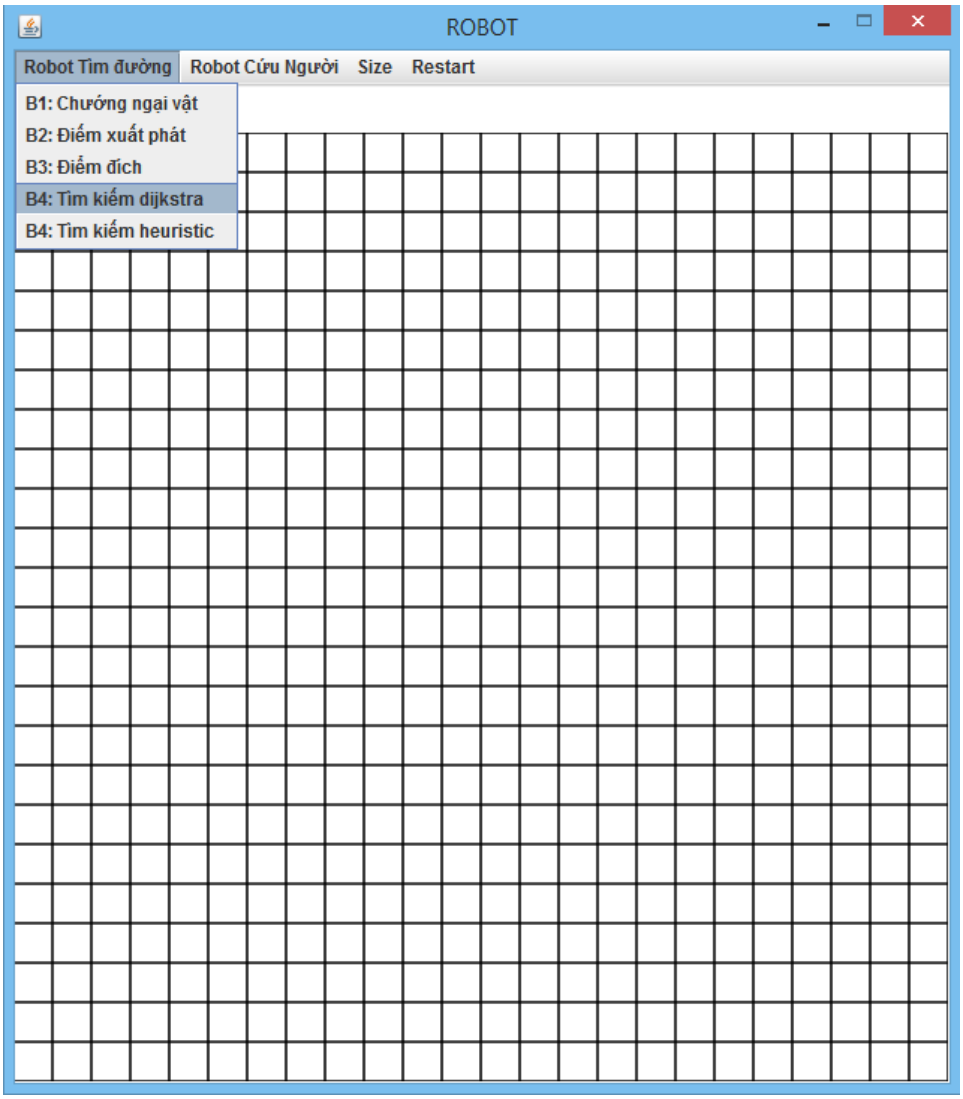
Giao diện người dùng gồm:

B1: Nhập chướng ngại vật, chỉ việc click chuột vào màn hình thì sẽ hiện chướng ngại vật tương ứng.

