

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN**

**BÁO CÁO THỰC NGHIỆM MÔN HỌC**

**TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI  
Tìm hiểu thuật toán tìm kiếm heuristic và ứng dụng vào bài toán tìm đường đi tốt nhất có sử dụng chi phí xử lý tại điểm đến.**

**Giáo viên:** Ths. Mai Thanh Hồng

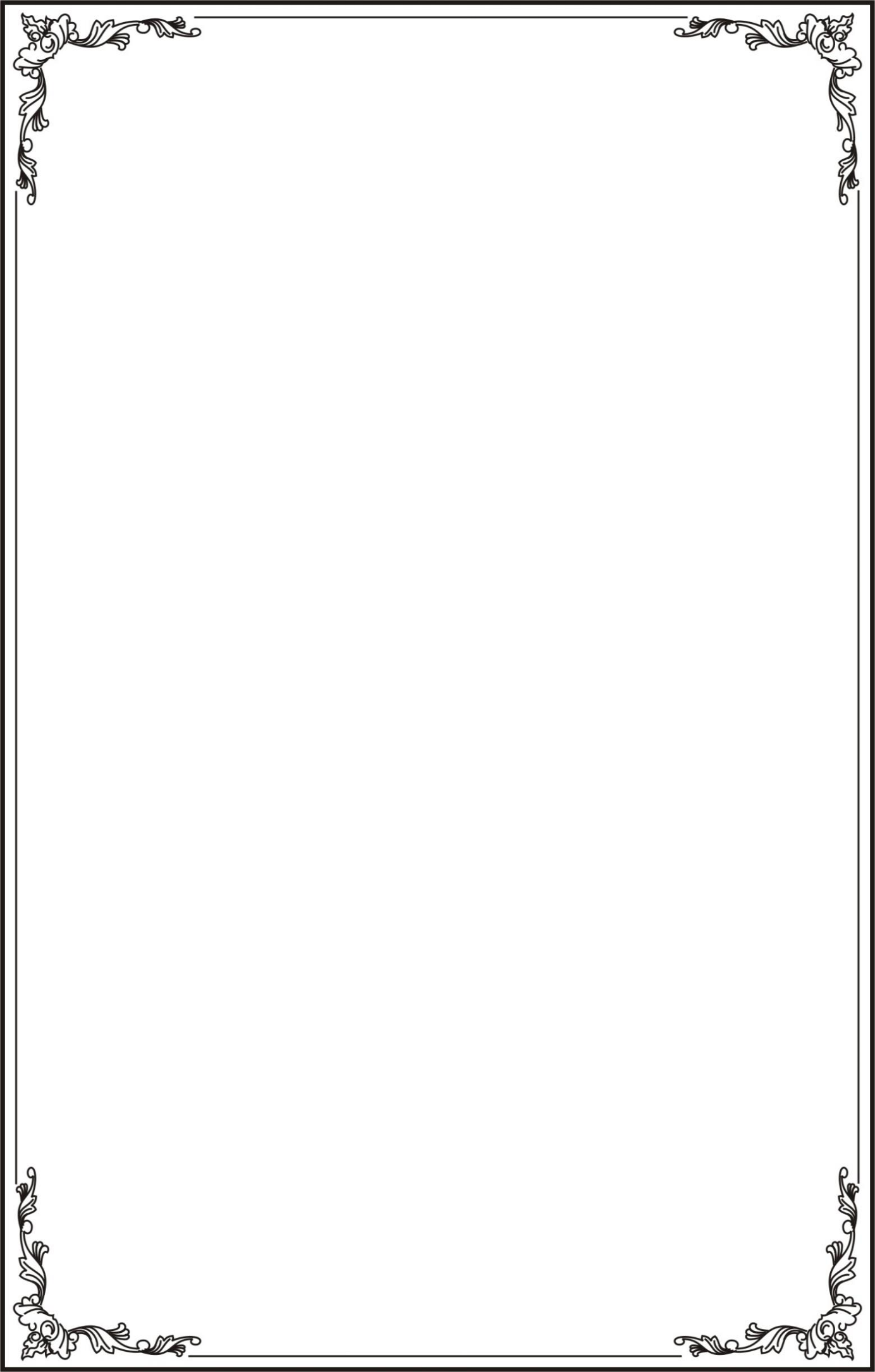
**Nhóm:** 13

**Lớp:** IT6094003 – K18

**Nhóm sinh viên thực hiện:**

|  |  |
| --- | --- |
| Nguyễn Sỹ Chính | MSV: 2023602031 |
| Nguyễn Hải Duy | MSV: 2023602605 |
| Văn Chí Hùng | MSV: 2023601886 |
| Tạ Công Lợi | MSV: 2023601856 |

Hà Nội, 2025



**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN**

**BÁO CÁO THỰC NGHIỆM MÔN HỌC**

**TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI  
Tìm hiểu thuật toán tìm kiếm heuristic và ứng dụng vào bài toán tìm đường đi tốt nhất có sử dụng chi phí xử lý tại điểm đến.**

**Giáo viên:** Ths. Mai Thanh Hồng

**Nhóm:** 13

**Lớp:** IT6094003 – K18

**Nhóm sinh viên thực hiện:**

|  |  |
| --- | --- |
| Nguyễn Sỹ Chính | MSV: 2023602031 |
| Nguyễn Hải Duy | MSV: 2023602605 |
| Văn Chí Hùng | MSV: 2023601886 |
| Tạ Công Lợi | MSV: 2023601856 |

Hà Nội, 2025

**MỤC LỤC**

[LỜI NÓI ĐẦU 1](#_Toc200548885)

[CHƯƠNG I: KHÔNG GIAN TRẠNG THÁI VÀ THUẬT TOÁN TÌM KIẾM HEURISTIC. 2](#_Toc200548886)

[Phần I. Không gian trạng thái. 2](#_Toc200548887)

[1. Mô tả trạng thái. 2](#_Toc200548888)

[2. Toán tử dịch chuyển trạng thái. 2](#_Toc200548889)

[3. Không gian trạng thái của bài toán. 2](#_Toc200548890)

[Phần II. Các thuật toán tìm kiếm heuristic. 4](#_Toc200548891)

[1.2.1. Tổng quan về giải thuật tìm kiếm Heuristic. 4](#_Toc200548892)

[1.2.1.1. Khái niệm 4](#_Toc200548893)

[1.2.1.2. Vai trò 4](#_Toc200548894)

[1.2.1.3. Ưu điểm 4](#_Toc200548895)

[1.2.1.4. Phương pháp xây dựng 4](#_Toc200548896)

[1.2.2. Tìm kiếm tối ưu(Best-First Search – BeFS) 4](#_Toc200548897)

[1.2.2.1. Khái quát về BeFS 4](#_Toc200548898)

[1.2.2.2. Cơ chế hoạt động của BeFS 4](#_Toc200548899)

[1.2.2.3. Ưu điểm nổi bật của BeFS 5](#_Toc200548900)

[1.2.2.4. Ứng dụng của BeFS 5](#_Toc200548901)

[1.2.2.5. Minh họa cho BeFS 5](#_Toc200548902)

