**DỊP KIM YẾN**

**Mã sMã só SV: 21DH112239**

**TRƯỜNG ĐẠI HỌC NGOẠI NGỮ- TIN HỌC TPHCM KHOA CÔNG NGHỆ THÔNG TIN**

**SỬ DỤNG GIẢI THUẬT BEST FIRST SEARCH ĐỂ GIẢI BÀI TOÁN ĐỐ 8-MIẾNG ( 8-PUZZLE)**

**ĐỒ ÁN MÔN HỌC: TRÍ TUỆ NHÂN TẠO**

**Giảng viên hướng dẫn:**

**PGS.TS DƯƠNG TUẤN ANH**

**Sinh viên thực hiện:**

**DƯƠNG TẤN ĐẠT**

**Mã số SV: 21DH113556**

**CAO THẢO VÂN**

**Mã số SV: 21DH113129**

**TP.Hồ Chí Minh, tháng 07 năm 2023**



# MỤC LỤC

[MỤC LỤC 2](#_Toc141131151)

[MỤC LỤC ẢNH 2](#_Toc141131152)

[MỤC LỤC BẢNG 3](#_Toc141131153)

[1. Giới thiệu thuật toán Best First Search 4](#_Toc141131154)

[**1.1. Thuật toán Best First Search là gì?** 4](#_Toc141131155)

[**1.2. Ý tưởng thuật toán** 4](#_Toc141131156)

[**1.3. Mô tả thuật toán bằng mã giả** 4](#_Toc141131157)

[**1.4. Mô tả bằng thuật toán Python** 5](#_Toc141131158)

[2. Áp dụng giải thuật Best First Search trong bài toán 6](#_Toc141131159)

[**2.1. Mô tả bài toán** 6](#_Toc141131160)

[**2.2.Phân tích giải thuật Best First Search trong bài toán bằng mã giả** 7](#_Toc141131161)

[**2.2.1. Thực nghiệm 1** 7](#_Toc141131162)

[**2.2.2. Thực nghiệm 2** 13](#_Toc141131163)

[3. Tổng kết 21](#_Toc141131164)

[**3.1. Thuật toán Best first search trong bài toán 8 miếng** 22](#_Toc141131165)

[**3.2. Phương hướng phát triển thuật toán trong tương lai** 22](#_Toc141131166)

[Tài liệu tham khảo 24](#_Toc141131167)

# MỤC LỤC ẢNH

[**Hình 1. Trạng thái ban đầu và trạng thái kết thúc của thực nghiệm 1 7**](#_Toc141129661)

[**Hình 2. Diễn tiến giải thuật Best First Search bài toán 8-PUZZLE thực nghiệm 1 8**](#_Toc141129662)

[**Hình 3. Kết quả đoạn mã Python 12**](#_Toc141129663)

[**Hình 4. Trạng thái ban đầu và trạng thái kết thúc của thực nghiệm 2 13**](#_Toc141129664)

[**Hình 5. Diễn tiến giải thuật Best First Search bài toán 8-PUZZLE thực nghiệm 2 15**](#_Toc141129665)

[**Hình 6. Kết quả đoạn mã Python 20**](#_Toc141129666)

# MỤC LỤC BẢNG

[**Bảng 1. Kết quả thực nhiệm 1 9**](#_Toc141131001)

[**Bảng 2. Kết quả thực nghiệm 2 17**](#_Toc141131002)

# 1. Giới thiệu thuật toán Best First Search

## **1.1. Thuật toán Best First Search là gì?**

Khác với 02 thuật toán DFS (tìm kiếm theo chiều sau) và BFS (tìm kiếm theo chiều rộng) là tìm kiếm một cách mù quáng mà không có sự ước tính về chi phí.

Thì thuật toán Tìm kiếm tốt nhất trước (Best first search) thuộc loại tìm kiếm dựa vào chi phí ước lượng từ nút gốc đến đích, có sử dụng thông tin hỗ trợ để di chuyển tới trạng thái đích nhanh hơn.

Thuật toán này là sự kết hợp giữa DFS (tìm kiếm theo chiều sau) và BFS (tìm kiếm theo chiều rộng) từ đó mở rộng đánh giá, tìm kiếm theo các nút có nhiều tiềm năng chứa trạng thái đích hơn các nút khác.

## **1.2. Ý tưởng thuật toán**

Sử dụng hàm đánh giá (heuristic function) để ước lượng giá trị của mỗi trạng thái và tìm kiếm dựa trên ước lượng này. Nó tập trung vào những trạng thái được xem là tiềm năng tốt nhất dựa trên hàm đánh giá.

