

Chương 2

ĐỀ QUY VÀ GIẢI THUẬT ĐỀ QUY

1. KHÁI NIỆM VỀ ĐỀ QUY

Ta nói một bài toán là đề quy nếu nó bao gồm chính nó như một bộ phận hoặc nó được định nghĩa dựa vào các bài toán nhỏ hơn.

Ví dụ: Trong toán học ta gặp các định nghĩa đề quy sau:

+ Số tự nhiên:

- 1 là số tự nhiên.

- n là số tự nhiên nếu $n-1$ là số tự nhiên.

+ Hàm giai thừa $n!$:

- $0! = 1$

- Nếu $n > 0$ thì $n! = n(n-1)!$

2. GIẢI THUẬT ĐỀ QUY VÀ THUẬT CĐỀ QUY

2.1. Giải thuật đề quy

Nếu bài giải của câu hỏi bài toán T được giải bằng bài giải của một bài toán T1, có dạng giống như T, thì bài giải đó được gọi là bài giải đề quy. Giải thuật tương ứng với bài giải đề quy gọi là giải thuật đề quy.

Ở đây T1 có dạng giống T nhưng theo một nghĩa nào đó T1 phải “nhỏ” hơn T.

Chúng ta xem ví dụ bài toán tính $n!$ thì $n!$ là bài toán T còn $(n-1)!$ là bài toán T1 ta thấy T1 cùng dạng với T nhưng nhỏ hơn $(n-1 < n)$.

Xét bài toán tìm một từ trong quyển từ điển. Có thể nêu giải thuật như sau:

if (từ điển là một trang)

tìm từ trong trang này

else

{

Mở từ điển vào trang “giữa”

Xác định xem nó nằm ở đâu và từ điển chứa từ cần tìm;

if (từ đó nằm ở nửa trước)

tìm từ đó ở nửa trước

else tìm từ đó ở n a sau.

}

Giới thuật này được gọi là giới thuật đệ quy. Việc tìm từ trong quyển từ điển được thực hiện bởi quy trình bằng bài toán nhúng hèn đó là việc tìm từ trong một n a thích hợp của quyển từ điển.

Ta thấy có hai điểm chính cần lưu ý:

1. Sau mỗi lần từ điển được tách làm đôi thì một n a thích hợp sẽ lại được tìm bằng một chiến thuật như đã dùng trước đó (n a này lại được tách đôi).

2. Có một trường hợp đặc biệt, đó là sau nhiều lần tách đôi từ điển chỉ còn một trang. Khi đó việc tách đôi ngừng lại và bài toán trở thành đơn giản để ta có thể tìm từ mong muốn bằng cách tìm tuần tự. Trường hợp này gọi là **trường hợp suy biến**.

2.2. Thuật toán đệ quy

Với giới thuật tìm kiếm như trên ta viết một thủ tục tổng quát như sau:

Void SEARCH(dict, word);

{Tìm từ word trong từ điển dict}

{

if (Từ điển chỉ còn là một trang)

 tìm từ word trong trang này

else

{

 mở từ điển vào trang giữa

 xác định xem n a nào của từ điển chứa từ word

if (từ word nằm ở n a trước của từ điển)

return SEARCH(dict 1, word)

else return SEARCH(dict 2, word)

}

{

Thủ tục trên được gọi là thủ tục đệ quy. Nó có những điểm cần lưu ý sau:

1. Trong thủ tục đệ quy có lời gọi đến chính thủ tục đó. Ở đây trong thủ tục SEARCH có lời gọi call SEARCH (lời gọi này được gọi là lời gọi đệ quy).

2. Sau mỗi lần có lời gọi đệ quy thì kích thước của bài toán được thu nhỏ hơn trước. Ở đây khi có lời gọi call SEARCH thì kích thước dần còn bằng một nửa so với trước đó.

3. Có một trường hợp đặc biệt, trường hợp suy biến là khi lời gọi call SEARCH với từ điển dict chỉ còn là một trang. Khi trường hợp này xảy ra thì bài toán còn lại sẽ được giải quyết theo một cách khác hơn (tìm từ word trong trang đó bằng cách tìm kiếm tuần tự) và việc gọi đệ quy cũng kết thúc. Chính tình trạng kích thước của bài toán giảm dần sau mỗi lần gọi đệ quy sẽ đem báo đến từ trường hợp suy biến.

Một số ngôn ngữ cấp cao như: Pascal, C, Algol v.v... cho phép viết các thủ tục đệ quy. Nếu thủ tục đệ quy chứa lời gọi đến chính nó thì gọi là đệ quy trực tiếp. Cũng có trường hợp thủ tục chứa lời gọi đến thủ tục khác mà thủ tục này lại chứa lời gọi đến nó. Trường hợp này gọi là đệ quy gián tiếp.