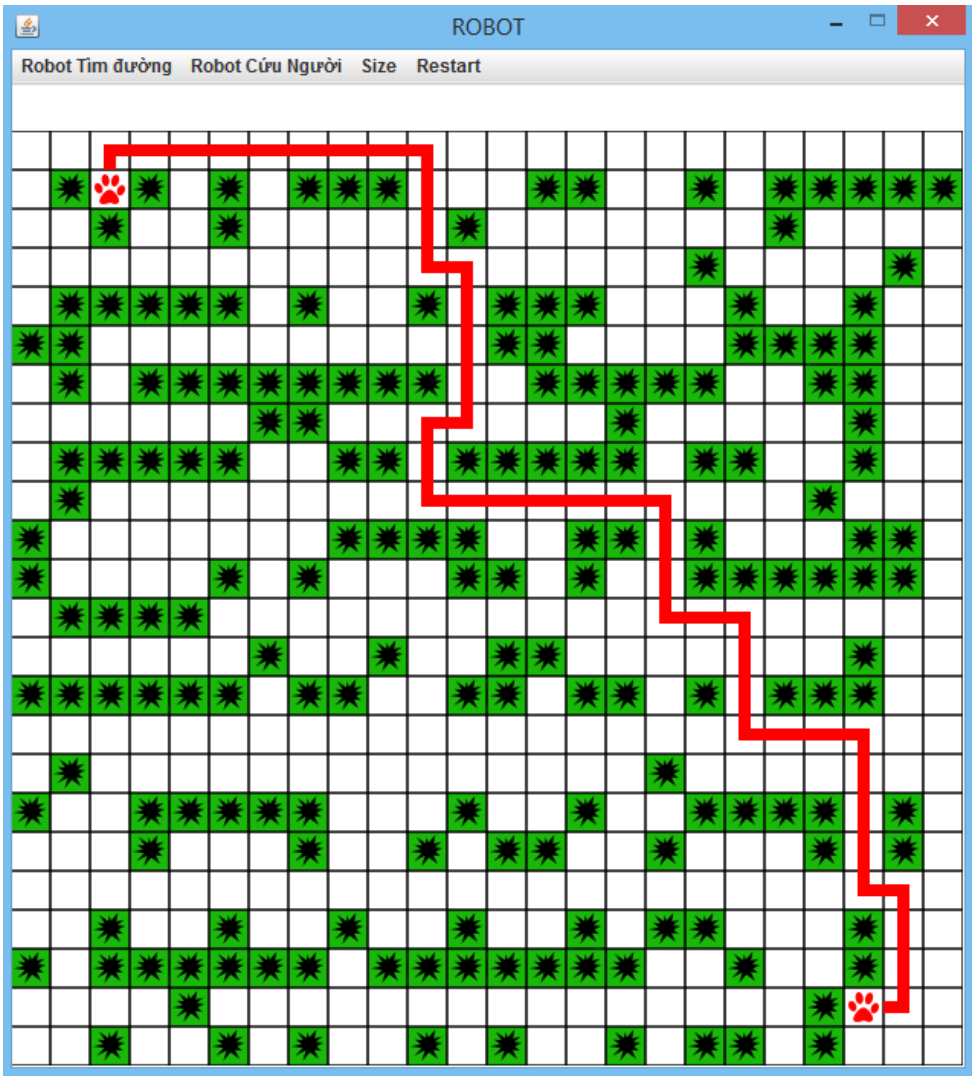
B2: Điểm xuất phát: Nhập điểm xuất phát của robot.

B3: Điểm đích: Nhập điểm kết thúc của robot.

B4: Lựa chọn thuật toán để robot thực hiện việc tìm kiếm đường đi. Hiển thị ra màn hình những ô vuông mà robot cần đi qua.

