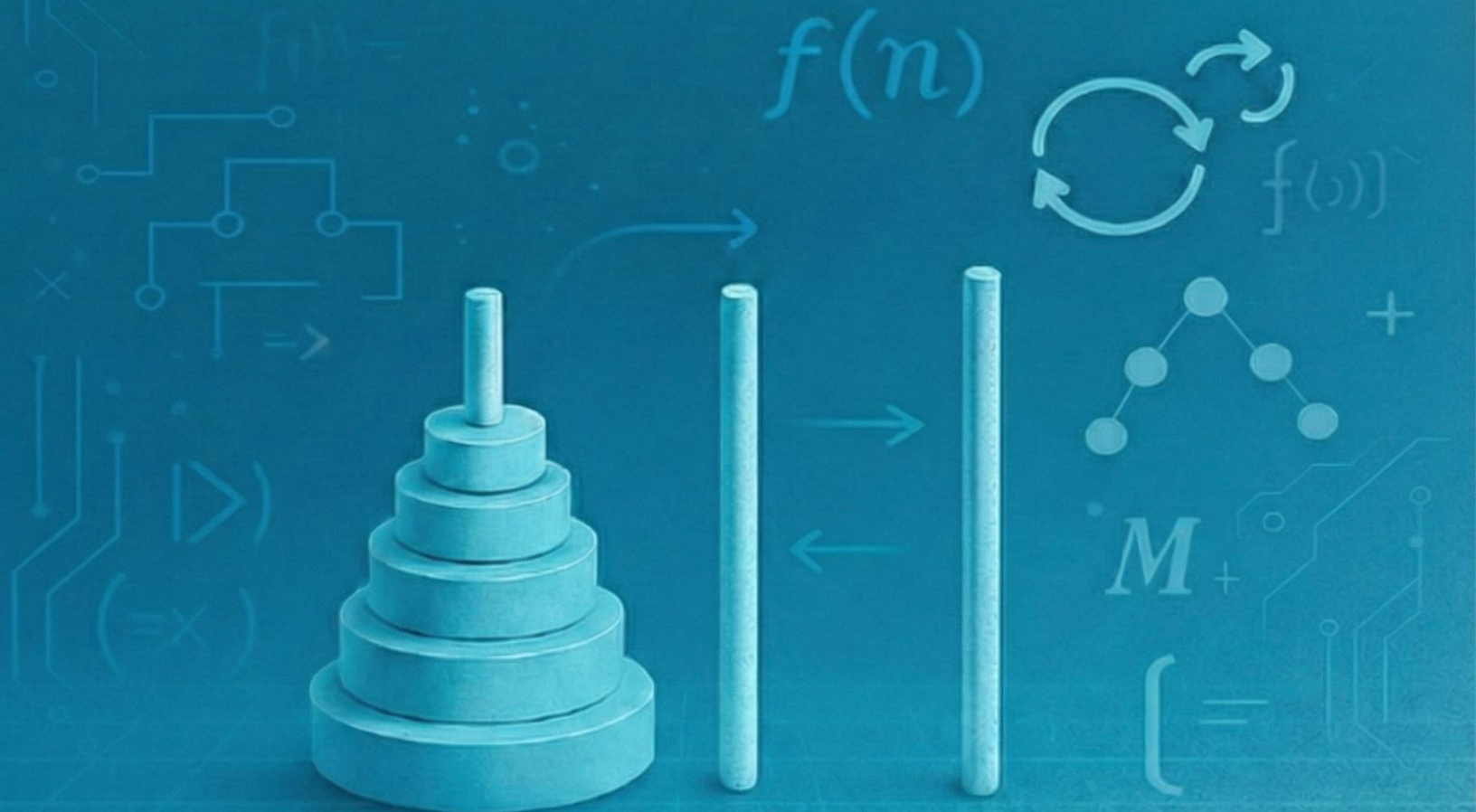
Nguyễn Xuân Phát
Dương Hải Đăng - Hà Trung Hiếu

THÁP HÀ NỘI - THUẬT TOÁN LẬP TRÌNH VÀ ỨNG DỤNG

Tài liệu đọc
Chuyên đề học tập 11.1 - Tin học lớp 11



Lưu hành nội bộ

LỜI NÓI ĐẦU

Trong thế giới khoa học máy tính, đệ quy không chỉ là một kỹ thuật lập trình mà còn là một cách tư duy đặc biệt: giải quyết vấn đề lớn bằng cách quay lại những phiên bản nhỏ hơn của chính nó. Tư duy ấy xuất hiện trong tự nhiên, trong toán học, và trở thành nền tảng của nhiều thuật toán quan trọng. Trong chương trình Tin học 11, việc học và thực hành đệ quy giúp học sinh hiểu sâu hơn về cấu trúc vấn đề, rèn luyện khả năng phân tích, mô hình hóa và phát triển tư duy thuật toán.

Chuyên đề học tập 11.1– được xây dựng nhằm giúp học sinh tiếp cận đệ quy một cách có hệ thống, trực quan và sinh động. Nội dung tài liệu xoay quanh bài toán kinh điển *Tháp Hà Nội*, một bài toán vừa mang màu sắc lịch sử thú vị, vừa sở hữu cấu trúc lý tưởng để minh họa cho bản chất của đệ quy và phương pháp *chia để trị*. Từ việc hiểu quy tắc, phân tích lời gọi hàm, đến lập trình mô phỏng bằng Python và mở rộng sang mô phỏng đồ họa, học sinh sẽ từng bước khám phá cách máy tính "suy nghĩ" và xử lý bài toán.

Tài liệu này còn đóng vai trò định hướng cho hoạt động dự án: học sinh được khuyến khích tìm hiểu, thực hành, gỡ lỗi, mô phỏng và sáng tạo sản phẩm học tập của riêng mình. Qua đó, các em không chỉ nắm được thuật toán mà còn phát triển năng lực giải quyết vấn đề, hợp tác nhóm, trình bày và đánh giá sản phẩm theo tiêu chí rõ ràng.

Với cách tiếp cận thực hành – trực quan – mở rộng, chúng tôi hy vọng tài liệu sẽ trở thành một người bạn đồng hành tin cậy, giúp học sinh tự tin chinh phục đệ quy và tạo nền tảng vững chắc cho các chủ đề thuật toán nâng cao trong những năm học tiếp theo.

Mục lục

1. GIỚI THIỆU VỀ ĐỆ QUY

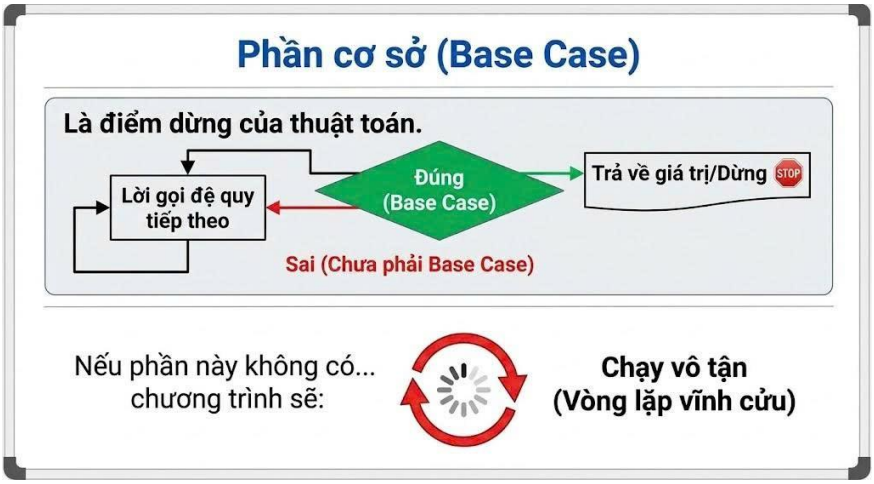
1.1. Khái niệm và tư duy về đệ quy

Định nghĩa: Trong Toán học, khái niệm đệ quy là một phương pháp giải toán, trong đó, lời giải của bài toán phụ thuộc vào một trường hợp nhỏ hơn của cùng một bài toán đó. Nói theo cách dễ hiểu hơn, đệ quy là một hàm mà hàm đó tự gọi chính nó [1].

Ví dụ:

Giai thừa của một số nguyên không âm ***n***, ký hiệu là ***n!,*** là tích của tất cả các số nguyên dương nhỏ hơn hoặc bằng ***n***. Ví dụ trong bài toán: 5! = 5 x 4 x 3 x 2 x 1

1.2. Hai thành phần cốt lõi của hàm đệ quy



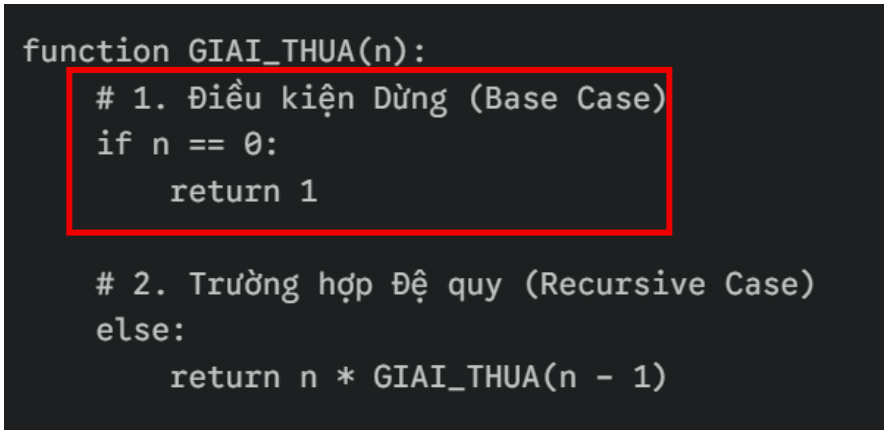
Phần cơ sở (Base Case): Là điểm dừng của thuật toán. Đây là trường hợp mà hàm sẽ trả về một giá trị cụ thể hoặc thực hiện một tác vụ đơn giản mà không cần thực hiện thêm bất kỳ lời gọi đệ quy nào nữa. Nếu phần này không có, xác định sai điều kiện dừng hoặc khai báo sai, chương trình sẽ chạy vô tận.

