TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

**KHOA CÔNG NGHỆ THÔNG TIN  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**



**BÁO CÁO THỰC NGHIỆM/THÍ NGHIỆM**

**HỌC PHẦN: TRÍ TUỆ NHÂN TẠO**

**Tìm hiểu thuật toán tìm kiếm heuristic và ứng dụng vào bài toán tháp Hà Nội**

**Sinh viên thực hiện: Nguyễn Quốc Phong – 2023602468**

**Đặng Đình Trọng – 2023602945**

**Thân Vũ Tuấn Anh – 2023601915**

**Hoàng Thanh Tùng - 2023607238**

**Nhóm thực hiện: Nhóm 11**

**Mã lớp học phần: 20242IT6094010**

**Khóa: 18**

**Giảng viên hướng dẫn: Th.S Mai Thanh Hồng**

***Hà Nội, năm 2025***

MỤC LỤC

[MỤC LỤC HÌNH ẢNH 2](#_Toc200556572)

[LỜI NÓI ĐẦU 3](#_Toc200556573)

[Chương 1: Không gian trạng thái và các thuật toán tìm kiếm 4](#_Toc200556574)

[1.1.Không gian trạng thái 4](#_Toc200556575)

[1.2.Toán tử dịch chuyển 5](#_Toc200556576)

[1.3.Thuật toán tìm kiếm mù 6](#_Toc200556577)

[1.3.1. Tìm kiếm theo chiều sâu (Depth First Search - DFS) 6](#_Toc200556578)

[1.3.2 Tìm kiếm theo chiều rộng (Breadth First Search - BFS) 7](#_Toc200556579)

[1.4. Các thuật toán tìm kiếm Heuristic 8](#_Toc200556580)

[1.4.1 Thuật toán A](#_Toc200556581)[T](#_Toc200556581) [9](#_Toc200556581)

[1.4.2 Thuật toán A](#_Toc200556582)[KT](#_Toc200556582) [(Algorithm for Knowlegeable Tree Search) 11](#_Toc200556582)

[1.4.3 Thuật toán A](#_Toc200556583)[\*](#_Toc200556583) [14](#_Toc200556583)

[Chương 2. Xây dựng và ứng dụng thuật toán tìm kiếm A](#_Toc200556584)[KT](#_Toc200556584) [vào bài toán tháp Hà Nội 18](#_Toc200556584)

[2.1 Lịch sử 18](#_Toc200556585)

[2.2 Mô tả bài toán 19](#_Toc200556586)

[2.3 Giải bài toán 22](#_Toc200556587)

[2.3.1 Không gian trạng thái của bài toán tháp Hà Nội 22](#_Toc200556588)

[2.3.2 Quy trình thực hiện 22](#_Toc200556589)

[2.3.3 Chương trình thực hiện: 23](#_Toc200556590)

[TỔNG KẾT 28](#_Toc200556591)

[TÀI LIỆU THAM KHẢO 29](#_Toc200556592)

# MỤC LỤC HÌNH ẢNH

[Hình 2.1. Tờ hướng dẫn chơi “Tòa tháp Hà Nội” 19](#_Toc200319681)

[Hình 2.2. Một bộ mẫu của Tháp Hà Nội 19](#_Toc200319682)

[Hình 2.3. Ví dụ trường hợp có số đĩa n = 3 21](#_Toc200319683)

# LỜI NÓI ĐẦU

Bài toán Tháp Hà Nội, với nguồn gốc từ công trình của Édouard Lucas năm 1883, là một minh họa sinh động cho khái niệm đệ quy và không gian trạng thái trong khoa học máy tính. Mặc dù quy tắc chuyển đĩa khá đơn giản, sự kết hợp giữa số lượng đĩa và ba cọc đã tạo ra một không gian trạng thái có kích thước tăng theo hàm mũ, khiến cho việc tìm kiếm lời giải nhanh chóng trở thành thách thức.

Trong bối cảnh phát triển mạnh mẽ của trí tuệ nhân tạo, các thuật toán heuristic như A\*, Greedy Best-First hay các hàm đánh giá đặc thù đã chứng tỏ hiệu quả vượt trội so với các phương pháp tìm kiếm thuần túy. Thông qua việc ước lượng chi phí còn lại đến đích, thuật toán heuristic giúp giảm thiểu đáng kể số trạng thái cần xem xét, tiết kiệm thời gian và bộ nhớ tính toán.

Báo cáo này tập trung vào việc giới thiệu cơ sở lý thuyết của các thuật toán tìm kiếm heuristic, thiết kế các hàm đánh giá phù hợp cho bài toán Tháp Hà Nội, đồng thời triển khai và đánh giá hiệu năng trong thực nghiệm. Kết quả thu được sẽ làm rõ ưu nhược điểm của từng thuật toán, từ đó cung cấp góc nhìn về cách áp dụng heuristic hiệu quả trong các bài toán lập kế hoạch và tối ưu tương tự.

