TÓM LƯỢC BÀI GIẢNG
NHẬP MÔN LẬP TRÌNH

(Vũ Quốc Hoàng, vqhoang@fit.hcmus.edu.vn, FIT-HCMUS, 2021)

BÀI 10A
CƠ BẢN THUẬT TOÁN VÀ ĐỆ QUI

Chủ đề

- Thuật toán
- Đệ qui trên số nguyên

Tài liệu

[1] Tony Gaddis, *Starting out with C++ From Control Structures through Objects*, Pearson, 8th edition, 2015.

[2] Vũ Quốc Hoàng, *Bí kíp luyện Lập trình C (Quyển 1)*, hBook, 2017.

Đọc tài liệu

- Đọc kĩ: Chương 19 [1], Bài 4.6 [2]

Kiến thức

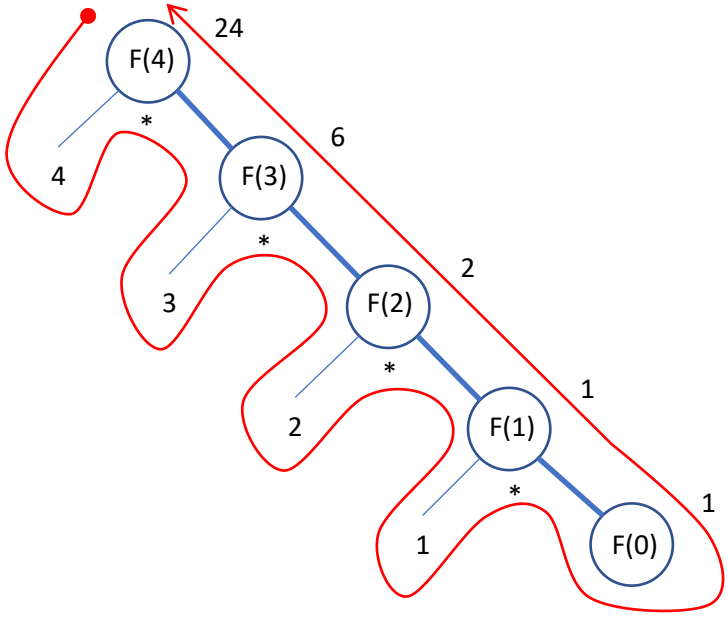
- **Bài toán** (problem) là một vấn đề **được định nghĩa tốt** (well-defined) cần giải quyết. Một **bài toán tính toán** (computational problem) được xác định bởi cặp **dữ liệu vào** (input) và **kết quả ra** (output). Dữ liệu vào còn được gọi là **tham số** (parameter) của bài toán và kết quả ra là **lời giải** (solution) của bài toán. Một giá trị cụ thể của tham số sẽ xác định một **trường hợp/thể hiện** (instance) cụ thể của bài toán.
- Ví dụ, bài toán "tính giai thừa" có tham số là một số nguyên không âm n và lời giải là số nguyên $n!$ thỏa $n! = 1 \times 2 \times \ldots \times n$. Một trường hợp cụ thể của bài toán này là "tính 4 giai thừa" có giá trị của tham số là $n = 4$ và lời giải là $24 = 4! = 1 \times 2 \times 3 \times 4$. Ta kí hiệu bài toán này là FACT và trường hợp cụ thể khi cho tham số n là $\mathrm{FACT}(n)$, hơn nữa, ta cũng kí hiệu lời giải là $\mathrm{FACT}(n)$.
- **Thuật toán** (algorithm) là một **thủ tục tính toán được đặc tả tốt** (well-specified computational procedure) để giải quyết một bài toán tính toán nào đó. Thuật toán, như vậy, gồm một dãy các bước tính toán biến đổi input của bài toán thành output (tức là "đi tìm" lời giải của bài toán từ tham số). Việc xây dựng thuật toán (tức là tìm thuật toán) cho một bài toán được gọi là **thiết kế thuật toán** (algorithm design).
- Một kĩ thuật giải quyết vấn đề (do đó là một kĩ thuật thiết kế thuật toán) rất hay dùng là **"thu-để-trị"** (reduce-and-conquer): để giải bài toán *A*, ta biến đổi (đưa/thu) nó về bài toán *B* (đã biết cách giải), sau đó giải *B* (bằng thuật toán đã biết) và dùng lời giải của *B* để xây dựng lời giải của *A*. Một thuật ngữ hay dùng hơn và tổng quát hơn cho "thu-để-trị" là **"chia-để-trị"** (divide-and-conquer), trong đó, ta đưa (hay chia) bài toán cần giải quyết về các bài toán đơn giản hơn.
- Một dạng rất hay dùng của kĩ thuật "thu-để-trị" là kĩ thuật **đệ qui** (recursion), trong đó ta đưa bài toán cần giải về bài toán có cùng dạng nhưng "đơn giản hơn", tức là đưa về trường hợp cùng bài toán nhưng tham số "nhỏ hơn". Để kĩ thuật đệ qui không quẩn ta cũng cần giải được các trường