Ví dụ:

Công thức tổng quát để tìm số Fibonacci thứ ***n*** là:

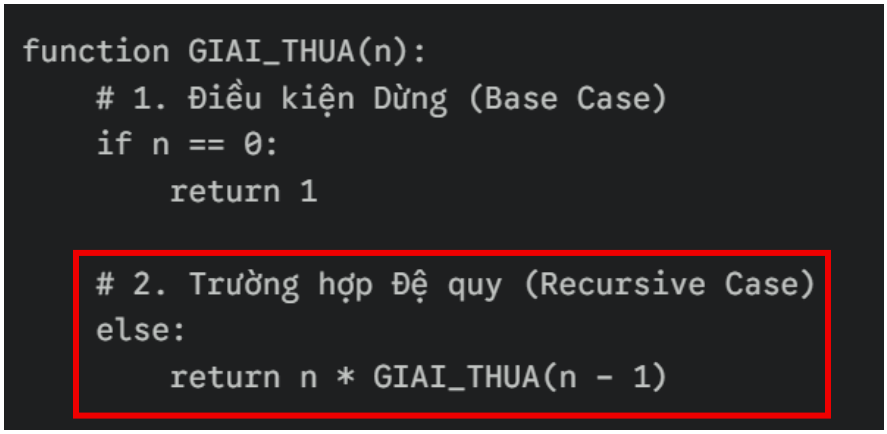
$$F_n = F_{n-1} + F_{n-2} \text{ (với } n > 1)$$

Điều kiện dừng của thuật toán: $F_0 = 0$ và $F_1 = 1$ (vì nó sẽ ngay lập tức trả về giá trị đã biết (0 hoặc 1) mà không gọi lại chính nó, đảm bảo thuật toán kết thúc).



Hình 1-Minh hoạ điều kiện dừng của thuật toán giai thừa

- Phần đệ quy (Recursive Step): Là phần mà hàm gọi lại chính nó với tham số đầu vào mới. Tham số này phải được biến đổi sao cho dữ liệu ngày càng nhỏ đi hoặc đơn giản hơn, và quan trọng nhất là phải có xu hướng tiến dần về Phần cơ sở sau mỗi lần gọi.



Hình 2-Minh hoạ phần đệ quy của thuật toán giai thừa

1.3. Các loại đệ quy:

Có một số loại và thuật ngữ đệ quy khác nhau. Bao gồm:

- Đệ quy trực tiếp (Direct recursion): Điều này được đặc trưng bởi việc triển khai giai thừa, trong đó các phương thức tự gọi chính nó.

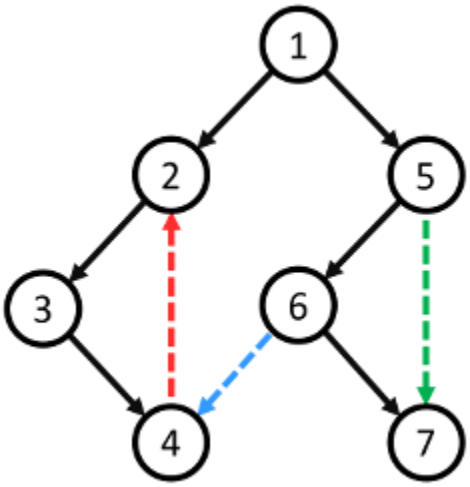
- Đệ quy gián tiếp (In-direct recursion): Điều này xảy ra khi một phương thức, chẳng hạn như phương thức A, gọi một phương thức B khác, sau đó phương thức B này lại gọi phương thức A. Điều này bao gồm hai hoặc nhiều phương thức, cuối cùng tạo ra một chuỗi gọi tuần hoàn.

- Đệ quy đầu (Head recursion)): Lệnh gọi đệ quy được thực hiện ở đầu phương thức.

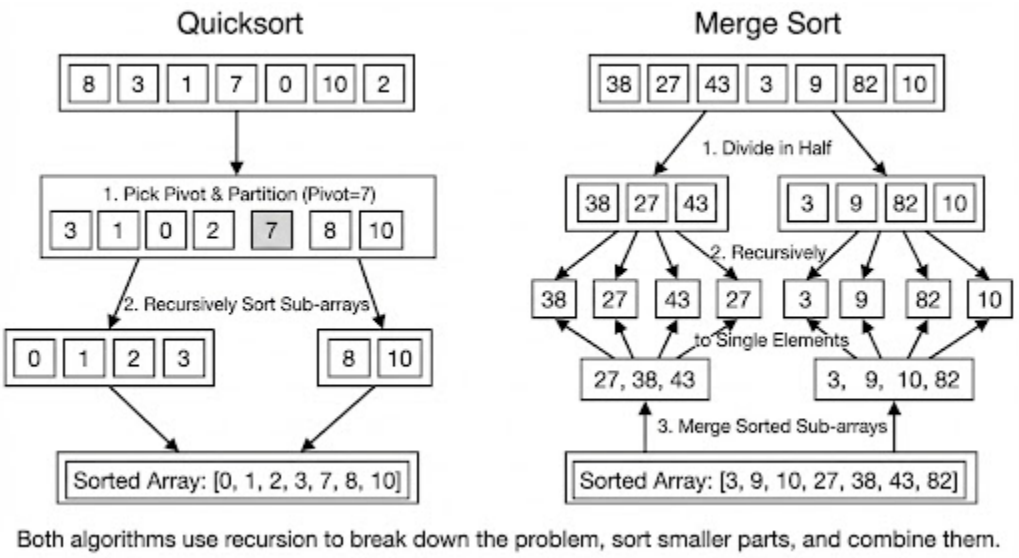
- Đệ quy đuôi (Tail recursion): Lệnh gọi đệ quy là câu lệnh cuối cùng.

1.4. Ứng dụng phổ biến của Đệ quy

- Duyệt cây và Đồ thị (Tree and Graph Traversal): Được sử dụng để khám phá các nút/đỉnh một cách có hệ thống trong các cấu trúc dữ liệu như cây và đồ thị.

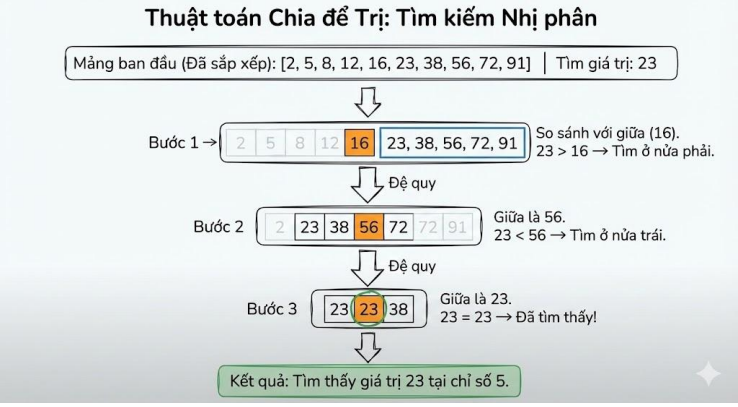


- Các thuật toán sắp xếp (Sorting Algorithm): Các thuật toán như Quicksort và Merge Sort chia dữ liệu thành các mảng con, sắp xếp chúng theo kiểu đệ quy và sau đó gộp chúng lại.

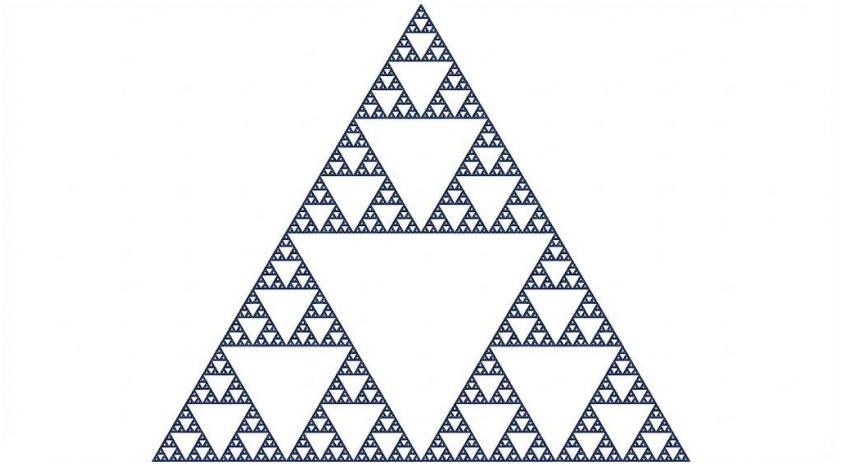


Both algorithms use recursion to break down the problem, sort smaller parts, and combine them.

- Các thuật toán Chia để Trị (Divide-and-Conquer Algorithms): Các thuật toán như Tìm kiếm nhị phân chia bài toán thành các bài toán con nhỏ hơn bằng đệ quy.



- Sinh Fractal: Đệ quy giúp tạo ra các mẫu fractal, chẳng hạn như tập Mandelbrot, bằng cách áp dụng lặp lại một công thức đệ quy.



- Các thuật toán Quay lui (Backtracking Algorithms): Được sử dụng cho các bài toán yêu cầu một chuỗi các quyết định, trong đó đệ quy khám phá tất cả các đường đi khả thi và quay lui khi cần.

- Ghi nhớ (Memoization): Bao gồm việc lưu trữ đệm kết quả của các lệnh gọi hàm đệ quy để tránh phải tính toán lại các bài toán con tốn kém.

Đây chỉ là một vài ví dụ về nhiều ứng dụng của đệ quy trong khoa học máy tính và lập trình. Đệ quy là một công cụ đa năng và mạnh mẽ có thể được sử dụng để giải quyết nhiều loại vấn đề khác nhau.

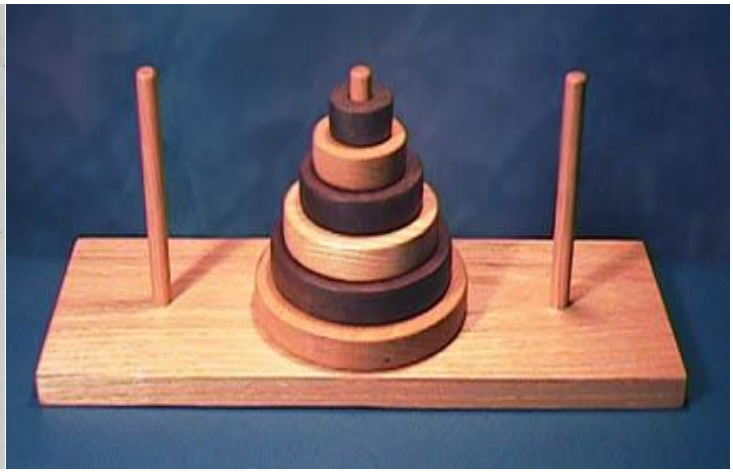
CÂU HỎI ÔN TẬP

1. *Trong các ví dụ sau, ví dụ nào không phải là một hình thức của đệ quy?*
   A. Hai chiếc gương đặt đối diện nhau tạo ra hình ảnh phản chiếu vô tận.
   B. Cây phả hệ (Gia phả) của một dòng họ.
   C. Một đoàn tàu hỏa gồm nhiều toa tàu giống hệt nhau nối đuôi nhau.
   D. Một thư mục trong máy tính chứa các thư mục con bên trong nó.
2. *Tại sao mọi hàm đệ quy trong lập trình đều bắt buộc phải có "Phần cơ sở" (Base Case)?*
   A. Để làm cho mã nguồn ngắn gọn hơn.
   B. Để máy tính biết khi nào cần dừng việc gọi lại hàm, tránh lỗi tràn bộ nhớ (Stack Overflow).
   C. Để hàm có thể trả về nhiều giá trị khác nhau.
   D. Để hàm có thể gọi được các hàm khác bên ngoài nó.
3. *Cho định nghĩa đệ quy của hàm tính tổng S(n) = 1 + 2 + ... + n như sau:*
   - Nếu n = 0 thì S(n) = 0.
   - Nếu n > 0 thì S(n) = n + S (n – 1).