[1.2.2.6. Thuật toán tìm kiếm tối ưu (Best-First Search - BeFS) 5](#_Toc200548903)

[1.2.2.7. Thông tin quá khứ và tương lai. 6](#_Toc200548904)

[1.2.3. Thuật toán AT 7](#_Toc200548905)

[1.2.4. Thuật toán AKT (Algorithm for Knowlegeable Tree Search) 9](#_Toc200548906)

[1.2.5. Thuật toán A\* 13](#_Toc200548907)

[CHƯƠNG II: ỨNG DỤNG THUẬT TOÁN HEURISTIC VÀO BÀI TOÁN TÌM ĐƯỜNG ĐI TỐT NHẤT CÓ SỬ DỤNG CHI PHÍ TẠI ĐIỂM ĐẾN. 16](#_Toc200548908)

[Phần I. Không gian trạng thái bài toán. 16](#_Toc200548909)

[1. Giới thiệu bài toán. 16](#_Toc200548910)

[2. Không gian trạng thái. 17](#_Toc200548911)

[Phần II. Phân tích các bước và cài đặt thuật toán. 17](#_Toc200548912)

[KẾT LUẬN 22](#_Toc200548913)

[TÀI LIỆU THAM KHẢO 23](#_Toc200548914)

**DANH MỤC HÌNH ẢNH**

[Hình 1.1: Ví dụ không gian trạng thái 2](#_Toc200132065)

[Hình 1.2: Tìm kiếm BeFS 5](#_Toc200132066)

[Hình 1.3: Phân biệt hàm g và h 7](#_Toc200132067)

[Hình 2.1: Đồ thị có trọng số và chi phí tại mỗi đỉnh 16](#_Toc200132068)

# LỜI NÓI ĐẦU

Trong bối cảnh khoa học máy tính và trí tuệ nhân tạo ngày càng phát triển, các bài toán tìm kiếm tối ưu đã và đang trở thành một trong những lĩnh vực nghiên cứu quan trọng, không chỉ về mặt lý thuyết mà còn có giá trị thực tiễn sâu sắc. Một trong những hướng tiếp cận nổi bật để giải quyết các bài toán này là thuật toán tìm kiếm heuristic – phương pháp dựa trên kinh nghiệm và đánh giá nhằm dẫn dắt quá trình tìm kiếm theo hướng hiệu quả hơn.

Bài toán tìm đường đi tốt nhất là một ví dụ điển hình, thường xuyên xuất hiện trong các ứng dụng như hệ thống định vị, lập lịch công việc, lập kế hoạch di chuyển robot, và nhiều lĩnh vực khác. Khi bài toán được mở rộng thêm yếu tố "chi phí xử lý tại điểm đến" – đại diện cho thời gian hoặc tài nguyên cần thiết để hoàn thành công việc sau khi đi đến điểm đích – việc tìm ra lời giải tối ưu trở nên phức tạp và đòi hỏi những cải tiến trong cách tiếp cận thuật toán.

Trước nhu cầu đó, đề tài "Tìm hiểu thuật toán tìm kiếm heuristic và ứng dụng vào bài toán tìm đường đi tốt nhất có sử dụng chi phí xử lý tại điểm đến" được thực hiện với mục tiêu nghiên cứu sâu về bản chất, cơ chế hoạt động và hiệu quả của các thuật toán heuristic như A\*, Greedy Best-First Search, v.v. Đồng thời, đề tài cũng tập trung đề xuất và thử nghiệm một mô hình áp dụng thực tế, nhằm đánh giá khả năng giải quyết bài toán một cách nhanh chóng và tối ưu trong điều kiện ràng buộc thực tế.

Hy vọng rằng thông qua đề tài này, chúng em mong người đọc không chỉ hiểu rõ hơn về các kỹ thuật heuristic mà còn có thể vận dụng chúng một cách linh hoạt trong các tình huống phức tạp của thế giới thực. Rất mong nhận được sự quan tâm, góp ý từ quý thầy cô và các bạn để đề tài được hoàn thiện hơn, góp phần vào sự phát triển chung của lĩnh vực trí tuệ nhân tạo và tối ưu hóa thuật toán.

# CHƯƠNG I: KHÔNG GIAN TRẠNG THÁI VÀ THUẬT TOÁN TÌM KIẾM HEURISTIC.

## Phần I. Không gian trạng thái.

#### 1. Mô tả trạng thái.

- Giải bài toán trong không gian trạng thái, trước hết phải xác định dạng mô tả trạng thái bài toán sao cho bài toán trở nên đơn giản hơn, phù hợp bản chất vật lý của bài toán (Có thể sử dụng các xâu ký hiệu, véctơ, mảng hai chiều, cây, danh sách,...).

- Mỗi trạng thái chính là mỗi hình trạng của bài toán, các tình trạng ban đầu và tình trạng cuối của bài toán gọi là trạng thái đầu và trạng thái cuối.

#### 2. Toán tử dịch chuyển trạng thái.

- Toán tử chuyển trạng thái thực chất là các phép biến đổi đưa từ trạng thái này sang trạng thái khác. Có hai cách dùng để biểu diễn các toán tử:

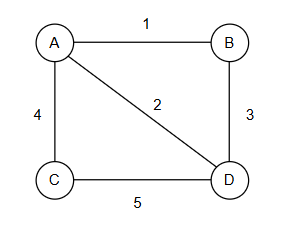
- Biểu diễn như một hàm xác định trên tập các trạng thái và nhận giá trị cũng trong tập này.

- Biểu diễn dưới dạng các quy tắc sản xuất S → A có nghĩa là nếu có trạng thái S thì có thể đưa đến trạng thái A.

#### 3. Không gian trạng thái của bài toán.

- Không gian trạng thái là tập tất cả các trạng thái có thể có và tập các toán tử của bài toán.

Ví dụ: Bài toán tìm đường đi ngắn nhất



Hình 1.1: Ví dụ không gian trạng thái

- Trạng thái (State): Mỗi đỉnh (node) trong đồ thị có thể coi là một trạng thái. Ví dụ: trạng thái "đang ở đỉnh A".

- Không gian trạng thái (State Space): Tập hợp tất cả các đỉnh trong đồ thị (V).

- Trạng thái ban đầu (Initial State): Đỉnh xuất phát (source node s).

- Trạng thái đích (Goal State): Đỉnh đích (destination node t).

- Toán tử dịch chuyển trạng thái (State Transition Operator):

- Hành động: Di chuyển từ một đỉnh u sang một đỉnh v kề với nó.

- Toán tử: Cho một trạng thái hiện tại là đỉnh u, toán tử này sẽ trả về tập hợp tất cả các đỉnh v kề với u (tức là tồn tại cạnh (u, v) trong E).

- Chi phí (Cost): Chi phí của việc áp dụng toán tử để chuyển từ u sang v chính là trọng số của cạnh (u, v), tức là w(u, v).