hợp "đơn giản nhất" mà không dùng đệ qui, còn gọi là các **trường hợp cơ sở** (hay **trường hợp dừng**) của bài toán.

- Để có thể giải bài toán bằng đệ qui, tức là thiết kế **thuật toán đệ qui** (recursive algorithm), trước hết, tham số của bài toán (hoặc thành phần của tham số) phải **đệ qui được**. Tham số đệ qui được điển hình là các tham số nguyên, trong đó, "đơn giản hơn" là nhỏ hơn (<) và "đơn giản nhất" là các giá trị như 0, 1. Sau đó, ta cần trả lời được câu hỏi: làm thế nào dùng lời giải của các trường hợp đơn giản hơn (nhỏ hơn) để xây dựng lời giải cho trường hợp phức tạp hơn (lớn hơn) của bài toán.
- Ví dụ, từ nhận xét "giai thừa của $(n-1)$ có thể được dùng để tính giai thừa của n" mà bài toán FACT có thể được tính đệ qui như sau:
  - *Trường hợp cơ sở* (base case): nếu $n = 0$ thì $\text{FACT}(n) = 1$
  - *Trường hợp đệ qui* (recursive case): nếu $n > 0$ thì $\text{FACT}(n) = n \times \text{FACT}(n-1)$
- Thuật toán đệ qui trên có thể được **cài đặt** (implement) trong C/C++ bằng **hàm đệ qui** (recursive function) như sau:

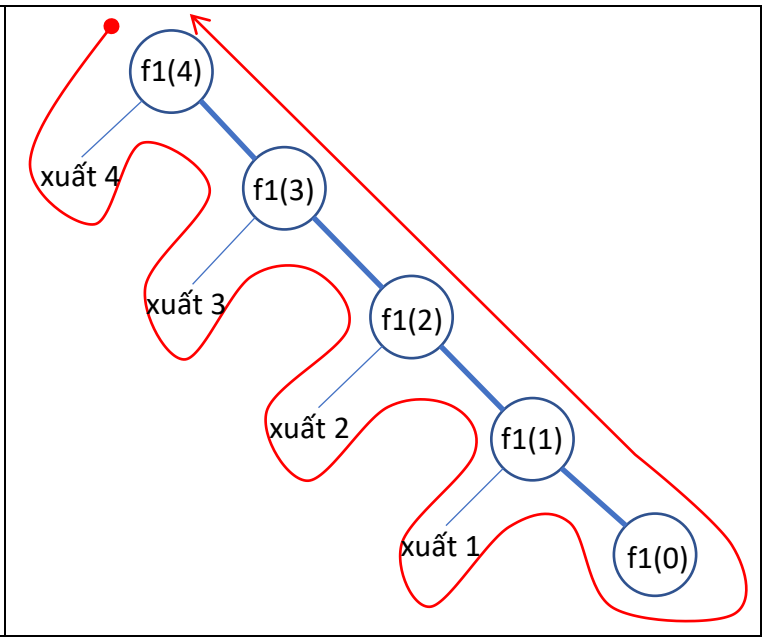
```
int fact(int n)
{
    if(n == 0)
        return 1;

    return n * fact(n - 1);
}
```

