Kỹ thuật cài đặt các thuật toán cơ bản

Nhập môn lập trình

Trình bày: …; Email: …@fit.hcmus.edu.vn

Nội dung

- Thuật giải rẽ nhánh và kỹ thuật cài đặt
- Tính toán lặp và kỹ thuật cài đặt
- Các vấn đề tìm hiểu mở rộng kiến thức nghề nghiệp
- Thuật ngữ và bài đọc thêm tiếng Anh

Thuật giải rẽ nhánh và kỹ thuật cài đặt

Bảng quyết định cho bài toán

- Nhằm mục đích mô tả rõ ràng các nhánh rẽ của một qui trình xử lý hay của các bước nào đó trong một thuật toán cần phải cài đặt.
- Có thể được trình bày dưới dạng văn bản (không câu hệ về hình thức, chỉ cần rõ ràng) và chỉ cần phân tích rõ các trường hợp logic xảy ra tránh để sót trường hợp.

Ví dụ bảng quyết định

- Tính căn bậc n của số thực x

Bảng quyết định	
x ≥ 0	Tính $\sqrt[n]{x}$ nhờ gọi pow(x, 1.0/n)
x < 0	n lẻ: $\sqrt[n]{x} = -\sqrt[n]{-x}$, tính $\sqrt[n]{x}$ nhờ gọi -pow(-x, 1.0/n)
	n chẵn: $\sqrt[n]{x}$ không tồn tại

Bảng quyết định (xảy ra trùng lắp)	
n lẻ	x ≥ 0: Tính $\sqrt[n]{x}$ nhờ gọi pow(x, 1.0/n)
	x < 0: $\sqrt[n]{x} = -\sqrt[n]{-x}$, tính $\sqrt[n]{x}$ nhờ gọi -pow(-x, 1.0/n)
n chẵn	x ≥ 0: Tính $\sqrt[n]{x}$ nhờ gọi pow(x, 1.0/n)
	x < 0: $\sqrt[n]{x}$ không tồn tại

Ý nghĩa của bảng quyết định

- Đa số các lỗi logic của chương trình (do xét không hết hay trùng lắp điều kiện) đều phát sinh từ việc lập trình viên suy nghĩ và viết trực tiếp mã nguồn mà không phân tích kỹ các trường hợp xảy ra.
- Việc chuẩn bị kỹ càng bảng quyết định sẽ giúp quá trình cài đặt chương trình được dễ dàng hơn.
- Bảng quyết định đóng vai trò quan trọng cho việc chuẩn bị những bộ dữ liệu kiểm thử (test case) để kiểm tra, chạy thử và bắt lỗi.

Ví dụ bảng kiểm thử

Bảng dữ liệu kiểm thử cho bài toán tính $\sqrt[n]{x}$				
Phạm vi n	**Phạm vi x**	**n**	**x**	**Giá trị/kết quả mong đợi**
n = 0	Bất kỳ	0.0	1.3	Không tồn tại
		0.0	0.0	
		0.0	-1.3	
n < 0	x = 0	-1.0	0.0	
		-20.7	0.0	
	x ≠ 0	-1.0	1.0	1.0
		-2.0	4.0	0.5
		-2.0	-1.7	Không tồn tại
		-3.0	-8.0	-0.5
n > 0	x ≥ 0	2.0	0.0	0.0
		102.0	1.0	1.0
		4.0	81.0	3.0
	x < 0	3.0	-125.0	-5.0 (khi n lẻ)
		4.0	-0.115	Không tồn tại (khi n chẵn)

Những ví dụ áp dụng

- Các bài toán về ngày tháng
  - Kiểm tra năm nhuận
  - Tính số ngày trong tháng
  - Tìm ngày hôm trước, hôm sau
- Tính tiền điện
- Tính tiền nước

(Xem chi tiết trong giáo trình NMLT trang 160-175)

Cài đặt đệ qui cho thuật giải rẽ nhánh

- Ý chính của kỹ thuật đệ qui là một hàm gọi lại chính nó nhằm để giải quyết một bài toán nhỏ hơn hay xử lý những trường hợp dễ hơn.
- Trong một số tình huống, xử lý rẽ nhánh chuyển sang một trường hợp mà lại qui về trường hợp đã giải quyết rồi. Lúc này người lập trình có thể gọi lại chính hàm đang viết với các đối số thích hợp để chuyển về nhánh rẽ đã giải quyết xong.