Hy vọng thông qua nghiên cứu này, người đọc sẽ có được cái nhìn sâu sắc hơn về vai trò của heuristic trong trí tuệ nhân tạo, cũng như những kỹ thuật thiết yếu để vận dụng vào các tình huống thực tiễn có không gian trạng thái phức tạp.

Xin chân thành cảm ơn sự hướng dẫn tận tình của cô Mai Thanh Hồng, sự hỗ trợ và đóng góp ý kiến quý báu từ các bạn trong lớp, đặc biệt là các thành viên nhóm 11 đã cùng nhau hợp tác, trao đổi để hoàn thành báo cáo này.

# Chương 1: Không gian trạng thái và các thuật toán tìm kiếm

## 1.1.Không gian trạng thái

***Mô tả trạng thái***

Để giải một bài toán bằng phương pháp tìm kiếm trong không gian trạng thái, bước đầu tiên là xác định cách biểu diễn trạng thái của bài toán. Cách biểu diễn cần phải nắm bắt được bản chất của bài toán, đủ thông tin để phân biệt các tình huống khác nhau, và thuận tiện cho việc xử lý. Các dạng biểu diễn phổ biến bao gồm vector, mảng, danh sách, cây, hoặc xâu ký tự.

Mỗi trạng thái tương ứng với một cấu hình (hình trạng) cụ thể của bài toán tại một thời điểm. Trạng thái ban đầu của bài toán được gọi là trạng thái đầu, và trạng thái (hoặc các trạng thái) thể hiện mục tiêu cần đạt được gọi là trạng thái cuối (hoặc trạng thái đích).

Ví dụ 1.: Bài toán Tháp Hà Nội

Xét bài toán Tháp Hà Nội với N đĩa và 3 cọc (ký hiệu là A, B, C).

Tại mỗi thời điểm, vị trí của tất cả N đĩa trên 3 cọc sẽ xác định hình trạng hiện tại của bài toán.

→ Ta có thể mô tả trạng thái bằng một vector (hoặc danh sách, mảng) V có N phần tử. Phần tử thứ i của vector, V[i], cho biết đĩa thứ i (với quy ước đĩa 1 là nhỏ nhất, đĩa N là lớn nhất) đang nằm ở cọc nào (A, B, hay C).

Ví dụ với n=3:

Trạng thái đầu: Tất cả 3 đĩa đều ở cọc A. Biểu diễn: [A, A, A] (nghĩa là đĩa 1 ở A, đĩa 2 ở A, đĩa 3 ở A).

Trạng thái cuối (đích): Tất cả 3 đĩa đều ở cọc C. Biểu diễn: [C, C, C].

Một trạng thái trung gian có thể là: Đĩa 1 ở B, đĩa 2 ở C, đĩa 3 ở A. Biểu diễn: [B, C, A].

## 1.2.Toán tử dịch chuyển

Toán tử dịch chuyển trạng thái (hay toán tử chuyển trạng thái, hành động) là các quy tắc, thao tác hợp lệ cho phép biến đổi bài toán từ một trạng thái này sang một trạng thái khác.

Có thể biểu diễn toán tử như một hàm ánh xạ từ một trạng thái sang một trạng thái khác, hoặc dưới dạng các quy tắc sản xuất S → A, nghĩa là nếu đang ở trạng thái S và điều kiện áp dụng được thỏa mãn, thì có thể chuyển sang trạng thái A bằng cách thực hiện toán tử tương ứng.

Ví dụ 2: Bài toán Tháp Hà Nội

Thao tác cơ bản duy nhất để chuyển trạng thái trong bài toán Tháp Hà Nội là: Di chuyển một đĩa từ cọc này sang cọc khác.

Tuy nhiên, việc di chuyển này phải tuân theo các luật chơi (điều kiện áp dụng):

Chỉ được di chuyển đĩa nằm trên cùng của một cọc.

Không được đặt đĩa lớn hơn lên trên đĩa nhỏ hơn.

Như vậy, nếu trạng thái đang xét là S, các trạng thái kế tiếp S' có thể chuyển đến được xác định bằng cách thử tất cả các phép di chuyển đĩa trên cùng từ một cọc bất kỳ sang một cọc khác, và chỉ giữ lại những phép di chuyển hợp lệ theo luật chơi.

Ví dụ với N=3, trạng thái đầu S = [A, A, A]:

Đĩa trên cùng cọc A: là đĩa 1.

Các phép di chuyển có thể thử:

Chuyển đĩa 1 từ A sang B: Hợp lệ (B rỗng). Trạng thái mới S1 = [B, A, A].

Chuyển đĩa 1 từ A sang C: Hợp lệ (C rỗng). Trạng thái mới S2 = [C, A, A].

