**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**ĐẠI HỌC KINH TẾ TP HỒ CHÍ MINH**

**TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ**

🖎🕮✍



**ĐỒ ÁN MÔN HỌC**

**ĐỀ TÀI:**

XÂY DỰNG GAME Ô CHỮ 8 SỐ DỰA TRÊN

GIẢI THUẬT TÌM KIẾM A\*

**Học phần**: Trí Tuệ Nhân Tạo

**Nhóm Sinh Viên**:

1. PHAN TRẦN SƠN BẢO
2. HỒ VIỄN TRIẾT

**Chuyên Ngành**: KHOA HỌC DỮ LIỆU

**Khóa**: K46

**Giảng Viên**: TS. Đặng Ngọc Hoàng Thành

**TP. Hồ Chí Minh, Ngày 29 tháng 5 năm 2022**

# MỤC LỤC

[MỤC LỤC 1](#_Toc104748662)

[CHƯƠNG 1. TỔNG QUAN 2](#_Toc104748663)

[1.1. Giới Thiệu Về Trò Chơi Ô Chữ 8 Số 2](#_Toc104748664)

[1.2. Phát Biểu Bài Toán 2](#_Toc104748665)

[1.3. Một Số Hướng Tiếp Cận Giải Quyết Bài Toán 2](#_Toc104748666)

[CHƯƠNG 2. GIẢI THUẬT TÌM KIẾM A\* 3](#_Toc104748667)

[2.1. Giới Thiệu Về Giải Thuật Tìm Kiếm A\* 3](#_Toc104748668)

[2.2. Ứng Dụng Giải Thuật Tìm Kiếm A\* Cho Bài Toán Ô Chữ 8 Số 3](#_Toc104748669)

[CHƯƠNG 3. CÁC KẾT QUẢ THỰC NGHIỆM 7](#_Toc104748670)

[3.1. Các Tình Huống 7](#_Toc104748671)

[3.2. Phân Tích và Đánh Giá 8](#_Toc104748672)

[CHƯƠNG 4. KẾT LUẬN 21](#_Toc104748673)

[4.1. Các Kết Quả Đạt Được 21](#_Toc104748674)

[4.2. Những Hạn Chế và Hướng Phát Triển 21](#_Toc104748675)

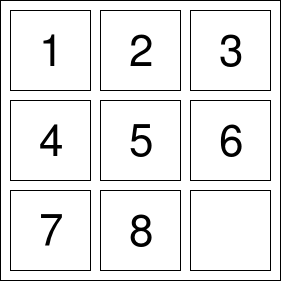
[TÀI LIỆU THAM KHẢO 23](#_Toc104748676)

[PHỤ LỤC 24](#_Toc104748677)

# CHƯƠNG 1. TỔNG QUAN

## 1.1. Giới Thiệu Về Trò Chơi Ô Chữ 8 Số

* Trò chơi ô chữ 8 số cũng giống như các trò chơi xếp hình, nó có kích thước là một 1 ô vuông có kích thước 3\*3, mỗi ô là số từ 1 đến 8 (không trùng lặp) và có 1 ô trống. Trạng thái ban đầu được khởi tạo một cách ngẫu nhiên (sắp xếp lộn xộn). Nhiệm vụ của người chơi là xếp lại các ô (chỉ di chuyển 1 bước theo chiều qua trái, qua phải, lên trên hoặc xuống dưới) sao cho đạt được trạng thái đích bên dưới càng nhanh và càng ít bước di chuyển thì thứ hạng càng cao. Goal State:



## 1.2. Phát Biểu Bài Toán

* Người chơi di chuyển ô trống qua trái, phải, lên hoặc xuống để đạt được trạng thái đích. Trong quá trình di chuyển, có thể có người chơi không chọn được bước di chuyển tối ưu nhất dẫn đến việc các bước di chuyển để đạt được trạng thái đích càng tăng và sẽ tốn nhiều thời gian để đạt được trạng thái đích. Vi vậy, ta cần thuật để tối ưu các bước di chuyển khi đi từ trạng thái ban đầu đến trạng thái đích.

## 1.3. Một Số Hướng Tiếp Cận Giải Quyết Bài Toán

* Ta có thể dùng các thuật toán tìm kiếm như BFS, DFS, A\*,… để giải quyết bài toán tìm kiếm trạng thái của puzzle trên. Nhưng các thuật toán tìm kiếm không có thông tin như BFS, DFS,… để đi từ trạng thái ban đầu đến trạng thái đích sẽ rất lâu và cần rất nhiều bước di chuyển để đạt được vì 2 thuật toán này sẽ duyệt từng trạng thái được sinh ra từ trạng thái ban đầu của nó. Vì vậy, các thuật toán tìm kiếm có thông tin như Greedy, A\*,… sẽ giúp ta giải quyết bài toán này nhanh hơn và ít bước di chuyển hơn so với các thuật toán tìm kiếm không có thông tin. Nhóm em đã chọn thuật toán tìm kiếm có thông tin A\* dựa theo hàm đánh giá Heuristic để đánh giá và tìm ra trạng thái tối ưu nhất trong các trạng thái được mở rộng nên tối ưu được các bước di chuyển cũng như tiết kiệm thời gian giải quyết cho bài toán tìm trạng thái đích của trò chơi ô chữ 8 số này.

# CHƯƠNG 2. GIẢI THUẬT TÌM KIẾM A\*

## 2.1. Giới Thiệu Về Giải Thuật Tìm Kiếm A\*

* Thuật toán A\* là một ví dụ của Best-first search. A\* là giải thuật tìm kiếm trong đồ thị, tìm đường đi từ một đỉnh hiện tại đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm Heuristic.
* Từ trạng thái hiện tại A\* xây dựng tất cả các đường đi có thể đi dùng hàm ước lượng khoảng cách (hàm Heuristic) để đánh giá đường đi tốt nhất có thể đi. Tùy theo mỗi dạng bài khác nhau mà hàm Heuristic sẽ được đánh giá khác nhau. A\* luôn tìm được đường đi ngắn nhất nếu tồn tại đường đi như thế.
* A\* lưu giữ một tập các đường đi qua đồ thị, từ đỉnh bắt đầu đến đỉnh kết thúc, tập các đỉnh có thể đi tiếp được lưu trong tập Open.
* Thứ tự ưu tiên cho một đường đi được quyết định bởi hàm Heuristic được đánh giá f(x) = g(x) + h(x)
  + g(x) là chi chi phí của đường đi từ điểm xuất phát cho đến thời điểm hiện tại.
  + h(x) là hàm ước lượng chi phí từ đỉnh hiện tại đến đỉnh đích.
  + f(x) thường có giá trị càng thấp thì độ ưu tiên càng cao.

## 2.2. Ứng Dụng Giải Thuật Tìm Kiếm A\* Cho Bài Toán Ô Chữ 8 Số

* Thuật toán A\* sẽ dựa vào hàm Heuristic để quyết định thứ tự ưu tiên cho đường đi, f(x) càng nhỏ thì độ ưu tiên càng cao. Thuật toán sẽ tiếp tục mở rộng xét các trạng thái con của trạng thái đã được chọn. Đối với các trạng thái có f(x) nhỏ nhất và bằng nhau thì thuật toán sẽ xét tất cả các trạng thái con của từng trạng thái với các trạng thái chưa được xét trước đó để chọn ra trạng thái có f(x) nhỏ nhất. Nếu trạng thái con trùng với trạng thái đích thì thuật toán kết thúc.
* Cài đặt thuật toán:

Begin Bước 1: Khởi tạo danh sách N và push trạng thái ban đầu vào.

