**GIẢI BÀI TOÁN 8 PUZZLE TRONG TRƯỜNG HỢP BẢNG BỊ CHE LẠI, CHỈ CÓ MỘT Ô TRỐNG ĐỂ CÓ THỂ NHÌN THẤY GIÁ TRỊ Ô ĐÓ**

**Hồ Đức Thiện­­1 18110206**

**Phùng Công Chiến1 18147171**

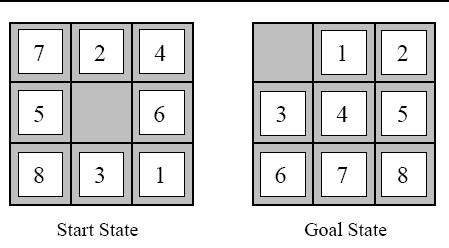
*1Trường Đại học Sư phạm Kỹ thuật TP.HCM*

Tóm tắt:

Bài toán (hay game) n-puzzle có lẽ rất quen thuộc với chúng ta cũng như những người mới bắt đầu tiếp cận với môn trí tuệ nhân tạo. Nó được biết đến với nhiều phiên bản và tên gọi khác nhau như 8-puzzle, 15-puzzle, Gem puzzle, Boss puzzle, Game of Fifteen, Mystic Square,… Ở mức độ đơn giản, nhóm em làm về phiên bản 8-puzzle nhưng đã được tùy biến. Bài toán gồm một bảng 3×3 với các ô số được đánh từ 1->8 và một ô trống. Ở trạng thái bắt đầu, các ô được sắp đặt ngẫu nhiên, và nhiệm vụ của người giải là tìm cách đưa chúng về đúng thứ tự kiểu lợp ngói (lớn dần từ trái qua phải và từ trên xuống). Mục đích chính của game này là giải bài toán nhanh nhất, ít tốn chi phí nhất mà vẫn chính xác. Điểm khác biệt của bài toán lần này so với bài toán 8 puzzle cơ bản là chỉ hiển thị giá trị 1 ô số, tất cả các ô còn lại đều bị che lại.

Từ khóa: 8-puzzle; Breadth-First Search; giải thuật tìm kiếm A\*;

1. **ĐẶT VẤN ĐỀ:**



**Hình 1.** Bài toán 8 puzzle.

Để đơn giản trong cách tiếp cận giải bài toán, người ta giả định chỉ có ô trống trong bảng là di chuyển đến những vị trí khác. Như vậy tại một trạng thái thì chỉ có tối đa 4 cách đi để chuyển sang trạng thái khác (trái, phải, lên, xuống). Người ta cũng nhận ra được rằng để có thể chuyển từ 1 trạng thái bất kì về trạng thái đích như trên thì trạng thái đầu đó phải theo một quy luật. Cho trạng thái đầu tiên, duyệt qua từng ô theo thứ tự từ trái qua và từ trên xuống, ở mỗi ô số duyệt đến, đếm số lượng gạch bị đặt sai vị trí và tìm tổng khoảng cách giữa mỗi khối và vị trí của nó trong cấu hình mục tiêu. Tuy nhiên, khác với bài toán cổ điển, ở bài toán này có sự thay đổi ở phần đầu vào. Start state chỉ xem được một ô duy nhất, các ô còn lại bị ẩn. Yêu cầu cần phải di chuyển ô đã hiển thị đi khắp bảng để mở hết các ô sau đó mới tiến hành giải bài toán. Nhóm em đã sử dụng cả 2 thuật toán tìm kiếm là A\* và BFS để giải, từ đó so sánh 2 thuật toán để xem bên nào tối ưu nhất, nhanh nhất và chính xác nhất.