Vậy từ trạng thái [A, A, A], có thể chuyển đến 2 trạng thái kế tiếp là [B, A, A] và [C, A, A].

[A, A, A] → {[B, A, A], [C, A, A]}

## 1.3.Thuật toán tìm kiếm mù

### 1.3.1. Tìm kiếm theo chiều sâu (Depth First Search - DFS)

1. Tư tưởng của chiến lược tìm kiếm theo chiều sâu  
   - Từ đỉnh xuất phát duyệt một đỉnh kề.  
   - Các đỉnh của đồ thị được duyệt theo các nhánh đến nút lá.  
   - Nếu chưa tìm thấy đỉnh TG thì quay lui tới một đỉnh nào đó để sang  
   nhánh khác.  
   - Việc tìm kiếm kết thúc khi tìm thấy đỉnh TG hoặc đã hết các đỉnh.

\* Thủ tục tìm kiếm theo chiều rộng

- Vào: + Đồ thị G = (V,E) với V: tập đỉnh, E: tập cung.

+ Đỉnh đầu T0 đến một đỉnh TG ∈ Goal

- Ra: Đường đi p từ T0 đến một đỉnh TG ∈ Goal

b. Thuật toán tìm kiếm theo chiều sâu  
- Lưu trữ: Sử dụng hai danh sách DONG và MO trong đó:  
+ DONG: Chứa các đỉnh đã xét hoạt động theo kiểu FIFO (hàng đợi)

+ MO: chứa các đỉnh đang xét hoạt động theo kiểu LIFO (ngăn xếp)

MO = ∅; MO = MO ∪{T0}

while (MO != ∅) {

n = get(MO) // lấy đỉnh đầu trong danh sach MO

if (n==TG) // nếu n là trạng thái kết thúc

return TRUE // tìm kiếm thành công, dừng

DONG = DONG ∪{n} //đánh dấu n đã được xét for các đỉnh kề v của n

if (v chưa đc xét) //v chưa ở trong DONG

MO = {v}∪MO //đưa v vào đầu DS MO

father(v)=n// lưu lại vết đường đi từ n đến v

}

### 1.3.2 Tìm kiếm theo chiều rộng (Breadth First Search - BFS)

1. Tư tưởng của chiến lược tìm kiếm theo chiều rộng

- Từ đỉnh xuất phát duyệt tất cả các đỉnh kề. 

- Làm tương tự với các đỉnh vừa được duyệt. 

- Quá trình duyệt kết thúc khi tìm thấy đỉnh TG hoặc đã hết các đỉnh để duyệt.

1. Thuật toán tìm kiếm theo chiều rộng

- Lưu trữ: Sử dụng hai danh sách DONG và MO hoạt động theo kiểu FIFO (hàng đợi).

+ DONG: Chứa các đỉnh đã xét

+ MO: chứa các đỉnh đang xét

MO = ∅; MO = MO ∪{T0}

while (MO != ∅)

{

n = get(MO) // lấy đỉnh đầu trong danh sach MO

if (n==TG) // nếu n là trạng thái kết thúc

return TRUE // tìm kiếm thành công, dừng

DONG = DONG ∪{n} //đánh dấu n đã được xét for các đỉnh kề v của n

if (v chưa đc xét) //v chưa ở trong DONG

MO = MO∪{v} //đưa v vào cuối DS MO

father(v)=n// lưu lại vết đường đi từ n đến v

}

## 1.4. Các thuật toán tìm kiếm Heuristic

- ***Heuristic*** là những tri thức được rút ra từ những kinh nghiệm, "trực giác" của con người. Heuristic có thế đúng hoặc sai. Heuristic thường được sử dụng trong những trường hợp sau:

+ Bài toán có thê không có nghiệm chính xác do các mênh đề không phát biều chặt chẽ hay thiếu dữ liệu đề khẳng định kết quả

+ Bài toán có nghiệm chính xác nhưng phí tổn tính toán đề tìm ra nghiệm là quá lớn (bùng nổ tổ hợp)

\* Thuật giải tìm kiếm tối ưu Heuristic

- Thuật toán Heuristic là mở rộng khái niệm thuật toán và có đặc điểm:

+ Thường tìm lời giải tốt nhưng không tốt nhất.

+ Nhanh chóng tìm ra kết quả hơn so với giải thuật tối ưu, vì vậy chi phí thấp hơn

+ Thường thể hiện khá tự nhiên, gần gũi với cách suy nghĩ và hành động của con người

- Đặt OPEN chứa trạng thái khởi đầu T0.

- Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện :

+ Chọn trạng thái tốt nhất (Tmax) trong OPEN (và xóa Tmax khi OPEN)

+ Nếu Tmax là trạng thái kết thúc thì thoát.

+ Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có t trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện : Tính f(Tk); Thêm Tk vào OPEN.