Bước 2: While:

* 1. Pop trạng thái ở đầu danh sách N
  2. If là trạng thái kết thúc:

Hiển thị đường đi

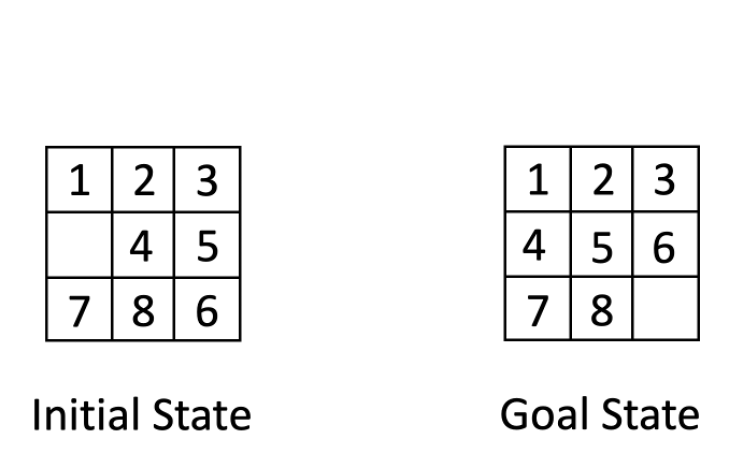
Stop

* 1. for mỗi trạng thái con được sinh ra từ với trạng thái mới được Pop ra:

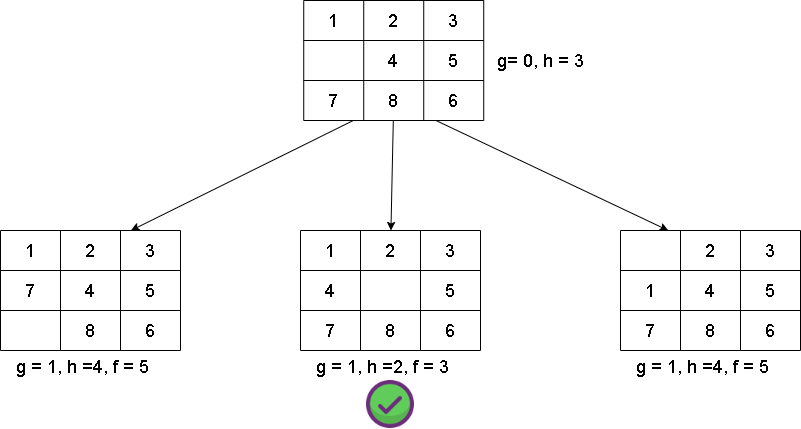
Push các trạng thái con vào danh sách N sao cho N được sắp theo thứ tự tăng dần theo hàm đánh giá

End

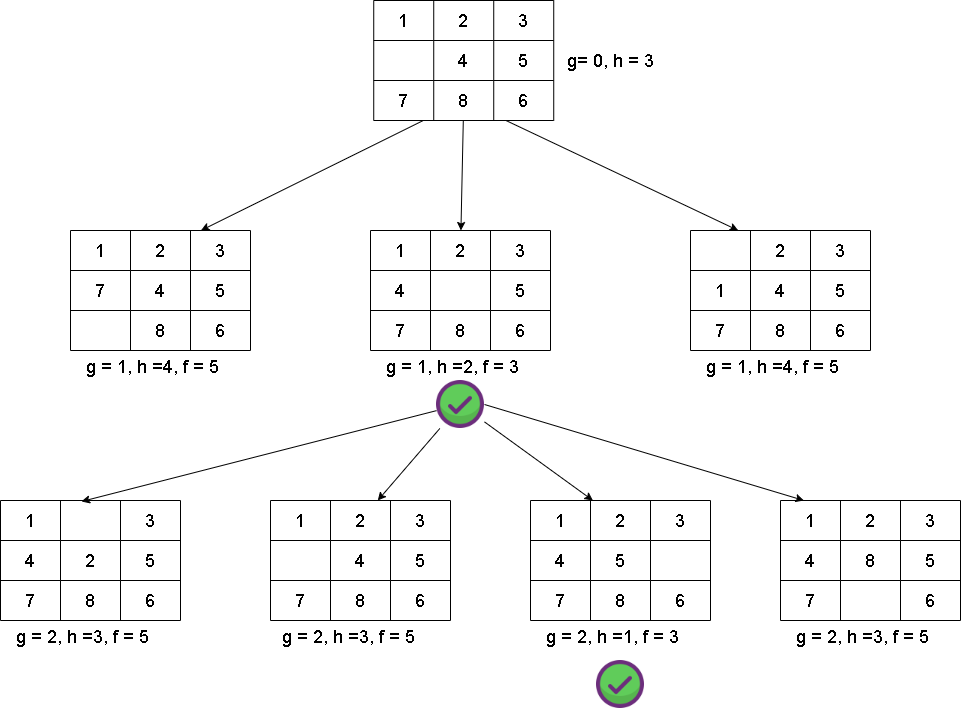
* Ví dụ:



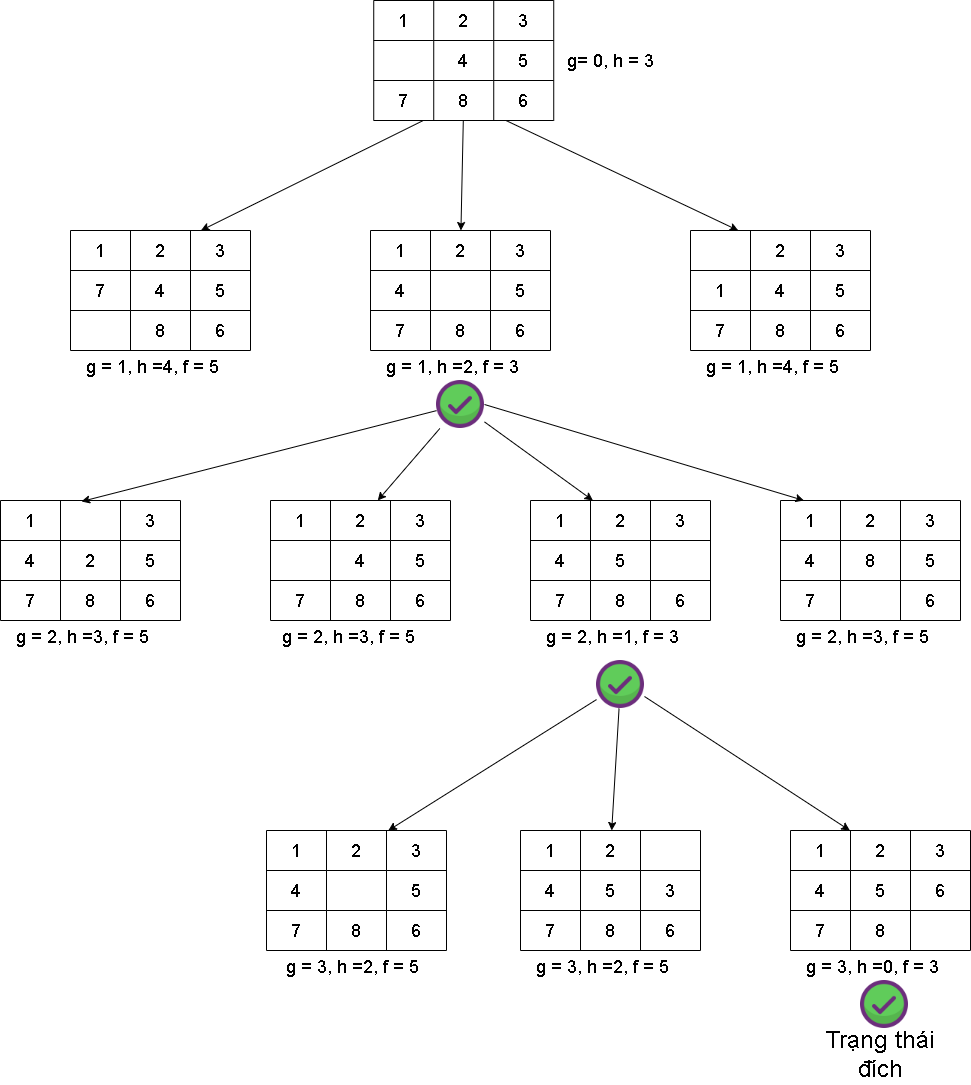
* Ta thấy từ trạng thái ban đầu, sẽ sinh ra 3 trạng thái con bên dưới (lần lượt 7 đi lên, 4 sang trái và 1 đi xuống).



* Vì trạng thái ở giữa có f = 3 là nhỏ nhất nên thuật toán sẽ chọn trạng thái tối ưu nhất là trạng thái ở giữa (có f = 3) trong 3 trạng thái con, tiếp tục tìm kiếm xét các trạng thái con từ trạng thái này.



* Tiếp đó, từ trạng thái tối ưu được chọn sinh ra 4 trạng thái tiếp theo như trên ảnh lần lượt có f = 5 (di chuyển 2 xuống) , f = 5 (di chuyển 4 sang phải), f = 3 (di chuyển 5 sang trái) và f = 5 (di chuyển 8 lên), thuật toán sẽ chọn trạng thái có f nhỏ nhất là f = 3 từ 6 trạng thái trong ảnh trên và tiếp tục tìm kiếm từ trạng thái này.

