KỸ THUẬT LẬP TRÌNH

Chương 2: Đệ qui

Mục tiêu

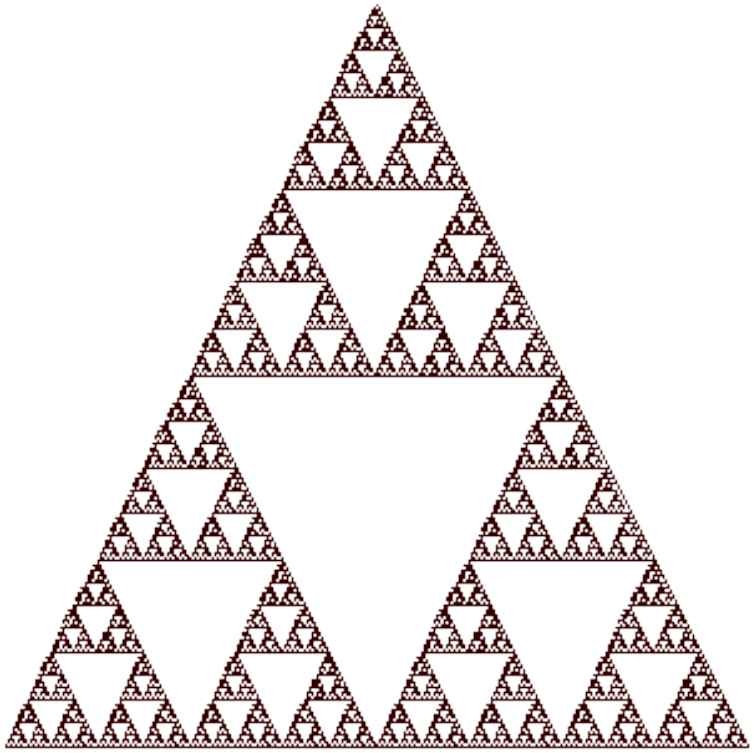
- Sau khi học xong chương này, người học có thể:

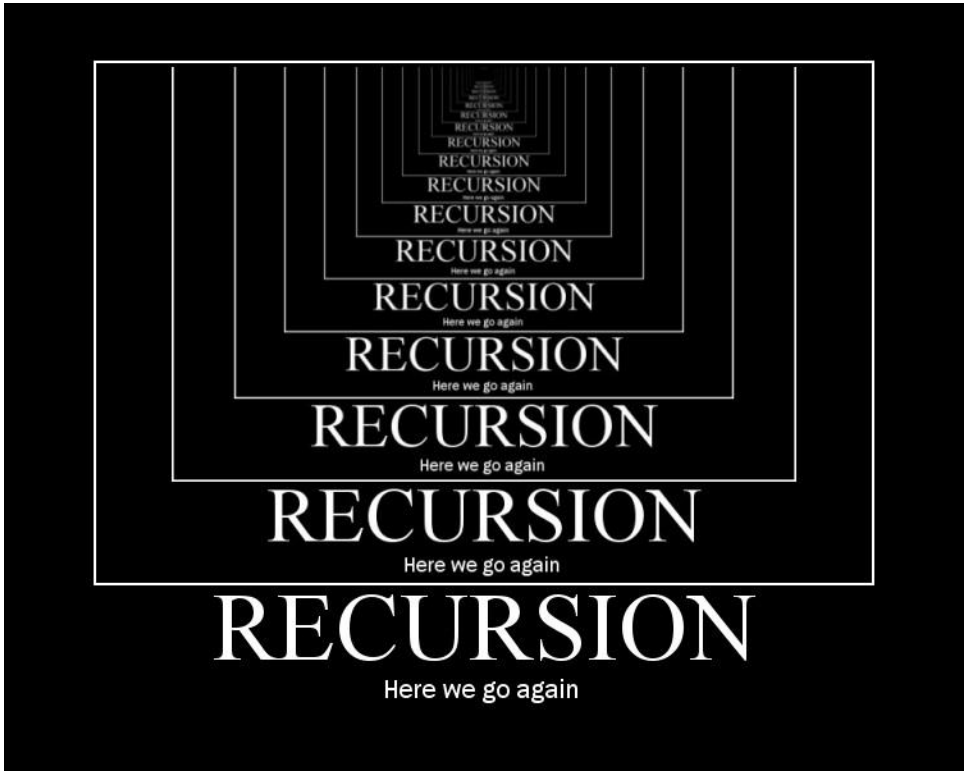
1. Hiểu được khái niệm và ý nghĩa của đệ qui
2. Hiểu và biết cách xây dựng hàm đệ qui