- BeFS khá đơn giản. Tuy vậy, trên thực tế, cũng như tìm kiếm chiều sâu và chiều rộng, hiếm khi ta dùng BeFS một cách trực tiếp. Thông thường, người ta thường dùng các phiên bản của BeFS là AT, AKT và A\*.

### 1.4.1 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

6

2

4

B C F

1

5

2

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

### 1.4.2 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 e Goal

Phương pháp: Sử dụng 2 danh sách CLOSE và OPEN

void A\_KT()

{

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  g = 0  h = 5  f = 5 |
| 1 | 6 | 4 |
| 7 |  | 5 |

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

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

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

g = 1

h = 7

f = 8

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

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

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

|  |  |  |
| --- | --- | --- |
|  | 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 |

|  |  |  |
| --- | --- | --- |
| 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

### 1.4.3 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

A

15

9

20

12

7

F

4

G

7

13

8

C

E

6

4

6

D

8

4

6

8

6

I

9

5

6

5

K

H

B

100

2

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

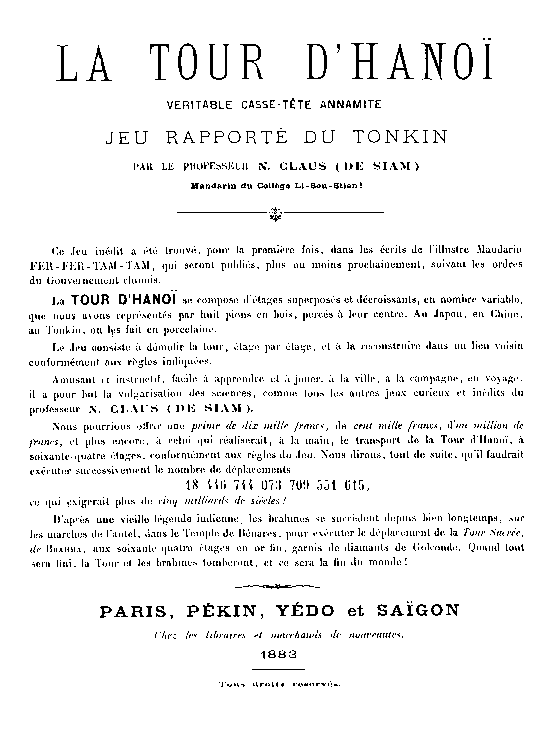
# Chương 2. Xây dựng và ứng dụng thuật toán tìm kiếm AKT vào bài toán tháp Hà Nội

## 2.1 Lịch sử

Như cái tên, tháp Hà Nội là một trò chơi toán học với tên gọi của nó gắn liền với yếu tố Việt Nam với tên đầy đủ là bài toán “Tòa tháp Hà Nội” do một người Pháp, Francois Lucas phát minh vào năm 1883. Nó bao gồm một chồng đĩa xếp lớn dưới, nhỏ trên và 3 cây cột. Yêu cầu là làm sao di dời đĩa qua cột khác, với sự trợ giúp của cột trung gian, nhưng không được để đĩa lớn trên đĩa nhỏ.

Cái tên “ Tòa tháp Hà Nội” gắn liền với một truyền thuyết rằng nó có một ngôi chùa ở Hà Nội, nơi đó các nhà sư đã cố giải bài toán này từ một ngàn năm trước. Họ có ba cây cột và 64 cái đĩa bằng vàng. Ngày ngày họ cứ nhấc đĩa từ cột này qua cột khác. Khi nào họ giải xong, thế gian sẽ thay đổi hoàn toàn.

Thuật toán để giải bài toán này là số lần dời đĩa tối thiếu là 2n - 1 với n là số đĩa. Trò chơi đã xuất hiện ở Đông Á từ thế kỷ 19 hoặc trước đó. Các đĩa cho trò chơi được làm bằng sứ ở các nước như Trung Quốc, Nhật Bản, Việt Nam. Khi được đưa sang pháp thì trò chơi đã được các nhà toán học nghiên cứu, sau đó trở thành ví dụ về phương pháp giải đệ quy kinh điển trong dạy học và tin học. Khi phiên bản đầu tiên được Pháp sản xuất, kèm theo tờ minh họa bìa và hai tờ hướng dẫn. các tờ hướng dẫn này chứa nhiều thông tin lịch sử quý báu về trò chơi này.



*Hình 2.1. Tờ hướng dẫn chơi “Tòa tháp Hà Nội”*

## 2.2 Mô tả bài toán

**Luật chơi**

- Bài toán Tháp Hà Nội được nhà toán học người Pháp Édouard Lucas giới thiệu vào năm 1883. Édouard Lucas (1842–1891) nổi tiếng với các nghiên cứu về lý thuyết số và các bài toán đố toán học.