Ví dụ

- Tính căn bậc n của số thực x

Bảng quyết định			
n = 0	$\sqrt[n]{x}$ không tồn tại		
n < 0	x = 0	$\sqrt[n]{x}$ không tồn tại	
	x ≠ 0	Đưa về trường hợp m = -n > 0 nhờ $\sqrt[n]{x} = \frac{1}{\sqrt[-n]{x}}$	
n > 0	x ≥ 0	Tính $\sqrt[n]{x}$ nhờ gọi pow(x, 1.0/n)	
	x < 0	n lẻ	Dựa vào $\sqrt[n]{x} = -\sqrt[n]{-x}$ ta đưa về trường hợp x ≥ 0
		n chẵn	$\sqrt[n]{x}$ không tồn tại

Ví dụ

- Chương trình tính căn bậc n của x

```
#include <math.h>
double sqrt_N(double x, int n, bool& errorFlag) {
	double Result = 0;
	errorFlag = false;
	if (n == 0) {
		errorFlag = true;
	}
	else if (n < 0) {
		if (x == 0)
			errorFlag = true;
		else
			Result = 1/sqrt_N(x, 1.0/n);	// Gọi đệ qui
	}
```

Ví dụ

```
    else {
        if (x >= 0)
            Result = pow(x, 1.0/n);
        else {
            if (n % 2 == 1)
                Result = -sqrt_N(-x, n, errorFlag);   // Gọi đệ qui
            ele
                errorFlag = true;
        }
    }
    return Result;
}
```

Đồ án lập trình

- Tính thuế sử dụng đất phi nông nghiệp
- Tính thuế thu nhập cá nhân

(Xem chi tiết trong giáo trình NMLT trang 182-186)

Tính toán lặp và kỹ thuật cài đặt

Ví dụ về tính toán lặp

- Tính tổng và tích các số từ 1 đến n

	Cách viết 1	Cách viết 2
1	void SumAndProduct(long n,	void SumAndProduct(long n,
2	long& S, long& P)	long& S, long& P)
3	{	{
4	S = 0; P = 1;	S = 0; P = 1;
5	for (int i = 1; i <= n; i++) {	for (; n >= 1; n--) {
6	S += i;	S += n;
7	P *= i;	P *= n;
8	}	}
9	}	}

Áp dụng của tính toán lặp

- Xem trong giáo trình NMLT trang 223-245
  - Các thuật toán tính tổng số hay tích số
  - Các thuật toán đếm
  - Tìm phần tử nhỏ nhất hay lớn nhất

Áp dụng của tính toán lặp

- Xem trong giáo trình NMLT trang 252- 295
  - Số nguyên tố
  - Ước chung lớn nhất
  - Tính lũy thừa nhanh, tính lũy thừa modulo
  - Phân tích ra thừa số nguyên tố
  - Tính toán số lớn
  - Tính toán dãy truy hồi
  - Tính xấp xỉ
  - Xử lý lặp trên mảng 1 chiều và 2 chiều
  - Sắp xếp theo thứ tự

Kỹ thuật sử dụng cờ hiệu

- Biến cờ hiệu thường dùng trong xử lý lặp để đánh dấu hay ghi nhận trạng thái của một quá trình tính toán nào đó.
  - Khi biến cờ hiệu thay đổi giá trị thì việc tính toán có thể rẽ nhánh một cách phù hợp.
  - Đối với một số trường hợp thì việc thay đổi giá trị cờ hiệu cũng kết thúc vòng lặp. Điều này xảy ra khi quá trình xử lý đã đáp ứng được yêu cầu của bài toán đặt ra và không còn cần phải tiếp tục lặp nữa.