   Hãy xác định giá trị của S(10) thông qua các bước gọi đệ quy.

2. GIỚI THIỆU BÀI TOÁN THÁP HÀ NỘI

2.1. Lịch sử và bối cảnh bài toán



Bài toán tháp Hà Nội có nguồn gốc từ một truyền thuyết Ấn Độ và đã được biết đến rộng rãi do nhà toán học người Pháp, Édouard Lucas, giới thiệu vào năm 1883 trong quyển sách "Récréations Mathématiques" [2].

Truyền thuyết kể về một ngôi đền cổ ở Ấn Độ, nơi có ba cây cột đá và 64 chiếc đĩa vàng khác kích cỡ. Các linh mục ở đền thờ phải chuyển toàn bộ các đĩa từ một cột sang cột khác, tuân theo quy tắc không bao giờ đặt đĩa lớn hơn lên đĩa nhỏ hơn. Truyền thuyết nói rằng, khi toàn bộ các đĩa được chuyển đến cột mới, thế giới sẽ kết thúc.

Édouard Lucas đã lấy cảm hứng từ truyền thuyết này để tạo ra bài toán tháp Hà Nội, đặt tên theo tên thủ đô của Việt Nam vào thời đó.

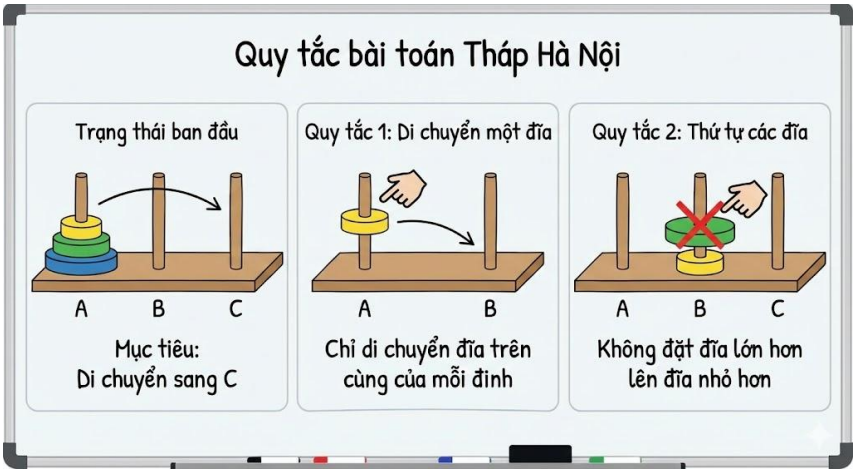
2.2. Quy tắc di chuyển đĩa

Quy tắc của bài toán như sau:

- **Di chuyển một đĩa**: Trong một lần di chuyển, chỉ được phép di chuyển duy nhất chiếc đĩa trên cùng của mỗi đinh.

- **Thứ tự các đĩa**: Không được đặt đĩa có kích thước lớn hơn lên trên đĩa có kích thước nhỏ hơn.

Bài toán tháp Hà Nội chủ yếu dựa vào sự di chuyển hợp lệ của các đĩa giữa các cây đinh, đồng thời tuân thủ các quy tắc để đạt được mục tiêu di chuyển tất cả các đĩa từ cây đinh A sang cây đinh C.



2.3. Xây dựng thuật toán đệ quy

Để giải được bài toán này, chúng ta có ba bước tổng quát:

- Di chuyển n-1 đĩa ở trên cùng ở chiếc đinh ban đầu đến chiếc đinh trung gian.

- Di chuyển chiếc đĩa lớn nhất ở dưới cùng của chiếc đinh đầu tiên đến chiếc đinh đích

- Di chuyển n-1 chiếc đĩa ở đinh trung gian sang đinh đích

Thực hiện lặp lại thuật toán này cho đến khi tất cả các đĩa từ đinh ban đầu đã nằm ở đinh đích và theo thứ tự.

Để dễ hiểu hơn, chúng ta sẽ thực hiện những bước sau:

Phân tích bài toán với n = 1, n = 2, n = 3

- Trường hợp 1 (n = 1): Đây là trường hợp đơn giản nhất, chỉ cần chuyển trực tiếp đĩa A sang đĩa C. *Số bước: 1.*

- Trường hợp 2 (n = 2): Mục tiêu là chuyển đĩa số 2 (lớn nhất sang C. Nhưng đĩa 1 đang nằm đè lên đĩa 2. Các bước giải quyết như sau:

+ Bước 1: Chuyển đĩa 1 từ A sang B (để giải phóng đĩa 2)

+ Bước 2: Chuyển đĩa 2 từ A sang C

+ Bước 3: Chuyển đĩa 1 từ B sang C (để hoàn thành chồng đĩa).

- Trường hợp 3 (n = 3): Mục tiêu là chuyển đĩa số 3 (lớn nhất) sang C. Nhưng hai đĩa 1, 2 đang nằm đè lên nó. Ta coi cụm đĩa 1, 2 là một khối thống nhất cần di chuyển.

+ Bước 1: Chuyển khối 2 đĩa 1, 2 từ A sang B và lấy cọc C làm trung gian.

+ Bước 2: Chuyển đĩa 3 từ A sang C.

+ Bước 3: Chuyển khối 2 đĩa 1, 2 từ B sang C và lấy cọc A làm trung gian.

Dù n là bao nhiêu, muốn chuyển đĩa to nhất (đĩa thứ n) sang cọc Đích, ta luôn phải dọn đường bằng cách chuyển toàn bộ khôi n -1 đĩa phía trên sang cọ Trung gian trước.

Xác định phần cơ sở (Base Case)

Trong mọi thuật toán đệ quy, phần cơ sở là điệu kiên để thuật toán dừng lại, tránh lặp vô hạn. Với Tháp Hà Nội thì điều kiện dừng và các bước thực hiện phần đệ quy được thể hiện như sau:

- Điều kiên dừng: Khi số đĩa cần chuyển chỉ còn là 1 (n = 1).
- Hành động: Chuyển trực tiếp đĩa Cọc nguồn sang Cọc đích.

(Lưu ý: Trong lập trình thực tế, đôi khi ta chọn n = 0 làm điểm dừng để code gọn hơn, nhưng về mặt tư duy thuật toán, n = 1 là điểm dừng tự nhiên nhất).

Xây dựng Phần đệ quy (gồm 3 bước cốt lõi)

Giả sử ta có hàm Hanoi(n, nguon, trung_gian, dich) để chuyển n đĩa. Dựa trên phân tích ở mục 2.3.1, ta tổng quá hóa thành 3 bước hành động cho trường hợp n > 1:

- Bước 1: Chuyển n – 1 đĩa từ cọc nguồn (A) sang cọc trung gian (B).

+ Để làm được việc này, ta cần dùng cọc đích (C) làm cọc hỗ trợ tạm thời.

+ Lời gọi đệ quy: Hanoi(n-1, A, C, B)

+ Giải thích tham số: Chuyển từ A, mượn C, đích đến là B.

- Bước 2: Chuyển đĩa n (đĩa lớn nhất) từ Cọc nguồn(A) sang cọc đích (C).

+ Đây là bước thực hiện trực tiếp, không cần đệ quy vì lúc này đĩa n đã nằm dưới cùng và không bị vướng.

+ Hành động: In ra màn hình "Chuyển đĩa n từ A sang C".

- Bước 3: Chuyển ***n-1*** đĩa từ Cọc trung gian (B) sang Cọc đích (C).

+ Lúc này ***n-1*** đĩa đang nằm ở B, ta cần đưa chúng về C đè lên đĩa n. Ta dùng Cọc nguồn (A) (lúc này đang trống) làm cọc hỗ trợ.

+ Lời gọi đệ quy: **Hanoi(n-1, B, A, C)**

Giải thích tham số: Chuyển từ B, mượn A, đích đến là C.

Ví dụ với ***n*** = 4:

1. Lời Gọi Ban Đầu (N = 4)

Gọi hàm gốc:

$$\text{Hanoi}(4, A, C, B)$$

Theo quy tắc đệ quy, ta có 3 bước:

1. **Hanoi(3, A, B, C)** – chuyển 3 đĩa từ A → B (dùng C làm trung gian)
2. **Di chuyển đĩa 4 từ A → C**
3. **Hanoi(3, B, C, A)** – chuyển 3 đĩa từ B → C (dùng A làm trung gian)

2. Phân Rã Bước 1 tại N = 3

Xét lời gọi:

$$\text{Hanoi}(3, A, B, C)$$

Lại gồm 3 bước:

1. **Hanoi(2, A, C, B)** – chuyển 2 đĩa từ A → C
2. **Di chuyển đĩa 3 từ A → B**
3. **Hanoi(2, C, B, A)** – chuyển 2 đĩa từ C → B

3. Phân Rã Bước 1.1 tại N = 2

Xét lời gọi:

$$\text{Hanoi}(2, A, C, B)$$

Lại gồm 3 bước:

1. **Hanoi(1, A, B, C)** – chuyển 1 đĩa từ A → B
2. **Di chuyển đĩa 2 từ A → C**
3. **Hanoi(1, B, C, A)** – chuyển 1 đĩa từ B → C

4. Điều Kiện Dừng – N = 1

Khi gặp lời gọi:

$$\text{Hanoi}(1,\text{Nguồn},\text{Đích},\text{Trung gian})$$

Ta chỉ thực hiện **1 bước duy nhất**:

- **Di chuyển đĩa 1 từ Nguồn → Đích**

Ví dụ từ bước trên:

$$\text{Hanoi}(1, A, B, C) \Rightarrow \text{Di chuyển đĩa 1 từ A} \rightarrow \text{B}$$

2.4. Thực hành

Vẽ bảng mô tả sơ đồ di chuyển đĩa với n=4.

Ví dụ:

1. Di chuyển 3 đĩa từ cọc 1 sang cọc 3: 1.1 Di chuyển 2 đĩa từ cọc 1 sang cọc 2: - Di chuyển 1 đĩa từ cọc 1 sang cọc 3. - Di chuyển 1 đĩa từ cọc 1 sang cọc 2. - Di chuyển 1 đĩa từ cọc 3 sang cọc 2. 1.2. Di chuyển 1 đĩa từ cọc 1 sang cọc 3.