Nội dung

Đệ qui (Recursion)

1. Giới thiệu, định nghĩa đệ qui
2. Hàm đệ qui
3. Một số loại đệ qui
4. Bài toán Tháp Hà Nội
5. So sánh đệ qui với lặp





Nguồn hình ảnh: internet

- Thuật giải đệ qui (*recursive algorithm*): tinh giảm vấn đề hiện tại.
  - Có 1 hoặc nhiều trường hợp cơ sở (basic case): giải trực tiếp
  - Trường hợp tổng quát (general case): tinh giảm dần để quay về basic case.

- Thuật giải đệ qui (recursive algorithm) được cài đặt gọi là HÀM ĐỆ QUI (recursive function)
- Định nghĩa đệ qui: là một vấn đề mà trong định nghĩa của nó có sử dụng chính khái niệm đó.
- Định nghĩa đệ qui: là các đối tượng được định nghĩa dưới dạng qui nạp từ những khái niệm đơn giản nhất cùng dạng với nó.

- Ví dụ: định nghĩa giai thừa của 1 số nguyên không âm?
- Định nghĩa đệ qui:

$$n! = \begin{cases} 1 & \text{nếu } n = 0 \\ n \times (n-1)! & \text{nếu } n > 0 \end{cases}$$

- Trong đó:
  - n! = 1 nếu n = 0 gọi là trường hợp cơ sở (basic case)
  - n! = n x (n – 1)! Nếu n > 0 gọi là trường hợp tổng quát (general case)

- Ví dụ: định nghĩa giai thừa của 1 số nguyên không âm
- Định nghĩa đệ qui:

$$n! = \begin{cases} 1 & \text{nếu } n = 0 \\ n \times (n-1)! & \text{nếu } n > 0 \end{cases}$$

- **Tính 2! = ???**
  - 2!: 2 > 0: *general case* nên kết quả = 2 x (2 – 1)! = 2 x 1!
  - 1!: 1 > 0: *general case* nên kết quả = 1 x (1 – 1)! = 1 x 0!
  - 0!: 0 = 0: *basic case* nên kết quả = 1
  - Khi đó kết quả tính toán cuối cùng = 2 x 1 x 1 = 2

- Hàm đệ qui (recursive function) là hàm gọi chính bản thân nó, được định nghĩa và khai báo như các hàm khác.
- Thân hàm đệ qui bao gồm:
  - Phần giải quyết trường hợp cơ sở (basic case)
  - Phần giải quyết trường hợp tổng quát (general case)

- Thân hàm đệ qui có dạng tổng quát như sau:

if (biểu thức điều kiện đúng trong trường hợp cơ sở**)**

- ✓ Biểu thức tính toán hay trả về kết quả (không gọi đệ qui chính nó)

else

{

- ✓ Chia thành vấn đề con đồng dạng
- ✓ Các vấn đề con gọi chính bản thân hàm đệ qui đang xây dựng
- ✓ Tính toán, tổng hợp và trả về kết quả cuối

}

- Ví dụ: viết hàm đệ qui tính giai thừa của một số nguyên không âm.
- **Hàm không đệ qui: sử dụng vòng lặp khi tính**

```
int giaiThua( int n)
{
    //assert ( n >= 0);
    int kq = 1;
    for ( int i = 2; i <= n; i++)
        kq *= i;
    return kq;
}
```

- Ví dụ: viết hàm đệ qui tính giai thừa của một số nguyên không âm.
- **Hàm đệ qui: KHÔNG sử dụng vòng lặp khi tính**

```
int giaithua( int n)
{
    //assert ( n >= 0);
    if ( n == 0)
        return 1;
    else
        return n * giaithua( n - 1);
}
```

Basic case

General case

- **Ví dụ: viết hàm đệ qui tính giai thừa của một số nguyên không âm.**

- **Hàm đệ qui: KHÔNG sử dụng vòng lặp khi tính**

```
int giaithua( int n)
{
    //assert ( n >= 0);
    if ( n == 0)
        return 1;
    else
        return n * giaithua( n - 1);
}
```

Tính 3!