Sử dụng hàng đợi ưu tiên (priority queue) để lưu trữ các trạng thái và sắp xếp chúng dựa trên giá trị đánh giá (ưu tiên). Trạng thái với giá trị đánh giá cao hơn sẽ được xem xét trước.

Việc tìm kiếm sẽ bắt đầu tại nút gốc và tiếp tục bằng cách duyệt lần lượt các nút tiếp theo có giá trị của hàm đánh giá (chi phí) là thấp nhất so với các nút đang trong hàng đợi.

## **1.3. Mô tả thuật toán bằng mã giả**

1. Đặt nút khởi đầu S vào list OPEN, tính f(S), và gắn trị hàm này với S.

Gắn một con trỏ null với S.

1. Nếu list OPEN rỗng, in ra thông báo “FAILURE” và dừng.
2. Chọn một nút N từ list OPEN sao cho f(N) ≤ f(M) với mọi nút M trong OPEN
3. Đặt nút N vào list CLOSED và xóa bỏ N ra khỏi list OPEN.
4. Nếu N là nút đích, trả về lối đi tìm thấy từ nút S đến nút N dựa vào chuỗi con trỏ khởi đầu bằng con trỏ liên kết với nút N. Và dừng giải thuật.
5. Sinh ra danh sách L gồm những nút kế cận của nút N. Loại trừ ra khỏi L những nút có hiện diện trong CLOSED.
6. Với mỗi nút J trong danh sách L mà không hề hiện diện trong OPEN :
   1. Đặt J vào OPEN.
   2. gắn J với con trỏ chỉ từ J ngược về N.
7. Quay lại Step 2.

## **1.4. Mô tả bằng thuật toán Python**

PREDECESSOR = {} COUNT = 0

**def** bestfs(start, goal):

#Thực hiện tìm kiếm tốt nhất đầu tiên từ đầu cho mục tiêu.

OPEN = [start] # Bước 1

CLOSED = []

PREDECESSOR[start] = None

**while** OPEN != []: # Bước 2 n = deleteMin(OPEN, FN) # Bước 3 CLOSED.append(n) # Bước 4 **global** COUNT COUNT += 1 **if** n == goal:

**return** extractPath(n) # Bước 5 lst = successors(n) # Bước 6 lst = listDifference(lst, CLOSED)

# successors được thêm vào tập OPEN. OPEN = lst +listDifference(OPEN, lst)  **for** elt **in** lst: # Bước 7

PREDECESSOR[elt] = n

# kết thúc vòng lặp. Bước 8

**Các hàm hỗ trợ:**

- Hàm deleteMin:

Hàm deleteMin để xác định phần tử “tốt nhất” dựa vào một hàm đánh giá. Hàm sẽ xóa bỏ phần tử này ra khỏi danh sách và trả nó về trình gọi.

**def** deleteMin(lst, fn):

minVal = 9999 # một giá trị cực lớn

minElt = None

**for** e **in** lst:

temp = fn[e] # fn là một tự điển

**if** temp < minVal: minVal = temp; minElt = e

lst.remove(minElt)

**return** minElt

- Hàm Test:

Để thử hàm bestfs, chúng ta định nghĩa hàm test như sau.

**def** test():

start = ’A’

goal = ’B’

print ("Starting a best-first graph search from " + start)

path = bestfs(start, goal)

print(path)

**global** COUNT

print(str(COUNT) + " nodes expanded during the search.")

# 2. Áp dụng giải thuật Best First Search trong bài toán

## **2.1. Mô tả bài toán**

Sử dụng giải thuật best-first-search để giải bài toán đố 8-Miếng (8-puzzle). Hàm ước lượng h đánh giá độ tốt của trạng thái hiện hành tính bằng số ô mà trạng thái hiện hành sai biệt so với trạng thái đích. Mỗi khi trạng thái kế tiếp được thêm vào hàng đợi ưu tiên (priority queue) theo giá trị , và ta sẽ tiếp tục duyệt đến khi đạt được trạng thái đích hoặc không còn trạng thái nào trong hàng đợi.

Yêu cầu: Dịch chuyển các miếng sao cho từ trạng thái ban đầu đạt đến một trạng thái đích nào đó  
Luật chơi: Cho phép trượt các miếng đến ô trống kề nó, không cho phép dịch huyển các miếng theo đường chéo.

