PHÂN HIỆU TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TẠI TP.HCM

BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

MÔN: TRÍ TUỆ NHÂN TẠO

ĐỀ TÀI: BÀI TOÁN GHÉP TRANH SỬ DỤNG THUẬT TOÁN A*

SVTH: Ngô Cao Kỳ

MSSV: 5551074024

LỚP: CNTT K55

GVHD: Nguyễn Thị Hải Bình

Tp. Hồ Chí Minh, 9/2017

MỤC LỤC

Lời nói đầu.

Đây là tài liệu dung để biểu diễn cơ bản thiết kế và giải quyết bài toán “trò chơi ghép tránh” sử dụng thuật toán A*. Tài liệu này giúp ta có cái nhìn toàn vẹn về ứng dụng thuật toán A* . Do thời gian có hạn nên bài làm không thể tối ưu được toàn bộ không gian trạng thái bài toán.

Thực hiện đề tài nhắm mục đích xây dựng một hệ thống giải quyết một bài toán thực tế dựa trên chiến lược tìm kiếm heuristic và không xây dung một trò chơi ứng dụng giải trí. Trong qua trình thực hiện đề tài không tránh khỏi những sai sót, monng sẽ nhận được sự góp ý và đánh giá của giáo viên hướng dẫn.

PHẦN I. GIỚI THIỆU BÀI TOÁN

Game ghép tranh(N-Puzzle) là một trò chơi khá hay và trí tuệ, nó được biết đến với nhiều phiên bản và tên gọi khác nhau như: 8-puzzle, 15-puzzle, Gem puzzle, Boss puzzle.

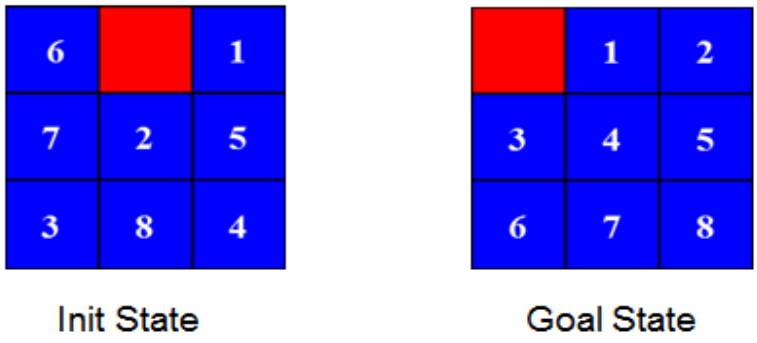
Bài toán N-puzzle là vấn đề cổ điển cho mô hình thuật toán liên quan đến trí tuệ nhân tạo. Bài toán đặt ra là phải tìm đường đi từ trạng thái hiện tại tới trạng thái đích. Và cho tới nay vẫn chưa có thuật toán tối ưu để giải bài toán này. Phần mềm N-Puzzle là một chương trình xây dựng trò chơi và giải quyết bài toán này. Phần mềm được viết trên nền Java, sử dụng giao diện đồ họa để mô phỏng trò chơi và thuật toán A* để tìm đường đi.

Người dùng có thể sử dụng chuột/bàn phím chơi với các kích thước khác nhau và với hình ảnh khác nhau hoặc có thể sử dụng chức năng tìm lời giải nhờ thuật toán A*. Yêu cầu xây dựng bảng ô vuông n hàng, n cột.

Bảng gồm 1 ô trống và n2 -1 ô chứa các số trong phạm vi [1, n2 -1]. Xuất phát từ một cách xếp bất kì, di chuyển ô trống lên trên, xuống dưới, sang phải, sang trái để đưa các ô về trạng thái đích. Sử dụng chuột hay phím chức năng để di chuyển ô trống.