*Hình 2.2. Một bộ mẫu của Tháp Hà Nội*

- Bài toán bao gồm:

**+ Ba cọc**: thường được gọi là A (nguồn), B (trung gian), và C (đích).

**+ n đĩa**: có kích thước khác nhau, được xếp chồng trên cọc A theo thứ tự từ lớn nhất ở dưới đến nhỏ nhất ở trên.

- Nhiệm vụ của bài toán là di chuyển các đĩa có kích cỡ khác nhau sang cột khác sao cho vẫn đảm bảo thứ tự ban đầu của các đĩa: đĩa nhỏ nằm trên đĩa lớn.

- Quy tắc cho bài toán Tháp Hà Nội (Tower of Hanoi):

+ Mỗi lần chỉ có thể di chuyển một đĩa từ cột này sang cột khác.

+ Chỉ được di chuyển đĩa nằm trên cùng (không được di chuyển các đĩa nằm giữa).  
+ Đĩa có kích thước lớn hơn không thể được đặt trên đĩa có kích thước nhỏ hơn.

- Ví dụ: Xét bài toán tháp Hà nội với n = 2

Chọn hàm f(n) = g(n) + h(n)

Trong đó:

g(n) là số bước đã đi

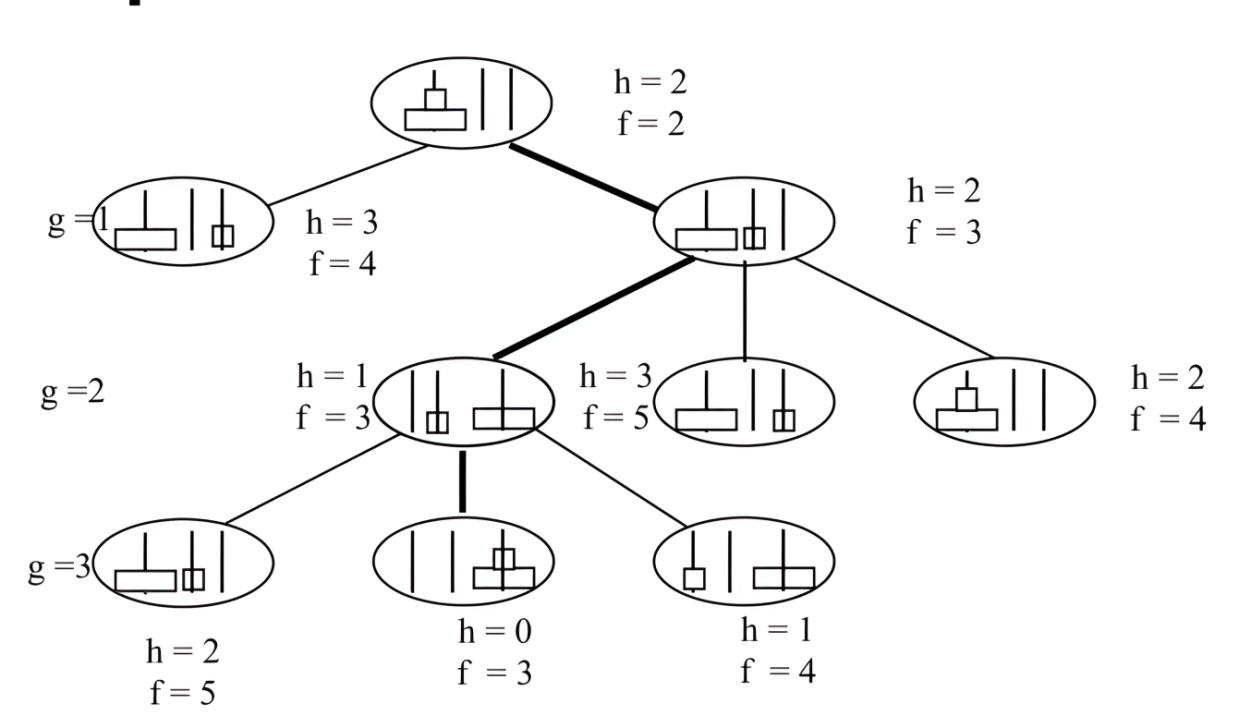
h(n) là thông tin liên quan đến số đĩa ở cọc 3