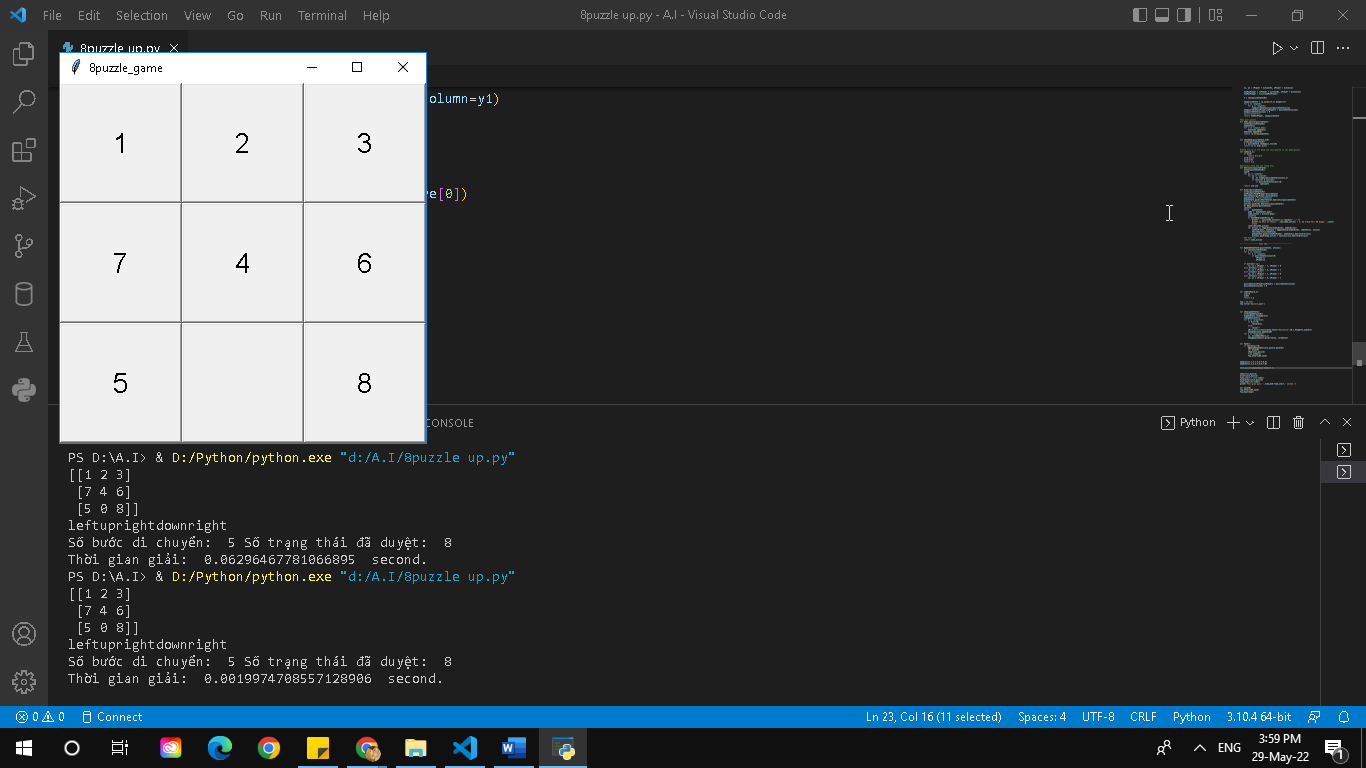


* Mở rộng được ba trạng thái con từ trạng thái tối ưu được chọn, thuật toán chọn trạng thái có f nhỏ nhất là f = 3 (nhỏ nhất trong 7 trạng thái như trên bằng cách di chuyển 6 lên) và đạt được trạng thái đích như yêu cầu và kết thúc thuật toán.