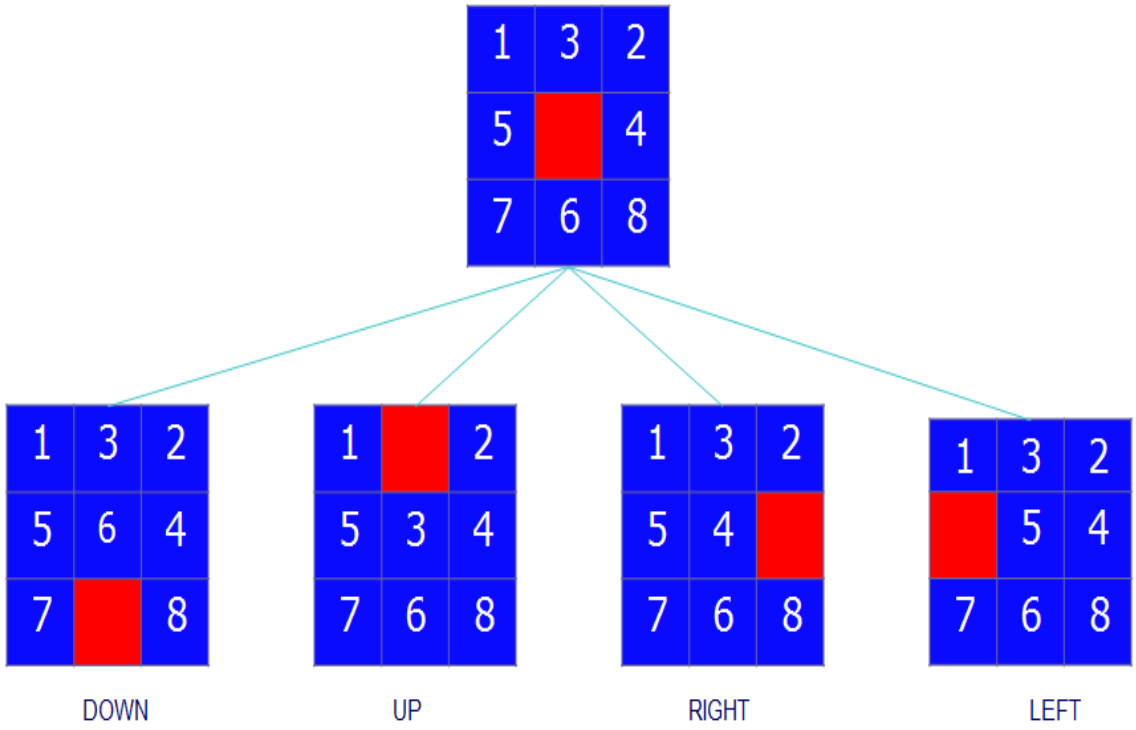
Chương trình có chức năng tự động chơi ở bất kì trạng thái nào đó. Mỗi trạng thái của bảng số là một hoán vị của n 2 phần tử. Ở đây ta có thể mở rộng bằng việc thêm hình ảnh vào game hoặc gắn số vào hình ảnh để gợi ý cho người chơi. Ở trạng thái ban đầu, các ô được sắp xếp ngẫu nhiên, và nhiệm vụ của người chơi là tìm được cách đưa chúng về trạng thái đích(ô đầu trống, các ô khác theo thứ tự tăng dần từ trái qua phải, từ trên xuống dưới).

Để đơn giản trong cách tiếp cận bài toán, ta giả định chỉ các ô trống di chuyển trong bảng là di chuyển đến các vị trí khác nhau. Như vậy tại một trạng thái bất kì có tối đa 4 cách di chuyển đến trạng thái khác(trái, phải, lên, xuống).