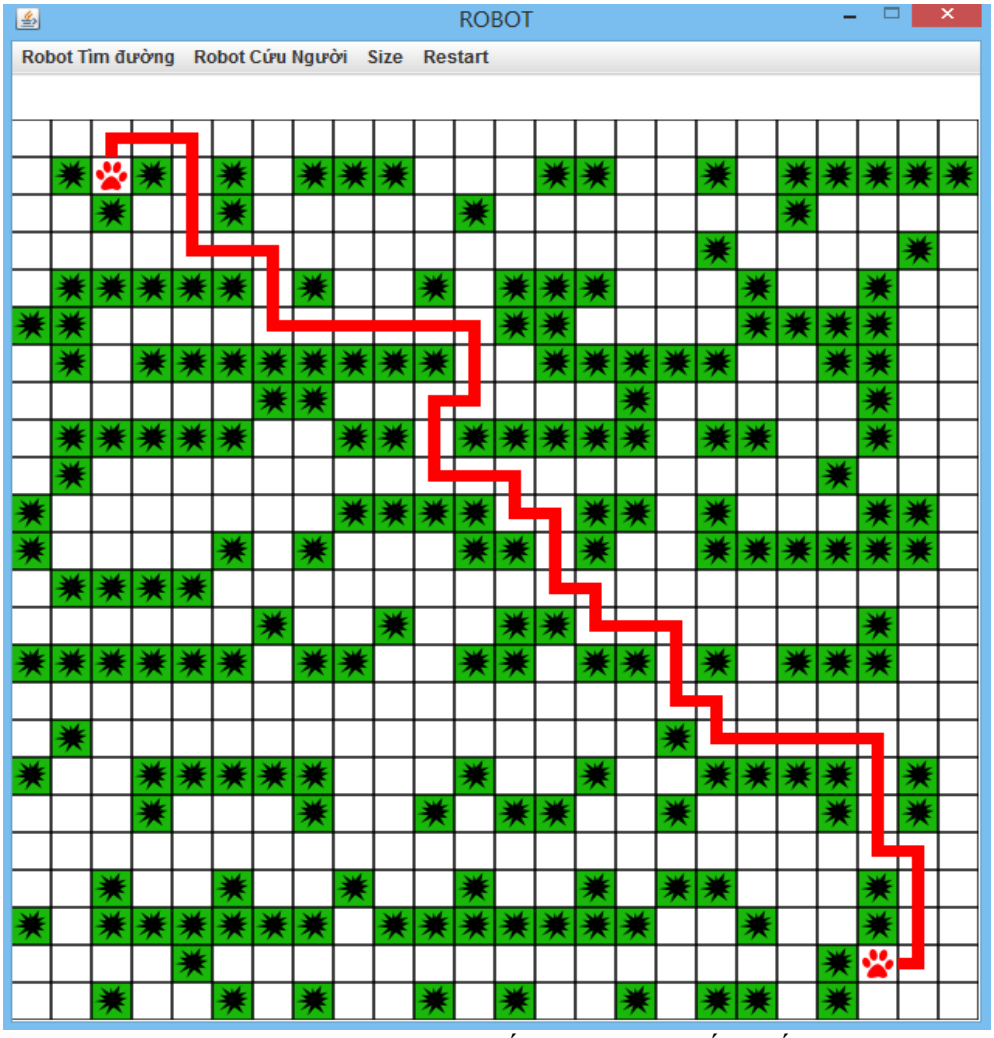


H6: Giao diện chính của sản phẩm, cho phép người dùng thao tác rất đơn giản.



H7: Trên đây là hình ảnh của chướng ngại vật(màu đỏ), điểm xuất phát(màu tím), điểm kết thúc(màu vàng). Và đường đi từ điểm đầu đến điểm cuối(màu xanh).

b) Thuật toán heuristic.



H8: Sử dụng heuristic trong việc tìm kiếm đường đi ngắn nhất.

