PHƯƠNG PHÁP CHIA ĐỂ TRỊ
(Divide and Conquer)

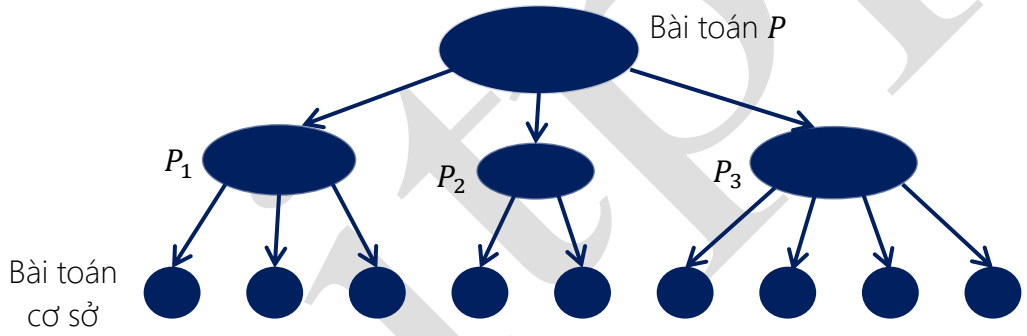
1. Khái niệm

Chia để trị là phương pháp phân rã bài toán ban đầu thành các bài toán con "đồng dạng" và dễ giải quyết hơn. Các bài toán con tiếp tục được phân rã cho đến khi đủ nhỏ để có thể tìm được nghiệm trực tiếp và nghiệm của các bài toán con sẽ được tổng hợp để tìm nghiệm của bài toán ban đầu.

Bài toán con "đồng dạng" là bài toán ban đầu nhưng với bộ tham số hoặc ít hơn hoặc tồn tại 1 tham số có giá trị nhỏ hơn (bài toán ban đầu nhưng với kích thước nhỏ hơn).

Chia để trị dùng kỹ thuật đệ quy quay lui để phân rã thành các bài toán con và tổng hợp nghiệm của chúng.

Mô hình tổng quát của phương pháp chia để trị



2. Các bước tiếp cận xây dựng thuật toán chia để trị

Thuật toán chia để trị được thiết kế dựa trên 3 bước cơ bản sau:

- Chia: phân rã bài toán ban đầu thành các bài toán con đồng dạng
- Trị: giải các bài toán con bằng cách tiếp tục phân rã chúng một cách đệ quy
- Gộp: kết hợp nghiệm của các bài toán con để tìm nghiệm của bài toán ban đầu

Điều kiện cần để áp dụng phương pháp chia để trị:

- Phân rã được bài toán ban đầu thành các bài toán con đồng dạng
- Định nghĩa được mô hình liên hệ giữa bài toán ban đầu và các bài toán con
- Xác định được nghiệm của bài toán con nhỏ nhất (trường hợp cơ sở)

Thuật toán chia để trị tổng quát

```
DAC(P)
Begin
    If (P đủ nhỏ) then
        exit(nghiệm của P)
    Else
    Begin
        Chia P thành các bài toán con con p1, p2, ..., pk
        Kết hợp (DAC(p1), DAC(p2), ..., DAC(pk))
    End;
End;
```

3. Đánh giá phương pháp chia để trị

Gọi $T(n)$ là thời gian giải bài toán kích thước n; a là số bài toán con; n/b là kích thước của từng bài toán con; $f(n)$ là thời gian phân rã, kết hợp các bài toán con. Ta có

$$T(n) = a.T\left(\frac{n}{b}\right) + f(n)$$

4. Một số bài toán minh họa

Bài toán 1: tìm kiếm nhị phân

Ý tưởng:

- Không gian tìm kiếm ban đầu từ l đến r
- So sánh x với $a[m]$ (m là vị trí giữa đoạn l, r)
    - Nếu $x = a[m]$ thì tìm kiếm thành công
    - Nếu $x < a[m]$ thì tìm kiếm từ l đến $m - 1$
    - Nếu $x > a[m]$ thì tìm kiếm từ $m + 1$ đến r

Cài đặt

```
int BS(int x, int l, int r)
{
    if (l > r) return -1;
    int m = (l + r)/2;
```

```
    if (x == a[m]) return m;
    if (x < a[m]) return BS(x, l, m-1);
    return BS(x, m+1, r);
}
```