Hình 1 Trạng thái bắt đầu và đích

Bước di chuyển của ô trống:



Hình 2 Bước di chuyển của ô trống

PHẦN II.GIỚI THIỆU THUẬT TOÁN SỬ DỤNG-THUẬT TOÁN A*

1. Giới thiệu thuật toán

Thuật toán A* được mô tả lần đầu tiên năm 1986 bởi Peter Hart, Nils Nilson và Bertram Raphael. Trong báo cáo của họ, thuật toán được gọi là thuật 6 toán A, khi sử dụng thuật toán này với một hàm đánh giá heuristic thích hợp sẽ thu được hoạt động tối ưu, do đó mà có tên là A*. Trong khoa học máy tính, A* là một thuật toán tìm kiếm trong đồ thị. Thuật toán này tìm một đường đi từ nút khởi đầu tới một nút đích cho trước(hoặc tới một nút thỏa mãn điều kiện đích). Thuật toán này sử dụng một đánh giá heuristic để xếp loại từng nút theo ước lượng về tuyến đường tốt nhất đi qua nút đó. Thuật toán này duyệt các nút theo thứ tự của đánh giá heuristic này. Do đó, thuật toán A* là một ví dụ của tìm kiếm theo lựa chọn tốt nhất(best-first search). Xét bài toán tìm đường – bài toán mà A* thường được dùng để giải. A* xây dựng tăng dần tất cả các tuyến đường từ điểm xuất phát cho tới khi nó tìm thấy một đường đi chạm tới đích. Tuy nhiên, cũng như tất cả các thuật toán tìm kiếm có thông tin nó chỉ xây dựng các tuyến đường có vẻ dần về đích. Để biết những tuyến đường nào có khả năng sẽ dẫn tới đích, A* sử dụng một hàm đánh giá heuristic về khoảng cách từ điểm bất kỳ cho tới đích. Trong trường hợp tìm đường đi, đánh giá này có thể là khoảng cách đường chim bay - một đánh giá xấy xỉ thường dùng cho khoảng cách của đường giao thông. Điểm khác biệt của A* đối với tìm kiếm theo lựa chọn tốt nhất là nó còn tính đến khoảng cách đã đi qua. Điều đó làm cho A* đầy đủ và tối ưu, nghĩa là A* sẽ luôn tìm thấy đường đi ngắn nhất nếu tồn tại một đường đi như thế. A* không đảm bảo sẽ chạy nhanh hơn các thuật toán tìm kiếm đơn giản hơn. Trong một môi trường dạng mê cung, cách duy nhất để đến đích có thể là trước hết phải đi về phía xa đích và cuối cùng mới quay trở lại. Trong trường hợp đó, việc thử các nút theo thứ tự "gần đích hơn thì được thử trước" có thể gây tốn thời gian.

2. Mô tả thuật toán

Giả sử n là một trạng thái đạt tới(có đường đi từ trạng thái ban đầu n0 tới n). Ta xác định hàm đánh giá: $f(n) = g(n) + h(n)$

- g(n) là chi phí từ nút gốc n0 tới nút hiện tại n
- h(n) chi phí ước lượng từ nút hiện tại n tới đích
- f(n) chi phí tổng thể ước lượng của đường đi qua nút hiện tại n đến đích

Một ước lượng heuristic h(n) được xem là chấp nhận được nếu với mọi nút n: $0 \leq h(n) \leq h^*(n)$

Trong đó $h^*(n)$ là chi phí thật(thực tế) để đi từ nút n đến đích.

3. Cài đặt thuật toán

OPEN(FRINGE): tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến. OPEN là một hàng đợi ưu tiên mà trong đó phần tử có độ ưu tiên cao nhất là phần tử tốt nhất.

CLOSE: tập chứa các trạng thái đã được xét đến. Chúng ta cần lưu trữ những trạng thái này trong bộ nhớ để phòng trường hợp khi có một trạng thái mới được tạo ra lại trùng với một trạng thái mà ta đã xét đến trước đó.

Khi xét đến một trạng thái ni trong OPEN bên cạnh việc lưu trữ 3 giá trị cơ bản g, h, f để phản ánh độ tốt của trạng thái đó, A* còn lưu trữ thêm hai thông số sau:

- Trạng thái cha của trạng thái ni (ký hiệu Cha(ni)): cho biết trạng thái dẫn đến trạng thái ni .
- Danh sách các trạng thái tiếp theo của ni : danh sách này lưu trữ các trạng thái kế tiếp nk của ni sao cho chi phí đến nk thông qua ni từ trạng thái ban đầu là thấp nhất. Thực chất danh sách này có thể được tính từ thuộc tính Cha của các trạng thái đã được lưu trữ. Tuy nhiên việc tính toán này có thể mất nhiều thời gian(khi tập OPEN,CLOSE được mở rộng) nên người ta thường lưu trữ ra một danh sách riêng.