1.3. Di chuyển 2 đĩa từ cọc 2 sang cọc 3:

- Di chuyển 1 đĩa từ cọc 2 sang cọc 1.
- Di chuyển 1 đĩa từ cọc 2 sang cọc 3
- Di chuyển 1 đĩa từ cọc 1 sang cọc 3.

2. Di chuyển 1 đĩa từ cọc 1 sang cọc 2.

3. Di chuyển 3 đĩa từ cọc 3 sang cọc 2:

3.1 Di chuyển 2 đĩa từ cọc 3 sang cọc 1:

- Di chuyển 1 đĩa từ cọc 3 sang cọc 2. 3.1.2
- Di chuyển 1 đĩa từ cọc 3 sang cọc 1.
- Di chuyển 1 đĩa từ cọc 2 sang cọc 1.

3.2 Di chuyển 1 đĩa từ cọc 3 sang cọc 2.

3.3 Di chuyển 2 đĩa từ cọc 1 sang cọc 2:

- Di chuyển 1 đĩa từ cọc 1 sang cọc 3.
- Di chuyển 1 đĩa từ cọc 1 sang cọc 2.
- Di chuyển 1 đĩa từ cọc 3 sang cọc 2.

Vậy, tổng số bước để di chuyển 4 đĩa theo quy trình trên là:

- Di chuyển 3 đĩa từ cọc 1 sang cọc 2: 7 bước
- Di chuyển đĩa còn lại từ cọc 1 sang cọc 3: 1 bước
- Di chuyển 3 đĩa từ cọc 2 sang cọc 3: 7 bước

Vậy tổng số bước cần thiết để di chuyển 4 đĩa trong bài toán tháp Hà Nội là 15 bước.

CÂU HỎI ÔN TẬP

1. Trong trò chơi Tháp Hà Nội, quy tắc nào sau đây là SAI?

A. Mỗi lần chỉ được di chuyển một đĩa.

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

C. Có thể đặt đĩa lớn lên trên đĩa nhỏ hơn nếu cọc đó còn trống.

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

2. Để chuyển 3 đĩa từ cọc A sang cọc C (dùng B làm trung gian), bước đầu tiên theo tư duy đệ quy là gì?

A. Chuyển đĩa số 3 (lớn nhất) từ A sang C.

B. Chuyển đĩa số 1 (nhỏ nhất) từ A sang C.

C. Chuyển khối 2 đĩa (1 và 2) từ A sang B.

D. Chuyển khối 2 đĩa (1 và 2) từ A sang C.

3. Công thức tính số bước di chuyển tối thiểu để giải bài toán Tháp Hà Nội với n đĩa là S(n) = $2^n - 1$*. Vậy với n = 5, cần thực hiện ít nhất bao nhiêu bước di chuyển?*

A. 15 bước

B. 31 bước

C. 32 bước

D. 63 bước

4. Giả sử bạn đang giải bài toán Tháp Hà Nội với 4 đĩa. Tại một bước nào đó, bạn cần chuyển khối 3 đĩa từ cọc Trung gian về cọc Đích. Để làm được điều này, bạn cần chuyển đĩa lớn nhất (đĩa số 4) từ đâu sang đâu trước đó?

A. Từ cọc Nguồn sang cọc Đích.

B. Từ cọc Nguồn sang cọc Trung gian.

C. Từ cọc Trung gian sang cọc Đích.

D. Từ cọc Đích sang cọc Nguồn.

5. Tại sao khi số lượng đĩa tăng lên (ví dụ từ 3 lên 4), số bước di chuyển lại tăng gấp đôi cộng thêm 1 (tức là $2 * 7 + 1 = 15$*)? Hãy giải thích ngắn gọn dựa trên quy trình 3 bước của thuật toán đệ quy.*

3. THỰC HÀNH LẬP TRÌNH THÁP HÀ NỘI VỚI PYTHON

3.1. Các cú pháp khai báo trong Python

Các cú pháp được sử dụng trong bài:

1. Khai báo hàm: *def hanoi(n, a, b, c):*
2. Câu điều kiện (if): *if n == 1:*
3. Lệnh trả về (return): *return*
4. Lời gọi hàm (gọi đệ quy): *hanoi(n - 1, a, c, b)*
5. Lệnh in kết quả: *print(...)*
6. Chuỗi dạng f-string: *f"Di chuyển đĩa {n} từ {a} sang {b}"*

3.2. Mã nguồn (code) mẫu cho thuật toán Tháp Hà Nội

```
def hanoi(n, source, target, auxiliary):
    # Điều kiện dừng
    if n == 1:
        print(f"Di chuyển đĩa 1 từ {source} sang {target}")
        return

    # Bước 1: Chuyển n-1 đĩa sang cọc phụ
    hanoi(n - 1, source, auxiliary, target)

    # Bước 2: Chuyển đĩa lớn nhất
    print(f"Di chuyển đĩa {n} từ {source} sang {target}")

    # Bước 3: Chuyển n-1 đĩa từ cọc phụ sang cọc đích
    hanoi(n - 1, auxiliary, target, source)


# Gọi hàm
hanoi(4, 'A', 'C', 'B')
```

- Giải thích chi tiết mã nguồn:

Cách hàm** thap_hanoi() **tự gọi chính nó:

Khi hàm thap_ha_noi(3, ...) được gọi, máy tính không giải quyết ngay lập tức. Nó nhận thấy n > 1, nên nó sẽ tạm dừng công việc hiện tại để thực hiện Bước 1: gọi

thap_ha_noi(2, ...). Tương tự, hàm xử lý 2 đĩa lại tạm dừng để gọi hàm xử lý 1 đĩa. Quá trình này giống như việc xếp chồng các chiếc hộp (ngăn xếp - stack) lên nhau. Chỉ khi chiếc hộp trên cùng (trường hợp cơ sở n=1) được giải quyết và lấy đi, chiếc hộp bên dưới mới được tiếp tục xử lý.

Phần cơ sở (if n == 1): Đây là "chốt chặn" quan trọng nhất. Nếu thiếu điều kiện này, hàm sẽ gọi thap_ha_noi(0), thap_ha_noi(-1)... mãi mãi cho đến khi máy tính báo lỗi "Tràn ngăn xếp".

Sự thay đổi của các cột sau mỗi lần gọi:

def hanoi(n, source, target, auxiliary):

- Khai báo hàm với 4 tham số:
  - n: số đĩa (nguyên dương).
  - source: tên hoặc ký hiệu cọc nguồn (ví dụ 'A').
  - target: tên cọc đích (ví dụ 'C').
  - auxiliary: tên cọc phụ (ví dụ 'B').

if n == 1:

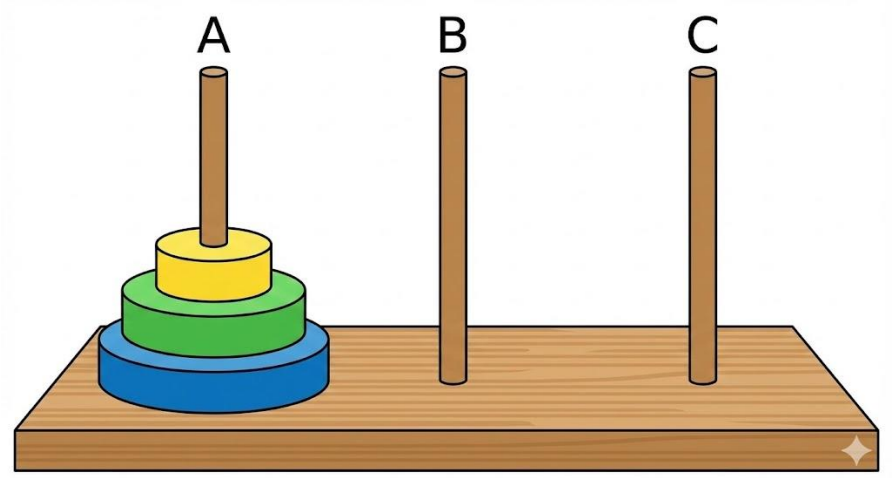
- Điều kiện dừng (base case) của đệ quy. Khi chỉ còn 1 đĩa, bài toán rất đơn giản: chỉ cần 1 bước di chuyển.

print (f"Di chuyển đĩa 1 từ {source} sang {target}")

- In ra hành động cụ thể: di chuyển đĩa số 1 từ cọc source sang target.
- f"..." là **f-string** — cú pháp để nhúng biến vào chuỗi.

return

- Kết thúc hàm cho trường hợp n == 1. Không gọi đệ quy nữa.



Ba bước đệ quy (trường hợp n > 1)

hanoi(n - 1, source, auxiliary, target)

- Bước 1: Chuyển n-1 đĩa nhỏ nhất từ source sang auxiliary, sử dụng target làm cọc phụ tạm. Lưu ý: đây là lời gọi đệ quy — hàm tự gọi lại chính nó với n-1.

print(f"Di chuyển đĩa {n} từ {source} sang {target}")

- Bước 2: Sau khi n-1 đĩa đã được chuyển sang auxiliary, ta có thể di chuyển đĩa lớn nhất (đĩa số n) từ source sang target. In hành động này.

hanoi(n - 1, auxiliary, target, source)

- Bước 3: Chuyển n-1 đĩa từ auxiliary lên target (lên trên đĩa n) sử dụng source làm cọc phụ.

3.3. Hướng dẫn chạy thử và gỡ lỗi (Debugging)

Chuẩn bị môi trường

Học sinh có thể dùng một trong các môi trường sau:

Cách 1: Dùng website (dễ nhất)

- Truy cập các website có sẵn IDE Python (online-python.com, onlinegdb.com, …)
- Dán mã nguồn vào khung soạn thảo.
- Nhấn **Run** để chạy.

Cách 2: Dùng Python cài trong máy

- Cài Python từ python.org
- Lưu file thành: hanoi.py
- Chạy bằng lệnh: python hanoi.py

Chạy thử chương trình

Học sinh nên bắt đầu với số đĩa nhỏ, chẳng hạn:

hanoi(2, 'A', 'C', 'B')

Với 2 hoặc 3 đĩa, số bước ít, học sinh dễ theo dõi hơn và hiểu rõ cách chương trình hoạt động.

Để quan sát rõ hơn quá trình đệ quy, học sinh có thể chèn thêm dòng in thông báo:

print(f"Gọi hanoi({n}, {source}, {target}, {auxiliary})")

Điều này giúp học sinh nhìn được trình tự gọi hàm bên trong.