Thông tin đánh giá heuristic của bài toán là số miếng đặt không đúng chỗ tại mỗi trạng thái khi nó được so sánh với trạng thái đích.  
Trạng thái nào có ít miếng đặt không đúng chỗ nhất so với trạng thái đích có là trạng thái tốt nhất được chọn duyệt qua để tiến đến đích.  
Nếu có hai trạng thái có cùng thông tin đánh giá heuristic, thì lúc này trạng thái được chọn để duyệt là trạng thái gần trạng thái gốc nhất của đồ thị. Trạng thái này sẽ có đường ngắn nhất để đi đến đích.

Khoảng cách từ trạng thái ban đầu đến các con cháu của nó được đo bằng cách tính độ sâu cho mỗi trạng thái. Tính 0 cho trạng thái ban đầu và được tăng lên 1 cho mỗi mức kế theo phép tính độ sâu này phản ánh số phép dịch chuyển thực sự mà đã được sử dụng đi từ trạng thái ban đầu trong việc tìm kiếm đến mỗi trạng thái con cháu, vì thế nó phải được bổ sung vào việc đánh giá heuristic.

Do đó để có đầy đủ thông tin đánh giá heuristic cho mỗi trạng thái trong không gian trạng thái tìm kiếm của bài toán, một hàm đánh giá heuristic f được định nghĩa đó là tổng của hai thành phần: f(n) = g(n) + h(n)

- g(n) : số lần chuyển vị trí đã thực hiện

- h(n) : số miếng đặt không đúng chỗ so với trạng thái đích.

## **2.2.Phân tích giải thuật Best First Search trong bài toán bằng mã giả**

### **2.2.1. Thực nghiệm 1**

A close-up of a grid

Description automatically generated

Hình 1. Trạng thái ban đầu và trạng thái kết thúc của thực nghiệm 1

A diagram of a diagram

Description automatically generated

Hình 2. Diễn tiến giải thuật Best First Search bài toán 8-PUZZLE thực nghiệm 1

#### **2.2.1.1 Best First Search áp dụng vào thực nghiệm 1**

- Số tại đỉnh của mỗi trạng thái chỉ số thứ tự của trạng thái được chọn để duyệt từ danh sách OPEN  
- Các trạng thái không được đánh số thứ tự vì chúng vẫn có mặt trên danh sách OPEN khi giải thuật đã được hội tụ.  
- Các chữ cái trong danh sách biểu diễn các trạng thái và các chữ số biểu diễn các giá trị của hàm đánh giá heuristic tại các trạng thái đó.

Qua đồ thị ta có kết quả các trạng thái của danh sách OPEN và CLOSED được phát như sau:  
**Bước 0 (Khởi tạo):** Bắt đầu với trạng thái ban đầu là A4. Đặt trạng thái này vào hàng đợi OPEN.

OPEN = [A4]; CLOSED = []

**Bước 1:** Lấy trạng thái có giá trị đánh giá cao nhất từ hàng đợi OPEN, trong trường hợp này là A4. Đặt A4 vào CLOSED.

OPEN = [C4, B6, D6]; CLOSED = [A4]

**Bước 2:** Tạo ra các trạng thái kế tiếp từ A4 và thêm chúng vào hàng đợi OPEN. Đồng thời, đánh giá các trạng thái này bằng hàm heuristic và sắp xếp lại hàng đợi OPEN theo giá trị đánh giá.

OPEN = [E5, F5, G6, B6, D6]; CLOSED = [A4, C4]

**Bước 3:** Lấy trạng thái có giá trị đánh giá cao nhất từ hàng đợi OPEN, trong trường hợp này là E5. Đặt E5 vào CLOSED.

OPEN = [F5, H6, G6, B6, D6, I7]; CLOSED = [A4, C4, E5]

**Bước 4:** Tạo ra các trạng thái kế tiếp từ E5 và thêm chúng vào hàng đợi OPEN. Đồng thời, đánh giá các trạng thái này bằng hàm heuristic và sắp xếp lại hàng đợi OPEN theo giá trị đánh giá.

OPEN = [J5, H6, G6, B6, D6, K7, I7]; CLOSED = [A4, C4, E5, F5]

**Bước 5:** Lấy trạng thái có giá trị đánh giá cao nhất từ hàng đợi OPEN, trong trường hợp này là J5. Đặt J5 vào CLOSED.

OPEN = [L5, H6, G6, B6, D6, K7, I7]; CLOSED = [A4, C4, E5, F5, J5]

**Bước 6:** Tạo ra các trạng thái kế tiếp từ J5 và thêm chúng vào hàng đợi OPEN. Đồng thời, đánh giá các trạng thái này bằng hàm heuristic và sắp xếp lại hàng đợi OPEN theo giá trị đánh giá.