4. Thuật toán A*

function Astar(n0, ngoal)

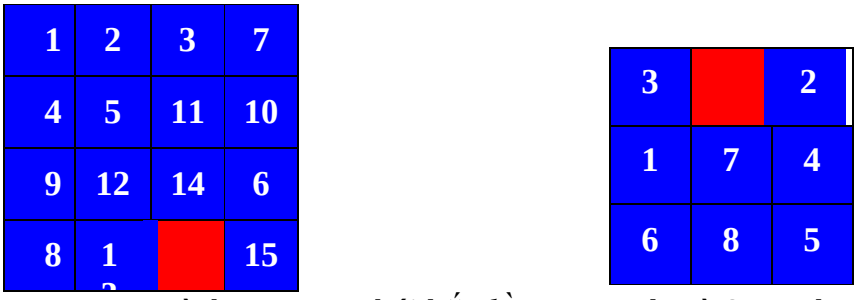
1) Đặt OPEN chỉ chứ n0. Đặt g(n0) = h(n0) = f(n0) = 0. Đặt CLOSE là tập rỗng
2) Lặp lại các bước sau cho đến khi gặp điều kiện dừng
   2).a Nếu OPEN rỗng: bài toán vô nghiệm, thoát.
   2).b Ngược lại, chọn ni trong OPEN sao cho f(ni) sao cho f(ni)min
      2).b.1 Lấy ni ra khỏi OPEN và đưa ni vào CLOSE
      2).b.2 Nếu ni chính là đích ngoal thì thoát và thông báo lời giải là ni
      2).b.3 Nếu ni không phải là đích. Tạo danh sách tất cả các trạng thái kế tiếp của ni
         Gọi một trạng thái này là nk. Với mỗi nk, làm các bước sau:
         2).b.3.1 Tính g(nk) = g(ni) + cost(ni , nk); h(nk);
            f(nk) = g(nk) + h(nk). 2.b.3.2
         2).b.3.2 Đặt Cha(nk) = ni
         2).b.3.3 Nếu nk chưa xuất hiện trong OPEN và CLOSE thì thêm nk vào OPEN

PHẦN III.CÀI ĐẶT THUẬT TOÁN

1. Trạng thái xuất phát

Rất dễ thấy mỗi trạng thái của bảng số là một hoán vị của n2 phần tử(với n là kích thước cạnh), như vậy không gian trạng thái của nó là n2 !, 8-puzzle là 9! = 362880(n = 3) và 15-puzzle là 16! = 20922789888000(n = 4),…. Khi m tăng lên 1 đơn vị thì không gian trạng thái sẽ tăng lên rất nhanh, điều này khiến cho việc giải quyết với các phiên bản n > 3 ít được áp dụng.

Để tạo ra một trạng thái ban đầu của trò chơi ta dùng mảng 1 chiều và sinh ngẫu nhiên một mảng trạng thái là hoán vị của n 2 phần tử trong đoạn (0; n2 -1). Với việc sinh ngẫu nhiên thì sẽ tạo ra những trạng thái không hợp lệ(trạng thái không dẫn tới trạng thái đích của bài toán), thì ta cần phải kiểm tra xem trạng thái có dẫn tới đích được hay không trước khi khởi tạo giao diện. Ngoài ra ta có thể trộn ngẫu nhiên trạng thái đích đến một trạng thái bất kì.