- **giaithua(3):**

return 3 * giaithua(2)

2 * giaithua(1)

1 * giaithua(0)

1

Trả kết quả

6

3 * 2

2 * 1

1

- Ví dụ: Viết hàm đệ qui tính tổng các số từ 1 đến n (n là số nguyên dương)

```
int tinhTong (int  n)
{
      if (n == 1)
            return 1;
      else
            return n + tinhTong(n - 1);
}
```

- Ví dụ: Viết hàm đệ qui để tính dãy số Fibonacci với số nguyên n >= 0:

$$f_n = \begin{cases} 0 & \text{nếu } n = 0 \\ 1 & \text{nếu } n = 1 \\ f_{n-1} + f_{n-2} & \text{nếu } n > 1 \end{cases}$$

- Basic case?
- General case?
- Recursive function?
- Fibonacci(5)

Bài tập

- Bài 1: Viết hàm đệ qui để tính tổng $S = 1^2 + 2^2 + \ldots + n^2$ với n là số nguyên dương. Sau đó viết chương trình để kiểm tra hàm vừa viết.
- Bài 2: Viết hàm đệ qui để đếm số chữ số của 1 số nguyên dương. Sau đó viết chương trình để kiểm tra hàm vừa viết.
- Bài 3: Viết hàm đệ qui để tính số đảo ngược của 1 số nguyên dương. Sau đó viết chương trình để kiểm tra hàm vừa viết.
- Bài 4: Viết hàm đệ qui để chuyển 1 số nguyên dương từ hệ thập phân sang nhị phân. Sau đó viết chương trình để kiểm tra hàm đã viết.

- **Đệ qui tuyến tính (linear recursion)**: trong thân hàm đệ qui chỉ 1 lần gọi lại chính bản thân nó.