OPEN = [M5, H6, G6, B6, D6, K7, I7, N7]; CLOSED = [A4, C4, E5, F5, J5, L5]

**Bước 7:** Lấy trạng thái có giá trị đánh giá cao nhất từ hàng đợi OPEN, trong trường hợp này là M5. Đây là trạng thái đích, do đó quá trình giải thuật kết thúc với thành công.

Kết quả cuối cùng là M5, đạt được bằng cách thực hiện các bước đi từ A4 đến M5.

|  |  |  |  |
| --- | --- | --- | --- |
| Lần | n | OPEN | CLOSED |
| 0  1  2  3  4  5  6  7 | A4  C4  E5  F5  J5  L5  M5 | [A4]  [C4, B6, D6]  [E5, F5, G6, B6, D6]  [F5, H6, G6, B6, D6, I7]  [J5, H6, G6, B6, D6, K7, I7]  [L5, H6, G6, B6, D6, K7, I7]  [M5, H6, G6, B6, D6, K7, I7, N7] | []  [A4]  [A4, C4]  [A4, C4, E5]  [A4, C4, E5, F5]  [A4, C4, E5, F5, J5]  [A4, C4, E5, F5, J5, L5] |

Bảng 1. Kết quả thực nhiệm 1

#### **2.2.1.2 Đoạn mã Python giải thuậtBest First Searchthực nghiệm 1**

import math

from queue import PriorityQueue

# Function to check if the achieved state is the end state

def is\_goal(state):

goal\_state = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]

return state == goal\_state

# Function to generate a list of child states from the current state

def generate\_states(state):

states = []

moves = [(0, 1), (0, -1), (1, 0), (-1, 0)] # Travel directions (right, left, down, up)

# Find the position of the blank cell (0)

empty\_pos = None

for i in range(3):

for j in range(3):

if state[i][j] == 0:

empty\_pos = (i, j)

break

# Create new states by moving the blank cell

for move in moves:

dx, dy = move

new\_pos = (empty\_pos[0] + dx, empty\_pos[1] + dy)

# Check if the new state is valid

if 0 <= new\_pos[0] < 3 and 0 <= new\_pos[1] < 3:

new\_state = [row[:] for row in state] # Copy the current state

new\_state[empty\_pos[0]][empty\_pos[1]] = new\_state[new\_pos[0]][new\_pos[1]]

new\_state[new\_pos[0]][new\_pos[1]] = 0

states.append(new\_state)

return states

# Function to calculate the h(n) value - heuristic

def calculate\_heuristic(state):

goal\_state = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]

misplaced\_tiles = 0

for i in range(3):

for j in range(3):

if state[i][j] != goal\_state[i][j]:

misplaced\_tiles += 1

return misplaced\_tiles

# Function that implements the Best-First Search algorithm

def best\_first\_search(start\_state):

queue = PriorityQueue()

queue.put((0, start\_state)) # Add the initial state to the priority queue

visited = set() # Keep track of visited states

visited.add(tuple(map(tuple, start\_state))) # Add the initial state to the visited set

visited\_states = [] # Keep track of visited states in order

while not queue.empty():

current\_node = queue.get()[1] # Get the next state from the priority queue

visited\_states.append(current\_node) # Add the current state to the visited states list

if is\_goal(current\_node): # Check if the current state has reached the end state

return visited\_states

next\_states = generate\_states(current\_node) # Generate child states from the current state

# Add child states to the priority queue with heuristic priority value

for state in next\_states:

if tuple(map(tuple, state)) not in visited: # Check if the state has been visited

visited.add(tuple(map(tuple, state))) # Mark the state as visited

priority = calculate\_heuristic(state)

queue.put((priority, state))

return None

# Function to print the steps that move from the initial state to the end state

def print\_solution(solution):

if solution is None:

print("No solution found.")

return

print("Solution:")

for i, state in enumerate(solution):

print(f"Step {i}:")

for row in state:

print(row)

print()

# The initial state of the 8-Puzzle problem

start\_state = [[2, 8, 3], [1, 6, 4], [7, 0, 5]]

# Implement the Best-First Search algorithm

solution = best\_first\_search(start\_state)

# Print the result

print\_solution(solution)

A screenshot of a computer

Description automatically generated

Hình 3. Kết quả đoạn mã Python

#### **2.2.1.3 Nhận xétthực nghiệm 1**

Trong phân tích chi tiết cho bài toán cụ thể này, chúng ta đã bắt đầu với trạng thái ban đầu là [2, 8, 3, 1, 6, 4, 7, 0, 5] và trạng thái đích là [1, 2, 3, 8, 0, 4, 7, 6, 5]. Sau mỗi bước của quá trình giải thuật Best-First Search, chúng ta đã phân tích các trạng thái và thêm vào hàng đợi ưu tiên, kiểm tra và lưu trữ trạng thái đã xét trong CLOSED.