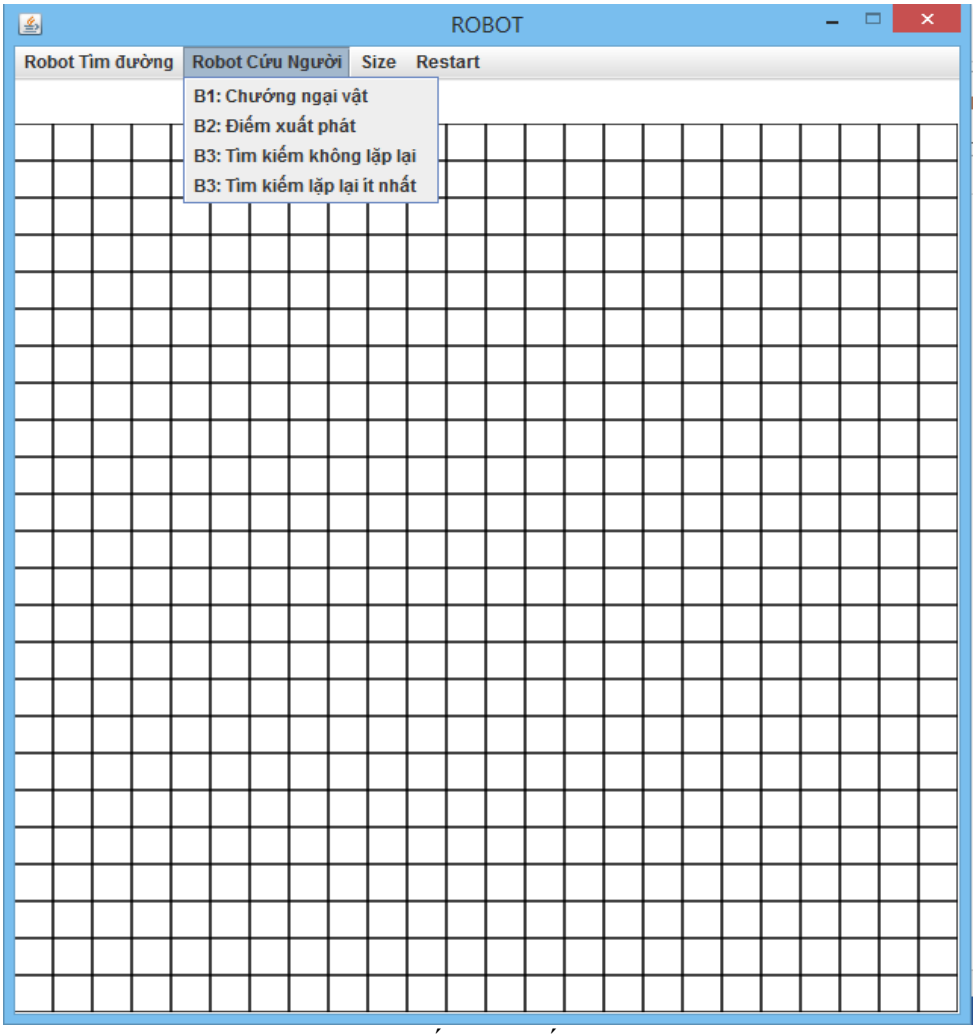
Nhận xét về việc sử dụng 2 thuật toán vào việc tìm đường đi ngắn nhất.

Thuật toán "loang" cho chúng ta kết quả chính xác nhất, thời gian tối ưu.

Thuật toán Heuritic cho kết quả không tối ưu, nhưng có thể chấp nhận được, thời gian thực hiện chương trình rất nhanh. Đặc biệt dựa vào 2 hình ảnh trên là kết quả của 2 thuật toán, chúng ta thấy rằng thuật toán heuristic luôn cố gắng đi thẳng đến đích cũng giống như ý tưởng của hàm lượng giá vậy.

2) Cài đặt thuật toán tìm đường đi dài nhất.

a) Không lặp lại đường đi.