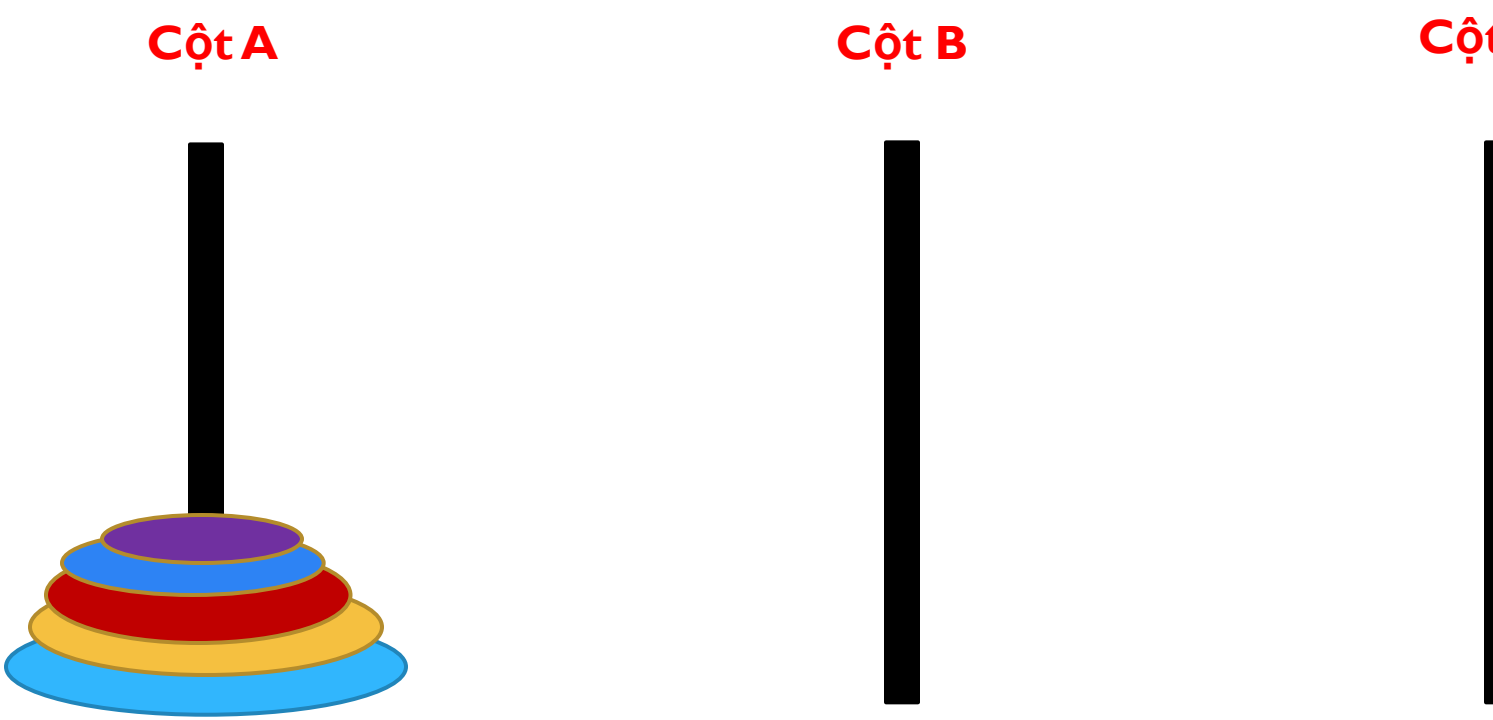
Ví dụ: hàm đệ qui tính giai thừa của số nguyên không âm.

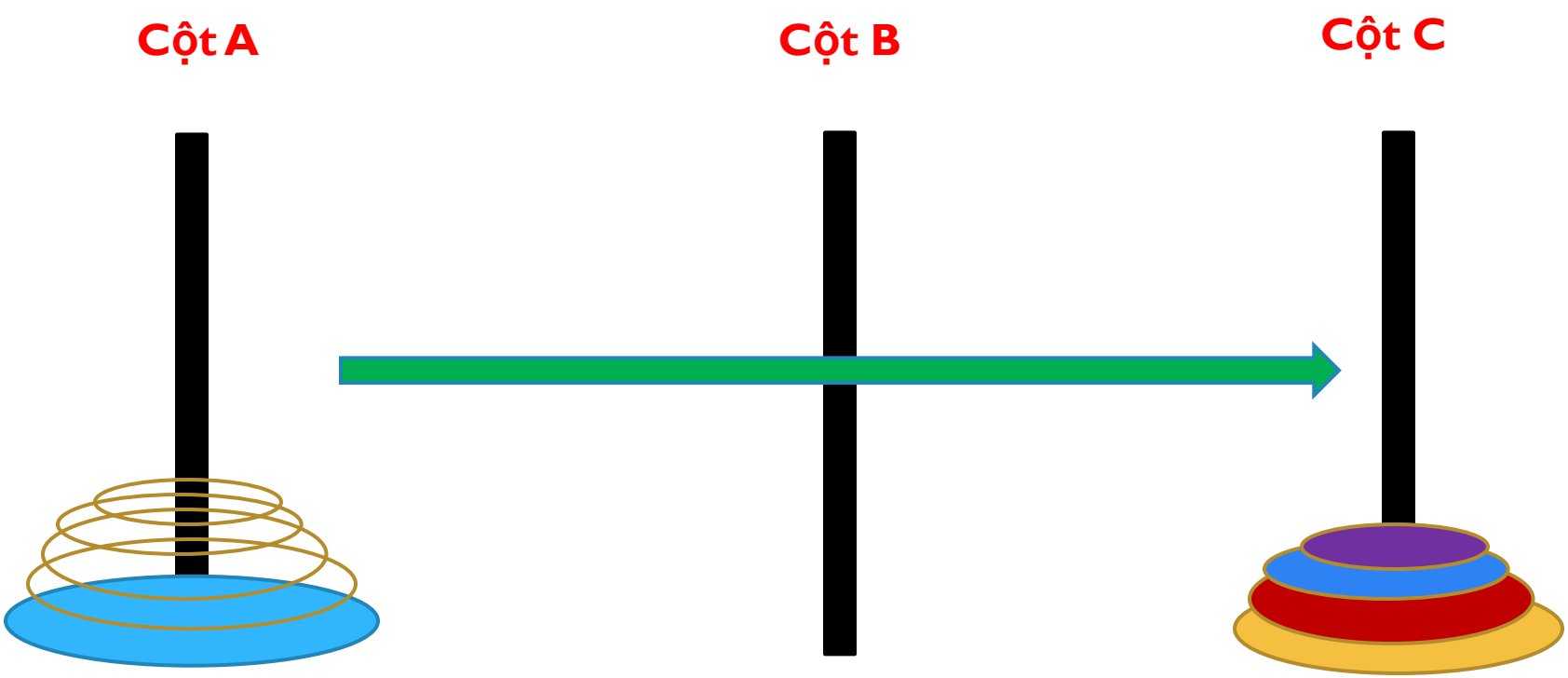
- **Đệ qui nhị phân (binary recursion)** – đệ qui nhánh: trong thân hàm đệ qui có 2 lần gọi đến chính bản thân nó.