Quá trình này tiếp tục cho đến khi tìm thấy trạng thái đích. Trạng thái đích trong bài toán này là [1, 2, 3, 8, 0, 4, 7, 6, 5]. Khi tìm thấy trạng thái này, quá trình giải thuật kết thúc.

Độ dài của lời giải là 5 bước. Điều này có nghĩa là cần thực hiện 5 lần di chuyển ô trống để chuyển từ trạng thái ban đầu sang trạng thái kết thúc.

### **2.2.2. Thực nghiệm 2**

A screenshot of a game

Description automatically generated

Hình 4. Trạng thái ban đầu và trạng thái kết thúc của thực nghiệm 2

A diagram of a diagram

Description automatically generated

A diagram of a diagram

Description automatically generated

Hình 5. Diễn tiến giải thuật Best First Search bài toán 8-PUZZLE thực nghiệm 2

#### **2.2.2.1 Best First Search áp dụng vào thực nghiệm 2**

- Số tại đỉnh của mỗi trạng thái chỉ số thứ tự của trạng thái được chọn để duyệt từ danh sách OPEN  
- Các trạng thái không được đánh số thứ tự vì chúng vẫn có mặt trên danh sách OPEN khi giải thuật đã được hội tụ.  
- Các chữ cái trong danh sách biểu diễn các trạng thái và các chữ số biểu diễn các giá trị của hàm đánh giá heuristic tại các trạng thái đó.

Qua đồ thị ta có kết quả các trạng thái của danh sách OPEN và CLOSED được phát như sau:  
**Bước 0 (Khởi tạo):** Bắt đầu với trạng thái ban đầu là A6. Đặt trạng thái này vào hàng đợi OPEN.

OPEN = [A6]; CLOSED = []

**Bước 1:** Lấy trạng thái có giá trị đánh giá cao nhất từ hàng đợi OPEN, trong trường hợp này là A6. Đặt A6 vào CLOSED.

OPEN = [B6, D7, C8]; CLOSED = [A6]

**Bước 2:** Tạo ra các trạng thái kế tiếp từ A6 và thêm chúng vào hàng đợi OPEN. Đồng thời, đánh giá các trạng thái này bằng hàm heuristic và sắp xếp lại hàng đợi OPEN theo giá trị đánh giá.

OPEN = [G6, D7, E7, F7, C8]; CLOSED = [A6, B6]

**Bước 3:** Lấy trạng thái có giá trị đánh giá cao nhất từ hàng đợi OPEN, trong trường hợp này là G6. Đặt G6 vào CLOSED.

OPEN = [D7, E7, F7, J7, I8, C8]; CLOSED = [A6, B6, G6]

**Bước 4:** Tạo ra các trạng thái kế tiếp từ G6 và thêm chúng vào hàng đợi OPEN. Đồng thời, đánh giá các trạng thái này bằng hàm heuristic và sắp xếp lại hàng đợi OPEN theo giá trị đánh giá.

OPEN = [E7, F7, J7, I8, C8, H9]; CLOSED = [A6, B6, G6, D7]

**Bước 5:** Lấy trạng thái có giá trị đánh giá cao nhất từ hàng đợi OPEN, trong trường hợp này là E7. Đặt E7 vào CLOSED.

OPEN = [F7, J7, I8, C8, L8, H9, M9]; CLOSED = [A6, B6, G6, D7, E7]

**Bước 6:** Tạo ra các trạng thái kế tiếp từ E5 và thêm chúng vào hàng đợi OPEN. Đồng thời, đánh giá các trạng thái này bằng hàm heuristic và sắp xếp lại hàng đợi OPEN theo giá trị đánh giá.

OPEN = [J7, N7, I8, C8, L8, H9, M9, O9]]; CLOSED = [A6, B6, G6, D7, E7, F7]

**Bước 7:** Lấy trạng thái có giá trị đánh giá cao nhất từ hàng đợi OPEN, trong trường hợp này là J7. Đặt J7 vào CLOSED.

OPEN = [N7, I8, C8, L8, H9, M9, O9, K9]; CLOSED = [A6, B6, G6, D7, E7, F7, J7]

**Bước 8:** Tạo ra các trạng thái kế tiếp từ J7 và thêm chúng vào hàng đợi OPEN. Đồng thời, đánh giá các trạng thái này bằng hàm heuristic và sắp xếp lại hàng đợi OPEN theo giá trị đánh giá.