+ Nếu cọc 3 chưa có đĩa nào thì h = 2

+ Nếu cọc 3 có 1 đĩa nhỏ thì h = 3

+ Nếu cọc 3 có 1 đĩa to thì h = 1

+ Nếu cọc 3 có 2 đĩa và đĩa nhỏ ở trên đĩa to thì h = 0

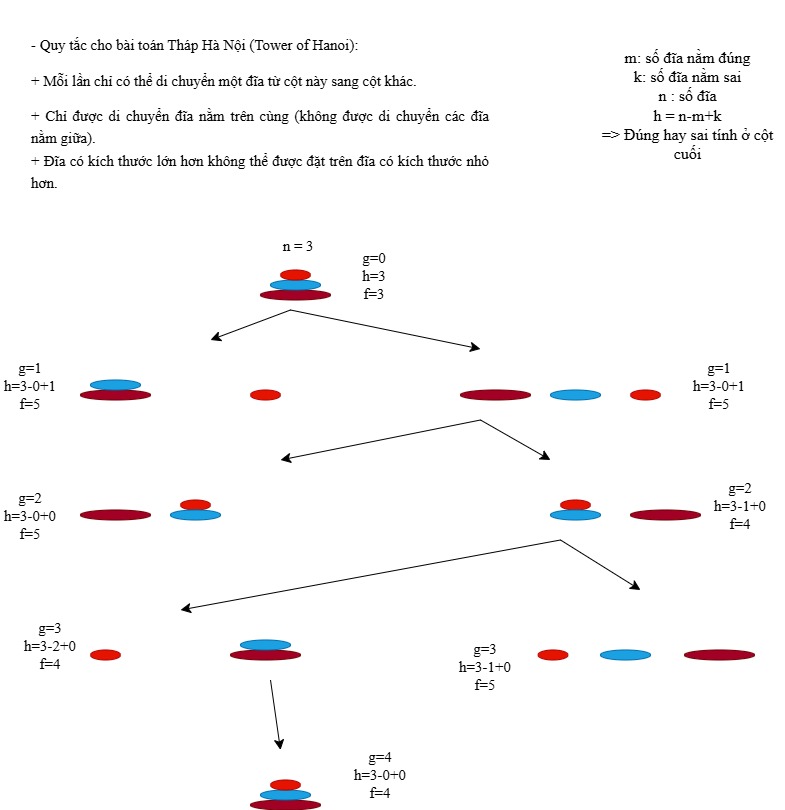


→ Vậy với trường hợp số đĩa n = 2 thì đường đi tìm được theo thuật toán là:

+ Chuyển đĩa 1 từ cột A sang cột B

+ Chuyển đĩa 2 từ cột A sang cột C

+ Chuyển đĩa 1 từ cột B sang cột C



*Hình 2.3. Ví dụ trường hợp có số đĩa n = 3*

## 2.3 Giải bài toán

### 2.3.1 Không gian trạng thái của bài toán tháp Hà Nội

- Không gian trạng thái

+ Trạng thái (x,y,z)

* Trạng thái đầu: ([1,2,3,..], 0, 0]
* Trạng thái cuối: (0, 0, [1,2,3,..]]

+ Toán tử di chuyển

* Di chuyển đĩa nằm trên cùng (không được di chuyển các đĩa nằm giữa)
* Đĩa có kích thước lớn hơn không được đặt trên đĩa có kích thước nhỏ hơn

+ Vẽ không gian trạng thái: Ở mỗi trường hợp với n là số đĩa khác nhau ta sẽ có một không gian trạng thái khác nhau nhưng có điểm chung:

* Hàm f(n) = g(n) + h(n)

Trong đó:

* g(n) là số bước đã đi
* h(n) là thông tin liên quan đến số đĩa ở cọc 3 (số đĩa chưa đúng vị trí so với trạng thái đích)

### 2.3.2 Quy trình thực hiện

Chương trình áp dụng giải thuật AKT , một trạng thái được mô tả bằng mảng hai chiều. Để ước lượng đối với trạng thái của trò chơi, ta sẽ tính trạng thái của cột thứ ba. Tại một trạng thái, thì số bước để đưa cột thứ ba về đúng như trạng thái đích là bao nhiêu? Ta thấy tại một trạng thái của cột thứ ba thì có một số đĩa nằm đúng vị trí của nó, và cũng có một số đĩa không nằm đúng vị trí của nó. Số lượt để ta có thể đưa cột thứ ba về đúng trạng thái đích bằng tổng số lượt mang những đĩa không đúng vị trí ra khỏi cột thứ ba cộng với số lượt mang những đĩa còn lại vào cho đúng vị trí của nó trong cột thứ ba.

Mặc dù thuật giải tương đối đơn giản, bài toán với n đĩa sẽ cần ít nhất 2n - 1 lần di chuyển. Tuy nhiên với số lượng Cọc nhiều hơn 3 thì vẫn chưa biết được sẽ cần ít nhất bao nhiêu lần di chuyển để giải bài toán. Do vậy việc áp dụng bước tiến dãy (tiếng Anh sequential advancement) để xác định vị trí của một số lượng lớn các đĩa trên ba cọc sau một số lớn tuỳ ý các bước tiến là không thực tế. Lời giải tối ưu cho bài toán Tháp Hà Nội với bốn cọc hay nhiều hơn vẫn còn là một bài toán mở.

Chỉ được di chuyển đĩa nằm trên cùng (không được di chuyển các đĩa nằm giữa). Đĩa có kích thước lớn hơden không thể được đặt trên đĩa có kích thước nhỏ hơn.