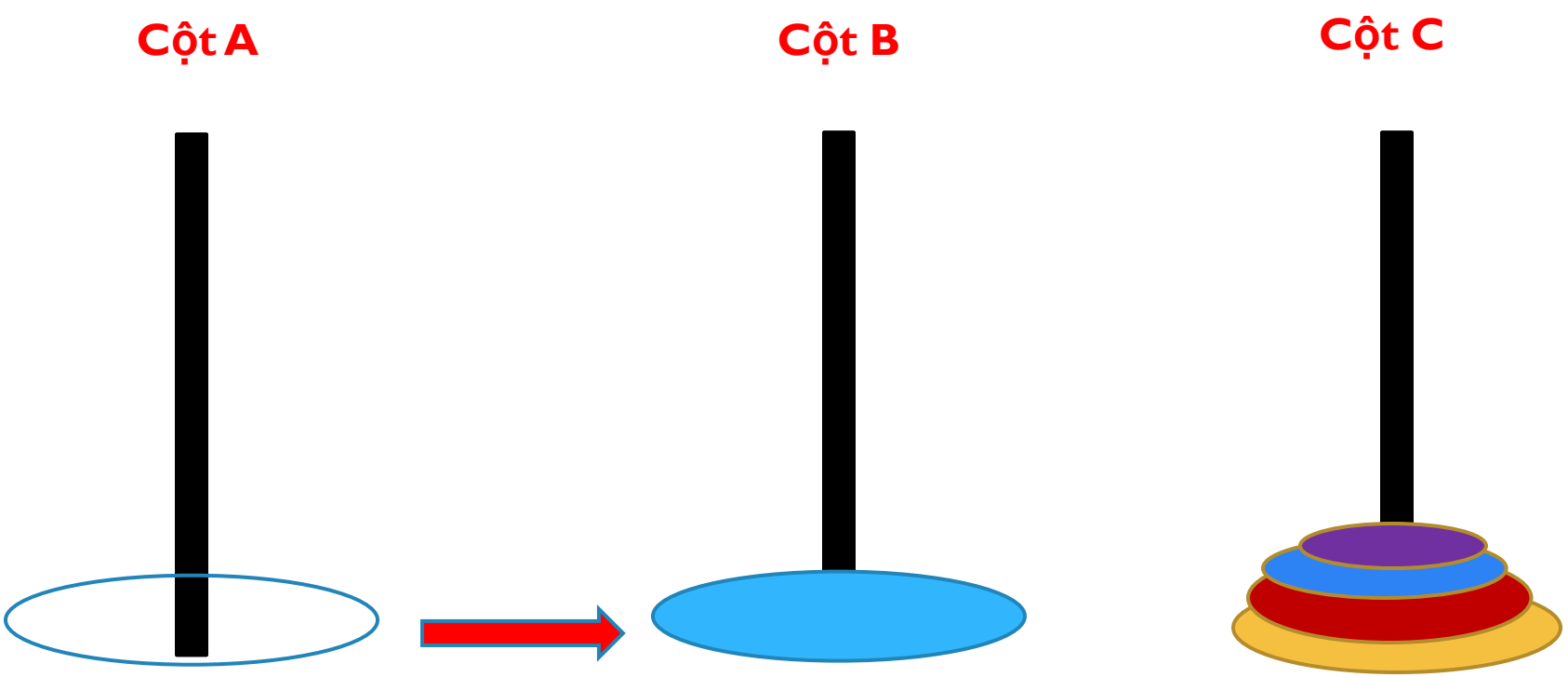
Ví dụ: hàm đệ qui tính Fibonacci thứ n.