OPEN = [P7, I8, C8, L8, H9, M9, O9, K9]; CLOSED = [A6, B6, G6, D7, E7, F7, J7, N7]

**Bước 9:** Lấy trạng thái có giá trị đánh giá cao nhất từ hàng đợi OPEN, trong trường hợp này là P7. Đặt P7 vào CLOSED.

OPEN = [Q7, I8, C8, L8, H9, M9, O9, K9, R9]; CLOSED = [A6, B6, G6, D7, E7, F7, J7, N7, P7]

**Bước 10:** Tạo ra các trạng thái kế tiếp từ P7 và thêm chúng vào hàng đợi OPEN. Đồng thời, đánh giá các trạng thái này bằng hàm heuristic và sắp xếp lại hàng đợi OPEN theo giá trị đánh giá.

OPEN = [U7, I8, C8, L8, H9, M9, O9, K9, R9, S9, T9]; CLOSED = [A6, B6, G6, D7, E7, F7, J7, N7, P7, Q7]

**Bước 11:** Lấy trạng thái có giá trị đánh giá cao nhất từ hàng đợi OPEN, trong trường hợp này là U7. Đặt U7 vào CLOSED.

OPEN = [W7, I8, C8, L8, H9, M9, O9, K9, R9, S9, T9, V9]; CLOSED = [A6, B6, G6, D7, E7, F7, J7, N7, P7, Q7, U7]

**Bước 12:** Tạo ra các trạng thái kế tiếp từ U7 và thêm chúng vào hàng đợi OPEN. Đồng thời, đánh giá các trạng thái này bằng hàm heuristic và sắp xếp lại hàng đợi OPEN theo giá trị đánh giá.

Lấy trạng thái có giá trị đánh giá cao nhất từ hàng đợi OPEN, trong trường hợp này là W7. Đây là trạng thái đích, do đó quá trình giải thuật kết thúc với thành công.

Kết quả cuối cùng là W7, đạt được bằng cách thực hiện các bước đi từ A6 đến W7.

|  |  |  |  |
| --- | --- | --- | --- |
| Lần | n | OPEN | CLOSED |
| 0  1  2  3  4  5  6  7  8  9  10  11  12 | A6  B6  G6  D7  E7  F7  J7  N7  P7  Q7  U7  W7 | [A6]  [B6, D7, C8]  [G6, D7, E7, F7, C8]  [D7, E7, F7, J7, I8, C8]  [E7, F7, J7, I8, C8, H9]  [F7, J7, I8, C8, L8, H9, M9]  [J7, N7, I8, C8, L8, H9, M9, O9]  [N7, I8, C8, L8, H9, M9, O9, K9]  [P7, I8, C8, L8, H9, M9, O9, K9]  [Q7, I8, C8, L8, H9, M9, O9, K9, R9]  [U7, I8, C8, L8, H9, M9, O9, K9, R9, S9, T9]  [W7, I8, C8, L8, H9, M9, O9, K9, R9, S9, T9, V9] | []  [A6]  [A6, B6]  [A6, B6, G6]  [A6, B6, G6, D7]  [A6, B6, G6, D7, E7]  [A6, B6, G6, D7, E7, F7]  [A6, B6, G6, D7, E7, F7, J7]  [A6, B6, G6, D7, E7, F7, J7, N7]  [A6, B6, G6, D7, E7, F7, J7, N7, P7]  [A6, B6, G6, D7, E7, F7, J7, N7, P7, Q7]  [A6, B6, G6, D7, E7, F7, J7, N7, P7, Q7, U7] |

Bảng 2. Kết quả thực nghiệm 2

#### **2.2.2.2 Đoạn mã Python giải thuậtBest First Searchthực nghiệm 2**

import math

from queue import PriorityQueue

# Function to check if the achieved state is the end state

def is\_goal(state):

goal\_state = [[1, 2, 3], [8, 4, 5], [7, 6, 0]]

return state == goal\_state

# Function to generate a list of child states from the current state

def generate\_states(state):

states = []

moves = [(0, 1), (0, -1), (1, 0), (-1, 0)] # Travel directions (right, left, down, up)

# Find the position of the blank cell (0)

empty\_pos = None

for i in range(3):

for j in range(3):

if state[i][j] == 0:

empty\_pos = (i, j)

break

# Create new states by moving the blank cell

for move in moves:

dx, dy = move

new\_pos = (empty\_pos[0] + dx, empty\_pos[1] + dy)