### 2.3.3 Chương trình thực hiện:

- Khai báo thư viện cần dùng:

import heapq

- Hàm xác định hàm h(n) ~ thông tin liên quan đến số đĩa ở cọc 3 (số đĩa chưa đúng vị trí so với trạng thái đích)

def h\_n(coc, so\_dia):

so\_dia\_sai = 0  
 for i in range(so\_dia):  
 if len(coc[2]) > i and coc[2][i] == so\_dia - i:  
 continue  
 so\_dia\_sai += 1  
 return so\_dia\_sai

*’’’len(coc[2]) ~ kiểm tra số đĩa có ở cọc 3 coc[2][i] == so\_dia - i ~ Kiểm tra đĩa tại vị trí i trên cọc C có đúng với thứ tự mong muốn hay không.   
Đĩa lớn nhất (so\_dia) phải ở dưới cùng (i = 0), đĩa bé hơn (so\_dia - 1) ở trên, v.v’’’*

- Hàm tính hàm f(n) với f(n) = g(n) + h(n) ~ g(n) là số bước đã đi

def gia\_tri\_f(so\_buoc, coc, so\_dia):

*"""Hàm tính f(n) = g(n)// Số bước đi + h(n)"""*

return so\_buoc + h\_n(coc, so\_dia)

→ Từ đây ta có các giá trị f(n) qua mỗi trạng thái của cột và đĩa, từ đó có thể tạo và sử dụng các bước đi tối ưu ở sau.

- Hàm tạo nước đi

def tao\_nuoc\_di(coc):

*"""Tạo các bước di chuyển hợp lệ từ trạng thái hiện tại"""*

nuoc\_di = []  
 for coc\_nguon in range(3):  
 if not coc[coc\_nguon]:  
 continue #diadichuyen = coc[coc\_nguon][-1]  
 for coc\_dich in range(3):  
 if coc\_nguon != coc\_dich and (not coc[coc\_dich] or coc[coc\_dich][-1] > coc[coc\_nguon][-1]):  
 nuoc\_di.append((coc\_nguon, coc\_dich))  
 return nuoc\_di

"""coc\_nguon != coc\_dich ~ Không thể di chuyển đĩa vào cùng một cọc.  
 not coc[coc\_dich] ~ Cọc đích trống, có thể đặt bất kỳ đĩa nào lên.  
 coc[coc\_dich][-1] > coc[coc\_nguon][-1] ~ Nếu cọc đích có đĩa, nó phải lớn hơn đĩa đang di chuyển."""

- Hàm áp dụng nước đi

def ap\_dung\_nuoc\_di(coc, nuoc\_di):

*"""Áp dụng bước di chuyển và trả về trạng thái mới"""*

coc\_nguon, coc\_dich = nuoc\_di  
 coc\_moi = [c[:] for c in coc] #Tạo bản sao của từng cọc  
 coc\_moi[coc\_dich].append(coc\_moi[coc\_nguon].pop()) #Di chuyển đĩa từ cọc nguồn sang cọc đích

""".pop() lấy đĩa nhỏ nhất từ coc\_nguon.  
 .append() đặt đĩa đó lên coc\_dich."""

return coc\_moi

- Hàm kiểm tra đích:

def kiem\_tra\_dich(coc, so\_dia):  
 *"""Kiểm tra nếu tất cả các đĩa đã ở cọc C theo đúng thứ tự. """*

return coc[2] == list(range(so\_dia, 0, -1))

- Hàm giải bài toán Tháp Hà Nội

def thap\_Ha\_Noi\_AKT(so\_dia):

*"""Giải bài toán Tháp Hà Nội bằng thuật toán AKT"""*

coc\_ban\_dau = [[i for i in range(so\_dia, 0, -1)], [], []] #Khởi tạo  
 MO = [(gia\_tri\_f(0, coc\_ban\_dau, so\_dia), 0, coc\_ban\_dau, [])]

#(f(n), g(n), trạng thái\_cọc, danh\_sách\_các\_bước)  
 DONG = set()  
  
 while MO:  
 \_, so\_buoc, coc, cac\_buoc = heapq.heappop(MO)

#Lấy trạng thái có f(n) nhỏ nhất từ MO để tiếp tục xử lý  
 trang\_thai\_coc = tuple(tuple(c) for c in coc)