* **Đỉnh (Trạng thái):** {A, B, C, D}

Toán tử dịch chuyển trạng thái T hoạt động như sau:

* T(A) = {B, C, D} (Từ A có thể đi đến B, C, D)

Chi phí đi đến B: 1 (w(A, B) = 1)

Chi phí đi đến C: 4 (w(A, C) = 4)

Chi phí đi đến D: 2 (w(A, D) = 2)

* T(B) = {A, D} (Từ B có thể đi đến A, D)

Chi phí đi đến A: 1 (w(B, A) = 1)

Chi phí đi đến D: 3 (w(B, D) = 3)

* T(C) = {A, D} (Từ C có thể đi đến A, D)

Chi phí đi đến A: 4 (w(C, A) = 4)

Chi phí đi đến D: 5 (w(C, D) = 5)

* T(D) = {A, B, C} (Từ D có thể đi đến A, B, C)

Chi phí đi đến A: 2 (w(D, A) = 2)

Chi phí đi đến B: 3 (w(D, B) = 3)

Chi phí đi đến C: 5 (w(D, C) = 5)

## Phần II. Các thuật toán tìm kiếm heuristic.

### 1.2.1. Tổng quan về giải thuật tìm kiếm Heuristic.

#### 1.2.1.1. Khái niệm

Trong các bài toán có độ phức tạp thuộc cấp đa thức, việc xây dựng và triển khai các thuật toán giải chính xác là khả thi và hiệu quả.Tuy nhiên, đối với những bài toán có độ phức tạp cao hơn, đặc biệt là thuộc cấp hàm mũ, không gian tìm kiếm trở nên rất lớn, khiến việc duyệt qua toàn bộ các khả năng là bất khả thi trong thời gian thực tế.Trong những trường hợp như vậy, cần áp dụng các phương pháp tìm kiếm xấp xỉ hoặc gần đúng, điển hình là **thuật giải heuristic.**

#### 1.2.1.2. Vai trò

Heuristic là một kỹ thuật dựa trên các hàm đánh giá ước lượng, giúp định hướng quá trình tìm kiếm. Nhờ đó, hệ thống có thể tập trung vào những nhánh có triển vọng cao hơn, cải thiện hiệu suất tìm kiếm đáng kể.

#### 1.2.1.3. Ưu điểm

* Heuristic không đảm bảo tối ưu tuyệt đối, nhưng lại mang lại hiệu quả thực tiễn cao.
* Đặc biệt hữu ích khi bài toán:
  + Yêu cầu phản hồi nhanh.
  + Không thể duyệt hết toàn bộ không gian trạng thái.

#### 1.2.1.4. Phương pháp xây dựng

Hàm ước lượng trong heuristic thường được thiết kế để phản ánh mức độ gần gũi đến trạng thái đích của một trạng thái hiện tại. Việc chọn đỉnh tiếp theo để mở rộng sẽ dựa vào giá trị của hàm này. Nhờ vậy, hệ thống ưu tiên mở rộng các trạng thái có khả năng đạt mục tiêu nhanh hơn, giúp giảm đáng kể thời gian xử lý.

### 1.2.2. Tìm kiếm tối ưu(Best-First Search – BeFS)

#### 1.2.2.1. Khái quát về BeFS

Một trong những chiến lược phổ biến trong tìm kiếm heuristic là **tìm kiếm tối ưu (Best-First Search – BeFS)**. Phương pháp này được thiết kế nhằm kết hợp những ưu điểm nổi bật của hai chiến lược truyền thống: tìm kiếm theo chiều sâu và tìm kiếm theo chiều rộng.

Cụ thể, tìm kiếm theo chiều sâu giúp giới hạn lượng bộ nhớ cần dùng vì chỉ mở rộng một nhánh tại một thời điểm, trong khi tìm kiếm theo chiều rộng có khả năng đảm bảo không bỏ sót lời giải và tránh được các nhánh cụt. BeFS khai thác đồng thời hai lợi thế này bằng cách chỉ tập trung mở rộng một nhánh duy nhất tại mỗi bước, nhưng đồng thời vẫn duy trì và theo dõi tập các trạng thái tiềm năng khác đã được xem xét.

#### 1.2.2.2. Cơ chế hoạt động của BeFS

Tại mỗi bước tìm kiếm, thuật toán BeFS sẽ lựa chọn mở rộng trạng thái có giá trị ước lượng tốt nhất trong tập các trạng thái đã được mở rộng hoặc đang chờ xử lý. Đây là điểm khác biệt rõ rệt so với thuật toán leo đồi (hill climbing), vốn chỉ xem xét các trạng thái lân cận với trạng thái hiện tại và dễ rơi vào trạng thái cực trị cục bộ. BeFS cho phép chuyển hướng sang các nhánh khác nếu trong quá trình tìm kiếm phát hiện rằng nhánh hiện tại đang ngày càng kém hiệu quả.

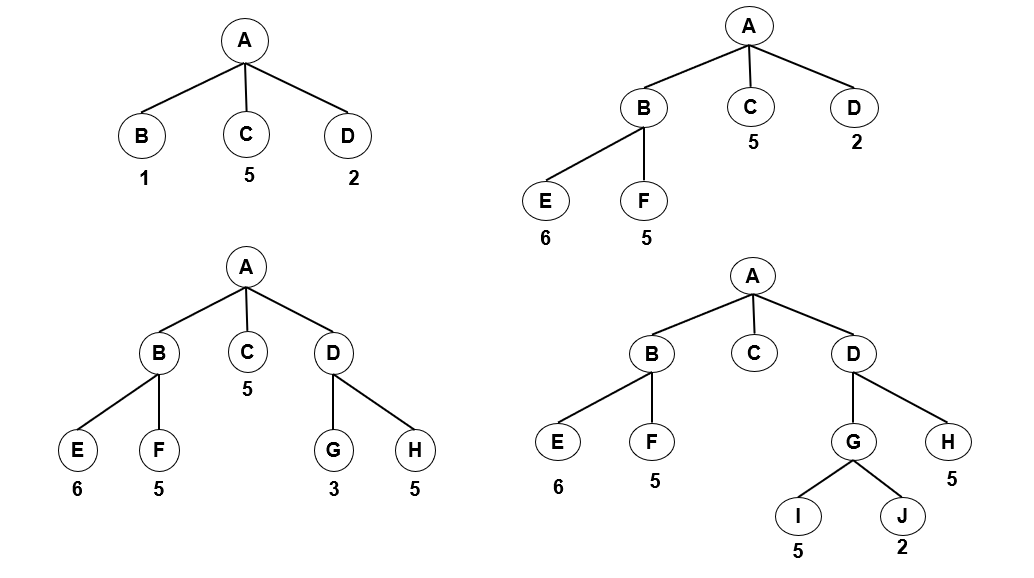
#### 1.2.2.3. Ưu điểm nổi bật của BeFS

Mặc dù đang theo một hướng tìm kiếm cụ thể, thuật toán vẫn luôn duy trì khả năng quay lại và chọn hướng đi tốt hơn nếu có. Chính khả năng này giúp BeFS duy trì sự cân bằng giữa khai thác sâu và khám phá rộng trong không gian trạng thái, từ đó nâng cao xác suất tìm được lời giải tối ưu hoặc gần tối ưu trong thời gian hợp lý.

