Khám phá Đệ quy (Recursion)

Hành trình chinh phục tri thức C++

Slide Learning C++

Ngày 16 tháng 1 năm 2026

Bức tranh toàn cảnh: Đệ quy là gì?

Khái niệm

Trong lập trình, **Đệ quy (Recursion)** đơn giản là việc một hàm tự gọi lại chính nó để giải quyết một vấn đề lớn bằng cách chia nhỏ nó thành các vấn đề tương tự nhưng có quy mô nhỏ hơn.

Phép ẩn dụ: Búp bê Nga (Matryoshka)

- Để lấy được con búp bê nhỏ nhất, bạn phải mở con búp bê lớn nhất.
- Bên trong là một con búp bê y hệt nhưng nhỏ hơn.
- Lặp lại cho đến khi chạm đến con búp bê cuối cùng không thể mở được nữa.

Cấu trúc của một hàm Đệ quy

Mọi hàm đệ quy cần 2 phần quan trọng để tránh lặp vô tận:

1. Điểm dừng (Base Case)

Đây là con búp bê nhỏ nhất. Khi chạm đến đây, hàm sẽ dừng lại và không gọi chính nó nữa.

2. Bước đệ quy (Recursive Step)

Hành động mở con búp bê hiện tại để tìm con búp bê nhỏ hơn. Đưa bài toán về phiên bản nhỏ hơn của chính nó.

Ví dụ minh họa: Tính Giai thừa (Factorial)

Giai thừa của n (ký hiệu $n!$) là tích các số từ 1 đến n. Ví dụ:
$4! = 4 \times 3 \times 2 \times 1 = 24$.
Công thức đệ quy: $n! = n \times (n-1)!$

```cpp
int tinhGiaiThua(int n) {
    // 1. Diem dung
    if (n <= 1) {
        return 1;
    }

    // 2. Buoc de quy
    return n * tinhGiaiThua(n - 1);
}

```

Listing 1: Tính giai thừa bằng C++

Cảnh báo: Nếu quên Điểm dừng?

Câu hỏi

Nếu một hàm đệ quy không có "Điểm dừng"(Base Case), điều gì sẽ xảy ra?

Cảnh báo: Nếu quên Điểm dừng?

Câu hỏi

Nếu một hàm đệ quy không có "Điểm dừng"(Base Case), điều gì sẽ xảy ra?

Hậu quả: Stack Overflow

- Hàm sẽ gọi chính nó mãi mãi (lặp vô tận).
- Bộ nhớ máy tính (Stack) bị đầy vì phải ghi nhớ các lần gọi hàm.
- Chương trình bị "treo"hoặc tắt đột ngột (Crash).

Cơ chế hoạt động bên trong: Ngăn xếp (Stack)

Tưởng tượng việc gọi hàm đệ quy giống như leo xuống cầu thang xoắn ốc.

1. **Giai đoạn Đi xuống (Winding):** Các hàm liên tục gọi nhau và "chồng"ghi chú vào ngăn xếp cho đến khi chạm **Điểm dừng**.
2. **Giai đoạn Quay ngược (Unwinding):** Khi có kết quả từ điểm dừng, máy tính gỡ từng ghi chú từ trên xuống dưới để tính kết quả cuối cùng.

Ứng dụng: Tìm kiếm Nhị phân (Binary Search)

Thay vì kiểm tra từng số (tuần tự), đệ quy giúp tìm kiếm nhanh hơn trên dãy đã sắp xếp:

- Nhìn vào số ở **giữa** dãy.
- Nếu số ở giữa $>$ số cần tìm $\rightarrow$ Đệ quy tìm ở nửa bên **trái**.
- Nếu số ở giữa $<$ số cần tìm $\rightarrow$ Đệ quy tìm ở nửa bên **phải**.

Điểm dừng của Binary Search là gì?

1. Khi tìm thấy số cần tìm ($mid == x$).
2. Khi khoảng tìm kiếm trống rỗng ($left > right$) $\rightarrow$ Không tìm thấy.

Code: Tìm kiếm Nhị phân Đệ quy

```cpp
int timKiemNhiPhan(int mang[], int trai, int phai, int x) {
    // Diem dung 1: Khong tim thay
    if (trai > phai) return -1;

    int giua = trai + (phai - trai) / 2;

    // Diem dung 2: Tim thay
    if (mang[giua] == x) return giua;

    // Buoc de quy: Tim ben trai
    if (mang[giua] > x)
        return timKiemNhiPhan(mang, trai, giua - 1, x);

    // Buoc de quy: Tim ben phai
    return timKiemNhiPhan(mang, giua + 1, phai, x);
}
```

Đệ quy trong nghệ thuật: Fractal

Đệ quy tạo ra các hình ảnh tự đồng dạng (phân thân) như bông tuyết, cành cây.