1. **PHƯƠNG PHÁP:**
   1. Xác định trạng thái ban đầu và trạng thái đích:

Trò chơi có trạng thái đầu (Start state) và trạng thái đích (Goal state). Đối với bài toán này, chúng ta cần di chuyển ô đã biết trên khắp vị trí của bảng để dần dần mở giá trị cho các ô khác, sau đó mới thực hiện bài toán. Trạng thái đầu được khởi tạo dựa trên hàm sinh ngẫu nghiên. Trạng thái đích của bài toán được chia làm nhiều dạng tùy theo hướng người lập trình, về cơ bản nó được sắp xếp theo thứ tự tăng dần, và ô thứ 9 là ô không có giá trị (nhằm thuận tiện cho việc lập trình, ta sẽ gán cho nó giá trị 0) để di chuyển được đặt ở đầu hoặc ở cuối.



**Hình 2.** Bảng bị che và trạng thái ban đầu sau khi tìm ra hết các giá trị.

* 1. Thuật toán tìm kiếm:

Đối với thuật toán Breadth-First Search BFS: thuật toán tìm kiếm ưu tiên tối ưu là sự kết hợp cả hai thuật toán tìm kiếm theo chiều sâu và thuật toán tìm kiếm theo chiều rộng, cho phép người dùng đi một con đường duy nhất tại một thời điểm, nhưng đồng thời vẫn xem xét các hướng đi khác. Nếu nhận thấy rằng hướng đang đi không có triển vọng bằn những con đường còn lại thì ta sẽ chuyển sang đi theo một trong những đường đó. Tại mỗi bước của thuật toán BFS ta chọn theo trạng thái có khả năng cao nhất trong số các trạng thái đã được xét cho đến thời điểm đó, như vậy với phương pháp này ta ưu tiên đi vào những nhánh tìm kiếm có khả năng cao nhất. Tư tưởng chủ đạo của Best First Search giúp ta sẽ không bị đi những vòng luẩn quẩn bởi vì nếu đi càng sâu vào một hướng mà ta phát hiện hướng đi này ngày càng tệ, thậm chí tệ hơn cả những đường ta chưa đi thì ra sẽ không tiếp tục trên hướng hiện tại nưa mà sẽ chọn theo một trong nhưng hướng tốt nhất trong tập hợp chưa đi.

Procedure BFS(*G, root*) is

Let Q be a queue

Label root as discovered

q.enqueue(root)

while Q is not empty do

v := Q.dequeue()

if v is the goal then

return v

for all edges from v to w in G.adjacentEdges(v) do

if w is not labeled as discovered them

label w as discovered

w.parent := v

Q.enqueue(w)

Mã giả BFS.

Đối với thuật toán A\*: là một thuật toán tìm đường đi từ điểm tới điểm, từ một nút khởi đầu đến một nút đích cho trước hoặc tới một nút thỏa mãn điều kiện và không sử dụng để tìm giải pháp cho vấn đề tìm đường đi ngắn nhất trong học thuyết đồ thị. Về bản chất, thuật toán này làm việc rất giống như Dijkstra. Nhưng hiệu quả hơn khi A\* sử dụng một hàm đánh giá bổ sung (đánh giá heuristic) để ước lượng về tuyến đường đi tốt nhất qua nút đó (Việc lựa chọn trạng thái tiếp theo sẽ phụ thuộc vào hàm lượng giá này). A\* Là một ví dụ cho thuật toán Best first search, là một thuật toán đầy đủ, luôn tìm thấy một lời giải nếu bài toán có lời giải. Vấn đề đưa ra giống với vấn đề được giải quyết bởi thuật toán tìm đường đi ngắn nhất Dijkstra. Cho một đồ thị (đồ thị có hướng và trọng số không âm) và hai node trên đồ thị (gọi là điểm bắt đầu và điểm cuối), cần tìm một con đường có chi phí đi từ đầu tới cuối là nhỏ nhất trong các đường đi có thể, kết quả là danh sách các node nó cần đi qua để tới đích với chi phí nhỏ nhất. Độ tốt của một trạng thái được tính dựa trên hai giá trị mà ta gọi là h(n)và g(n). Trong đó h(n) là một ước lượng về chi phí từ trạng thái hiện tại cho đến trạng thái đích. Còn g(n) là chiều dài quãng đường đã đi từ trạng thái ban đầu tới trạng thái đang xét. Một điểm cần lưu ý rằng g là chi phí thực sự còn h là chi phí ước lượng. Kết hợp g (n) và h(n) ta có hàm tổng chi phí ước lượng cho quãng đường đi từ nốt xuất phát tới nốt đích dọc theo đường đi của trạng thái hiện tại. Công thức ước lượng: f(n) = g(n)+h(n). Nếu giá trị f(n) càng thấp thì độ ưu tiên của n càng cao. Hai giá trị này đều quy ước là không âm để thuận tiện cho quá trình tính toán. Dưới đây là mã giả của thuật toán A\*:

1. Gọi Open: tập các trạng thái đã được sinh ra nhưng chưa được xét đến.
2. Close: tập các trạng thái đã được xét đến.
3. Cost(p,q): là khoảng cách giữa p, q.
4. g(p): khoảng cách từ trạng thái đầu đến trạng thái hiện tại.
5. h(q): giá trị ước lượng từ trạng thái hiện tại đến trạng thái đích.
6. f(p) = g(p) + h(p).
   1. Bước 1:
      * Open: = {S}
      * Close: = {}
   2. Bước 2: while (Open !={})
      * Chọn trạng thái (đỉnh) tốt nhất p trong Open (xóa p khỏi Open).
      * Nếu p là trạng thái kết thúc thì thoát.
      * Chuyển p qua Close và tạo ra các trạng thái kế tiếp q sau p.
        1. Nếu q đã có trong Open
           1. Nếu g(q)>g(p)+Cost(p,q)

g(q) = g(p) + Cost(p,q)

f(q) = g(q) + h(q)

prev(q) = p //đỉnh cha của q là p

* + - 1. Nếu q chưa có trong Open

g(q) = g(p) + Cost(p,q)

f(q) = g(q) + h(q)

prev(q) = p

Thêm q vào Open

* + - 1. Nếu q có trong Open
         1. Nếu g(q) > g(p) + Cost(p,q)

Bỏ q khỏi Close

Thêm q vào Open

* 1. Bước 3: Không tìm được.

1. **KẾT QUẢ:**

**So sánh thời gian chạy của thuật toán A\* và BFS**

Ta thấy thuật toán A\* sẽ tối ưu trong nhiều trường hợp sẽ tối ưu hơn thuật toán BFS. Về bản chất của thuật toán, ta cũng có thể dễ dàng nhận ra sự khác biệt này. BFS là thuật toán tìm kiếm mù đồng nghĩa với việc sẽ duyệt tất cả các trạng thái. Điều này sẽ làm cho thuật toán tốn bộ nhớ và thời gian. Trong khi đó A\* là thuật toán có hàm đánh giá Heuristic nên chỉ duyệt những trạng thái được đánh giá là tối ưu mà thôi. Cho nên sự chênh lệch thời gian ở bảng thống kê trên là minh chứng cho sự tiện ích mà hàm A\* mang lại.

1. **KẾT LUẬN:**

Trong bài toán này, nhóm chúng em đã ứng dụng được các hàm, các thuật toán cơ bản của môn học trí tuệ nhân tạo để giải quyết vấn đề. Thông qua đó hiểu rõ hơn được bản chất của 2 thuật toán đã sử dụng. Điểm nổi bật là thuật toán A\*, với việc tích hợp hàm Heuristic đã khiến cho A\* tối ưu hơn BFS một cách đáng kể và đã cho ra kết quả chính xác, ít tốn bộ nhớ và nhanh hơn khá nhiều so với BFS.

**TÀI LIỆU THAM KHẢO:**

[1]:<https://www.youtube.com/watch?v=G7XnNtF7UEE&list=PLQj93CJe0N73raStQrnbZ9oSLtYZv8zej&index=8>

[2]: <https://www.youtube.com/watch?v=VR0mhQfOAbQ>

[3]:<https://yinyangit.wordpress.com/2010/12/16/algorithm-phan-tich-va-gi%E1%BA%A3i-bai-toan-n-puzzle/>

[4]: <https://www.geeksforgeeks.org/a-search-algorithm/>

[5]:<https://www.researchgate.net/publication/312230941_8_Tiles_Puzzle_Solving_Using_IDS_and_A_Search>