# Check if the new state is valid

if 0 <= new\_pos[0] < 3 and 0 <= new\_pos[1] < 3:

new\_state = [row[:] for row in state] # Copy the current state

new\_state[empty\_pos[0]][empty\_pos[1]] = new\_state[new\_pos[0]][new\_pos[1]]

new\_state[new\_pos[0]][new\_pos[1]] = 0

states.append(new\_state)

return states

# Function to calculate the h(n) value - heuristic

def calculate\_heuristic(state):

goal\_state = [[1, 2, 3], [8, 4, 5], [7, 6, 0]]

misplaced\_tiles = 0

for i in range(3):

for j in range(3):

if state[i][j] != goal\_state[i][j]:

misplaced\_tiles += 1

return misplaced\_tiles

# Function that implements the Best-First Search algorithm

def best\_first\_search(start\_state):

queue = PriorityQueue()

queue.put((0, start\_state)) # Add the initial state to the priority queue

visited = set() # Keep track of visited states

visited.add(tuple(map(tuple, start\_state))) # Add the initial state to the visited set

visited\_states = [] # Keep track of visited states in order

while not queue.empty():

current\_node = queue.get()[1] # Get the next state from the priority queue

visited\_states.append(current\_node) # Add the current state to the visited states list

if is\_goal(current\_node): # Check if the current state has reached the end state

return visited\_states

next\_states = generate\_states(current\_node) # Generate child states from the current state

# Add child states to the priority queue with heuristic priority value

for state in next\_states:

if tuple(map(tuple, state)) not in visited: # Check if the state has been visited

visited.add(tuple(map(tuple, state))) # Mark the state as visited

priority = calculate\_heuristic(state)

queue.put((priority, state))

return None

# Function to print the steps that move from the initial state to the end state

def print\_solution(solution):

if solution is None:

print("No solution found.")

return

print("Solution:")

for i, state in enumerate(solution):

print(f"Step {i}:")

for row in state:

print(row)

print()

# The initial state of the 8-Puzzle problem

start\_state = [[2, 8, 3], [1, 6, 4], [7, 0, 5]]

# Implement the Best-First Search algorithm

solution = best\_first\_search(start\_state)

# Print the result

print\_solution(solution)

A white background with black dots

Description automatically generated

Hình 6. Kết quả đoạn mã Python

#### **2.2.2.2 Nhận xétthực nghiệm 2**

Giải thuật Best-First Search đã hoạt động hiệu quả trong bài toán này, và tìm được lời giải chính xác. Việc sử dụng hàm heuristic để ước tính chi phí từ trạng thái hiện tại đến trạng thái kết thúc đã giúp giải thuật tìm kiếm mở rộng các trạng thái tiếp theo theo hướng có khả năng làm giảm chi phí nhanh chóng.

Kết quả là một chuỗi các bước di chuyển từ trạng thái ban đầu đến trạng thái kết thúc, trong đó mỗi bước di chuyển là hoán đổi ô trống với một ô số lân cận. Kết quả này là một lời giải hợp lý và đúng đắn để chuyển từ trạng thái ban đầu [[2, 8, 3], [1, 6, 4], [7, 0, 5]] sang trạng thái kết thúc [[1, 2, 3], [8, 4, 5], [7, 6, 0]].

Độ dài của lời giải là 10 bước. Điều này có nghĩa là cần thực hiện 10 lần di chuyển ô trống để chuyển từ trạng thái ban đầu sang trạng thái kết thúc.

# 3. Tổng kết

Quá trình giải thuật Best-First Search bắt đầu với trạng thái ban đầu và sử dụng một hàng đợi ưu tiên (PriorityQueue) để lưu trữ và quản lý các trạng thái. Mỗi lần lấy trạng thái có giá trị đánh giá cao nhất từ hàng đợi ưu tiên, chúng ta kiểm tra xem trạng thái đó có phải là trạng thái đích hay không. Nếu không phải, chúng ta tạo ra các trạng thái kế tiếp từ trạng thái hiện tại và thêm chúng vào hàng đợi ưu tiên dựa trên giá trị đánh giá.

Quá trình này tiếp tục cho đến khi tìm thấy trạng thái đích hoặc không còn trạng thái nào trong hàng đợi ưu tiên. Nếu tìm thấy trạng thái đích, quá trình giải thuật kết thúc và chúng ta có được một chuỗi các bước di chuyển từ trạng thái ban đầu đến trạng thái đích.

## **3.1. Thuật toán Best first search trong bài toán 8 miếng**

Quá trình thực hiện thuật toán Best-First Search trong bài toán 8-Puzzle:

Khởi tạo hàng đợi ưu tiên và tập hợp các trạng thái đã thăm.