#### 1.2.2.4. Ứng dụng của BeFS

Tư tưởng chủ đạo của phương pháp tìm kiếm tối ưu là không ràng buộc vào một con đường cố định, mà linh hoạt thích ứng theo đánh giá về tiềm năng của các nhánh, nhờ vào cơ chế đánh giá heuristic. Nhờ vậy, BeFS được xem là một trong những phương pháp tìm kiếm hiệu quả và thực tiễn trong các hệ thống trí tuệ nhân tạo và ứng dụng cần phản hồi nhanh như chatbot, điều hướng bản đồ, và robot tự hành.

#### 1.2.2.5. Minh họa cho BeFS



*Hình 1.2: Tìm kiếm BeFS*

Để minh họa rõ ràng tư tưởng của thuật toán BeFS, ta xét đồ thị trong **Hình 1**. Từ đỉnh khởi đầu A, ta lần lượt xem xét các đỉnh con B, C và D. Trong số đó, đỉnh B có chi phí ước lượng nhỏ nhất nên được chọn để mở rộng trước.

Tuy nhiên, khi phát triển B và sinh ra các đỉnh E và F, ta nhận thấy các đỉnh này có chi phí lớn hơn một số đỉnh khác chưa được mở rộng, cụ thể là đỉnh D. Vì vậy, thuật toán quay lại mở rộng đỉnh D thay vì tiếp tục theo nhánh của B.

Quá trình này lặp lại – luôn ưu tiên mở rộng đỉnh có chi phí ước lượng thấp nhất trong tập các đỉnh đang xét – cho đến khi tìm được đích đến. Cách tiếp cận này giúp tránh rơi vào những nhánh không triển vọng và đảm bảo hướng tìm kiếm hiệu quả hơn.

#### 1.2.2.6. Thuật toán tìm kiếm tối ưu (Best-First Search - BeFS)

Thuật toán BeFS thực hiện tìm kiếm theo hướng ưu tiên mở rộng những trạng thái được đánh giá là “triển vọng nhất” thông qua một hàm ước lượng chi phí. Quá trình tìm kiếm diễn ra theo các bước như sau:

1. Khởi tạo tập OPEN (Open List) với trạng thái bắt đầu.
2. Lặp lại cho đến khi tìm được trạng thái đích hoặc OPEN rỗng:

a. chọn trạng thái tốt nhất (Tmax) trong OPEN (và xóa Tmax khỏi OPEN)

b. Nếu Tmax là trạng thái kết thúc thì thoát.

c. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện :

Tính f(Tk); Thêm Tk vào OPEN.

Mặc dù thuật toán Best-First Search (BeFS) có cấu trúc tương đối đơn giản và dễ triển khai, tuy nhiên trong thực tế, tương tự như các phương pháp tìm kiếm theo chiều sâu và chiều rộng, BeFS hiếm khi được sử dụng trực tiếp. Thay vào đó, các biến thể nâng cao của BeFS như AT, AKT và đặc biệt là A\* thường được ưu tiên áp dụng hơn nhờ khả năng tối ưu hóa quá trình tìm kiếm cũng như cải thiện hiệu quả tính toán trong nhiều bài toán thực tế.

#### 1.2.2.7. Thông tin quá khứ và tương lai.

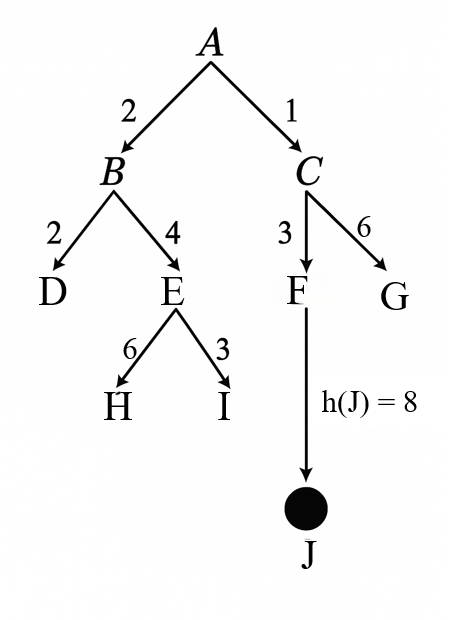
Trong các phương pháp tìm kiếm theo chiến lược Best-First Search (BeFS), chi phí tổng thể f(n) của một trạng thái n thường được tính dựa trên hai thành phần chính là g(n) và h(n).

Cụ thể, g(n) là chi phí thực tế từ trạng thái ban đầu đến trạng thái hiện tại n, đại diện cho **thông tin quá khứ**. Trong khi đó, h(n) là giá trị ước lượng chi phí từ trạng thái hiện tại n đến trạng thái đích, phản ánh **thông tin tương lai**.

Hàm chi phí tổng hợp f(n) được tính theo công thức:

**f(n) = g(n) + h(n)**

Việc kết hợp cả hai yếu tố này cho phép thuật toán BeFS đưa ra các quyết định tìm kiếm hiệu quả hơn, cân bằng giữa những gì đã biết (quá khứ) và những gì cần dự đoán (tương lai).

Ví dụ:

Hình 1.3: Phân biệt hàm g và h

Trong ví dụ này g(J) = 4 là chi phí thực sự từ A đến J, còn h(J) = 8 là chi phi ước lượng từ đến đỉnh đích (hình tròn màu đen), nên f(J) = g(J) + h(J) = 12.

### 1.2.3. Thuật toán AT

Thuật giải AT là một phương pháp tìm kiếm theo kiểu BeFS với chi phí của đỉnh là giá trị hàm g (tổng chiều dài thực sự của đường đi từ đỉnh bắt đầu đến đỉnh hiện tại).

Cho đồ thị G = (V, E) với V: tập đỉnh; E: Tập cung. Với mỗi một cung người ta gắn thêm một đại lượng được gọi là giá của cung.

C: E→ R+

e → C(e)

Khi đó đường đi p=nı, n2, ...nk có giá được tính theo công thức:

C(p) = (ni, ni+1)

Vấn đề đặt ra là tìm đường đi p từ T0 đến đỉnh TG ∈ Goal sao cho C(p) → min

Thuật toán AT

Vào:

+ Đồ thị G = (V, E)

+ C: E → R+

+ e → C(e)

+ Đỉnh đầu T0 và Goal chứa tập các đỉnh đích

Ra:

+ Đường đi p: T0 → TG ∈ Goal sao cho:

C(p) = g(nk) = min {g(n)/n ∈ Goal}.