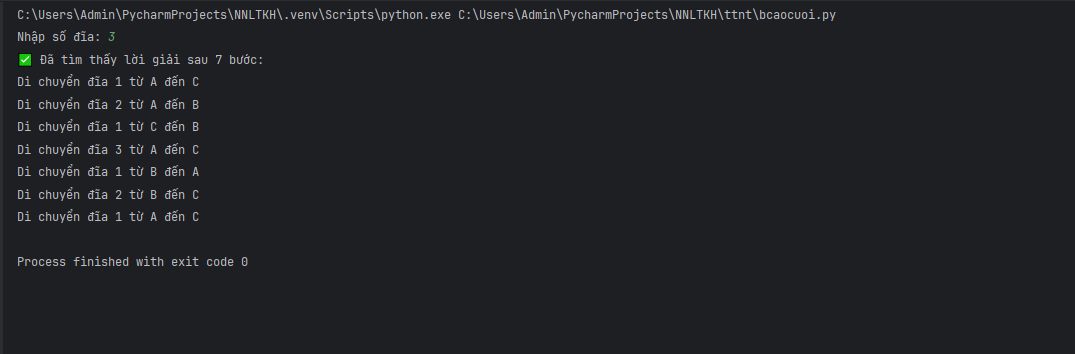
#Kiểm tra nếu trạng thái đã xét trước đó  
 if trang\_thai\_coc in DONG:  
 continue  
  
 DONG.add(trang\_thai\_coc)  
  
 if kiem\_tra\_dich(coc, so\_dia):  
 print(f"So buoc can di chuyen: {len(cac\_buoc)}")  
 for buoc in cac\_buoc:  
 print(buoc)  
 return

for nuoc\_di in tao\_nuoc\_di(coc):  
 coc\_moi = ap\_dung\_nuoc\_di(coc, nuoc\_di)  
 buoc\_moi = cac\_buoc + [f"Di chuyen dia {coc\_moi[nuoc\_di[1]][-1]} tu {chr(65 + nuoc\_di[0])} den {chr(65 + nuoc\_di[1])}"] #chr(65 + nuoc\_di[0]) → Ký tự cọc nguồn // ASCII  
 heapq.heappush(MO, (gia\_tri\_f(so\_buoc + 1, coc\_moi, so\_dia), so\_buoc + 1, coc\_moi, buoc\_moi))

- Nhập số liệu và áp dụng thuật toán để giải bài toán

if \_\_name\_\_ == "\_\_main\_\_":  
 so\_dia = int(input("Nhap so dia: "))  
 thap\_Ha\_Noi\_AKT(so\_dia)

- Kết quả chạy chương trình với n = 3:



# TỔNG KẾT

Qua quá trình nghiên cứu và phân tích, đề tài đã tập trung làm rõ cách tiếp cận và ứng dụng thuật toán AKT để giải bài toán Tháp Hà Nội một cách hiệu quả hơn so với phương pháp truyền thống. Bằng việc tìm hiểu cơ sở lý thuyết, nguyên lý hoạt động của thuật toán và tiến hành so sánh, đánh giá hiệu suất, đề tài đã chứng minh được những ưu điểm nổi bật của thuật toán AKT, đặc biệt trong việc giảm thiểu độ phức tạp tính toán và rút ngắn thời gian thực thi khi số lượng đĩa lớn.

Bên cạnh giá trị lý thuyết, bài toán Tháp Hà Nội còn có nhiều ứng dụng thực tiễn quan trọng. Cụ thể, bài toán được sử dụng trong giảng dạy để minh họa khái niệm đệ quy, thiết kế thuật toán và cấu trúc dữ liệu. Ngoài ra, nó còn được ứng dụng trong lĩnh vực sao lưu và phục hồi dữ liệu, trí tuệ nhân tạo, lập kế hoạch hậu cần, thiết kế mạch điện tử và hệ thống mạng. Những ứng dụng này cho thấy tầm quan trọng của bài toán Tháp Hà Nội không chỉ ở khía cạnh học thuật mà còn trong các lĩnh vực công nghệ và công nghiệp hiện đại.

Tuy nhiên, bên cạnh những kết quả tích cực, đề tài cũng nhận thấy còn nhiều cơ hội để tiếp tục mở rộng và phát triển, chẳng hạn như nghiên cứu khả năng kết hợp thuật toán AKT với các phương pháp tối ưu hóa khác hoặc áp dụng vào các bài toán có cấu trúc tương tự trong thực tế.

Hy vọng rằng những kết quả và kiến thức thu nhận được từ đề tài sẽ là nền tảng hữu ích cho những nghiên cứu tiếp theo, đồng thời đóng góp vào việc phát triển các thuật toán tối ưu trong khoa học máy tính.

Xin chân thành cảm ơn sự quan tâm và đóng góp ý kiến từ quý thầy cô và các bạn.

# TÀI LIỆU THAM KHẢO

1. Giáo trình Trí Tuệ Nhân Tạo - Trường Đại học Công Nghiệp
2. Đề cương bài giảng - Trường Đại học Công Nghiệp
3. Wikipedia - <https://vi.wikipedia.org/wiki/Tháp_Hà_Nội> [Truy cập ngày 18/05/2025]
4. Bài toán Tháp Hà Nội (Tower of Hanoi) - <https://quantrimang.com/cong-nghe/bai-toan-thap-ha-noi-tower-of-hanoi-156528> [Truy cập ngày 01/06/2025]