Hình 3. Trạng thái bắt đầu 15-puzzle và 8-puzzle

2. Cài đặt A*

Việc cài đặt thuật toán A* trong bài toán N-Puzzle cũng giống như phần trên đã nêu:

- FRINGE là tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến.
- M là tập các trạng thái tiếp theo của trạng thái ni
- KQ tập trạng thái kết quả, lưu các trạng thái từ trạng thái hiện tại tới đích

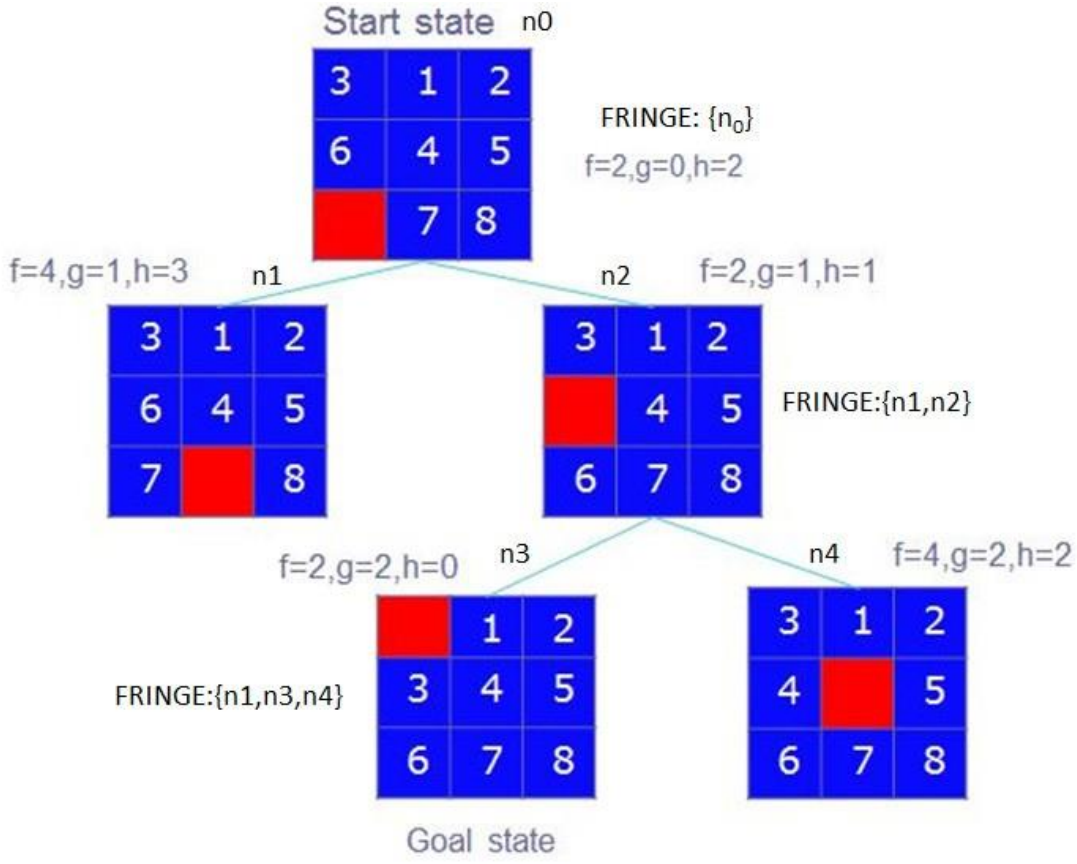
Đầu vào: trạng thái hiện tại, trạng thái đích

Đầu ra: tập các trạng thái từ trạng thái hiện tại tới trạng thái đích

Điều kiện dừng thuật toán: tìm thấy kết quả hoặc giới hạn thời gian hoặc người dùng cho phép dừng. Trong bài toán này ta có thể cải tiến bằng việc bỏ tập CLOSE. Ta thấy trong bước 2).b.3 ở trên sau khi tìm ra các trạng thái con của trạng thái ni . Khi đó ni có tối đa 4 trạng thái con, nhưng trong các trạng thái con của ni có 1 trạng thái trùng với trạng thái cha của ni . Vì vậy ta có thể loại bỏ trạng thái này để tránh việc xét lặp. Khi loại bỏ trạng thái này sẽ không tồn tại một trạng thái con nào trùng với một trạng thái trong tập CLOSE nữa. Việc loại bỏ này sẽ tránh được việc kiểm tra ở bước 2).b.3.3 gây tốn rất nhiều thời gian.