Phương pháp : Sử dụng hai danh sách CLOSE và OPEN

void AT()

{

OPEN = {T0}, g(T0) = 0, CLOSE = Ø

while OPEN Ø do

{

n ← getNew(OPEN) // lấy đỉnh n sao cho g(n) > min

if (n = TG) then return True

else

{

for each m ∈ A(n) do

if(m ∉ OPEN) and (m ∉ CLOSE) then

{

g(m) = g(n) + cost(m,n)

OPEN = OPEN ∪ {m}

}

else g(m) = min{g(m), gnew(m)}

CLOSE = CLOSE ∪ {n}

}

}

return False;

}

Ví dụ: Cho đồ thị sau: Đỉnh xuất phát A và Goal = {D , H}

A

2

6

4

B C F

2

1

5

8

D E G H

|  |  |
| --- | --- |
| CHA | CON |
| A | B |
| A | C |
| A | F |
| C | D |
| C | E |
| F | G |
| F | H |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Lặp | n | B(n) | MO | DONG |
| 0 |  |  | A(0) | ∅ |
| 1 | A(0) | B,C,F | B(2), C(4), F(6) | A |
| 2 | B(2) | ∅ | C(4), F(6) | A, B |
| 3 | C(4) | D,E | D(12), E(6), F(6) | A, B, C |
| 4 | E(6) | ∅ | D(12), F(6) | A, B, C, E |
| 5 | F(6) | G, H | D(12), G(11), H(7) | A, B, C, E, F |
| 6 | H(7) | → Đích |  |  |

H → Cha(H) = F → Cha(F) = A

=> Đường đi: A → F→ H

### 1.2.4. Thuật toán AKT (Algorithm for Knowlegeable Tree Search)

Thuật giải AT trong quá trình tìm đường đi chỉ xét đến các đỉnh và giá của chúng. Nghĩa là việc tìm đỉnh triển vọng chỉ phụ thuộc hàm g(n) (thông tin quá khứ). Tuy nhiên thuật giải này không còn phù hợp khi gặp phải những bài toán phức tạp (độ phức tạp cấp hàm mũ) do ta phải tháo một lượng nút lớn. Để khắc phục nhược điểm này, người ta sử dụng thêm các thông tin bổ sung xuất phát từ bản thân bài toán để tìm ra các đỉnh có triển vọng, tức là đường đi tối ưu sẽ tập trung xung quanh đường đi tốt nhất nếu sử dụng các thông tin đặc tả về bài toán (thông tin quá tương lai).

Theo thuật giải này, chi phí của đỉnh được xác định:

**f(n) = g(n) + h(n)**

+ Đỉnh n được chọn nếu f(n) → min.

-Việc xác định hàm ước lượng h(n) được thực hiện dựa theo:

+ Chọn toán tử xây dựng cung sao cho có thể loại bớt các đỉnh không liên quan và tìm ra các đỉnh có triển vọng.

+ Sử dụng dụng thêm các thông tin bổ sung nhằm xây dựng tập OPEN và cách lấy các đỉnh trong tập OPEN.

- Để làm được việc này người ta phải đưa ra độ đo, tiêu chuẩn để tìm ra các đỉnh có triển vọng. Các hàm sử dụng các kỹ thuật này gọi là hàm đánh giá. Sau đây là một số phương pháp xây dựng hàm đánh giá:

+ Dựa vào xác suất của đỉnh trên đường đi tối ưu.

+ Dựa vào khoảng cách, sự sai khác của trạng thái đang xét với trạng thái đích hoặc các thông tin liên quan đến trạng thái đích.

- Thuật toán AKT

Vào:

+ Đồ thị G = (V, E) trong đó V là tập đỉnh, E là tập cung.

+ f: V→R+ ( f(n): hàm ước lượng)

+ Đỉnh đầu To và tập các đỉnh đích

Ra:

+ Đường đi p: To→ TG thuộc Goal

Phương pháp: Sử dụng 2 danh sách CLOSE và OPEN

void AKT()

{

OPEN = {T0}, g(T0) = 0

Tính h(To), f(To) = g(To) + h(To)

while OPEN ≠ Ø do

{

N ← getNew(OPEN) // lấy đỉnh n sao cho f(n) → min

if(n = TG) then return True

else

{

for each me ∈ A(n) do

{

g(m) = g(n) + cost(m, n)

Tính h(m), f(m) = g(m) + h(m)

OPEN = OPEN ∪ {m}

}

}

}

return False;

}

Ví dụ: Bài toán trò chơi 8 số

Trạng thái đầu Trạng thái đích

|  |  |  |
| --- | --- | --- |
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 |  | 5 |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 8 |  | 4 |
| 7 | 6 | 5 |

Chọn hàm f(n) = g(n) + h(n)

+ g(n) là giá của đường đi hiện tại từ đỉnh To tới đỉnh n (số lần dịch chuyển ô trống từ trạng thái s đến trạng thái n).

+ h(n): số các con số không nằm đúng vị trí của nó so với trạng thái đích.

|  |  |  |
| --- | --- | --- |
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 |  | 5 |

g = 0

h = 5

f = 5

|  |  |  |
| --- | --- | --- |
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | 5 |  |

|  |  |  |
| --- | --- | --- |
| 2 | 8 | 3 |
| 1 |  | 4 |
| 7 | 6 | 5 |

|  |  |  |
| --- | --- | --- |
| 2 | 8 | 3 |
| 1 | 6 | 4 |
|  | 7 | 5 |

g = 1

h = 7

f = 8

g = 1

h = 5

f = 6

g = 1

h = 7

f = 8

|  |  |  |
| --- | --- | --- |
| 2 |  | 3  g = 2  h = 4  f = 6 |
| 1 | 8 | 4 |
| 7 | 6 | 5 |

|  |  |  |
| --- | --- | --- |
| 2 | 8 | 3 |
| 1 | 4 |  |
| 7 | 6 | 5 |

|  |  |  |
| --- | --- | --- |
| 2 | 8 | 3 |
|  | 1 | 4 |
| 7 | 6 | 5 |

g = 2

h = 4

f = 6

g = 2

h = 5

f = 7

|  |  |  |
| --- | --- | --- |
|  | 8 | 3  g = 3  h = 4  f = 7 |
| 2 | 1 | 4 |
| 7 | 6 | 5 |

|  |  |  |
| --- | --- | --- |
| 2 | 8 | 3  g = 3  h = 5  f = 8 |
| 7 | 1 | 4 |
|  | 6 | 5 |

|  |  |  |
| --- | --- | --- |
|  | 2 | 3  g = 3  h = 3  f = 6 |
| 1 | 8 | 4 |
| 7 | 6 | 5 |

|  |  |  |
| --- | --- | --- |
| 2 | 3 | g = 3  h = 5  f = 8 |
| 1 | 8 | 4 |
| 7 | 6 | 5 |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3  g = 4  h = 2  f = 6 |
|  | 8 | 4 |
| 7 | 6 | 5 |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3  g = 5  h = 3  f = 8 |
| 7 | 8 | 4 |
|  | 6 | 5 |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3  g = 5  h = 0  f = 5 |
| 8 |  | 4 |
| 7 | 6 | 5 |

→ Đích

### 1.2.5. Thuật toán A\*

A\* là một phiên bản đặc biệt của AKT áp dụng cho trường hợp đồ thị. Thuật giải A\* có sử dụng tập hợp CLOSE để lưu trữ những trường hợp đã được xét đến. A*\** mở rộng AKT bằng cách bổ sung cách giải quyết trường hợp khi mở một nút mà nút này đã có sẵn trong OPEN hoặc CLOSE. Khi xét đến một trạng thái Ti, bên cạnh việc lưu trữ 3 giá trị cơ bản g, h, f để phản ánh chi phí của trạng thái đó, A\* còn lưu trữ thêm hai thông số sau:

Trạng thái cha của trạng thái Ti (ký hiệu là Cha(Ti)*:* cho biết trạng thái dẫn đến trạng thái Ti. Trong trường hợp có nhiều trạng thái dẫn đến Ti, thì chọn Cha(Ti) sao cho chi phí đi từ trạng thái khởi đầu đến Ti là thấp nhất, nghĩa là :

g(Ti)= g(Tcha) + cost(Tcha, Ti) là thấp nhất.