Kỹ thuật sử dụng cờ hiệu

- Ví dụ:
  - Khi tìm phần tử nhỏ nhất (gọi là min) thỏa điều kiện K thì biến cờ hiệu sẽ chuyển trạng thái khi gặp phần tử đầu tiên thỏa K, lúc này biến min mới có tác dụng lưu phần tử nhỏ nhất thỏa điều kiện K.
  - Khi kiểm tra số có phải số nguyên tố hay không thì quá trình kiểm tra dừng lập tức khi gặp một ước số lớn hơn 1 và nhỏ hơn số đó.

Kỹ thuật sử dụng cờ hiệu

- Thông thường để cài đặt biến cờ hiệu, chúng ta có thể dùng một biến luận lý (kiểu bool trong C++ hay kiểu int trong C) và khởi động biến với giá trị true (hay 1).
- Khi tính toán lặp đến một giai đoạn cần thiết thì biến cờ hiệu được sử thành false (hay 0).
- Nếu cần thiết thì xử lý lặp sẽ dừng (có thể dùng break trong C/C++) ngay khi biến cờ hiệu thay đổi giá trị.

Khái niệm bất biến của vòng lặp

- Bất biến của vòng lặp là một biểu thức, công thức hay một điều kiện có thể qui về một mệnh đề logic (là một vị từ theo các biến của vòng lặp) mà luôn có chân trị đúng lúc bắt đầu của mỗi lần lặp và sau khi vòng lặp kết thúc.
- Đây là khái niệm nâng cao và quan trọng trong lý thuyết lập trình.

Ví dụ về bất biến của vòng lặp

- Mã nguồn sau tính: $sum = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}$

```
double y = 1;
for (int i = 0; i < n; i++)
        y *= x;
```

- Bất biến của vòng lặp trên là:

$$1 \le i \le n \Rightarrow sum = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{i}$$

$$\Leftrightarrow \neg(1 \le i \le n) \vee sum = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{i}$$

$$\Leftrightarrow (i < 1) \vee (i > n) \vee \left(sum = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{i} \right)$$

Ví dụ về bất biến của vòng lặp

- Mã nguồn sau tính: $y = x^n$

```
double y = 1;
for (int i = 0; i < n; i++)
        y *= x;
```

- Bất biến của vòng lặp trên là:

$$(i < 0) \vee (i \geq n) \vee \left(y = x^{i+1}\right)$$

Khái niệm bất biến của vòng lặp

- Các bất biến của vòng lặp thường được sử dụng trong việc chứng minh tính đúng đắn của chương trình hay khảo sát chặt chẽ ngữ nghĩa của các chương trình về mặt lý thuyết.
- Đây là vấn đề không đơn giản, phải kết hợp giữa kiến thức lập trình và logic toán học nên thường không được áp dụng rộng rãi trong thực tế mà chỉ dùng giới hạn trong nghiên cứu.

Cài đặt đệ qui cho tính toán lặp

- Đa số thuật toán lặp đều có thể qui về cách cài đặt đệ qui, tức là viết các hàm mà gọi lại chính hàm đó.
- Ví dụ tính $n! = 1 \times 2 \times \cdots \times n$

$\Rightarrow$ Dựa vào tính chất $n! = (n-1)! \times n$

```
long Factorization(int n) {
    if (n <= 0)
        return 1;
    return Factorization(n – 1) * n;          // Gọi đệ qui
}
```