Ngoài ra, trong game này còn cài đặt thêm thuật toán A* sâu dần(IDA*) là một biến thể của thuật toán tìm kiếm A*. IDA* giúp loại bỏ hạn chế bộ nhớ của A* mà ko hy sinh giải pháp tối ưu. Mỗi lần lặp của thuật toán là quá trình tìm kiếm theo chiều sâu, f(n) = g(n) + h(n), tạo nút mới. Khi một nút được tạo ra có chi phí vượt quá một ngưỡng cutoff thì nút đó sẽ bị cắt giảm, quá trình tìm kiếm backtracks trước khi tiếp tục. Các ngưỡng chi phí được khởi tạo ước lượng heuristic của trạng thái ban đầu, và trong mỗi lần lặp kế tiếp làm tăng tổng chi phí của các nút có chi phí thấp đã được cắt tỉa trước đó. Thuật toán chấm dứt khi trạng thái đích có tổng chi phí không vượt quá ngưỡng hiện tại.

Minh họa A*

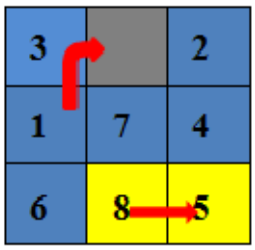


*Hình 4.Minh họa A**

3 Hàm ước lượng heuristic

3.1 Các hàm ước lượng heuristic

3.1.1 Heuristic1 = tổng khoảng cách dịch chuyển (←,→,↑,↓) ngắn nhất để dịch chuyển các ô sai về vị trí đúng của nó(khoảng cách Manhattan)



Hình 5. Minh họa

Giả sử: + rowđ, colđ là tọa độ(dòng và cột) của ô ở vị trí đúng

+ rows , cols là tọa độ(dòng và cột) của ô ở vị trí sai

+ xd = |cols – colđ|

+ yd = |rows – rowđ|

=> d = xd + yd là khoảng cách ngắn nhất để di chuyển 1 ô về đúng vị trí

=>h1 = ∑d

Trong bảng số 3x3 trên, để di chuyển ô số 8 về vị trí đúng cần 1 lần, để di chuyển ô số 1 về vị trí đúng cần 2 lần(qua 2 ô khác). Để thu được kết quả này ta làm phép tính đơn giản: lấy tổng khoảng cách của các dòng và cột giữa hai vị trí(vị trí hiện tại và vị trí đúng)

- Lấy tọa độ của ô số 1 ở vị trí hiện tại ta có: rows = 1, cols = 0

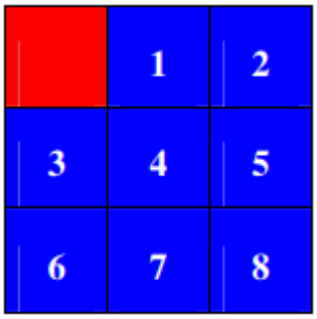
- Lấy tọa độ ô số 1 ở vị trí đúng : rowđ = 1/3 = 0; cols = 1%3 = 1

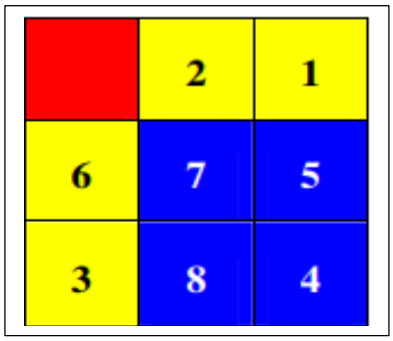
- Vậy khoảng cách ngắn nhất để di chuyển ô số 1 về vị trí đúng:

d1 = |rows – rowđ| + |cols – colđ| = |1 – 0| + |0 – 1| = 2

Tương tự, tính tất cả các khoảng cách d của các ô sai còn lại ta được

3.1.2 heuristic2 = tổng khoảng cách dịch chuyển (←,→,↑,↓) ngắn nhất để dịch chuyển các ô sai về vị trí đúng của nó cộng thêm chỉ số phạt cặp ô hàng xóm với nhau đang nằm ngược vị trí của nhau.