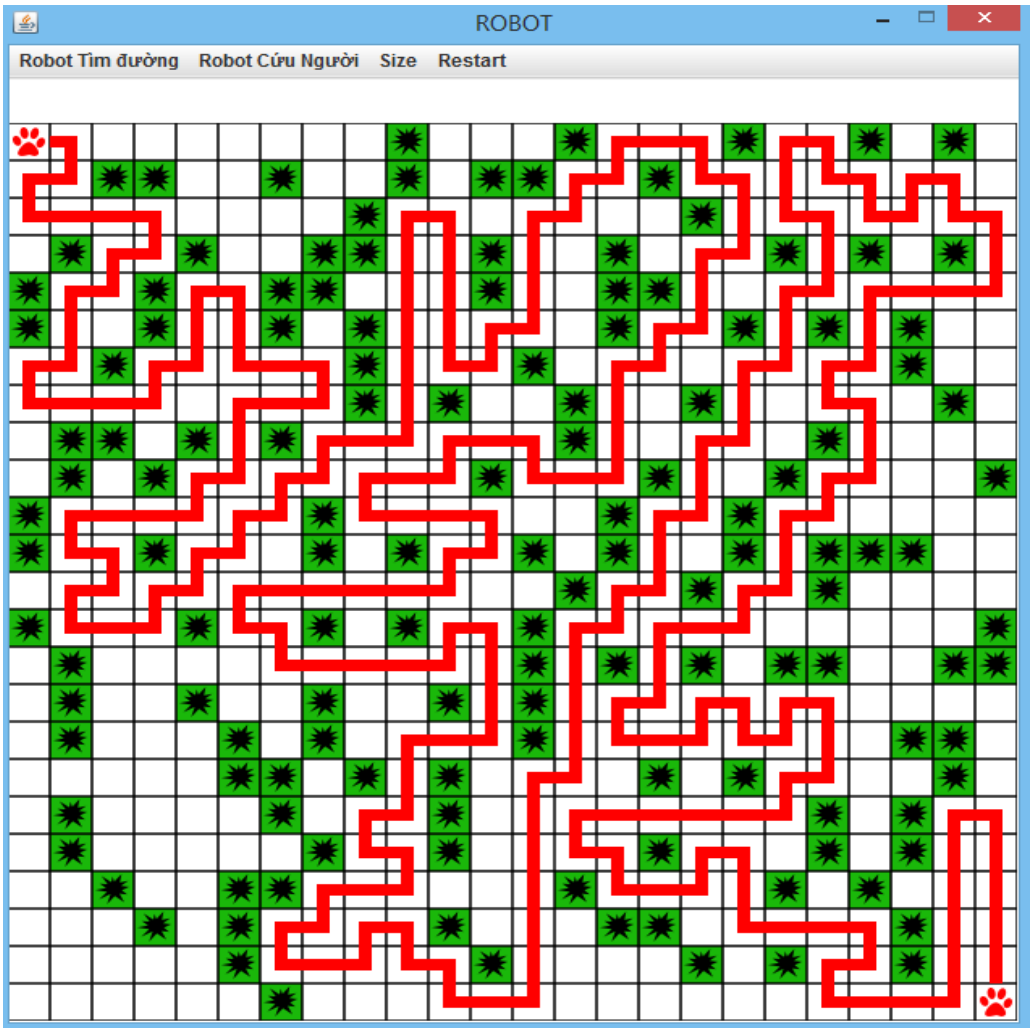
H9: Giao diện khi người dùng muốn tìm kiếm.

Các bước để có thể thao tác là:

B1: Nhập dữ liệu là các chướng ngại vật.

B2: Nhập điểm đầu và điểm cuối(Trong chương trình chúng ta đã mặc định cho điểm đầu là (0,0) và điểm cuối là (size-1, size-1).

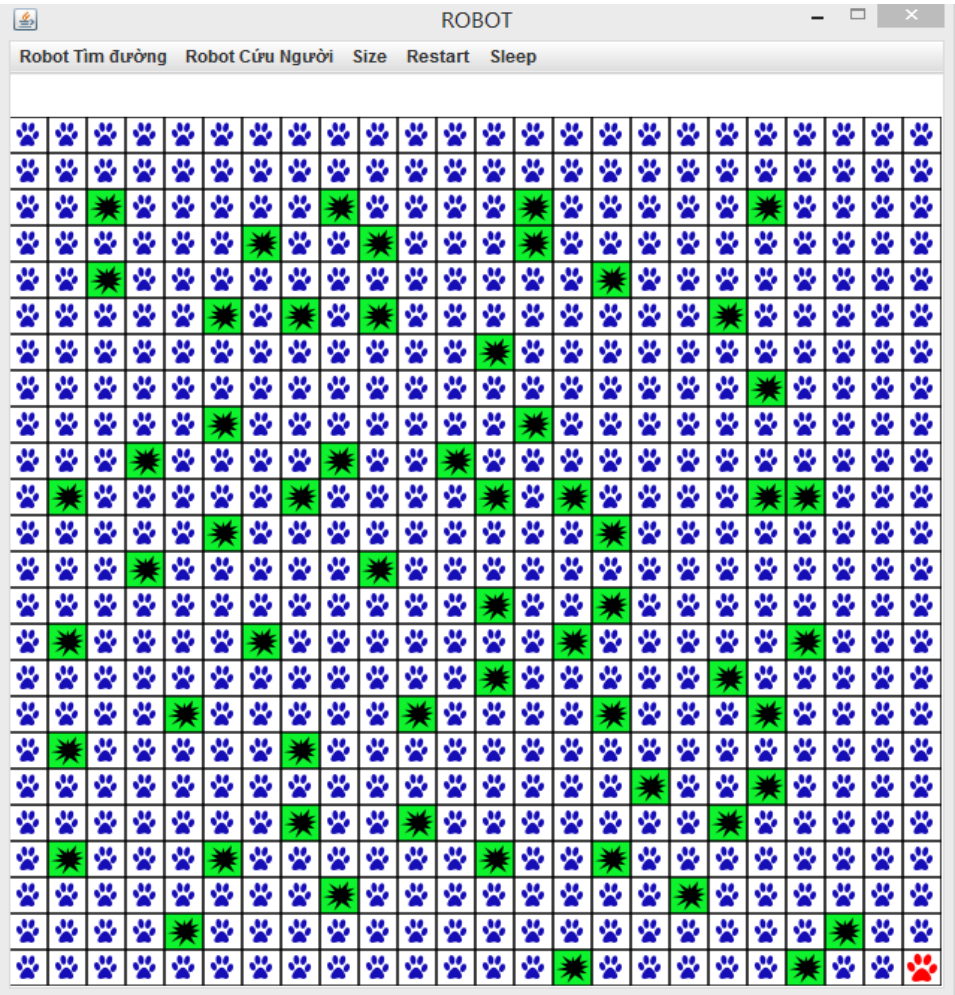
B3: Chọn thuật toán cho tìm kiếm.