Danh sách các trạng thái kế tiếp của Ti: danh sách này lưu trữ các trạng thái kế tiếp Tk của Ti sao cho chi phí đến Tk thông qua Ti từ trạng thái ban đầu là thấp nhất.

- Thuật toán A\*

void Astar()

{

OPEN = {To}, CLOSE = Ø,

g(To) = 0, tính h(To), f(To) = g(To) + h(To)

while (OPEN ≠ Ø)

{

n → getNew(OPEN) // lấy đỉnh n sao cho f(n) → min

if(n= TG) then return path T0→ TG

else

{

for each m ∈ A(n) do

if (m ∉ OPEN + CLOSE) then

{

tính h(m), g(m)

f(m) = g(m) + h(m)

cha(m) = n

OPEN = OPEN ∪ {m}

}

else

{

g(m) = min{gold (m), gnew(m)}

Cập nhật lại OPEN

}

}

CLOSE = CLOSE ∪ {n}

}

return False;

}

Ví dụ: Trạng thái ban đầu T0 =A , trạng thái đích Goal = {B}, các số ghi cạnh các cung là độ dài đường đi, các số cạnh các đỉnh là giá trị của hàm h

14

4

12

15

9

20

7

13

7

6

8

6

4

8

4

6

8

6

9

5

5

6

2

100

0

|  |  |
| --- | --- |
| CHA | CON |
| A | C |
| A | D |
| A → D | E |
| A | F |
| D | H |
| E | K |
| E | I |
| K | B |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Lặp | n | A(n) | MO | DONG |
| 0 |  |  | A014 | ∅ |
| 1 | A | C, D, E, F | C924, D713, E1321, F2027 | A |
| 2 | D | H, E | C924, F2027, H1525, E1119 | A, D |
| 3 | E | K, I | C924, F2027, H1525, K1719, I1924 | A, D, E |
| 4 | K | B | C924, F2027, H1525, I1924 , B2324 | A, D, E, K |
| 5 | B | → Đích |  |  |

B → Cha(B) = K → Cha(K) = E → Cha(E) = D → Cha(D) = A

=> Đường đi: A → D → E → K → B

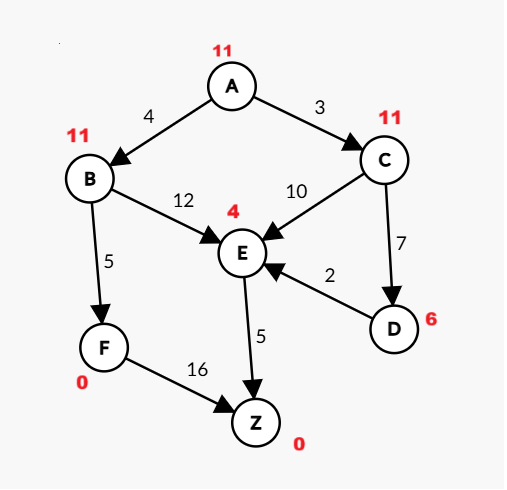
# CHƯƠNG II: ỨNG DỤNG THUẬT TOÁN HEURISTIC VÀO BÀI TOÁN TÌM ĐƯỜNG ĐI TỐT NHẤT CÓ SỬ DỤNG CHI PHÍ TẠI ĐIỂM ĐẾN.

## Phần I. Không gian trạng thái bài toán.

### 1. Giới thiệu bài toán.

Bài toán tìm đường đi tốt nhất có sử dụng chi phí tại điểm đến là một bài toán khá quen thuộc với những người mới tiếp cận với môn Trí Tuệ Nhân Tạo. Bài toán có nhiều phiên bản khác nhau như: bài toán tham quan thành phố,…và ở đây chúng em xét bài toán tìm đường đi tốt nhất có sử dụng chi phí tại điểm đến.

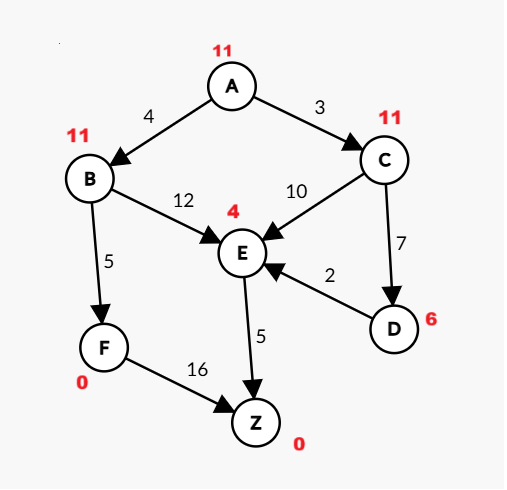
Bài toán tìm đường đi tốt nhất có sử dụng chi phí tại điểm đến gồm có một đồ thị có trọng số và chi phí tại mỗi đỉnh. Nhiệm vụ của bài toán là tìm kiếm đường đi từ điểm đầu tiên cho tới điểm kết thúc mà chi phí sử dụng là nhỏ nhất. Ví dụ có một đồ thị:



Hình 2.1: Đồ thị có trọng số và chi phí tại mỗi đỉnh

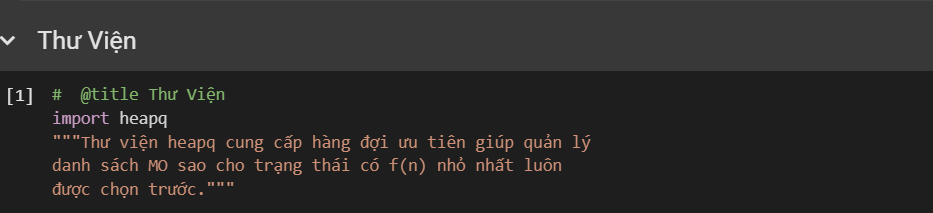
Trong quá trình giải bài toán tại mỗi đỉnh ta phải xét các đỉnh tiếp theo xem đỉnh nào có chi phí nhỏ nhất