Cài đặt đệ qui cho tính toán lặp

- Ví dụ tính $E_n = \frac{1}{1^k} + \frac{1}{2^k} + \cdots + \frac{1}{n^k}$

$\Rightarrow$ Dựa vào tính chất $E_n = E_{n-1} + \frac{1}{n^k}$ và $E_0 = 0$.

```
#include <math.h>
double EulerFunc(int n, int k) {
      if (n <= 0 || k < 1)
            return 0;
      return EulerFunc(n – 1, k) + 1/pow(n, k);          // Gọi đệ qui
}
```

Cài đặt đệ qui cho tính toán lặp

- Ví dụ tính tổng n phần tử của mảng a

$\Rightarrow$ Dựa vào tính chất tổng n phần tử bằng tổng n – 1 phần tử cộng phần tử cuối cùng.

```
double Sum(double a[], int n) {
    if (n <= 0)
        return 0;
    return Sum(a, n – 1) + a[n – 1];          // Gọi đệ qui
}
```

Cài đặt đệ qui cho tính toán lặp

- Ví dụ tìm phần tử lớn nhất của mảng a

$\Rightarrow$ Dựa vào tính chất tổng n phần tử bằng tổng n – 1 phần tử cộng phần tử cuối cùng.

```
double Max(double a[], int n) {
        double Result;
        if (n <= 1)
                return a[0];
        Result = Max(a, n – 1);
        if (Result < a[n - 1])
                Result = a[n – 1];
        return Result;              // Gọi đệ qui
}
```

Cài đặt đệ qui cho tính toán lặp

- Ví dụ tính nhanh x^n với n là số nguyên

$\Rightarrow$ Dựa vào tính chất $x^n = \left(x^{\frac{n}{2}}\right)^2$ nếu n chẵn và $x^n = \left(x^{\frac{n-1}{2}}\right)^2 . x$ nếu n lẻ

```
double Power(double x, int n) {
    double xLast;
    if (n < 0) return 1/Power(x, -n);
    if (n == 0) return 1;
    if (n == 1) return x;
    xLast = Power(x, n/2);
    if (n % 2 == 0) return xLast * xLast;
    else return xLast * xLast * x;
}
```

Đồ án lập trình

- Tính các dãy truy hồi để tính sinx, cos, e^x, …

Các vấn đề tìm hiểu mở rộng kiến thức nghề nghiệp

Tìm hiểu thêm

- Các phương pháp tính và ứng dụng trong khoa học kỹ thuật.
- Các thuật toán lặp trong số học (giới thiệu về số học thuật toán)
- Một số bài toán xử lý lặp chưa có lời giải về điều kiện dừng
- Những nỗ lực để giảm độ phức tạp tính toán

Thuật ngữ và bài đọc thêm tiếng Anh

Thuật ngữ tiếng Anh

- ***code duplication***: sự trùng lặp mã nguồn
- ***code reuse***: tái sử dụng (dùng lại) mã nguồn
- ***coditional expression***: biểu thức điều kiện
- ***composite number***: hợp số
- ***coprime***: hai số nguyên tố cùng nhau
- ***counting***: đếm
- ***decision table***: bảng quyết định
- ***expected result***: kết quả mong đợi
- ***greatest common devisor (GCD)***: ước số chung lớn nhất
- ***increase, decrease***: tăng giảm
- ***leap year***: năm nhuận
- ***loop invariant***: bất biến của vòng lặp
- ***maximum, minimum***: lớn nhất, nhỏ nhất

Thuật ngữ tiếng Anh

- ***prime***: số nguyên tố
- ***recursive, recursive implementation***: tính đệ qui, cài đặt đệ qui
- ***search algorithm***: thuật toán tìm kiếm
- ***sort algorithm***: thuật toán sắp xếp
- ***test case***: bộ dữ liệu kiểm thử

Bài đọc thêm tiếng Anh

- **Thinking in C**, Bruce Eckel, E-book, 2006.
- **Theory and Problems of Fundamentals of Computing with C++**, John R.Hubbard, Schaum's Outlines Series, McGraw-Hill, 1998.