Ví dụ:

```
def thap_ha_noi_debug(n, nguon, dich, trung_gian, indent=0):
```

```
    # Tạo khoảng trắng thụt đầu dòng để mô phỏng độ sâu của ngăn xếp
    spaces = "    " * indent
    print(f"{spaces}-> Bắt đầu hàm với n={n}. Mục tiêu: {nguon} -> {dich}")

    if n == 1:
        print(f"{spaces}    * HÀNH ĐỘNG: Chuyển đĩa 1 từ {nguon} sang {dich}")
        print(f"{spaces}<- Kết thúc hàm n=1")
        return

    thap_ha_noi_debug(n - 1, nguon, trung_gian, dich, indent + 1)

    print(f"{spaces}    * HÀNH ĐỘNG: Chuyển đĩa {n} từ {nguon} sang {dich}")

    thap_ha_noi_debug(n - 1, trung_gian, dich, nguon, indent + 1)

    print(f"{spaces}<- Kết thúc hàm n={n}")

thap_ha_noi_debug(3, 'A', 'C', 'B')
```

Các lỗi thường gặp và cách gỡ lỗi

Lỗi 1: Chương trình không in gì

- Nguyên nhân: gọi hàm sai tham số, ví dụ chỉ gọi hanoi().
- Cách sửa: gọi đầy đủ 4 tham số, ví dụ:
- hanoi(3, 'A', 'C', 'B')

Lỗi 2: Thứ tự các bước bị sai

Lỗi này xảy ra khi học sinh đổi nhầm vị trí của cọc đích và cọc phụ trong lời gọi đệ quy.

Thứ tự đúng phải luôn là:

1. Chuyển n-1 đĩa sang cọc phụ
2. Chuyển đĩa lớn nhất sang cọc đích
3. Chuyển n-1 đĩa từ cọc phụ sang cọc đích

Học sinh cần kiểm tra kỹ lại hai lời gọi:

hanoi(n - 1, source, auxiliary, target)

hanoi(n - 1, auxiliary, target, source)

Lỗi 3: Đệ quy lặp vô hạn

- Nguyên nhân: thiếu điều kiện dừng.

- Điều kiện dừng đúng phải là:

if n == 1:

return

Lỗi 4: Tên cọc in ra bị sai

Học sinh thường ghi nhầm cố định tên cọc, ví dụ:

print("Di chuyển đĩa", n, "từ A sang C") # sai

Cần dùng biến:

print(f"Di chuyển đĩa {n} từ {source} sang {target}") # đúng

Kiểm tra kết quả đúng hay sai

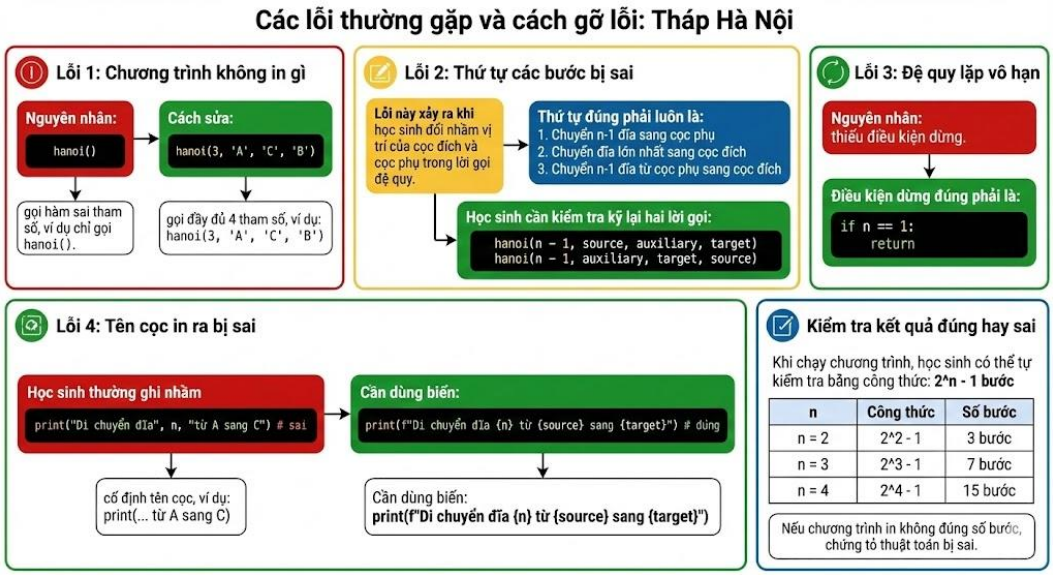
Khi chạy chương trình, học sinh có thể tự kiểm tra bằng công thức:

$$\text{số bước} = 2^n - 1$$

Ví dụ:

- n = 2 → 3 bước
- n = 3 → 7 bước
- n = 4 → 15 bước

Nếu chương trình in không đúng số bước, chứng tỏ thuật toán bị sai.



CÂU HỎI ÔN TẬP

Câu 1: Trong Python, từ khóa nào được sử dụng để bắt đầu định nghĩa một hàm?

A. function
B. define
C. def
D. func

Câu 2: Xem đoạn mã nguồn hàm Tháp Hà Nội dưới đây

```
def hanoi(n, nguon, dich, trung_gian):
    if n == 1:
        print(f"Chuyển {nguon} -> {dich}")
        return
    hanoi(n - 1, nguon, trung_gian, dich)  # Dòng A
    print(f"Chuyển {nguon} -> {dich}")
    hanoi(n - 1, trung_gian, dich, nguon)  # Dòng B
```

Tại dòng A (Bước 1 của thuật toán), vai trò của cọc dich lúc này là gì?

A. Cọc nguồn
B. Cọc đích
C. Cọc trung giang
D. Không đóng vai trò gì

Câu 3: Chương trình sau đây đang bị lỗi gì?

```
def sum(n):
    return n + sum(n - 1)

if __name__ == "__main__":
    n = 5
    print(sum(n))
```

A. Chương trình chạy bình thường như khôgn in ra kết quả
B. Chương trình báo lỗi cú pháp (Syntax Error)
C. Chương trình báo lỗi sai tham số
D. Chương trình lặp vô hạn dẫn đến lỗi tràn ngăn xếp (RecursionError)

Câu 4: Để theo dõi (trace) giá trị của các tham số thay đổi như thế nào qua từng bước gọi đệ quy nhằm mục đích gỡ lỗi (debugging), phương pháp đơn giản nhất là gì?

A. Viết lại toàn bộ chương trình bằng vòng lặp
B. Thêm lệnh print () để in giá trị các tham số ngay đầu thân hàm.
C. Đổi tên các biến thành chữ in hoa.
D. Xóa bớt các tham số không cần thiết.

Câu 5: Khi chạy hàm hanoi(2, 'A', 'C', 'B'), kết quả in ra màn hình sẽ có thứ tự nào sau đây?

A. Chuyển A -> B, Chuyển A -> C, Chuyển B -> C.
B. Chuyển A -> C, Chuyển A -> B, Chuyển B -> C.
C. Chuyển A -> C, Chuyển B -> C, Chuyển A -> B.
D. Chuyển A -> B, Chuyển B -> C, Chuyển A -> C.

Câu 6: Sửa lỗi tràn ngăn xếp cho chương trình sau:

```
def sum(n):
    return n + sum(n - 1)

if __name__ == "__main__":
    n = 5
    print(sum(n))
```

4. LẬP TRÌNH MÔ PHỎNG ĐỒ HỌA (Mở rộng)

4.1. Giới thiệu thư viện đồ họa

Để vẽ hình và tạo chuyển động trong Python, ta có thể dùng thư viện **turtle** – một thư viện đồ họa đơn giản, phù hợp cho người mới bắt đầu.

Em có thể hiểu **turtle** như một "con rùa vẽ" trên màn hình:

- Màn hình là một tờ giấy trắng.
- Con rùa là cây bút biết di chuyển lên – xuống – sang trái – sang phải.
- Mỗi lần rùa di chuyển, nó để lại nét vẽ (nếu bút đang được "hạ xuống").

Một ví dụ tối giản:

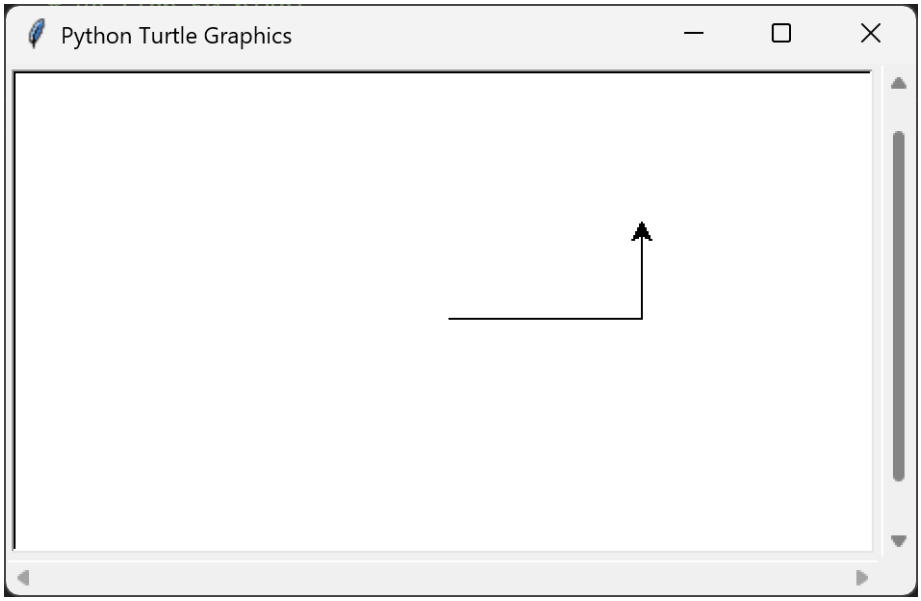
```
import turtle

screen = turtle.Screen()      # Tạo cửa sổ
pen = turtle.Turtle()         # Tạo một "con rùa" (bút vẽ)

pen.forward(100)              # Vẽ đoạn thẳng dài 100 pixel
pen.left(90)                  # Quay trái 90 độ
pen.forward(50)               # Vẽ tiếp 50 pixel

screen.mainloop()             # Giữ cửa sổ không bị đóng
ngay
```

Khi chạy đoạn code này, em sẽ thấy một cửa sổ hiện ra và một đường gấp khúc được vẽ.



Hình 3-Minh họa vẽ mũi tên bằng Python Turtle Graphics

Trong dự án Tháp Hà Nội, chúng ta **không cần** vẽ những hình quá phức tạp. Chỉ cần:

- Một vài **hình chữ nhật hẹp, dài** để làm **cọc** (peg).
- Một số **hình chữ nhật rộng** với độ rộng khác nhau để làm **đĩa** (disk).

Ta sẽ quy ước:

- Vẽ 3 cọc thẳng đứng đặt đều nhau trên màn hình (trái – giữa – phải).
- Các đĩa ban đầu xếp chồng lên **cọc bên trái**.
- Mỗi lần di chuyển đĩa, ta "cập nhật" lại hình vẽ để đĩa đó chuyển sang cọc mới.