3. THIẾT KẾ GIẢI THUẬT Đệ QUY

Khi bài toán đang xét hoặc dữ liệu đang xử lý được định nghĩa dưới dạng đệ quy thì việc thiết kế các giải thuật đệ quy trở nên thuận lợi. Hầu như nó phản ánh rất sát nội dung của định nghĩa đó.

Ta xét một số bài toán sau:

3.1. Hàm n!

Hàm này được định nghĩa như sau:

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

Giải thuật đệ quy được viết dưới dạng hàm dưới đây:

```
void Factorial(n)
{
    if (n=0) Factorial=1;
    else    Factorial = n*Factorial(n-1);
}
```

Trong hàm trên nếu gọi được nó nằm ở câu lệnh gắn sau else.

Mỗi lần gọi đệ quy đến Factorial, thì giá trị của n giảm đi 1. Ví dụ, Factorial(4) gọi đến Factorial(3), gọi đến Factorial(2), gọi đến Factorial(1), gọi đến Factorial(0) đây là trường hợp suy biến, nó được tính theo cách đặc biệt Factorial(0) = 1.

3.2. Bài toán dãy số FIBONACCI

Dãy số Fibonacci bắt nguồn từ bài toán cổ về việc sinh sản của các cặp thỏ. Bài toán được đặt ra như sau:

- *Các con thỏ không bao giờ chết.*

- Hai tháng sau khi ra đời một cặp thỏ mới sinh ra một cặp thỏ con.

- Khi đã sinh con rồi thì cứ sau mỗi tháng chúng lại sinh được một cặp con mới.

Giả sử bắt đầu từ một cặp thỏ mới ra đời thì đến tháng thứ n sẽ có bao nhiêu cặp?

Ví dụ với $n = 6$, ta thấy.

Tháng thứ 1: 1 cặp (cặp ban đầu)

Tháng thứ 2: 1 cặp (cặp ban đầu vẫn chưa đẻ)

Tháng thứ 3: 2 cặp (đã có thêm 1 cặp con)

Tháng thứ 4: 3 cặp (cặp đầu vẫn đẻ thêm)

Tháng thứ 5: 5 cặp (cặp con bắt đầu đẻ)

Tháng thứ 6: 8 cặp (cặp con vẫn đẻ tiếp)

Đặt $F(n)$ là số cặp thỏ ở tháng thứ n . Ta thấy chính xác cặp thỏ đã có ở tháng thứ $n-2$ mới sinh con ở tháng thứ n do đó số cặp thỏ ở tháng thứ n là:

$F(n) = F(n-2) + F(n-1)$ vì vậy $F(n)$ có thể được tính như sau:

$$F(n) = \begin{cases} 1 & \text{nếu } n \leq 2 \\ F(n-2) + F(n-1) & \text{nếu } n > 2 \end{cases}$$

Dãy số thứ n của $F(n)$ ứng với các giá trị của $n = 1, 2, 3, 4, \dots$, có dạng

1 1 2 3 5 8 13 21 34 55.... nó

được gọi là dãy số Fibonacci. Nó là mô hình của rất nhiều hiện tượng tự nhiên và cũng được sử dụng nhiều trong tin học.

Sau đây là cách thuật toán quy hoạch hàm để tính $F(n)$.

Int Fibonacci(n)

```
{  
    If (n<=2)    return 1;  
    else return F(n-2) + F(n-1);  
}
```

Ở đây trình hợp suy biến ở 2 giá trị $F(1) = 1$ và $F(2) = 1$.

Chú ý

Đối với hai bài toán nêu trên thì việc thiết kế các giải thuật đệ quy tổng quát khá thú vị vì có hai đầu thu được đồng tính giá trị hàm mà định nghĩa của nó xác định được một cách dễ dàng.

Nhưng không phải lúc nào tính đệ quy trong cách giải bài toán cũng thể hiện rõ nét và đơn giản như vậy. Mà việc thiết kế một giải thuật đệ quy đòi hỏi phải giải đáp được các câu hỏi sau:

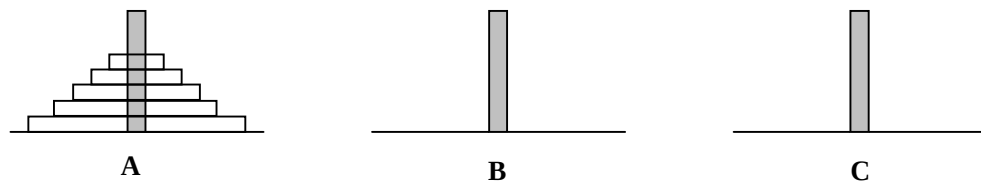
- Có thể định nghĩa được bài toán dưới dạng một bài toán cùng loại, nhưng nhỏ hơn như thế nào?
- Như thế nào là kích thước của bài toán để có giải thuật đệ quy?
- Trình hợp được biết nào của bài toán để gọi là trình hợp suy biến?

