**GIẢI THUẬT A\* VÀ BÀI TOÁN 15 PUZZLE**

Sinh viên thực hiện:

<1542228 - Phạm Trung Hiếu>

<1542242 - Hồ Hải Nam>

<1542244 - Vũ Hoàng Nam>

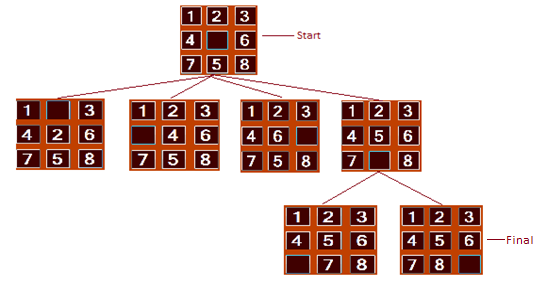
<1542259 - Hứa Chí Phước>

**1.Giới Thiệu**

Bài toán N-Puzzle là một trò chơi được biết đến với phiên bản phổ biến là 8-puzzle (bảng 3\*3) và 15-puzzle( bảng 4\*4).

N-puzzle là một bảng gồm các ô số như bàn cờ, trong đó có 1 ô bị thiếu. Mục tiêu của bài toán này là sắp xếp thứ tự của các ô số từ một trang thái bắt đầu đến trạng thái đích bằng cách trượt các sô số vào ô trống để tạo ra những bước di chuyển từ đó thay đổi trật tự của các ô số. Người chơi chỉ được di chuyển ô trống ngang, dọc và cố gắng để đạt được trạng thái đích bằng ít bước di chuyển nhất có thể.

Ví dụ với bài toán 8-pluzze



-Trạng thái bắt đầu “Start”

-Trạng thái kết thúc “Final”

Mỗi trạng thái tương ứng với mỗi đỉnh trong đồ thị, theo ví dụ trên ta thấy cần di chuyển ba bước sẽ tới trạng thái đích.

**Mục tiêu:** Tìm ra lời giải cho bài toán 15 puzzle bằng thuật giải A\* với số bước chuyển ít nhất có thể.

**Giới hạn:** Số lượng trạng thái có thể của 15 puzzle là 16! hay 2.092279 x 10^13 do đó một số trường hợp bài toán có lời giải chưa tối ưu và giải pháp thay thế.

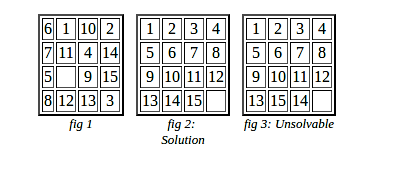
**2.Thuật Giải**

**2.1 Giới thiệu về giải thuật A\***

Trong khoa học máy tính, A\* (đọc là A sao) là một thuật toán tìm kiếm trong đồ thị. Thuật toán này tìm một đường đi từ một nút khởi đầu tới một nút đích cho trước (hoặc tới một nút thỏa mãn một 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). (Theo Wikipedia.)

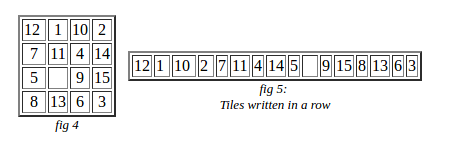
**2.2 Xét các trạng thái có thể giải quyết được**

Như đã đề cập ở trên, bài toán 15 puzzle có khoảng một nửa trạng thái (hay cấu hình ) của bài toán 15 puzzle là không thể giải quyết được vì vậy trước khi bắt đầu dùng thuật giải A\* cần xác định trạng thái đã cho (trạng thái bắt đầu) là có thể giải quyết được.



Hình 1.1 Trang thái thứ 3 (fig3) là trạng thái không thể giải quyết được.

**Công thức để xác định các trạng thái có thể giải quyết được**



Hình 1.2 Ví dụ về 1 trạng thái có thể giải quyết được.

* Khái niệm về sự đảo ngược

Một sự đảo ngược được xác định khi một ô số ta đang xét đứng trước một ô số khác nhỏ hơn nó hay nói cách khác là có một ô số nhỏ hơn đứng sau nó.