Quy tắc vẽ cây

1. Vẽ thân cây.
2. Tại ngọn, rẽ hai nhánh con.
3. **Đệ quy:** Lặp lại quy trình với nhánh con.

Điểm dừng: Khi chiều dài cành cây nhỏ hơn một mức nhất định (ví dụ < 10) để tránh vẽ vô tận.

Dãy số Fibonacci

Dãy số: 0, 1, 1, 2, 3, 5, 8, 13...
Quy luật: Số sau bằng tổng hai số trước đó.

```
1 int fibonacci(int n) {
2     // 1. Diem dung
3     if (n == 0) return 0;
4     if (n == 1) return 1;
5 
6     // 2. Buoc de quy: Goi lai chinh no 2 lan!
7     return fibonacci(n - 1) + fibonacci(n - 2);
8 }
9 
```

Listing 2: Fibonacci Đệ quy

Lưu ý hiệu năng

Hàm này gọi lại chính nó 2 lần mỗi bước, dẫn đến số lượng tính toán khổng lồ khi n lớn. Cần kỹ thuật "Ghi nhớ"(Memoization) để tối ưu.

Bài tập: Đảo ngược chuỗi

Sử dụng tính chất của Stack: "Vào trước, Ra sau".

```
1  void daoNguoc(string s) {
2      // 1. Diem dung
3      if (s.length() <= 1) {
4          cout << s;
5          return;
6      }
7
8      // 2. Buoc de quy: cat chuoi tu vi tri 1 den het
9      daoNguoc(s.substr(1));
10
11     // 3. In ky tu dau tien SAU KHI de quy xong
12     cout << s[0];
13 }
14
```

Nếu ‘cout‘ đặt **trước** lời gọi đệ quy, chuỗi sẽ in xuôi. Nếu đặt **sau**, máy tính phải "leo"xuống hết đáy rồi mới in khi quay ngược lên → Chuỗi đảo ngược.

Bài tập: Tính tổng các chữ số

Ví dụ: $123 \rightarrow 1 + 2 + 3 = 6$. Công cụ: Phép chia lấy dư (‘

```
int tinhTongChuSo(int n) {
    // 1. Diem dung: Neu n chi con 1 chu so
    if (n < 10) {
        return n;
    }

    // 2. Buoc de quy
    return (n % 10) + tinhTongChuSo(n / 10);
}

```

Luồng chạy với n = 123:

- Bước 1: $3 + \text{tong}(12)$
- Bước 2: $3 + (2 + \text{tong}(1))$
- Bước 3 (Base Case): $3 + (2 + 1) = 6$

So sánh: Đệ quy vs. Vòng lặp

Đặc điểm	**Đệ quy (Recursion)**	**Vòng lặp (Iteration)**
Cách viết	Ngắn gọn, giống toán học	Dài hơn, rõ ràng từng bước
Bộ nhớ	Tốn hơn (do Stack)	Tiết kiệm hơn
Tốc độ	Có thể chậm nếu không tối ưu	Thường nhanh hơn
Ứng dụng	Cấu trúc cây, Fractal, Sort	Xử lý danh sách đơn giản

Kiểm tra cuối khóa

Câu hỏi tư duy

Trong hàm tính giai thừa $n!$, nếu ta đổi Điểm dừng thành: `if (n == 0) return 0;` (thay vì `return 1`)
Thì kết quả của phép tính giai thừa (ví dụ 4!) sẽ là bao nhiêu?

Kiểm tra cuối khóa

Câu hỏi tư duy

Trong hàm tính giai thừa $n!$, nếu ta đổi Điểm dừng thành: `if (n == 0) return 0;` (thay vì `return 1`)
Thì kết quả của phép tính giai thừa (ví dụ 4!) sẽ là bao nhiêu?

Đáp án: 0

Vì đệ quy là chuỗi phép nhân liên tiếp. Nếu điểm dừng trả về 0, nó sẽ nhân với tất cả các kết quả trước đó. $4 \times 3 \times 2 \times 1 \times 0 = 0$.
Bài học: Điểm dừng đóng góp giá trị khởi đầu cho quá trình quay ngược.

Tổng kết hành trình

- **Bản chất:** Chia nhỏ vấn đề (Búp bê Nga).
- **Cốt lõi:** Phải luôn có **Điểm dừng** và **Bước đệ quy**.
- **Cơ chế:** Sử dụng Ngăn xếp (Stack) để ghi nhớ và quay ngược.

Chúc bạn áp dụng thành công tư duy Đệ quy vào các bài toán phức tạp!