Như vậy, việc em đã hiểu thuật toán (phần 2 và 3) sẽ giúp rất nhiều:

Thuật toán vẫn thế, chỉ thêm "phần vẽ" để minh họa.

4.2. Xây dựng đối tượng Đĩa và Cọc

Để mô phỏng bằng đồ họa, em cần làm rõ:

- **Cọc** là gì trong chương trình?
- **Đĩa** là gì trong chương trình?
- Làm sao biết đĩa nào đang nằm trên cọc nào?

Thay vì nghĩ "đĩa" và "cọc" chỉ là khái niệm toán học, ta sẽ **biểu diễn** chúng bằng dữ liệu:

1. Biểu diễn cọc (peg)

Ta có 3 cọc: A, B, C.

Trong màn hình đồ họa, mỗi cọc tương ứng với **một vị trí x** cố định:

```
peg_positions = {
    'A': -150,   # cọc bên trái
    'B': 0,      # cọc giữa
    'C': 150     # cọc bên
phải
}
```

Mỗi cọc là **một đường thẳng đứng**, có thể vẽ bằng hàm:

```
def draw_peg(pen, x):
    pen.penup()
    pen.goto(x, -100)   # chân cọc
    pen.pendown()
    pen.goto(x, 100)    # đỉnh cọc
```

2. Biểu diễn đĩa (disk)

Giả sử có n đĩa, ta đánh số từ 1 (nhỏ nhất) đến n (lớn nhất).

Một cách đơn giản:

- Mỗi đĩa được biểu diễn bằng **một hình chữ nhật** có:
  - **Chiều rộng** phụ thuộc vào số thứ tự đĩa (đĩa lớn rộng hơn).
  - **Chiều cao** giống nhau cho tất cả các đĩa.
- Vị trí của đĩa trên màn hình được quyết định bởi:
  - Cọc nó đang đứng (A/B/C).
  - Tầng (thứ tự từ dưới lên) trên cọc.

Ta có thể lưu trạng thái của các cọc bằng một **từ điển**:

```
# Ví dụ với 3 đĩa, ban đầu tất cả ở cọc A
state = {
    'A': [3, 2, 1],  # từ dưới lên: 3 (to), 2, 1 (nhỏ
nhất)
    'B': [],
    'C': []
}
```

Khi vẽ một đĩa, ta cần:

- Biết nó đang ở cọc nào (A/B/C).
- Biết nó ở tầng thứ mấy trên cọc (0, 1, 2, …) để tính **tọa độ y**.

Ví dụ một hàm vẽ một đĩa:

```
def draw_disk(pen, peg, disk_size, level):
    x = peg_positions[peg]
    width = 40 + disk_size * 20    # Đĩa lớn rộng hơn
    height = 20                    # Chiều cao đĩa
    y = -100 + level * height      # Mỗi tầng cao thêm 20
pixel


    pen.penup()
    pen.goto(x - width // 2, y)    # Góc trái dưới
    pen.pendown()
    pen.begin_fill()
    for _ in range(2):
        pen.forward(width)
        pen.left(90)
        pen.forward(height)
        pen.left(90)
    pen.end_fill()
```

Ở đây, em không cần thuộc hết code, chỉ cần hiểu ý:

Mỗi lần vẽ, ta đọc "trạng thái" của cọc và đĩa → chuyển nó thành hình trên màn hình.

Sau khi có draw_peg và draw_disk, ta có thể viết thêm hàm draw_state để xóa màn hình cũ và vẽ lại toàn bộ Tháp Hà Nội với trạng thái mới.

4.3. Lập trình chuyển động

Bây giờ, nhiệm vụ chính: **làm cho đĩa "chuyển động"** mỗi khi thuật toán đệ quy quyết định "Di chuyển đĩa n từ A sang C".

Ý tưởng rất giống phần in chữ trong 3.2, chỉ khác ở chỗ:

- Trước đây: khi di chuyển, ta **in ra một dòng chữ**.

- Bây giờ: khi di chuyển, ta **cập nhật trạng thái các cọc**, rồi **vẽ lại** hình.

- Bước 1. Viết hàm cập nhật trạng thái

Khi di chuyển đĩa từ cọc source sang target, ta:

1. Lấy đĩa trên cùng của cọc source.
2. Bỏ đĩa đó vào đỉnh cọc target.

Ví dụ:

```
def move_disk_state(state, source, target):
    disk = state[source].pop()    # Lấy đĩa trên
cùng
    state[target].append(disk)    # Đặt lên cọc đích
```

Bước 2. Gắn trạng thái với việc vẽ

Sau khi move_disk_state thay đổi state, ta gọi draw_state(pen, state) để vẽ lại cả ba cọc và các đĩa theo trạng thái mới.

Ví dụ:

```
def draw_state(pen, state):
    pen.clear()          # Xóa hình cũ
    # Vẽ 3 cọc
    for peg in ['A', 'B', 'C']:
        draw_peg(pen, peg_positions[peg])
    # Vẽ các đĩa trên từng cọc
    for peg in ['A', 'B', 'C']:
        peg_disks = state[peg]
        for level, disk in
enumerate(peg_disks):
            draw_disk(pen, peg, disk, level)
```

Bước 3. Kết nối với thuật toán đệ quy

Thay vì chỉ print(...) trong hàm hanoi, ta sẽ:

- Cập nhật state bằng move_disk_state.
- Vẽ lại trạng thái bằng draw_state.
- Thêm một chút time.sleep() để em thấy được chuyển động.

Ví dụ:

```
import time

def hanoi_graphic(n, source, target, auxiliary, state, pen):
    if n == 1:
        move_disk_state(state, source, target)
        draw_state(pen, state)
        time.sleep(0.5)    # Tạm dừng 0.5 giây cho dễ quan sát
        return

    hanoi_graphic(n - 1, source, auxiliary, target, state, pen)
    move_disk_state(state, source, target)
    draw_state(pen, state)
    time.sleep(0.5)
    hanoi_graphic(n - 1, auxiliary, target, source, state, pen)
```

Khi chương trình chạy:

1. hanoi_graphic hoạt động **y hệt thuật toán đệ quy** em đã học, chỉ thêm vài dòng code để **chuyển trạng thái → vẽ lại**.
2. Mỗi bước di chuyển, em sẽ thấy **một đĩa "nhảy" sang cọc mới** trên màn hình.
3. Sau cùng, tất cả đĩa sẽ được xếp gọn trên cọc đích, đúng thứ tự.

Như vậy, phần 4 chính là **"bọc" lại thuật toán ở phần 3** bằng lớp "đồ họa" bên ngoài, giúp em thấy trực quan cách đệ quy đang hoạt động.

4.4. Kết quả mô phỏng

Khi hoàn thành, chương trình mô phỏng đồ họa Tháp Hà Nội của em sẽ:

- Mở ra một cửa sổ với **3 cọc thẳng đứng**.
- Các đĩa có kích thước khác nhau xếp chồng lên nhau ở cọc A (bên trái).
- Từng đĩa **lần lượt "trượt" sang cọc khác**, đúng theo thứ tự các bước của thuật toán đệ quy.
- Kết thúc, tất cả đĩa sẽ nằm gọn trên cọc C (bên phải), từ lớn đến nhỏ.

Giáo viên có thể dùng sản phẩm này để:

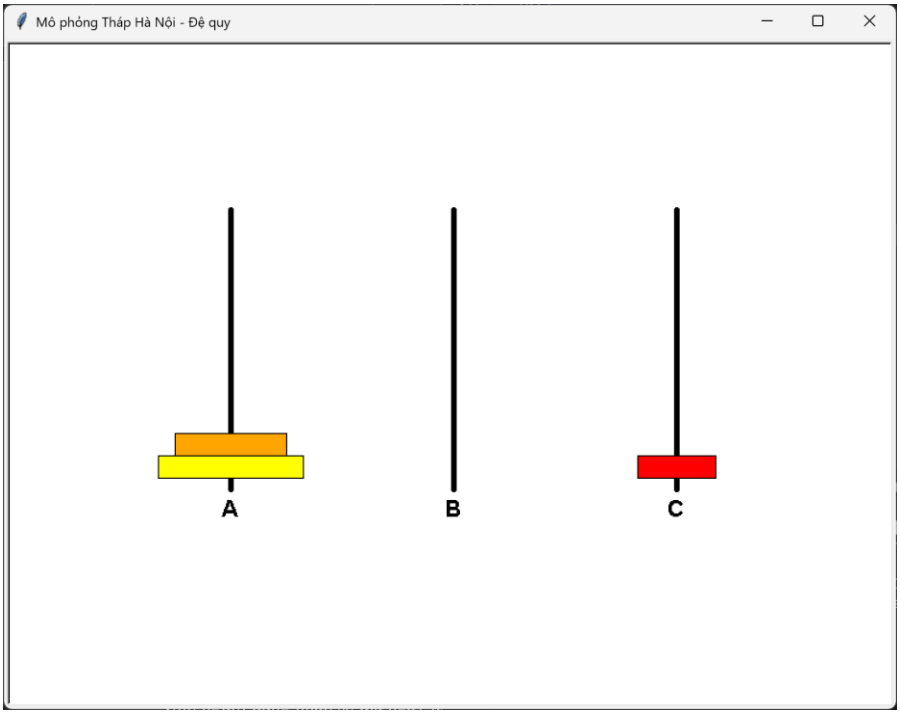
- **Minh họa trực quan trong lớp**: học sinh nhìn thấy từng bước di chuyển đĩa, dễ hiểu logic hơn so với chỉ nhìn mã nguồn.
- Là **sản phẩm dự án** của nhóm em trong chuyên đề 11.1 (file .py + video quay lại màn hình mô phỏng).

Để **khuyến khích sáng tạo**, em có thể tự mở rộng:

- Thay đổi màu sắc mỗi đĩa.
- Cho người dùng nhập số đĩa ngay từ đầu.
- Thêm bộ đếm số bước di chuyển hiển thị trên màn hình.
- Thêm nút "Tăng tốc/giảm tốc" bằng cách thay đổi thời gian sleep.

Những ý tưởng này không bắt buộc, nhưng là cơ hội để em:

- Vừa ôn lại **thuật toán đệ quy**,
- Vừa rèn luyện **kỹ năng lập trình đồ họa** – bước đệm tốt cho lập trình game, mô phỏng, và các dự án Tin học nâng cao sau này.