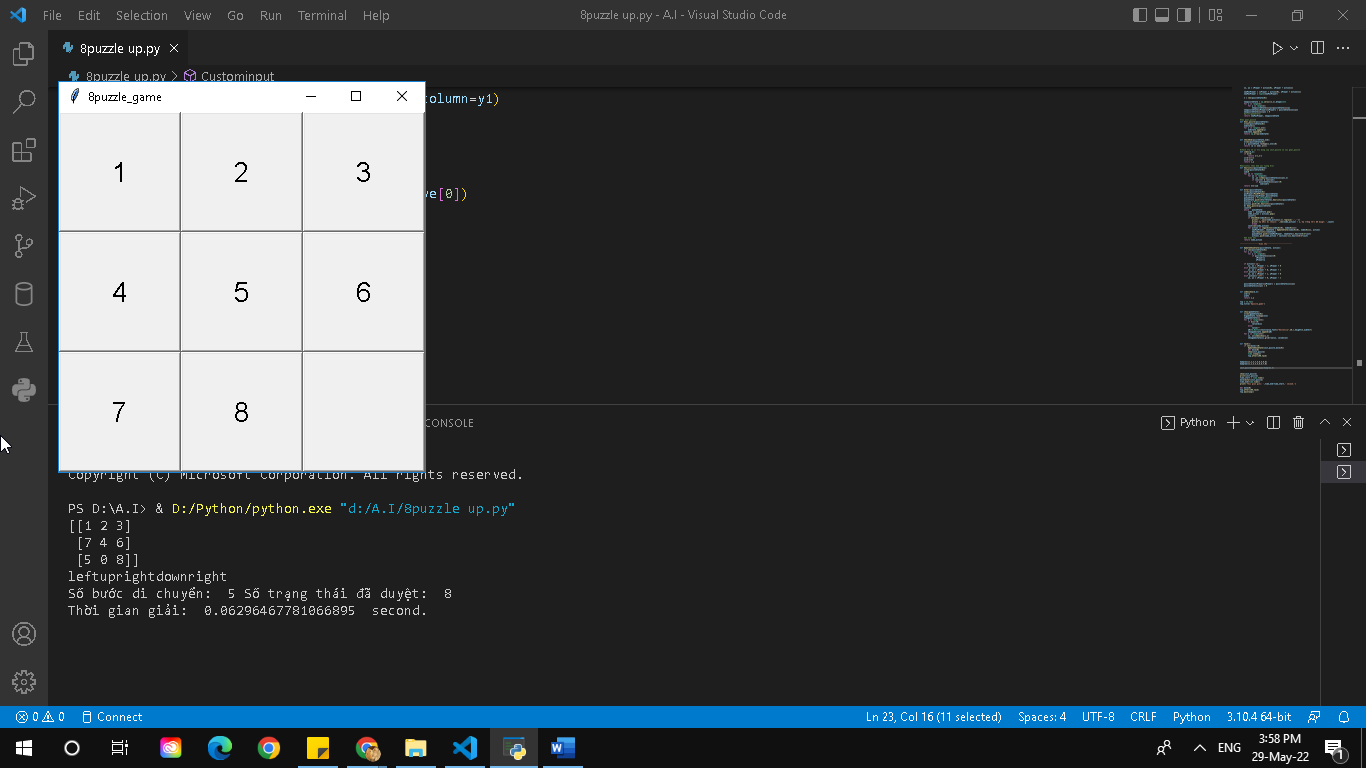
# CHƯƠNG 3. CÁC KẾT QUẢ THỰC NGHIỆM

## 3.1. Các Tình Huống

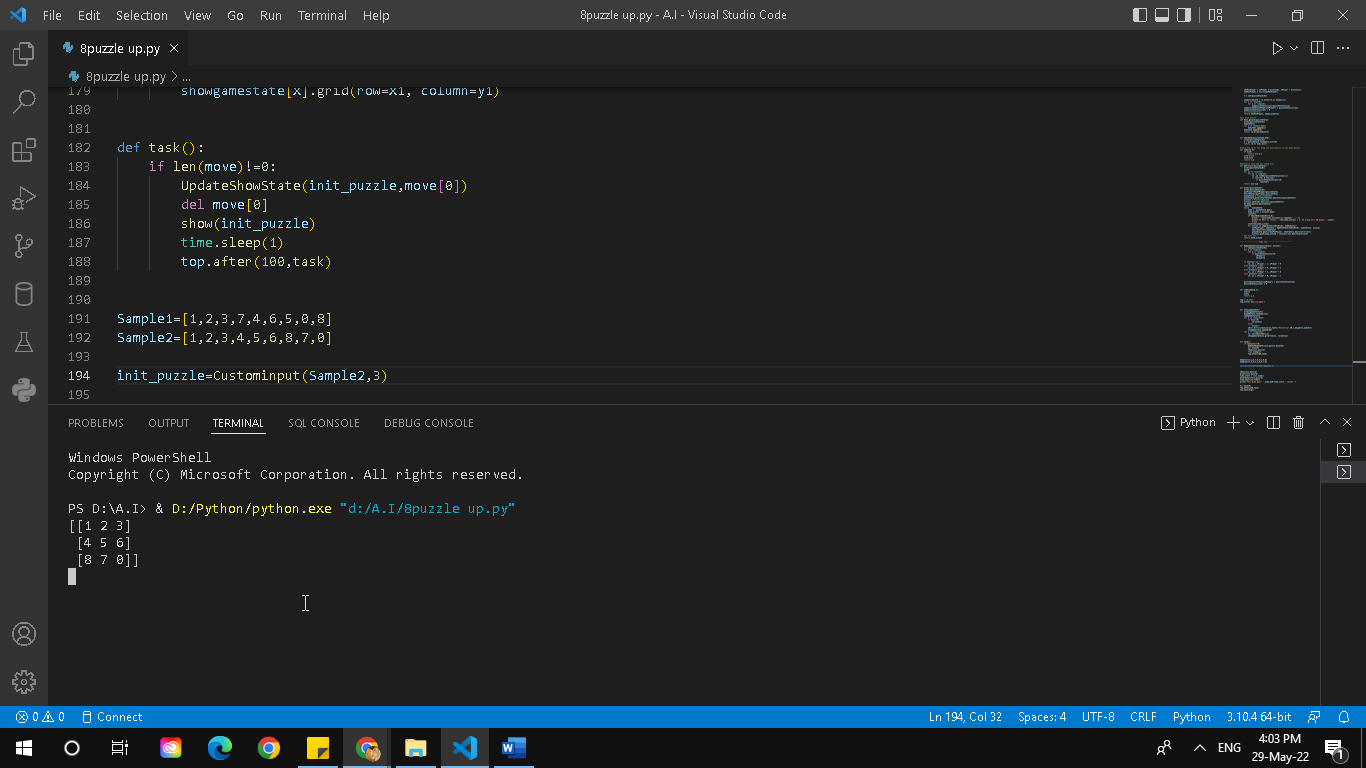
* Chạy Sample 1: Puzzle [[1,2,3][7,4,6][5,0,8]]
  + Chạy chương trình:



* + Kết quả chương trình: Thuật toán A\* giải quyết được Sample 1 với tổng số bước di chuyển là 5 và tổng số trạng thái đã duyệt là 8.



* Chạy Sample 2: Puzzle [[1,2,3][4,5,6][8,7,0]]
  + Chạy chương trình:

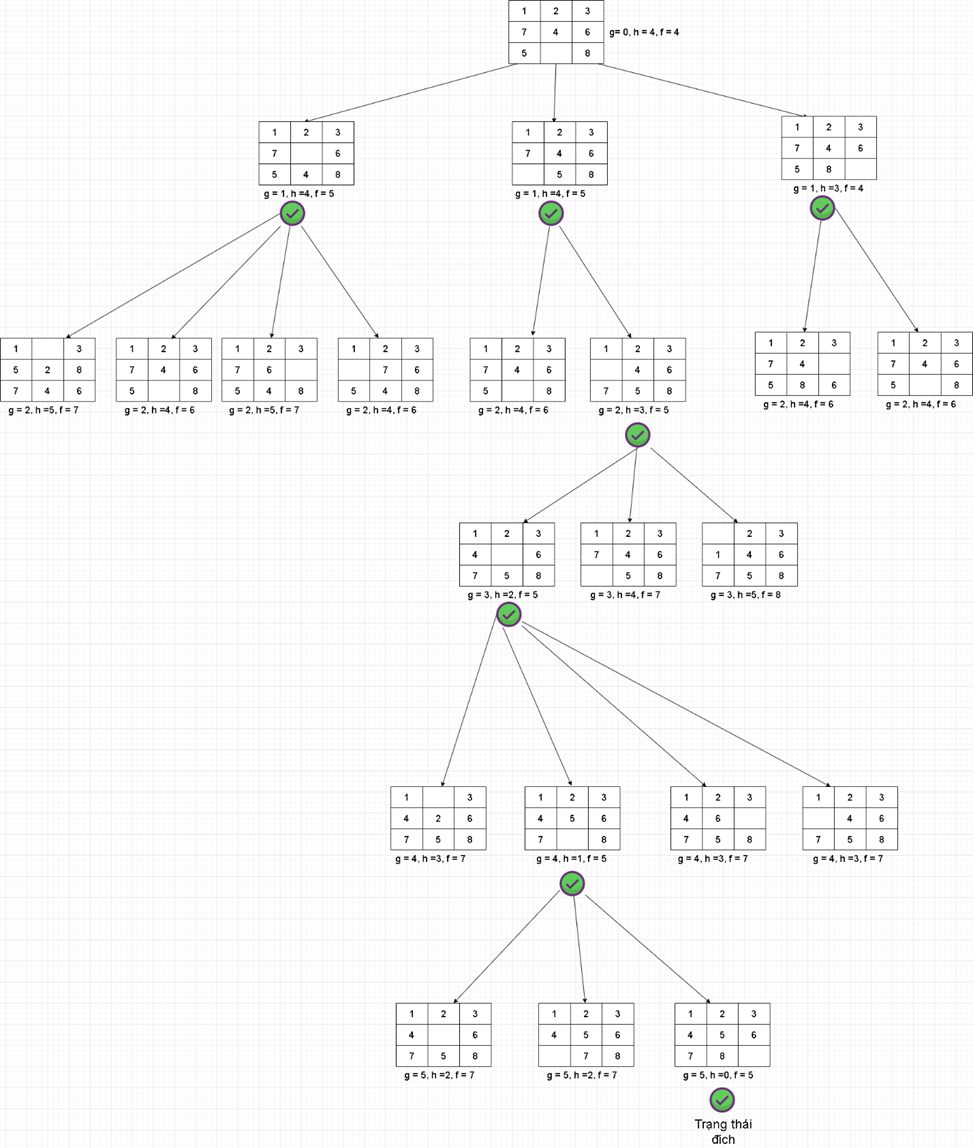


* + Kết quả chương trình: Thuật toán A\* không thể giải quyết được Sample 2.

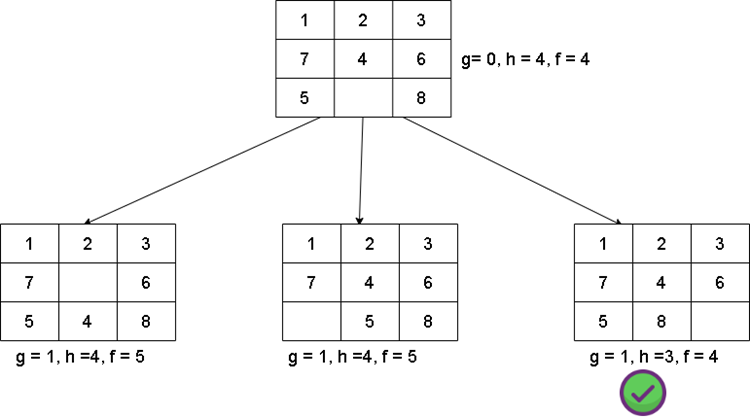
## 3.2. Phân Tích và Đánh Giá

* Cây biểu diễn phân tích Sample 1:
  + Link Drive:

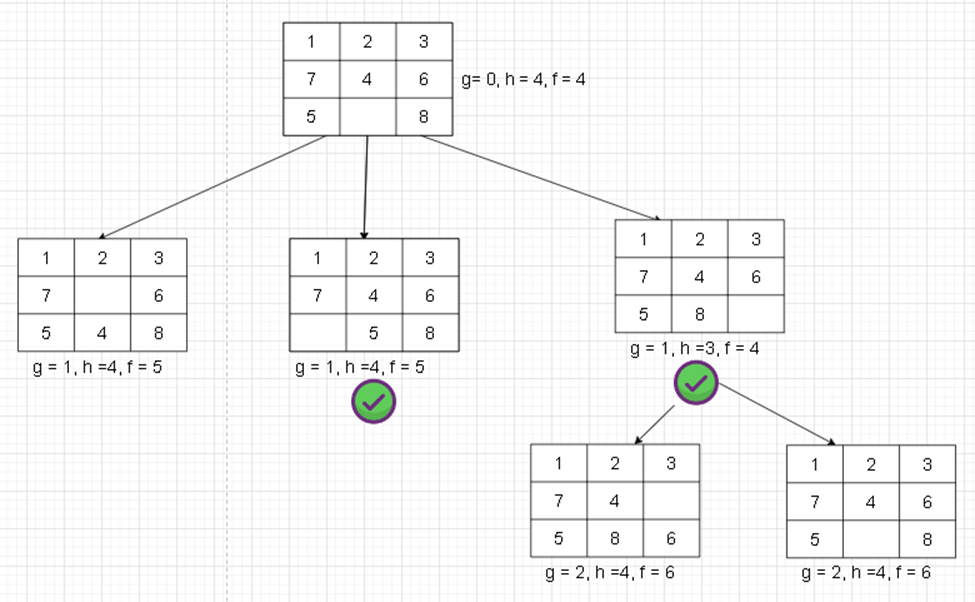
<https://drive.google.com/file/d/1RMudMRlMJE-GxGkmhhvGSAru1J4vf-JO/view?usp=sharing>



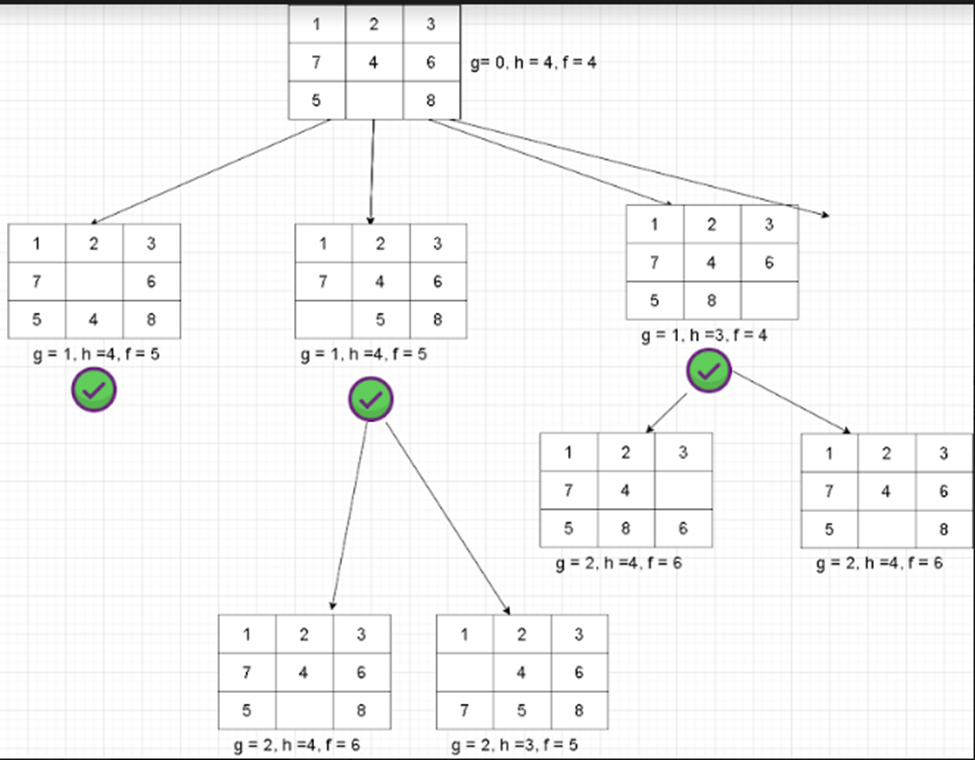
* + Mở rộng và lựa chọn trạng thái tối ưu lần 1:



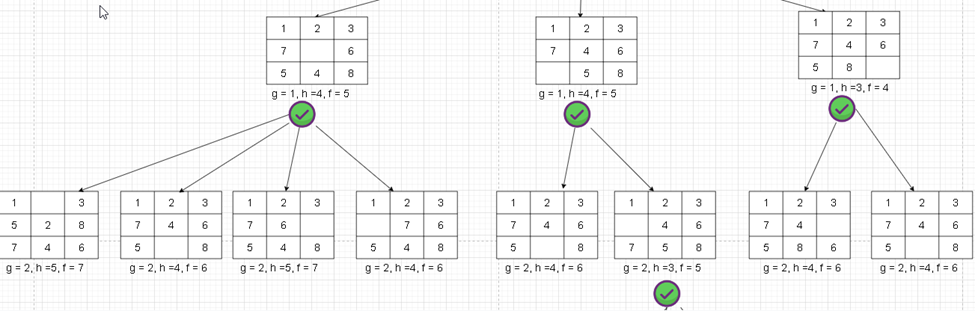
* + - Có 3 trạng thái được mở rộng từ trạng thái ban đầu lần lượt từ trái sang phải là 4 xuống, 5 sang phải và 8 sang trái. Push các trạng thái vào PriorityQueue. Số vị trí sai của các trạng thái so với trạng thái đích từ trái sang phải lần lượt là 4, 4 và 3. Từ đó, ta có được f của các trạng thái lần lượt từ trái sang phải là 5, 5 và 4.
    - => Pop ra trạng thái có f nhỏ nhất (f = 4: 8 sang trái). Kiểm tra: Chưa trùng trạng thái đích => Tiếp tục
  + Mở rộng và chọn trạng thái tối ưu lần 2:



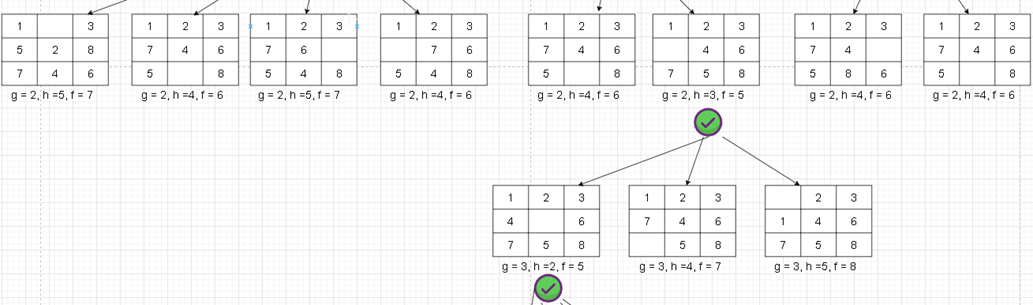
* + - Có 2 trạng thái được mở rộng từ trạng thái tối ưu 1 sau khi được Pop: lần lượt từ trái sang phải là 6 xuống và 8 sang phải. Tiếp tục Push các trạng thái này vào PriorityQueue. Số vị trí sai của các trạng thái so với trạng thái đích đều bằng 4. Từ đó, ta có được f của các trạng thái này đều bằng 6.
    - => Pop ra trạng thái có f nhỏ nhất (f = 5). Kiểm tra: Chưa trùng trạng thái đích => Tiếp tục
  + Mở rộng và chọn trạng thái tối ưu lần 3:



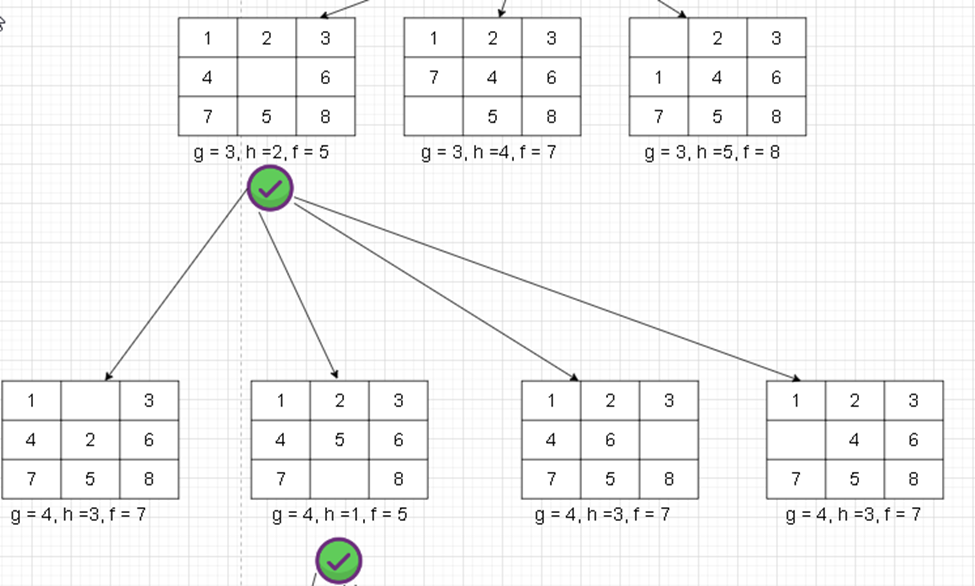
* + - Có 2 trạng thái được mở rộng từ trạng thái tối ưu 2 sau khi được Pop: lần lượt từ trái sang phải là 7 xuống và 5 sang trái. Tiếp tục Push các trạng thái này vào PriorityQueue. Số vị trí sai của các trạng thái so với trạng thái đích lần lượt từ trái sang phải là 4 và 3.. Từ đó, ta có được f của các trạng thái này là 6 và 5..
    - => Pop ra trạng thái có f nhỏ nhất (f = 5) như trong ảnh trên, vì trạng thái này đã được Push vào trong PriorityQueue trước nên sẽ Pop ra trước. Kiểm tra: Chưa trùng trạng thái đích => Tiếp tục
  + Mở rộng và chọn trạng thái tối ưu lần 4:



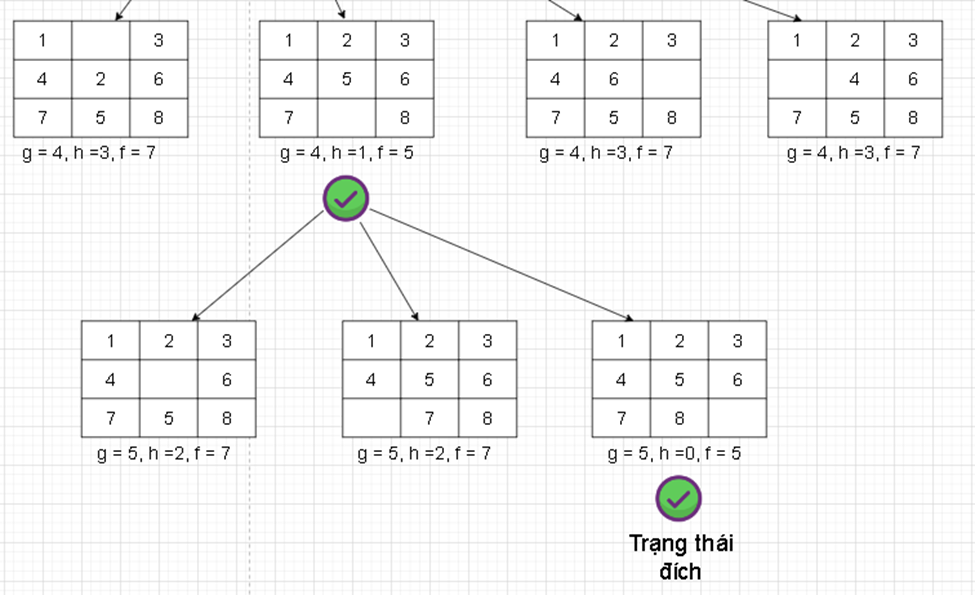
* + - Có 4 trạng thái được mở rộng từ trạng thái tối ưu 3 sau khi được Pop: lần lượt từ trái sang phải là 2 xuống, 4 lên, 6 sang trái và 7 sang phải. Tiếp tục Push các trạng thái này vào PriorityQueue. Số vị trí sai của các trạng thái so với trạng thái đích lần lượt từ trái sang phải là 5, 4, 5, 4. Từ đó, ta có được f của các trạng thái này lần lượt là 7, 6, 7, 6.
    - => Pop ra trạng thái có f nhỏ nhất (f = 5) như trong ảnh trên. Kiểm tra: Chưa trùng trạng thái đích => Tiếp tục
  + Mở rộng và chọn trạng thái tối ưu lần 5:



* + - Có 3 trạng thái được mở rộng từ trạng thái tối ưu 4 sau khi được Pop: lần lượt từ trái sang phải là 4 sang trái, 7 lên và 1 xuống. Tiếp tục Push các trạng thái này vào PriorityQueue. Số vị trí sai của các trạng thái so với trạng thái đích lần lượt từ trái sang phải là 2, 4, 5. Từ đó, ta có được f của các trạng thái này lần lượt là 5, 7, 8.
    - => Pop ra trạng thái có f nhỏ nhất (f = 5) như trong ảnh trên. Kiểm tra: Chưa trùng trạng thái đích => Tiếp tục
  + Mở rộng và chọn trạng thái tối ưu lần 6:

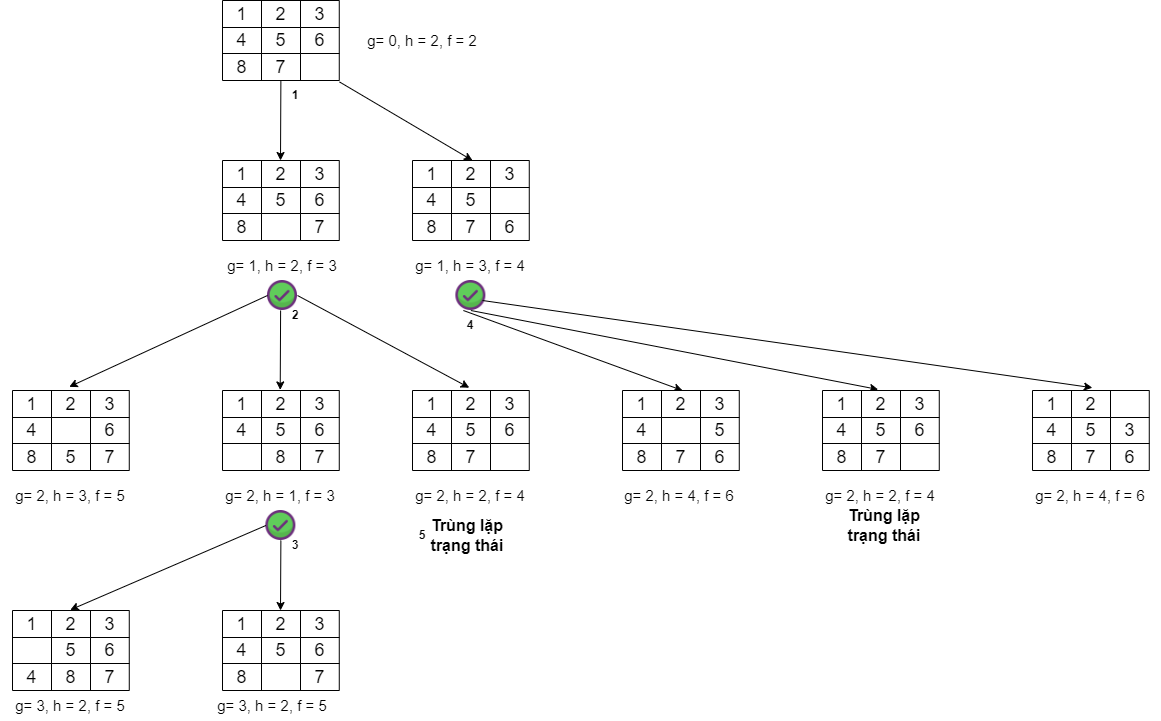


* + - Có 4 trạng thái được mở rộng từ trạng thái tối ưu 5 sau khi được Pop: lần lượt từ trái sang phải là 2 xuống, 5 lên, 6 sang trái và 4 sang phải. Tiếp tục Push các trạng thái này vào PriorityQueue. Số vị trí sai của các trạng thái so với trạng thái đích lần lượt từ trái sang phải là 3, 1, 3, 3. Từ đó, ta có được f của các trạng thái này lần lượt là 7, 5, 7, 7.
    - => Pop ra trạng thái có f nhỏ nhất (f = 5) như trong ảnh trên. Kiểm tra: Chưa trùng trạng thái đích => Tiếp tục
  + Mở rộng và chọn trạng thái tối ưu lần 7:

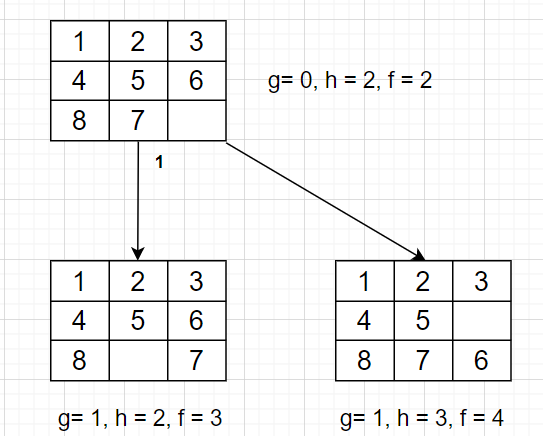


* + - Có 3 trạng thái được mở rộng từ trạng thái tối ưu 6 sau khi được Pop: lần lượt từ trái sang phải là 5 xuống, 7 sang trái và 8 sang phải. Tiếp tục Push các trạng thái này vào PriorityQueue. Số vị trí sai của các trạng thái so với trạng thái đích lần lượt từ trái sang phải là 2, 2, 0. Từ đó, ta có được f của các trạng thái này lần lượt là 7, 7, 5.
    - => Pop ra trạng thái có f nhỏ nhất (f = 5) như trong ảnh trên. Kiểm tra: Trùng trạng thái đích => Kết thúc quá trình tìm kiếm.
* Cây biểu diễn phân tích Sample 2:
  + Link Drive:

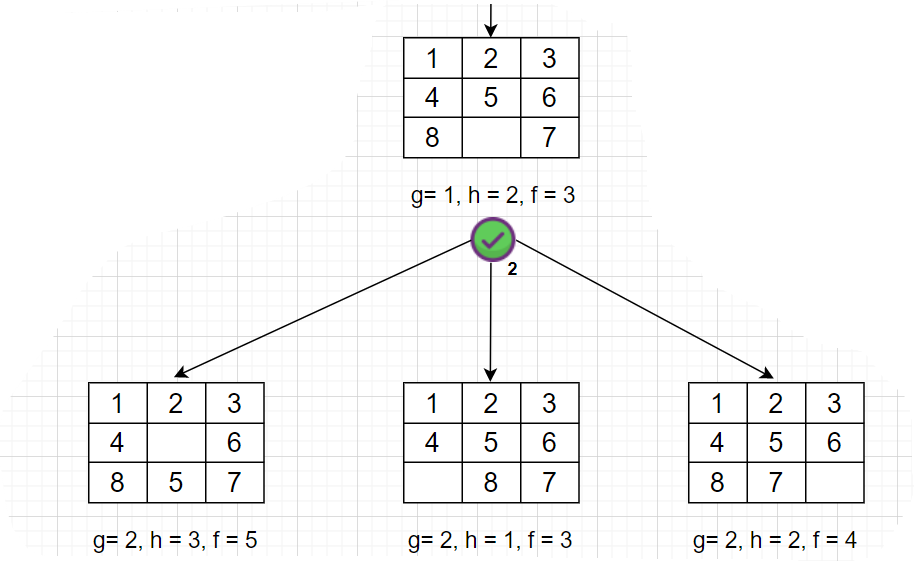
<https://drive.google.com/file/d/138Nmmbwl8gzn6Y1CgDJD2dcsuP6aS03S/view?usp=sharing>



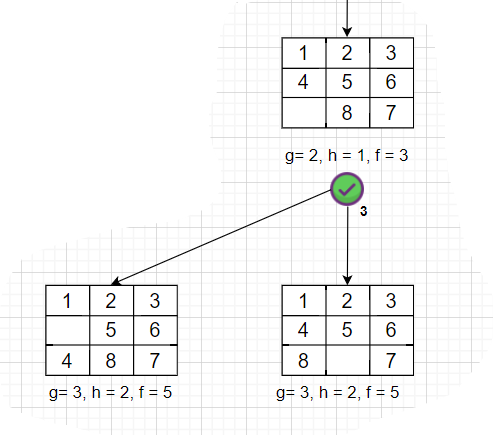
* + Mở rộng và lựa chọn trạng thái tối ưu lần thứ 1:



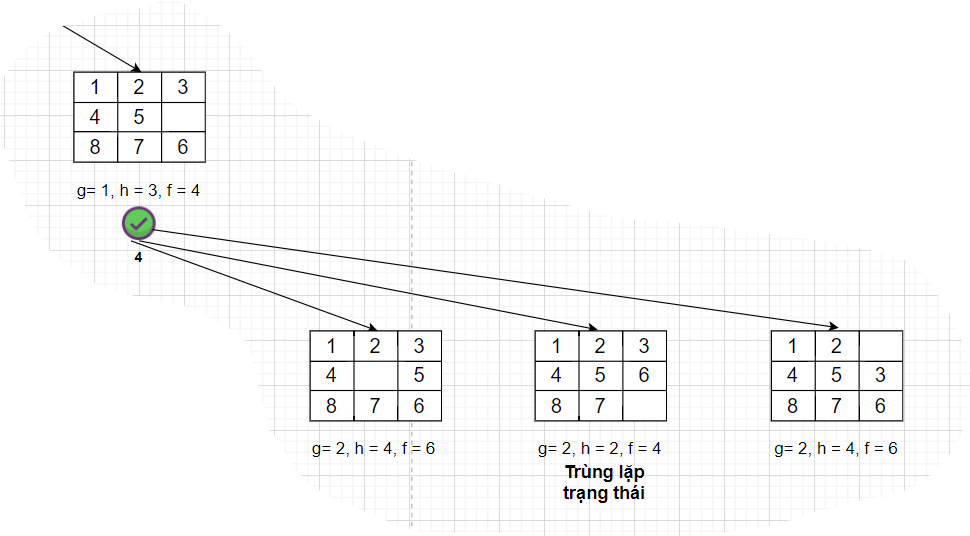
* + - Có 2 trạng thái được mở rộng từ trạng thái ban đầu lần lượt từ trái sang phải là 7 sang phải và 6 xuống. Push các trạng thái vào PriorityQueue. Số vị trí sai của các trạng thái so với trạng thái đích từ trái sang phải lần lượt là 2 va 3. Từ đó, ta có được f của các trạng thái lần lượt từ trái sang phải là 3 và 4.
    - ⇒Pop ra trạng thái có f nhỏ nhất (f = 3: 7 sang phải). Kiểm tra: Chưa trùng trạng thái đích => Tiếp tục
  + Mở rộng và lựa chọn trạng thái tối ưu lần thứ 2:



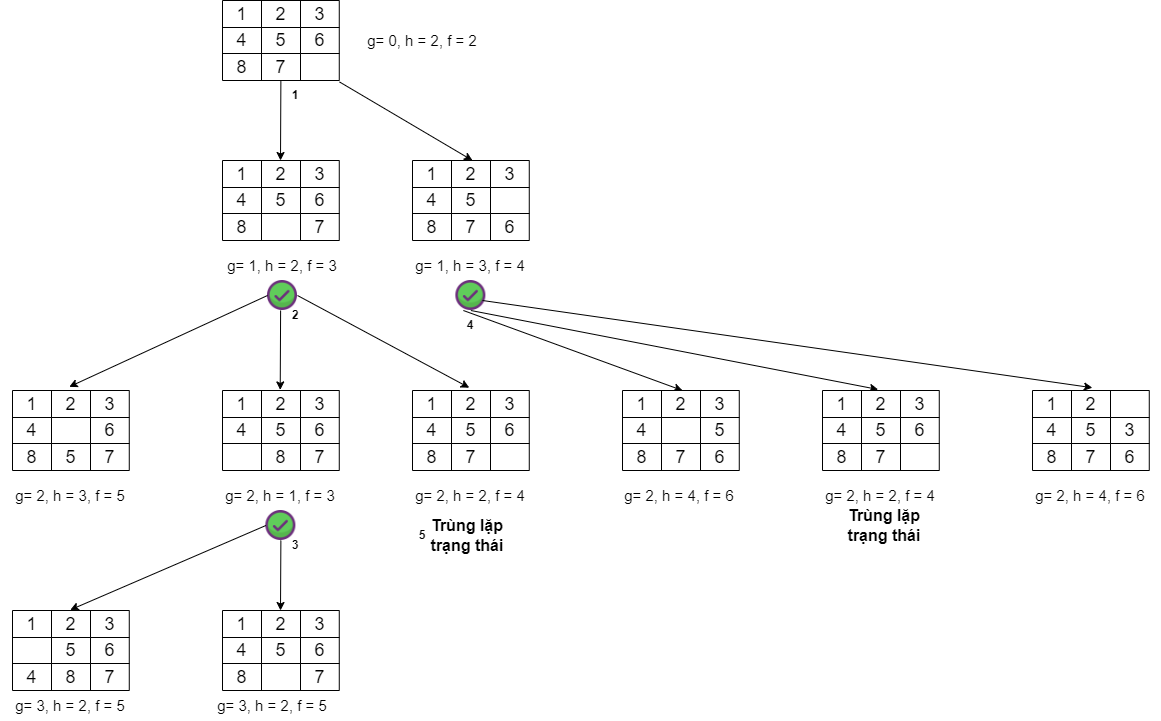
* + - Có 3 trạng thái được mở rộng từ trạng thái tối ưu 1 sau khi được Pop: lần lượt từ trái sang phải là 5 xuống, 8 sang phải và 7 sang trái. Tiếp tục Push các trạng thái này vào PriorityQueue. Số vị trí sai của các trạng thái so với trạng thái đích từ trái sang phải lần lượt là 3, 1 và 2. Từ đó, ta có được f của các trạng thái lần lượt từ trái sang phải là 5, 3 và 4.
    - ⇒Pop ra trạng thái có f nhỏ nhất (f = 3: 8 sang phải). Kiểm tra: Chưa trùng trạng thái đích => Tiếp tục
  + Mở rộng và lựa chọn trạng thái tối ưu lần thứ 3:



* + - Có 2 trạng thái được mở rộng từ trạng thái tối ưu 2 sau khi được Pop: lần lượt từ trái sang phải là 4 xuống và 8 sang trái. Tiếp tục Push các trạng thái này vào PriorityQueue. Số vị trí sai của các trạng thái so với trạng thái đích từ trái sang phải lần lượt là 2 và 2. Từ đó, ta có được f của các trạng thái lần lượt từ trái sang phải là 5 và 5.
    - ⇒ Pop ra trạng thái có f nhỏ nhất (f = 4). Kiểm tra: Chưa trùng trạng thái đích => Tiếp tục.
    - Lúc này có hai trạng thái có f = 4 lần lượt là:
      * Trạng thái A: [[1,2,3][4,5,6][8,7,0]] (Trùng lặp trạng thái ban đầu ⇒ Tạm thời bỏ qua, sẽ được đề cập sau để phục vụ cho việc giải thích vì sao thuật toán A\* không giải quyết được Sample 2)
      * Trạng thái B: [[1,2,3][4,5,0][8,7,6]] (Chọn để mở rộng và lựa chọn trạng thái tối ưu lần thứ 4)
  + Mở rộng và lựa chọn trạng thái tối ưu lần thứ 4:



* + - Có 3 trạng thái được mở rộng từ trạng thái B tối ưu 3 sau khi được Pop: lần lượt từ trái sang phải là 5 sang phải, 6 lên và 3 xuống. Tiếp tục Push các trạng thái này vào PriorityQueue. Số vị trí sai của các trạng thái so với trạng thái đích từ trái sang phải lần lượt là 4, 2 và 4. Từ đó, ta có được f của các trạng thái lần lượt từ trái sang phải là 6, 4 và 6.
    - ⇒ Pop ra trạng thái có f nhỏ nhất (Trạng thái C: [[1,2,3][4,5,6][8,7,0]] với f = 4). Kiểm tra: Chưa trùng trạng thái đích => Tiếp tục.
  + Để ý ta thấy cả trạng thái A và trạng thái C đều có dạng [[1,2,3][4,5,6][8,7,0]], trùng lặp với trạng thái ban đầu của Sample khiến cho thuật toán rơi vào vòng lặp vô hạn.



* + ⇒Lý giải vì sao thuật toán A\* không đủ tối ưu để giải quyết Sample 2 [[1,2,3][4,5,6][8,7,0]] vì bị rơi vào vòng lặp vô hạn không thể tìm ra cách giải quyết thích hợp để giải quyết bài toán.

# CHƯƠNG 4. KẾT LUẬN

## 4.1. Các Kết Quả Đạt Được

* Nhận xét:
  + A\* có tính “đầy đủ” (hoàn chỉnh) - có nghĩa nó sẽ luôn tìm thấy lời giải nếu bài toán có lời giải (chưa xét đến giới hạn thời gian và xử lý bộ nhớ)
    - Nếu không gian các trạng thái là hữu hạn và có 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)
  + A\* còn có tính “tối ưu”. Không có thuật toán nào cũng sử dụng hàm đánh giá heuristic đó mà chỉ phải mở rộng ít nút hơn A\*, A\* có tính hiệu quả một cách tối ưu.
    - Muốn A\* tối ưu thì hàm h(n) phải có tính chấp nhận được - tức là nó không bao giờ đánh giá cao hơn chi phí nhỏ nhất thực sự của việc đi tới đích (trong trường hợp tập đóng).
    - Nếu không dùng tập đóng, thì h(n) phải có tính đơn điệu - tức là nó không bao giờ đánh giá chi phí đi từ một nút tới một nút kề nó cao hơn chi phí thực.
    - Một cách hình thức, nếu m là nút kế tiếp của n thì:

**h(n) <= g(m) - g(n) + h(m)**

* + - Tuy nhiên, không thể đảm bảo A\* luôn chạy nhanh hơn các thuật toán khác
* Độ phức tạp:
  + Độ phức tạp của A\* phụ thuộc vào hàm đánh giá h(n). Độ phức tạp đó sẽ là hàm đa thức nếu h(n) thỏa mãn điều kiện sau:

**h(n) – h\*(n)| O(log h\*(n))**

* + - Với h\*(n) là hàm cho kết quả là chi phí chính xác để đi từ nút đang xét n tới đích.
    - Từ công thức trên ta thấy được: sai số của h(n) không nên tăng nhanh hơn log h\*(n) ⇒ Nếu không, độ phức tạp sẽ tăng lên hàm mũ dẫn đến vấn đề xử lý bộ nhớ.

## 4.2. Những Hạn Chế và Hướng Phát Triển

- Có hai hạn chế đáng chú ý là:

* Hạn chế 1: Trạng thái đầu truyền vào là một mảng random nên không thể đảm bảo chương trình luôn có thể tìm ra lời giải thích hợp để giải quyết bài toán với mảng random này được.
* Hướng phát triển: Tạo sẵn một list các mảng đầu vào đã được kiểm tra từ trước đảm bảo chương trình luôn có thể giải quyết được. Sau đó cho random ngẫu nhiên giữa các mảng này để làm trạng thái đầu truyền vào.
* Hạn chế 2: Chưa có hàm kiểm tra để tránh việc xét lặp lại các trạng thái nên dẫn đến việc lặp lại các trạng thái đã xét, thường sẽ dẫn đến tốn bộ nhớ lưu trữ, trong trường hợp tệ nhất khiến chương trình rơi vào vòng lặp vô hạn.
* Hướng phát triển: Xây dựng để kiểm tra các trạng thái được Push vào và Pop ra đã từng được xét trước hay chưa, nếu rồi thì bỏ qua trạng thái đó, còn không thì tiếp tục thực hiện chương trình.

# TÀI LIỆU THAM KHẢO

1. Slide bài giảng môn Trí Tuệ Nhân Tạo - Giảng Viên: TS. Đặng Ngọc Hoàng Thành
2. Stuart Russell and Peter Norvig (2009), Artificial Intelligence: A Modern Approach 3rd Edition, Pearson, United Kingdom.
3. Prateek Joshi (2017), Artificial Intelligence with Python: A Comprehensive Guide to Building Intelligent Apps for Python Beginners and Developers, Packt Publishing, United Kingdom.
4. <https://www.geeksforgeeks.org/a-search-algorithm/>
5. <https://github.com/baokhanhle123/HK211_DoAnAI-1234-HCMUT>

# PHỤ LỤC

Link Github:

https://github.com/sonbao0901/8-puzzle-game