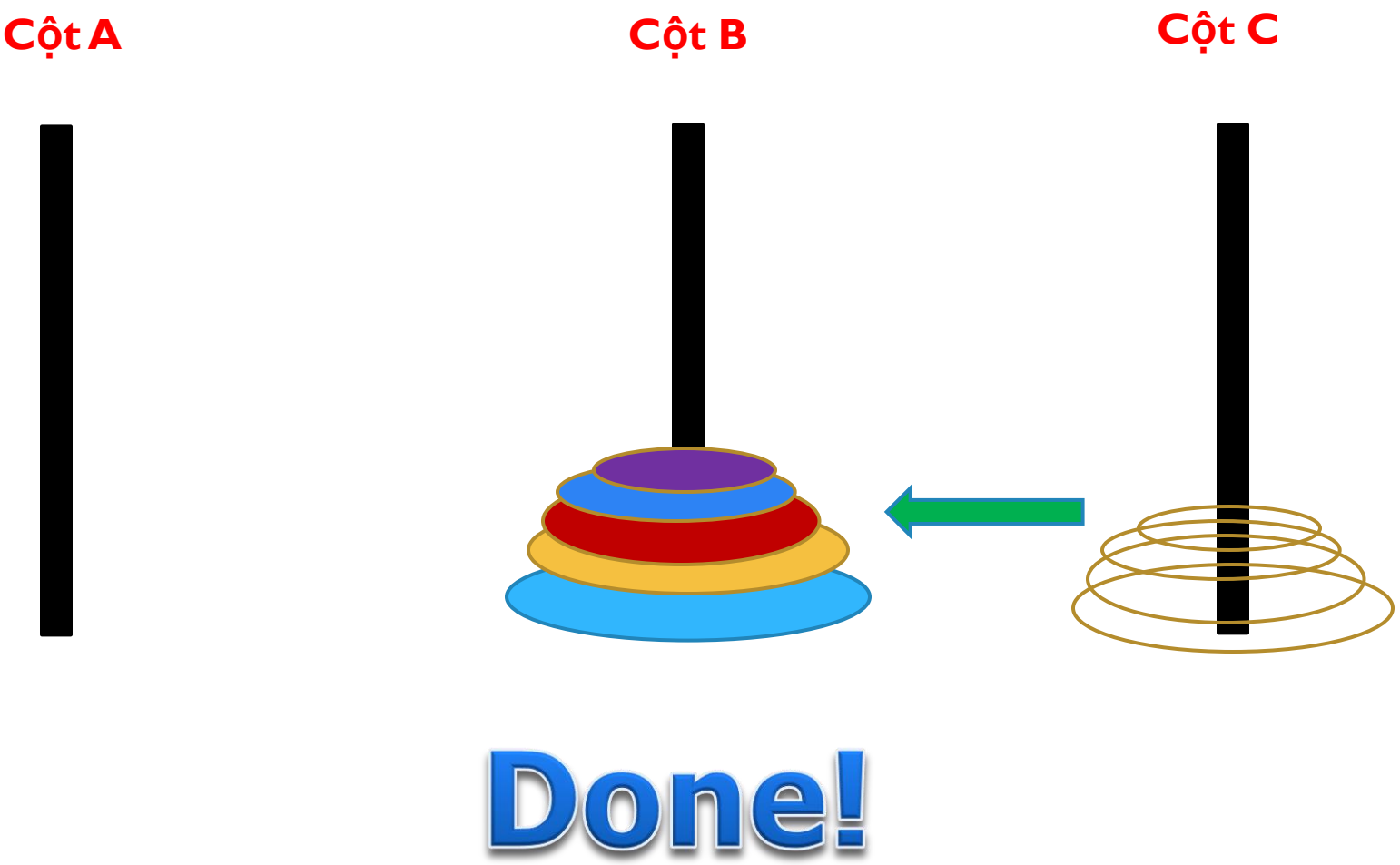
- **Đệ qui đuôi (tail recursion)**: là 1 dạng của đệ qui tuyến tính. Trong câu lệnh gọi đệ qui chỉ có gọi lại chính bản thân nó mà không có kết hợp với giá trị hay phép toán nào khác.
- **Đệ qui phi tuyến**: trong thân hàm đệ qui có vòng lặp gọi chính bản thân nó 1 số lần (nhiều lần). Thường xuất hiện trong hàm đệ qui tính theo công thức truy hồi.
- **Đệ qui hỗ tương**: các hàm đệ qui gọi lẫn nhau.