Hình 4-Mô phỏng THN

CÂU HỎI ÔN TẬP

Câu 1: *Trong thư viện turtle của Python, để di chuyển bút vẽ đến một tọa độ (x, y) bất kỳ mà không để lại nét vẽ trên màn hình (ví dụ: di chuyển từ chân cọc lên đỉnh cọc), ta cần sử dụng cặp lệnh nào?*

A. pen.forward() và pen.backward()

B. pen.penup() và pen.goto(x, y)

C. pen.pendown() và pen.goto(x, y)

D. pen.clear() và pen.update()

Câu 2: Để tạo hiệu ứng "hoạt hình" (animation) cho các đĩa di chuyển trong bài toán Tháp Hà Nội, quy trình nào sau đây là chính xác nhất được đề cập trong tài liệu?

A. Vẽ đĩa mới đè lên đĩa cũ.

B. Xóa toàn bộ màn hình (clear) -> Cập nhật trạng thái vị trí các đĩa -> Vẽ lại toàn bộ (draw) -> Tạm dừng (sleep)

C. Chỉ cần cập nhật dữ liệu trong danh sách (list), hình ảnh sẽ tự thay đổi.

D. Vẽ đĩa di chuyển từ từ từng pixel một từ cọc này sang cọc khác.

Câu 3: Trong việc biểu diễn dữ liệu cho đồ họa, tại sao chúng ta sử dụng cấu trúc dữ liệu Danh sách (List) hoặc Ngăn xếp (Stack) để lưu trữ các đĩa trên 3 cọc A, B, C (ví dụ: state = {'A': [3, 2, 1], 'B': [], 'C': []})?

A. Vì List có màu sắc đẹp hơn.

B. Vì List trong Python tự động vẽ hình chữ nhật

C. Vì các thao tác pop() (lấy ra) và append() (thêm vào) của List mô phỏng chính xác hành động lấy đĩa trên cùng và đặt đĩa xuống cọc mới.

D. Vì thư viện Turtle chỉ làm việc được với List.

Câu 4: Dựa vào kiến thức phần 4.2 (Xây dựng đối tượng Đĩa và Cọc) , hãy viết hàm draw_rectangle(pen, color, width, height) sử dụng thư viện turtle để vẽ một hình chữ nhật đặc (dùng làm đĩa). Yêu cầu: Hàm nhận vào bút vẽ, màu sắc, chiều rộng, chiều cao. Hình chữ nhật được tô màu bên trong.

Câu 5: Trong phần 4.3 (Lập trình chuyển động), chuyển động của đĩa thực chất là việc cập nhật dữ liệu trạng thái. Hãy viết hàm move_disk_logic(state, source, target) để thực hiện logic di chuyển một đĩa:

Input: Biến state (từ điển chứa danh sách đĩa ở 3 cọc), tên cọc nguồn source, tên cọc đích target.

Xử lý: Lấy đĩa trên cùng của cọc nguồn và đặt nó sang cọc đích.

5. PHÂN TÍCH HIỆU SUẤT THUẬT TOÁN (Nâng cao)

5.1. Phân tích độ phức tạp thời gian (Time Complexity)

Khi học lập trình, chúng ta không chỉ quan tâm *chương trình chạy đúng*, mà còn phải xem *chạy nhanh hay chậm*, đặc biệt khi dữ liệu lớn. Với thuật toán Tháp Hà Nội, thời gian chạy phụ thuộc vào **số lượng đĩa n**.

Điều quan trọng là: **Mỗi lần tăng thêm 1 đĩa, số bước di chuyển tăng gấp đôi**

Vì sao vậy?

Khi có **n đĩa**, muốn chuyển toàn bộ sang cọc đích, ta phải:

1. Chuyển **n–1 đĩa** sang cọc trung gian
2. Chuyển **đĩa lớn nhất**
3. Chuyển **n–1 đĩa** còn lại sang cọc đích

Tức là công việc gấp đôi lượng của bài toán n – 1, rồi thêm 1 bước chuyển đĩa lớn nhất.

Vì vậy, số bước tăng rất nhanh. Thuật toán này được xem là thuật toán có **độ phức tạp theo cấp số nhân**, ký hiệu là:

$$\mathbf{O(2^n)}$$

Nghĩa là: Khi n tăng lên một chút, thời gian chạy tăng rất mạnh - gấp đôi mỗi lần tăng 1 đĩa.

5.2. Xây dựng công thức truy hồi tính số bước di chuyển

Hãy gọi **T(n)** là số bước cần thiết để di chuyển n đĩa.

Dựa vào quy tắc 3 bước của thuật toán, ta có:

1. Chuyển **n–1 đĩa** sang cọc phụ → mất **T(n–1)** bước
2. Chuyển **đĩa lớn nhất** → mất **1 bước**
3. Chuyển **n–1 đĩa** sang cọc đích → mất **T(n–1)** bước

Gộp lại:

$$\mathbf{T(n) = 2 \times T(n–1) + 1}$$

Đây là công thức gọi là **công thức truy hồi** – vì giá trị hiện tại phụ thuộc vào giá trị của bước trước đó (n–1).

Ví dụ:

- $T(1) = 1$
- $T(2) = 2 \cdot T(1) + 1 = 3$
- $T(3) = 2 \cdot T(2) + 1 = 7$
- $T(4) = 2 \cdot T(3) + 1 = 15$

5.3. Chứng minh công thức tổng quát

Từ bảng ví dụ:

n	***T(n)***	***$2^n - 1$***
1	1	1
2	3	3
3	7	7
4	15	15

Ở phần trên, chúng ta đã tìm được công thức truy hồi là: **T(n) = 2 × T(n–1) + 1** và đã khảo sát bằng việc thay ***n*** thành giá trị trong khoảng từ [1,4]. Sau khi tính, ta có thể nhận thấy:

- $T(1) = 1 = 2^1 - 1$
- $T(2) = 3 = 2^2 - 1$
- $T(3) = 7 = 2^3 - 1$
- $T(4) = 15 = 2^4 - 1$

Từ đó, ta có thể phỏng đoán được công thức tổng quát là ***$T(n) = 2^n - 1$***

Cụ thể hơn, ta tiến hành kiểm chứng phỏng đoán bằng phương pháp quy nạp như sau:

Ta đã có **công thức truy hồi**:

$$T(1) = 1, T(n) = 2T(n-1) + 1 \text{ với } n \geq 2$$

Ta **phỏng đoán** (từ bảng giá trị) rằng:

$$T(n) = 2^n - 1$$

Bây giờ ta sẽ **chứng minh bằng quy nạp** trên n.

- Bước 1 – Kiểm tra với n = 1 (bước cơ sở)

Từ đề bài:

$$T(1) = 1$$

Còn công thức ta muốn chứng minh cho n = 1:

$$2^1 - 1 = 2 - 1 = 1$$

Ta thấy hai giá trị bằng nhau, nên **công thức đúng với n = 1**.

- Bước 2 – Giả sử đúng với n = k (giả thuyết quy nạp)

Giả sử rằng với một số nguyên dương k nào đó, ta đã có:

$$T(k) = 2^k - 1$$

*(Đây là giả thiết tạm thời, **chưa kết luận**, chỉ là "giả sử đúng tới k".)*

- Bước 3 – Chứng minh đúng với n = k + 1

Dùng công thức truy hồi:

$$T(k+1) = 2T(k) + 1$$

Thay $T(k)$bằng biểu thức trong giả thuyết quy nạp:

$$T(k+1) = 2(2^k - 1) + 1$$

Rút gọn:

$$T(k+1) = 2^{k+1} - 2 + 1 = 2^{k+1} - 1$$

Đây chính là **dạng cần chứng minh cho n = k+1**.

Theo **nguyên lý quy nạp toán học**, suy ra:

$$\boxed{T(n) = 2^n - 1 \text{ đúng với mọi } n \geq 1}$$

Vậy có thể kết luận công thức tổng quát là ***T(n) = 2ⁿ – 1.***

CÂU HỎI ÔN TẬP

Câu 1: Phân tích độ phức tạp thuật toán:

Quan sát hàm đệ quy hanoi(n, ...) mô phỏng bài toán Tháp Hà Nội dưới đây:

```
def hanoi(n, ...):
    if n == 1:
        # Tốn 1 đơn vị thời gian (c) để in ra màn hình
        print(...)
        return

    # Bước 1: Gọi đệ quy cho n-1 đĩa
    hanoi(n - 1, ...)

    # Bước 2: Hành động in ra màn hình (tốn hằng số thời gian c)
    print(...)

    # Bước 3: Gọi đệ quy cho n-1 đĩa
    hanoi(n - 1, ...)
```

Yêu cầu:

a) Gọi T(n) là thời gian thực hiện của hàm với n đĩa. Hãy viết biểu thức liên hệ giữa T(n) và T(n-1).

b) Dựa trên biểu thức đó, hãy giải thích ngắn gọn tại sao độ phức tạp thời gian (Big-O) của thuật toán này lại là O(2^n) mà không phải là O(n) hay O(n^2).

Câu 2: Xây dựng công thức tổng quát

Một học sinh đã chạy thử chương trình Tháp Hà Nội và ghi lại số bước di chuyển tối thiểu S_n cần thiết cho các trường hợp n đĩa như bảng sau:

Số đĩa (n)	**1**	**2**	**3**	**4**	**5**
Số bước ($\boldsymbol{S_n}$)	1	3	7	15	?

a) Hãy điền giá trị số bước cho trường hợp n=5 vào bảng trên.

b) Giả sử máy tính có thể thực hiện 1 tỷ (10^9) phép di chuyển mỗi giây. Hãy tính thời gian (theo giây) để máy tính giải quyết bài toán với n=30. (Lấy $2^{10} \approx 10^3$).

6. THẢO LUẬN

*1) Tại sao thuật toán có độ phức tạp **O(2^n)**?*

Vì công thức truy hồi T(n) = 2T(n–1) + 1 giống như một cấp số nhân.

Mỗi lần tăng 1 đĩa → số bước gần như **tăng gấp đôi**.

Cụ thể:

- n = 5 → T(5) = 31 bước
- n = 10 → T(10) = 1023 bước
- n = 20 → T(20) = 1,048,575 bước
- n = 30 → hơn **1 tỉ** bước

Suy ra thuật toán chạy rất chậm khi n lớn.

2) Thời gian cần thiết để giải bài toán với n=20, n=64?

Nếu giải bài toán với n = 20?

Ta dùng công thức:

$$T(20) = 2^{20} - 1 = 1{,}048{,}575 \text{ bước}$$

Nếu mỗi bước mất 1 mili–giây (0.001 giây), thời gian chạy ≈ 1,048,575 ms ~ **17.5 phút**

Nếu giải bài toán với n = 64?

$$T(64) = 2^{64} - 1$$
$$= 18{,}446{,}744{,}073{,}709{,}551{,}615 \text{ bước}$$
$$\approx \mathbf{1.8 \times 10^{19} \text{ bước}}$$

Nếu máy tính thực hiện 1 tỉ bước/giây (rất mạnh!), thời gian sương sương có thể… khoảng **584 năm.** Vì vậy, không thể giải Tháp Hà Nội với 64 đĩa bằng mô phỏng thông thường.