Thêm trạng thái ban đầu vào hàng đợi ưu tiên với giá trị ưu tiên là heuristic của trạng thái ban đầu.

Lặp cho đến khi hàng đợi ưu tiên trống hoặc tìm được trạng thái kết thúc:

**a)** Lấy trạng thái tiếp theo từ hàng đợi ưu tiên.

**b)** Kiểm tra xem trạng thái hiện tại đã là trạng thái kết thúc chưa. Nếu phải, trả về danh sách các trạng thái đã thăm theo thứ tự.

**c)** Nếu không phải trạng thái kết thúc, tạo các trạng thái con từ trạng thái hiện tại và thêm vào hàng đợi ưu tiên với giá trị ưu tiên là heuristic của các trạng thái con đó.

**d)**  Đánh dấu trạng thái hiện tại là đã thăm và thêm vào danh sách các trạng thái đã thăm theo thứ tự.

Nếu không tìm thấy lời giải, trả về giá trị None.

Thuật toán Best-First Search có thể tìm kiếm nhanh chóng và hiệu quả các lời giải cho bài toán 8-Puzzle. Tuy nhiên, nó không đảm bảo tìm kiếm được lời giải tối ưu nhất. Điều này phụ thuộc vào việc chọn hàm heuristic thích hợp và cách sắp xếp các trạng thái trong hàng đợi ưu tiên.

## **3.2. Phương hướng phát triển thuật toán trong tương lai**

Trong tương lai, có một số phương hướng phát triển tiềm năng cho thuật toán Best-First Search và giải quyết bài toán 8-Puzzle:

**1)** Cải tiến heuristic: Hiện nay, heuristic trong thuật toán Best-First Search được sử dụng là số ô không đúng vị trí so với trạng thái kết thúc. Phương hướng phát triển là nghiên cứu và phát triển các heuristic mới và tốt hơn để đánh giá chính xác hơn khoảng cách từ trạng thái hiện tại đến trạng thái kết thúc. Heuristic càng chính xác, thuật toán càng có khả năng tìm kiếm lời giải tối ưu hơn.

**2)** Sử dụng thuật toán tìm kiếm thông minh khác: Best-First Search là một trong nhiều thuật toán tìm kiếm thông minh. Có thể nghiên cứu sử dụng các thuật toán tìm kiếm khác như A\* (A Star) hoặc thuật toán tối ưu hóa thông minh (Intelligent Optimization Algorithms) để giải quyết bài toán 8-Puzzle. Các thuật toán này có thể đem lại hiệu quả tốt hơn trong việc tìm kiếm lời giải tối ưu.

**3)** Dùng Machine Learning trong heuristic: Một hướng phát triển tiềm năng khác là sử dụng Machine Learning để học heuristic từ các dữ liệu huấn luyện. Thay vì thiết kế heuristic một cách thủ công, Machine Learning có thể học cách đánh giá trạng thái và tối ưu hóa chúng dựa trên các dữ liệu mẫu. Điều này có thể dẫn đến các heuristic tốt hơn và phù hợp với các trường hợp cụ thể.

**4)** Tối ưu thuật toán tạo trạng thái con: Trong quá trình tạo trạng thái con từ trạng thái hiện tại, có thể có các phương pháp tối ưu hơn để loại bỏ các trạng thái con không cần thiết. Điều này giúp giảm không gian tìm kiếm và cải thiện hiệu suất của thuật toán.

**5)** Tối ưu hóa cài đặt: Đối với bài toán 8-Puzzle, có thể có các cài đặt tối ưu hơn để xử lý các trạng thái và tính toán heuristic một cách hiệu quả. Các cấu trúc dữ liệu và thuật toán tối ưu hóa có thể giúp thuật toán Best-First Search hoạt động nhanh chóng và hiệu quả hơn.

Những phương hướng trên đều đòi hỏi nghiên cứu và nỗ lực trong lĩnh vực Trí tuệ nhân tạo và tối ưu hóa thuật toán. Sự tiến bộ trong những lĩnh vực này sẽ giúp cải thiện hiệu suất và khả năng giải quyết của thuật toán Best-First Search và giải quyết bài toán 8-Puzzle một cách tốt hơn.

# Tài liệu tham khảo

1. - PGS.TS Dương Tuấn Anh, Slide bài giảng Môn Trí tuệ nhân tạo, Chương 3 Tìm kiếm trên không gian trạng thái.
2. - GeekforGeeks, Best First Search (Informed Search), <https://www.geeksforgeeks.org/best-first-search-informed-search/>
3. - Sách Artificial Intelligence: A Modern Approach do Stuart J. Russell và Peter Norvig viết