### 2. Không gian trạng thái.

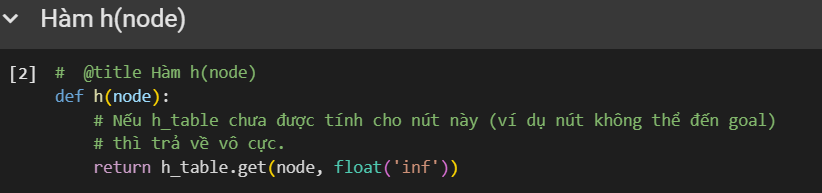


## Phần II. Phân tích các bước và cài đặt thuật toán.

* Khai báo thư viện

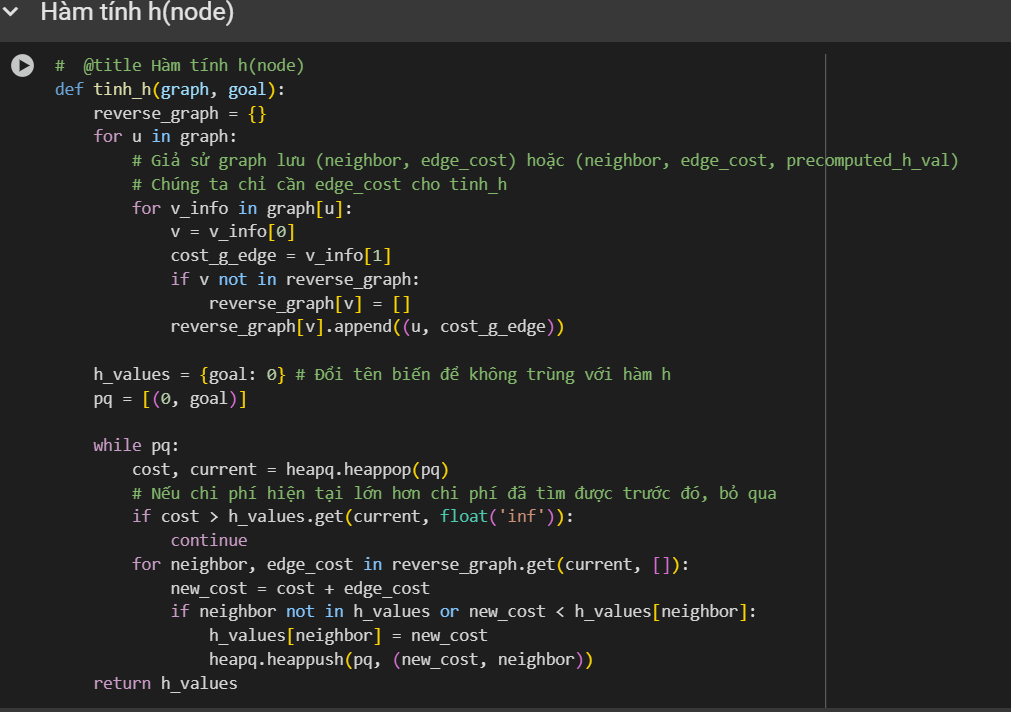


* Hàm h(node)



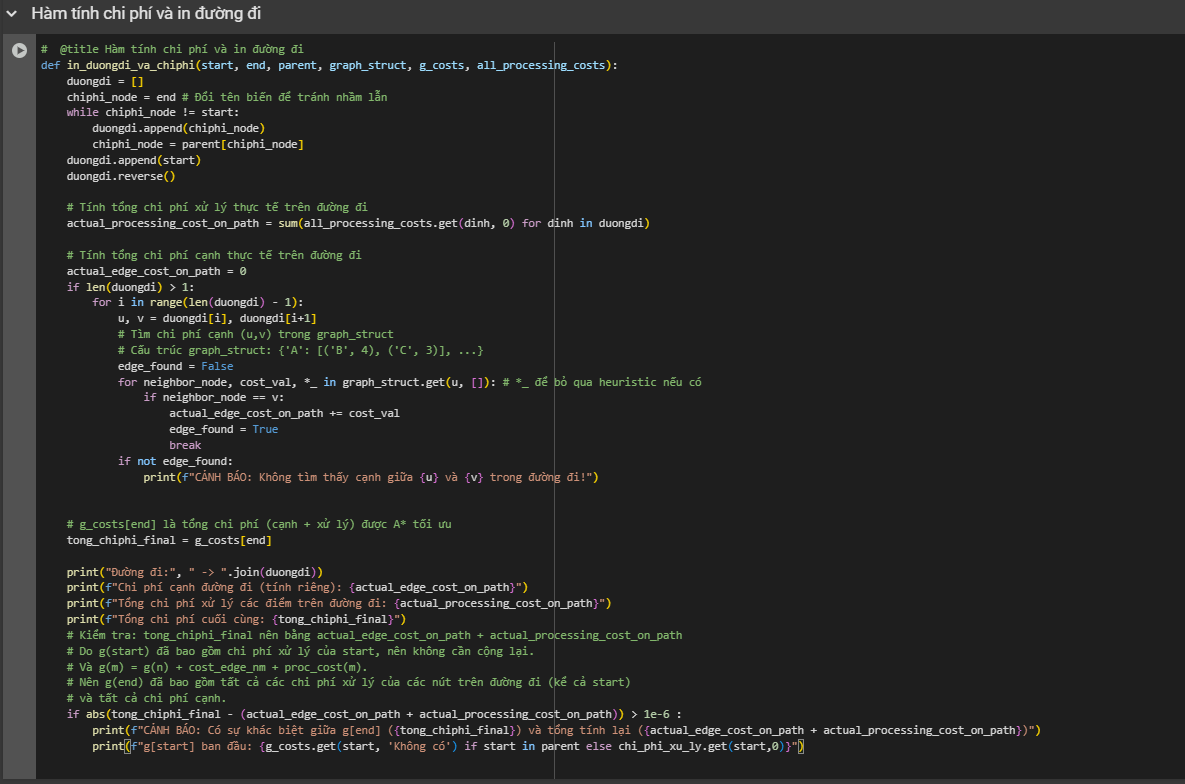
Hàm h là hàm chưa thông tin chi phí của mỗi đỉnh trong đồ thị

* Hàm tính h

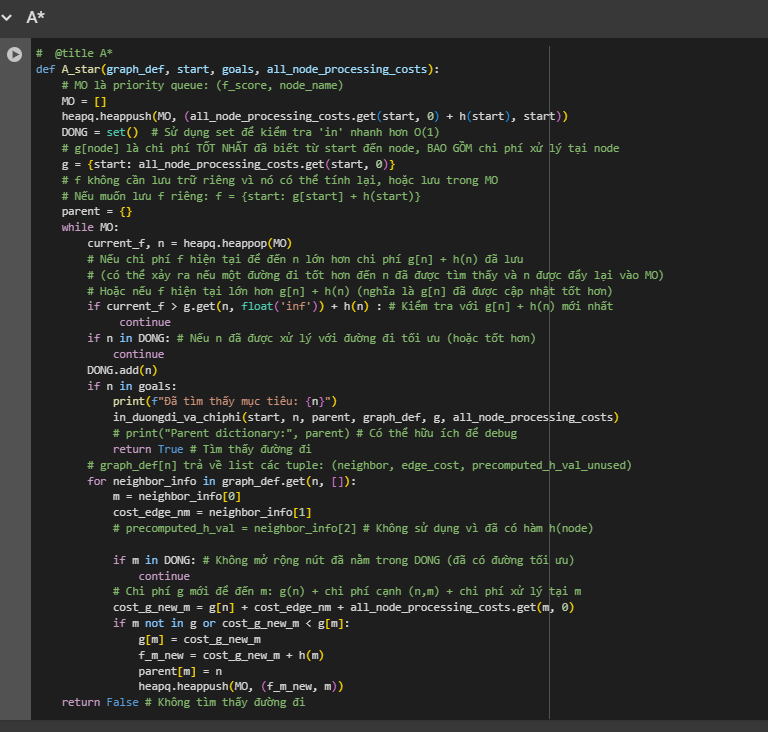


Hàm tính h dùng để tính toán chi phí đi từ n0 đến goal từ đó giúp cho quá trình tính toán chi phí nhanh hơn