PHỤ LỤC 1: CÁC CÔNG CỤ HỖ TRỢ THỰC HIỆN DỰ ÁN

- Các công cụ có thể khởi động được chương trình Python:

+ Google Colab: là một dịch vụ Jupyter Notebook được lưu trữ, không yêu cầu thiết lập để sử dụng và cung cấp quyền truy cập miễn phí vào các tài nguyên máy tính, bao gồm GPU và TPU. Colab đặc biệt phù hợp cho lĩnh vực học máy (machine learning), khoa học dữ liệu (data science) và giáo dục. Đây là nền tảng phù hợp để khởi động và kiểm tra đồ án trực tuyến.

+ Kaggle: tương tự như Google Colab.

+ Visual Studio Code: là một trình soạn thảo mã nguồn miễn phí, đa nền tảng, được phát triển bởi Microsoft. Đây là một IDE *(Integrated Development Environment – Môi trường phát triển tích hợp)* rất phù hợp cho việc lập trình, đặc biệt là ngôn ngữ lập trình Python.

- Các công cụ lưu trữ mã nguồn:

+ Google Drive: phổ biến và đi kèm với nhóm Google, có thể liên kết và lưu trữ với Google Colab.

+ GitHub: nền tảng lưu trữ mã nguồn lớn nhất thế giới, là nơi lưu trữ mã nguồn chuyên dùng cho các lập trình viên.

- Các công cụ khác gợi ý cho việc báo cáo:

+ Tạo slide thuyết trình: PowerPoint, Canva, Google Slides, …

+ Viết báo cáo: Microsoft Word, WPS, overleaf, …

Khuyến khích học sinh sử dụng trí tuệ nhân tạo (các công cụ AI như ChatGPT, Gemini…) nhưng không được lạm dụng hoặc phụ thuộc.

PHỤ LỤC 2: CÁC THÔNG TIN ĐÍNH KÈM

- A. Các bài toán đệ quy kinh điển khác (Giai thừa, Fibonacci).
- B. Mẫu Phiếu giao nhiệm vụ dự án.
- C. Mẫu Rubric đánh giá sản phẩm.
- D. Tài liệu tham khảo.

PHỤ LỤC 3: ĐÁP ÁN CỦA CÁC CÂU HỎI ÔN TẬP

1. GIỚI THIỆU VỀ ĐỆ QUY

1. C (Đoàn tàu chỉ là sự lặp lại tuyến tính, không có tính chất "chứa đựng chính nó" ở mức độ cấu trúc sâu như gương hay cây thư mục).
2. B.
3. Bài làm tham khảo:

Phân tích các bước gọi đệ quy (Quá trình đi xuống):

Theo công thức S(n) = n + S(n-1), ta có:

- S(10) = 10 + S(9)
- S(9) = 9 + S(8)
- S(8) = 8 + S(7)
- ... (tiếp tục giảm dần n) ...
- S(2) = 2 + S(1)
- S(1) = 1 + S(0)
- S(0) = 0 (Đây là **phần cơ sở** - điều kiện dừng của đệ quy)

Thay thế ngược để tính giá trị (Quá trình đi lên):

Bây giờ, ta thay giá trị từ dưới lên trên:

- S(0) = 0
- S(1) = 1 + 0 = 1
- S(2) = 2 + 1 = 3
- S(3) = 3 + 3 = 6
- S(4) = 4 + 6 = 10
- S(5) = 5 + 10 = 15
- S(6) = 6 + 15 = 21
- S(7) = 7 + 21 = 28
- S(8) = 8 + 28 = 36
- S(9) = 9 + 36 = 45
- S(10) = 10 + 45 = 55

Kết luận: Giá trị của S(10) là 55.

2. GIỚI THIỆU VỀ BÀI TOÁN THÁP HÀ NỘI

1. C (Quy tắc bắt buộc: Không bao giờ được đặt đĩa lớn lên đĩa nhỏ, dù cọc có trống hay không thì đĩa đặt xuống phải là đĩa nhỏ hơn đĩa đang nằm dưới nó - trừ khi đặt vào cọc trống).
2. C (Tư duy đệ quy: Muốn chuyển đĩa to nhất (3) sang C, phải dọn đường bằng cách chuyển n-1 đĩa (1,2) sang cọc phụ B trước)
3. B ($2^5 - 1 = 32 - 1 = 31$)
4. A (Quy trình: Chuyển 3 đĩa từ Nguồn → Trung gian => Chuyển đĩa 4 từ Nguồn → Đích => Chuyển 3 đĩa từ Trung gian → Đích. Vậy đĩa 4 phải đi từ Nguồn sang Đích).
5. Gợi ý trả lời: Để chuyển n đĩa, ta cần:
    a. Chuyển n – 1 đĩa sang cọc phụ (tốn S(n-1) bước)
    b. Chuyển đĩa thứ n sang cọc đích (tốn 1 bước)
    c. Chuyển n – 1 đĩa từ cọc phụ về cọc đích (tốn thêm S(n-1) bước)

Tổng: $S(n-1) + 1 + S(n-1) = 2 * S(n-1) + 1$. Do đó số bước tăng gấp đôi cộng 1.

3. THỰC HÀNH LẬP TRÌNH THÁP HÀ NỘI VỚI PYTHON

1. C (def)
2. C (Trong dòng A: hanoi(n-1, nguồn, trung_gian, đích), tham số thứ 3 là đích thực chất đang đóng vai trò là cọc phụ cho bước chuyển n-1 đĩa này)
3. D (Lỗi RecursionError/Stack Overflow)
4. B (Logging/Print debugging)
5. A (Quy tắc: Chuyển đĩa nhỏ sang trung gian -> Chuyển đĩa lớn sang đích -> Chuyển đĩa nhỏ từ trung gian sang đích)
6. Gợi ý: Lỗi tràn ngăn xếp do thiếu trường hợp cơ sở do đó ta cần xác định bước cơ sở của bài toán.

```
def sum(n):
    # Trường hợp cơ sở (base-case)
    if n == 1:
        return 1

    return n + sum(n - 1)

if __name__ == "__main__":
    n = 5
    print(sum(n))
```

4. LẬP TRÌNH MÔ PHỎNG ĐỒ HỌA

1. B (Lệnh penup() nhấc bút lên để không vẽ, sau đó goto() để di chuyển)
2. B (Đây là nguyên tắc cơ bản của hoạt hình máy tính: Xóa hình cũ -> Vẽ hình mới tại vị trí mới -> Tạo độ trễ để mắt người nhìn thấy) .
3. C (Cấu trúc List/Stack với pop và append phản ánh đúng quy tắc LIFO - vào sau ra trước của Tháp Hà Nội).
4. Bài làm tham khảo:

```
import turtle


def draw_rectangle(pen, color, width, height):
    pen.fillcolor(color)  # Chọn màu tô
    pen.begin_fill()      # Bắt đầu tô
    for _ in range(2):    # Vẽ hình chữ nhật
        pen.forward(width)
        pen.left(90)
        pen.forward(height)
        pen.left(90)
    pen.end_fill()        # Kết thúc tô
```

5. Bài làm tham khảo:

```
def move_disk_logic(state, source, target):
    # Kiểm tra xem cọc nguồn có đĩa không
    if len(state[source]) > 0:
        disk = state[source].pop()  # Lấy đĩa trên cùng ra (Move out)
        state[target].append(disk)  # Đặt đĩa vào cọc đích (Move in)
```

5. PHÂN TÍCH HIỆU SUẤT THUẬT TOÁN

Đáp án câu 1:

a) Biểu thức truy hồi: $T(n) = 2 \cdot T(n-1) + c$.
Giải thích*: Để giải quyết bài toán n đĩa, ta phải giải quyết 2 bài toán con kích thước n-1 (gọi hàm 2 lần) và thực hiện 1 bước chuyển cơ bản (cộng thêm hằng số c).*

b) Giải thích độ phức tạp $O(2^n)$:
   - Mỗi lần tăng thêm 1 đĩa, số lượng công việc (số lần gọi hàm) tăng lên gấp đôi.
   - Mô hình này giống như một cây nhị phân đầy đủ, số lượng nút của cây tăng theo lũy thừa của 2. Do đó, độ phức tạp là hàm mũ $O(2^n)$.

Đáp án câu 2:

a) Với n=5, số bước là 31 (theo quy tắc $2 \times 15 + 1 = 31$

b) Tính toán thời gian:

- Số bước với n=30 là $2^{30} - 1 \approx 2^{30}$.
- Ta có $2^{10} \approx 10^3$ (một nghìn), suy ra $2^{30} = \left(2^{\{10\}}\right)^3 \approx (10^3)^3 = 10^9$ (một tỷ bước).
- Tốc độ máy tính: 10^9 bước/giây.
- Thời gian $= \frac{10^9}{10^9} = 1$ giây.

Kết luận: Với n=30, máy siêu tính này chỉ mất khoảng 1 giây để giải.

NGUỒN THAM KHẢO

[1] L. V. Hải, "200Lab," 200Lab, 21 9 2023. [Online]. Available: https://200lab.io/blog/de-quy-la-gi. [Accessed 30 11 2025].

[2] L. V. Hải, "200Lab," 200Lab, 28 9 2023. [Online]. Available: https://200lab.io/blog/bai-toan-thap-ha-noi. [Accessed 30 11 2025].

[3] kartik, "Introduction to Recursion - GeeksforGeeks," 25 October 2025. [Online]. Available: https://www.geeksforgeeks.org/dsa/introduction-to-recursion-2/. [Accessed 12 February 2025].

[4] Harendra, "Stack Data Structure - GeeksforGeeks," 13 November 2025. [Online]. Available: https://www.geeksforgeeks.org/dsa/stack-data-structure/. [Accessed 12 February 2025].

TÀI LIỆU ĐỌC THÊM

1. A. Sweigart, The Recursive Book of Recursion: Ace the Coding Interview with Python and JavaScript. No Starch Press, 2022.
2. M. Lutz, Programming Python: Powerful Object-Oriented Programming. “O’Reilly Media, Inc.,” 2010.
3. P. T. Long, B. V. Hà, Chuyên đề học tập Tin Học 11 Định hướng Khoa học Máy tính. Nhà xuất bản giáo dục Việt Nam, 2022.
4. Bùi Việt Hà, Python Cơ Bản. Nhà xuất bản Đại học Quốc gia Hà Nội, 2023
5. TS. Trần Minh Sơn, PGS. TS T. T. Hà, TS. L. D.Trinh, TS. L. H. Minh, Python dành cho người tự học (phần cơ bản), Nhà xuất bản Thanh niên

CHUYÊN ĐỀ HỌC TẬP 11.1 –

THÁP HÀ NỘI - THUẬT TOÁN LẬP TRÌNH VÀ ỨNG DỤNG