- **Bài toán Tháp Hà Nội:** Có 3 cột (giả sử A, B, C), trên cột A có 1 số lượng đĩa nhất định (có kích thước khác nhau) được đặt theo qui luật đĩa nhỏ đặt trên đĩa lớn hơn. Tiến hành dời toàn bộ số đĩa ở cột A sang cột khác (giả sử cột B) sao cho số bước là nhỏ nhất theo yêu cầu sau:
  - Di chuyển đĩa phải đặt vào 1 cột
  - Mỗi lần chỉ di chuyển 1 đĩa
  - Khi di chuyển phải đảm bảo đĩa nhỏ phải luôn nằm trên đĩa lớn
  - Có thể dùng cột C làm trung gian


Cột A
Cột B
Cột C


Cột A
Cột B
Cột C


Cột A
Cột B
Cột C


Cột A
Cột B
Cột C
Done!

- **Giải thuật đệ qui bài toán Tháp Hà Nội:**
  - Di chuyển (số đĩa – 1) từ cột A sang cột C (cột trung gian)
  - Di chuyển đĩa cuối cùng từ cột A sang cột B (cột đích)
  - Di chuyển (số đĩa – 1) từ cột C sang cột B dùng cột A làm trung gian.

```
void chuyendia(unsigned n, char a, char b, char c)
{
    if (n > 0) {
        chuyendia(n - 1, a, c, b);
        cout << "Chuyen dia " << n << " tu "
             << a << " -> " << b << endl;
        chuyendia(n - 1, c, b, a);
    }
}
```

- **Giải thuật đệ qui bài toán Tháp Hà Nội:**
  - Di chuyển (số đĩa – 1) từ cột A sang cột C (cột trung gian)
  - Di chuyển đĩa cuối cùng từ cột A sang cột B (cột đích)
  - Di chuyển (số đĩa – 1) từ cột C sang cột B dùng cột A làm trung gian.

```
int main()
{
    chuyendia(3, 'A', 'B', 'C');
}
```

	Đệ qui (recursion)	**Lặp (iteration)**
Dùng cấu trúc điều khiển	Cấu trúc lựa chọn	Cấu trúc lặp
Lặp lại	Lặp lại qua lời gọi hàm	Dùng cấu trúc lặp tường minh
Kiểm tra diều kiện dừng	Trường hợp cơ sở	Điều kiện lặp có giá trị sai
Liên tục tiếp cận điều kiện dừng	Phiên bản mới đơn giản	Cập nhật biến đếm
Lặp vô tận	Nếu bước đệ qui không giảm bớt vấn đề, không thể hội tụ về trường hợp cơ sở	Nếu điều kiện lặp luôn thỏa

Q & A