* Hàm tính chi phí và in đường đi



* Thuật toán A\*

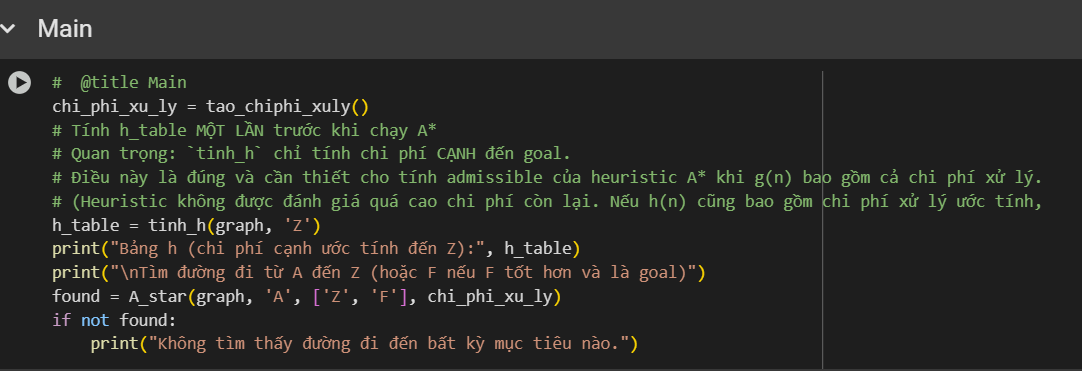


Thuật toán A\* đã được cài tiến để phù hợp với bài toán và những thay đổi đó là:

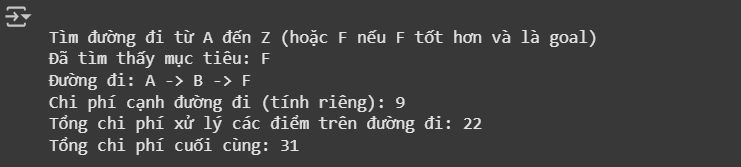
* Dùng heapq thay vì dùng list cho MO và MO hiện giờ là một priority queue
* Dùng set thay vì list cho DONG để có thể kiểm tra n in DONG nhanh hơn
* Đồ thị và chi phí tại mỗi đỉnh



* Hàm Main



* Kết quả sau khi chạy code:



Trong đó:

* Dòng thứ 2 đã tìm được mục tiêu là F bởi vì trong 2 đỉnh Z và đỉnh F thì chi phí để đi đến đỉnh F là nhỏ hơn nên đã chọn đỉnh F
* Dòng thứ 3 là đường đi tốt nhất đã tìm được khi xuất phát từ n0 đến Goal.
* Dòng thứ 4 là chi phí đường đi khi đi qua từ A đến B là 4 và chi phí từ B đến F là 5 tổng chi phí cả đường đi là 9.
* Dòng thứ 5 là chi phí tại mỗi đỉnh trong đó đỉnh A và đỉnh B có chi phí là 11 còn đỉnh F có chi phí là 0 nên tổng chi phí tại mỗi đỉnh là 22
* Dòng 6 là tổng chi phí của toàn bộ đường đi( chi phí đường đi + chi phí mỗi đỉnh) = 31

# KẾT LUẬN

Qua quá trình nghiên cứu và triển khai, bài tập lớn với đề tài "Tìm hiểu thuật toán tìm kiếm Heuristic và ứng dụng vào bài toán tìm đường đi tốt nhất có sử dụng chi phí xử lý tại điểm đến" đã đạt được những mục tiêu đề ra. Báo cáo đã tập trung làm rõ được cơ sở lý thuyết, phương pháp tiếp cận và các bước triển khai thuật toán nhằm giải quyết một bài toán tối ưu hóa đường đi mang tính ứng dụng thực tiễn cao, nơi chi phí không chỉ đơn thuần là khoảng cách mà còn bao gồm cả chi phí phát sinh tại các điểm trung gian và đích đến.

Điểm cốt lõi của bài tập lớn là việc vận dụng và điều chỉnh thuật toán A\* để giải quyết bài toán tìm đường đi tốt nhất có tính đến "chi phí xử lý tại điểm đến". Cụ thể, hàm chi phí thực tế g(n) đã được định nghĩa lại để bao gồm cả chi phí di chuyển qua các cạnh và tổng chi phí xử lý tại tất cả các nút trên đường đi. Đồng thời, hàm ước lượng heuristic h(n) được thiết kế một cách thông minh, phản ánh chi phí cạnh ước tính từ nút hiện tại đến đích, đảm bảo tính chấp nhận được cho thuật toán. Quá trình cài đặt thuật toán A\* đã được tối ưu hóa bằng việc sử dụng hàng đợi ưu tiên (heapq) và cấu trúc dữ liệu tập hợp (set), giúp nâng cao hiệu suất tính toán.

Kết quả thực nghiệm thu được từ việc triển khai thuật toán trên đồ thị mẫu với các chi phí cạnh và chi phí xử lý tại đỉnh cụ thể đã minh chứng cho khả năng của thuật toán A\* trong việc tìm ra đường đi tối ưu, cân bằng hiệu quả giữa chi phí di chuyển và chi phí xử lý.

Bên cạnh những kết quả tích cực đã đạt được, đề tài cũng nhận thấy còn nhiều cơ hội để tiếp tục mở rộng và phát triển. Chẳng hạn, việc thực hiện các nghiên cứu so sánh hiệu năng với các thuật toán tối ưu hóa khác, hay áp dụng thuật toán vào các bài toán thực tế cụ thể.

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | S. J. &. N. P. Russell, Artificial Intelligence: A Modern Approach (4th ed.). Pearson Education., 2021. |
| [2] | D. M. A. &. G. R. Poole, Computational Intelligence: A Logical Approach. Oxford University Press., 1998. |
| [3] | J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley., 1984. |
| [4] | W. contributors, “ A search algorithm\*. Wikipedia,” 2025. [Trực tuyến]. Available: https://en.wikipedia.org/wiki/A\*\_search\_algorithm. |
| [5] | “Best First Search (Informed Search).,” GeeksforGeeks. (n.d.), [Trực tuyến]. Available: https://www.geeksforgeeks.org/best-first-search-informed-search/. [Đã truy cập 2025]. |
| [6] | N. P. N. Trần Hùng Cường, “Giáo trình trí tuệ nhân tạo,” Trường Đại học Công nghiệp Hà Nội, NXB thống kê, 2014. |