H10: Đường đi trong đồ thị khi hoàn thành việc tìm kiếm.

b) Đường đi dài nhất có lặp lại.

Giống như bài toán không lặp lại, ngoài việc tìm kiếm đường đi dài nhất không lặp lại, ta phải tìm thêm thành phần liên thông với chính nó nữa.



H11: Đường đi dài nhất.

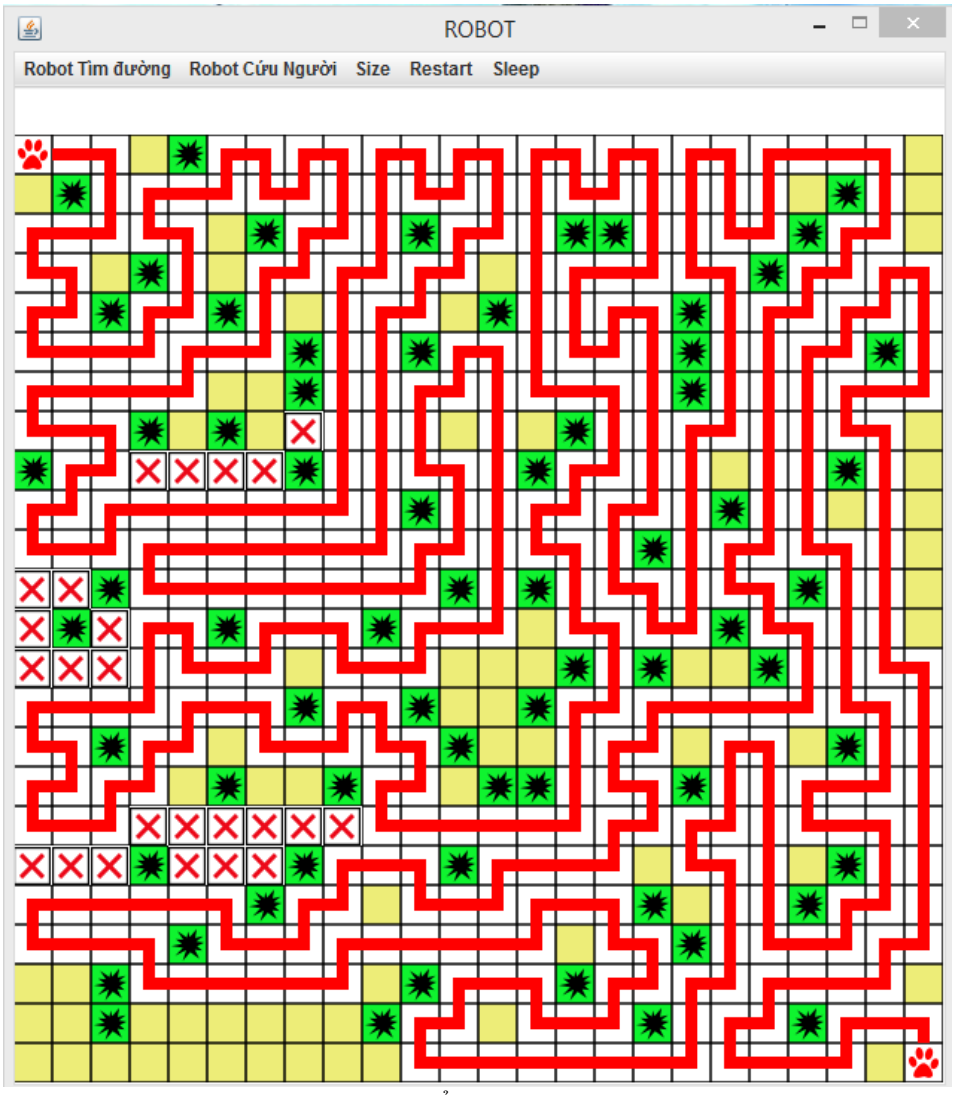
c) Đường đi có vật cản cố định.