* công thức

Theo hình 1.2, ta có một bảng số 4\*4. Ta sẽ bắt đầu xét các ô số lần lượt theo thứ tự từ trái qua phải và từ trên xuống dưới. Xét số 12, có 11 số nhỏ hơn đứng sau nó nên tại ô số 12 có 11 sự đảo ngược. Vậy ta có số lượng sự đảo ngược ứng với từng ô số

* ô số 12 có 11.
* ô số 1 có 0.
* ô số 10 có 8
* ô số có 0.
* ô số 7 có 4.
* ô số 11 có 6.
* ô số 4 có 1
* ô số 14 có 6
* ô số 5 có 1
* ô số 9 có 3
* ô số 15 có 4
* ô số 8 có 2
* ô số 13 có 2
* ô số 6 có 1

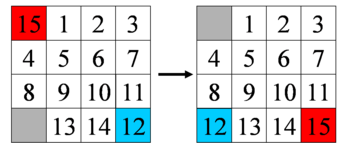
Gọi N là tổng số lượng sụ đảo ngược. Vậy trong bảng số trên ta có tổng cộng N=49 sự đảo ngược

Áp dụng cho bài toán 15-pluzze. Các trạng thái có thể giải phải thỏa mản một trong hai trường hợp sau:

* N mod 2 = 0 và ô trống nằm trên dòng chẵn tính từ trên xuống
* N mod 2 = 1 và ô trống nằm trên dòng lẻ tính từ trên xuống

**2.3 Các loại Heuristic có thể áp dụng trong giải thuật A\* đối với bài toán 15 puzzle**

**2.3.1 Theo số ô nằm sai vị trí**



Hình 1.3 Heuristic được tính theo số ô năm sai vị trí.

Heuristic h được tính bằng tổng số ô nằm sai vị trí của trạng thái hiện tại so với trạng thái đích. Theo hình 1.3 thì ô số 15 và ô số 12 năm sai vị trí so với trạng thái đích :

h = 1 + 1 = 2

**2.3.2 Theo khoảng cách mahatan**

Tổng khoảng cách dịch chuyển ngắn nhất để dịch chuyển các ô sai số về vị trí đúng của nó.

**d = |toạ độ dòng sai - toạ độ dòng đúng| + |toạ độ cột sai - toạ độ cột đúng|**

Theo hình 1.3

d15 = |0-3| + | 0-3| = 6

d12 = |3-3| + |3-0| = 3

=> h = d15 + d12 = 6 + 3 = 9.

**2.4 Giải Thuật**

Để giải quyết bài toán 15-puzzle ta sẽ tính toán chi phí dựa theo công thức sau:

Chi phí = G + H

G: số bước di chuyển

H: chi phí tính theo heuristic

Giải quyết bài toán

1. Kiểm tra trạng thái (trạng thái bắt đầu) có thể giải quyết hay không
2. Nếu trạng thái có thể giải quyết ⇒ trạng thái đầu tiên == trạng thái bắt đầu(Start state)
3. Chọn các trạng thái (v) có thể di chuyển từ trạng thái đầu tiên (v không nằm trong openlist và closed list)
4. Đưa các trạng thái trên vào “open list”
5. Đưa trạng thái đang xử lý vào “closed list”
6. Chọn trạng thái kế tiếp là trạng thái có chi phí nhỏ nhất trong “open list”
7. Lặp lại các bước 2-6 cho tới khi gặp trạng thái kết thúc(Final state) hoặc không còn trạng thái nào để kiểm tra

Áp dụng thực tế vào bài toán 15-puzzle. Do số trạng thái trong bài toán là rất lớn nên ta sẽ áp dụng các cài đặt sau vào trong giải thuật:

* Áp dụng hàng đợi ưu tiên
* Sử dụng Hash tables của mỗi trạng thái để tìm kiếm một trạng thái trong “open list” / “closed list” nhanh hơn thay vì so sánh từng phần tử trong 15-puzzle

1. **Chương Trình Và Kết Quả**

**3.1 Chương Trình**

Chương trình chạy tham số dòng lệnh có dạng sau:

**<Tên chương trình> <Tập tin đầu vào> <Tập tin đầu ra>**

* **Cấu trúc file dữ liệu đầu vào**: hai ma trận trạng thái đầu và trạng thái đích
* **Cấu trúc file dữ liệu đầu ra:**  
  + Nếu không tìm được lời giải ghi là “NULL”  
  + Nếu tìm được lời giải thì:

Dòng đầu tiên: số bước di chuyển

Dòng tiếp theo: lần lượt xuất ra các ma trận trạng thái các bước di chuyển

**3.2 Kết Luận**

- Áp dụng code thử nghiệm thì một số trạng thái có thể tìm ra lời giải nhanh chóng (số bước di chuyển < 50)

-Một số trạng thái có lời giải chưa tối ưu do số trạng thái được thêm vào openlist quá lớn và thời gian chạy xong thuật toán quá lâu. Các trạng thái này đều cần hơn 50 bước di chuyển để di chuyển về trạng thái đích

Do các hạn chế của A\* nên để sử giải quyết các bài toán n-puzzle với n > 3 giải pháp thay thế là sử dụng thuật toán Iterative-Deepening-A\* (IDA\*)