Đánh giá

- Gọi $T(n)$ là thời gian tìm kiếm trên dãy gồm n phần tử, ta có

$$T(n) = T\left(\frac{n}{2}\right) + 1 = T(0) + logn$$

Bài toán 2: tính lũy thừa

Cho 2 số nguyên dương a, n. Tính a^n

Thuật toán 1: lặp với độ phức tạp $O(n)$

Thuật toán 2: chia để trị độ phức tạp $O(n)$

$$a^n = \begin{cases} 1, & n = 0 \\ a \times a^{n-1}, & n > 0 \end{cases}$$

Cài đặt

```
int pow(int a, int n)
{
    if (n == 0) return 1;
    return (a*pow(a, n-1));
}
```

Thuật toán 3: chia để trị độ phức tạp $O(logn)$

$$a^n = \begin{cases} 1, & n = 0 \\ \left(a^{\frac{n}{2}}\right)^2, & n\ chẵn \\ a \times \left(a^{\frac{n}{2}}\right)^2, & n\ lẻ \end{cases}$$

Nhận xét

- Trường hợp cơ sở $a^0 = 1$
- Nếu n chẵn thì $a^n = a^m * a^m, m = n$ div 2
- Nếu n lẻ thì $a^n = a * a^m * a^m, m = n$ div 2

Cài đặt

```
int pow(int a, int n)
{
    if (n == 0) return 1;
    int t = pow(a, n / 2);
    if (n % 2 == 0)
        return t*t;

    return a*t*t;
}
```

Đánh giá

- Gọi $T(n)$ là thời gian tính a^n của, ta có

$$T(n) = T(n \text{ div } 2) + 1 = T(0) + \log n$$

Bài toán 3: tháp Hà Nội với n đĩa

Xét bài toán Hà Nội với 2 đĩa, 3 đĩa và tổng quát cho trường hợp n đĩa.

Mô hình chia để trị chuyển n đĩa từ cột x sang cột z (thông qua cột trung gian y).

- Xem n đĩa như trường hợp 2 đĩa với đĩa trên cùng gồm $n-1$ đĩa.
- Chuyển $n-1$ đĩa trên cùng từ cột x sang cột y.
- Chuyển 1 đĩa từ cột x sang cột z.
- Chuyển $n-1$ đĩa từ cột y sang cột z.

Cài đặt

```
void move(int n, int x, int z)
{
    if (n == 1)
        cout<<x<<"->"<<z;
    else
    {
        move(n-1, x, 6-x-z);
        move(1, x, z);
        move(n-1, 6-x-z, z);
    }
}
```

Đánh giá

Gọi $T(n)$ là thời gian giải bài toán chuyển n đĩa. Ta có

$$T(n) = 2 * T(n-1) + 1 = 2^n T(0) + (1 + 2 + 2^2 + \cdots + 2^{n-1}) = 2^n - 1$$

Bài toán 4: tìm số Fibonacci thứ n

Dãy số Fibonacci được định nghĩa theo công thức

$$F_n = \begin{cases} 1, & n \leq 2 \\ F_{n-1} + F_{n-2}, & n > 2 \end{cases}$$

Cài đặt

```
int fibo(int n)
{
    if (n <= 2) return 1;
    return fibo(n-1) + fibo(n-2);
}
```

Nhận xét:

- Fibo(n) = Fibo(n-1) + Fibo(n-2) và Fibo(n-1) = Fibo(n-2) + Fibo(n-3). Trong lời gọi của Fibo(n-1) sẽ lại gọi Fibo(n-2) (đã được gọi trong Fibo(n)) do đó các bài toán con sẽ bị giải lại nhiều lần gây lãng phí thời gian và bộ nhớ.

Bài toán 5: tính tổ hợp chập k của n phần tử

Tổ hợp chập k của n phần tử được tính theo công thức sau:

$$C_n^k = \begin{cases} 1, n = k \text{ hoặc } k = 0 \\ C_{n-1}^k + C_{n-1}^{k-1} \end{cases}$$

Cài đặt

```
int C(int n, int k)
{
    if (k == 0 || n == k)
        return 1;
```

```
    return C(n-1,k) + C(n-1,k-1);
}
```

Nhận xét:

- Lời gọi các bài toán con sẽ bị gọi lại nhiều lần gây tốn thời gian và bộ nhớ hệ thống.

Kết luận

Chia để trị là một phương pháp tiếp cận để giải các bài toán có bản chất đệ quy. Kỹ thuật đệ quy giúp cho việc cài đặt thuật toán ngắn gọn và đơn giản. Tuy nhiên việc lạm dụng kỹ thuật đệ quy đôi khi làm cho chương trình trở nên chậm chạp và tốn kém bộ nhớ.

Chia để trị không phù hợp đối với những bài toán mặc dù có bản chất đệ quy nhưng bài toán ban đầu bị phân rã thành các bài toán toán con có "giao nhau" (ví dụ bài tính tổ hợp, tìm số Fibonacci).