Những ô màu xanh là vật cản cố định và có từ đầu.

Những gạch X màu đỏ là chướng ngại vật di chuyển vừa được nhập khi chương trình hoạt động.

Đương thẳng màu đỏ là đương đi tốt nhất mà Robot tìm được.

Ô màu vàng là ô trống, không có chướng ngại vật.



H12: Đường đi có vật cản di chuyển.

V) Nhận xét.

Sau đây là kết quả khi chạy một số bộ Test.

1) Đường đi dài nhất không lặp lại.

STT	Số ô tối đa có thể đi được	Số ô vuông đã đi được	Phần trăm(%)	Số ô chướng ngại vật
1	576	553	96	0
2	562	541	96,26	10
3	536	523	97,57	20
4	505	527	95,83	30
5	512	497	97,07	42
6	471	486	96,92	50
7	435	463	93,95	60
8	363	333	91,74	100
KQ			95,67	

2) Đường đi ngắn nhất trong đồ thị.

STT	Đường đi ngắn nhất	Thuật toán Loang	Phần trăm(%)	Thuật toán Heuristic	Phần trăm(%)	Số chướng ngại vật.
1	18	18	100	18	100	10
2	32	32	100	32	100	30
3	22	22	100	26	84,62	60
4	78	78	100	115	67,83	90
5	108	108	100	196	55,10	120
6	117	117	100	135	86,67	150
KQ			100%		82,37%	

3) Nhận xét

Trong các thuật trên chúng em có sử dụng thuật toán có độ chính xác tuyệt đối, kèm theo đó là những thuật có bổ sung tri thức nhằm tăng tốc độ tìm kiếm, nhưng không đảm bảo là tối ưu.

Thuật toán heuristic là thuật toán có bổ sung các hàm đánh giá tại các nút xét duyệt, và đi tiếp chứ không quay lại bước trước đó nữa, đây cũng chính là làm sai lệch đáp án.

Nhưng trong bài toán này chúng em đã cố gắng tối ưu hàm lượng giá nhằm đảm bảo kết quả tìm ra có thể chấp nhận được. Ước tính thuật toán đúng trên 80%.

Đôi khi thời gian xử lý thuật toán là quan trọng hơn là một kết quả chính xác, thay vì vậy một lời giải ngắn gọn cho chúng ta kết quả ngay tức thời, và có thể chấp nhận được.

Chúng em đã áp dụng toàn bộ những kiến thức môn "trí tuệ nhân tạo" vào việc tìm kiếm ra những thuật toán, và viết chương trình.

Đôi lúc trong một số bộ Test chương trình có bị treo vì sự sai sót nhỏ, mong thầy bỏ qua.

Trong quá trình hình thành thuật toán, và biến thành một chương trình còn có nhiều sai sót mong thầy đóng góp ý kiến để chúng hoàn thiện hơn.

Chúng em xin chân thành cảm ơn.

VI) Tài liệu tham khảo.

Bài giảng "Trí tuệ nhân tạo" của thầy Phạm Văn Hải.

"Cẩm nang Thuật Toán" của Robert Sedgewich.