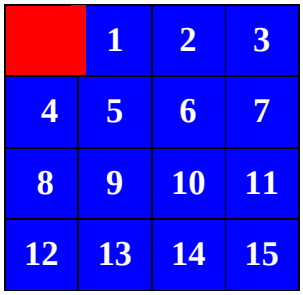


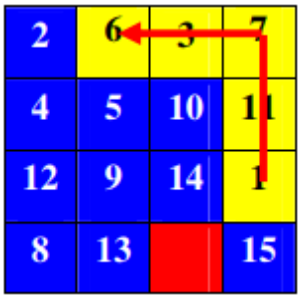
Hình 6. Đích Hình 7. Minh họa

=>h2 = ∑d + a

Trong đó a là chỉ số phạt cặp ô hàng xóm đang nằm ngược vị trí. Cặp (2,1) muốn về đúng vị trí cần dịch chuyển ít nhất 4 bước(không để ý tới các ô khác), 2 bước đã được tính trong ∑d nên a = 2. Vì vậy trong 1.1.6b có 2 cặp hàng xóm nằm ngược vị trí nên ở đây a = 2+2 = 4.

3.1.3 heuristic3





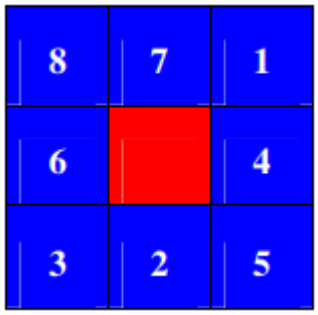
Hình 7 đích Hình 8. Minh họa

Xét 1 ô nằm sai vị trí: d = |rows – rowđ| 2 + |cols – colđ| 2

Đặt d3 = ∑d

=>h3 = d3 – [0.15*d3] + a

3.2 Ví dụ so sánh 3 hàm heuristic



Xét trạng thái hình trên

+ heuristic1 = 4 + 2 + 1 + 1 + 1 + 1 + 3 + 1 = 14

+ heuristic2 = heuristic1 + 2 = 16 (a = 2)

+ d34 = 8 + 4 + 1 + 1 + 1 + 1 + 5 + 1 = 22

+ heuristic3 = d34 – [0.15*d34] + 2 = 21

+ heuristic4 = d34 + 2 = 24

PHẦN III- KẾT LUẬN

Thông qua việc tìm hiểu và nghiên cứu đề tài này giúp chúng tôi có cái nhìn toàn diện hơn trong việc ứng dụng trí tuệ nhân tạo vào giải quyết vấn đề thực tế. Đây là bài toán cổ điển trong trí tuệ nhân tạo cho các thuật toán mô hình hóa liên quan đến tìm kiếm có tri thức bổ sung. Đề tài đã được nhiều người nghiên cứu giải quyết, nhưng cho đến nay vẫn chưa có cách giải quyết tối ưu cho tất cả không gian trạng thái trò chơi vì kích thước tăng không gian trạng thái sẽ tăng lên rất nhanh. Hy vọng những nghiên cứu đánh giá của chúng tôi sẽ góp phần bổ sung thêm một hướng giải quyết cho bài toán. Do thời gian có hạn nên đề tài không tránh khỏi những sai sót, mong thầy góp ý, đánh giá giúp chúng tôi hoàn thiện đề tài.

PHẦN IV-TÀI LIỆU THAM KHẢO

1. Bài giảng môn khai phá dữ liệu cô Nguyễn Thị Hải Bình
2. Tài liệu trên các trang web, diễn đàng.