- Việc thực thi thuật toán đệ qui (tương ứng là thực thi hàm đệ qui) là quá trình lặp lại nhiều lần các bước thu, đệ qui, tổng hợp lời giải mà **cây đệ qui** (recursion tree) là công cụ quan trọng giúp trực quan hóa quá trình này. Hình sau minh họa quá trình tính toán đệ qui cho trường hợp bài toán cụ thể $\text{FACT}(4)$ (tương ứng là lời gọi hàm `fact(4)`)



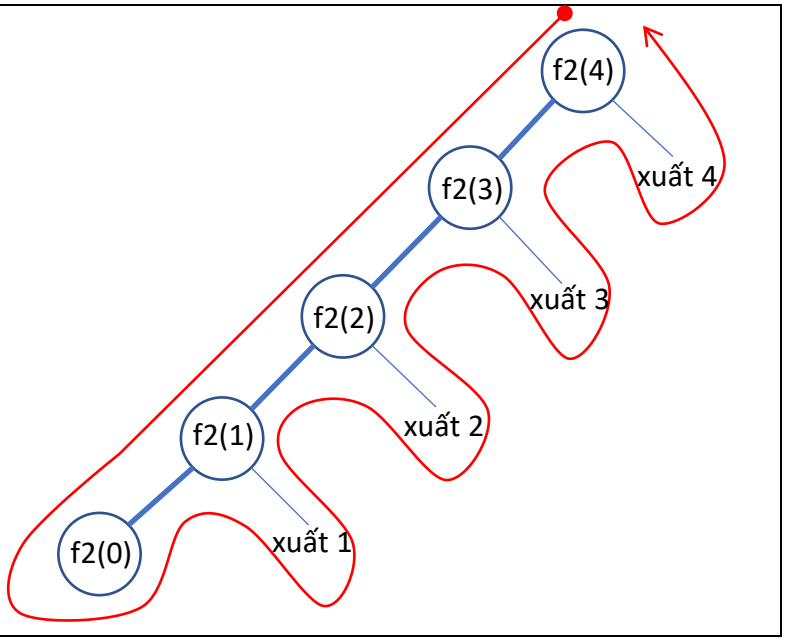
- Để hiểu rõ hoạt động của hàm đệ qui `f1` sau, ta xây dựng cây đệ qui cho việc thực thi lời gọi hàm `f1(4)` mà kết quả xuất ra là: 4 3 2 1.

```
void f1(int n)
{
    if(n > 0)
    {
        cout << n << endl;
        f1(n - 1);
    }
}

// Hàm không đệ qui tương ứng
void f1(int n)
{
    for(int i = n; i > 0; i--)
        cout << i << endl;
}
```



- Tương tự, để hiểu rõ hoạt động của hàm đệ qui `f2` sau, ta xây dựng cây đệ qui cho lời gọi hàm `f2(4)` mà kết quả xuất ra là: 1 2 3 4.

```
void f2(int n)
{
    if(n > 0)
    {
        f2(n - 1);
        cout << n << endl;
    }
}

// Hàm không đệ qui tương ứng
void f2(int n)
{
    for(int i = 1; i <= n; i++)
        cout << i << endl;
}
```



Kĩ năng

- Đọc hiểu và viết được các hàm đệ qui
- Biết cách dùng cây đệ qui (vẽ và duyệt cây) để hiểu rõ quá trình thực thi đệ qui
- Biết cách vận dụng chia-để-trị, thu-để-trị và đặc biệt là đệ qui để thiết kế thuật toán cho các bài toán đơn giản

Lưu ý

- Chia-để-trị, thu-để-trị và đặc biệt đệ qui là những phương pháp giải quyết vấn đề, phương pháp tư duy và lập trình rất quan trọng nên sinh viên cần rèn luyện nhiều để quen thuộc và thành thạo

Bài tập

1. Làm các bài tập trong Bài 4.6 [2].
2. Làm các bài tập trong Chương 19 [1].