Sau đây ta xét thêm bài toán phân hoạch.

3.3. Bài toán “Tháp Hà Nội”

Bài toán này mang tính chất là một trò chơi, nội dung như sau:

Có n đĩa, kích thước nhỏ dần, mỗi đĩa có lỗ giữa. Có thể xếp chồng chúng lên nhau xuyên qua một cọc, đĩa to dưới, đĩa nhỏ trên để cuối cùng có một chồng đĩa dạng như hình tháp như hình 2.1.



Hình 2.1: Chồng đĩa trước khi chuyển

Yêu cầu đưa ra là:

Chuyển chồng đĩa từ cọc A sang cọc khác, chồng hùn cọc C, theo những điều kiện:

- Mỗi lần chỉ được chuyển một đĩa.
- Không khi nào có tình huống đĩa to ở trên đĩa nhỏ (dù là tạm thời).
- Được phép sử dụng một cọc trung gian, chồng hùn cọc B để đặt tạm đĩa (gọi là cọc trung gian).

Để đi tìm cách giải tổng quát, trước hết ta xét vài trường hợp đơn giản.

***) Trường hợp có 1 đĩa:** - Chuyển đĩa từ cọc A sang cọc C.

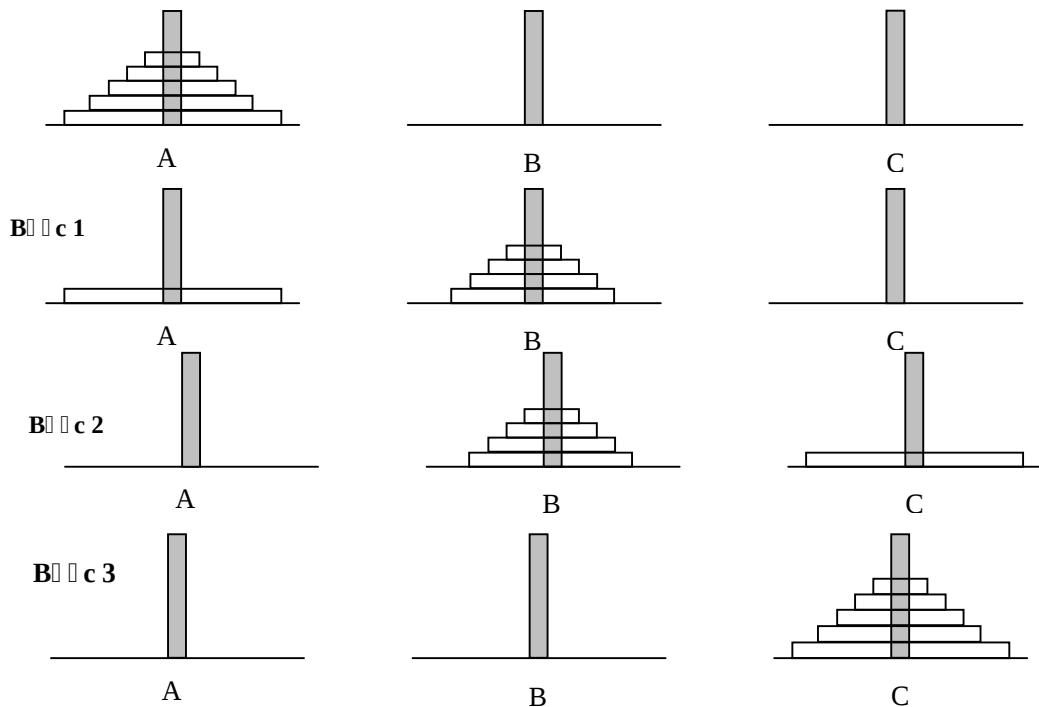
***) Trường hợp có 2 đĩa:**

- Chuyển đĩa nhỏ nhất từ cọc A sang cọc B.
- Chuyển đĩa hai từ cọc A sang cọc C.
- Chuyển đĩa nhỏ nhất từ cọc B sang cọc C.

Ta thấy với trường hợp n đĩa ($n > 2$) nếu coi $n-1$ đĩa ở trên, đóng vai trò như đĩa nhỏ nhất thì có thể xây lý giải như trường hợp 2 đĩa được, nghĩa là:

- Chuyển $n-1$ đĩa trên từ A sang B.
- Chuyển đĩa thứ n từ A sang C.
- Chuyển $n-1$ đĩa từ B sang C.

Lúc đó thể hiện 3 bước này như sau:



Hình 2.2: Các bước chuyển đĩa

Như vậy, bài toán “Tháp Hà Nội” tổng quát với n đĩa đã được đưa về bài toán tổng quát với kích thước nhỏ hơn, chính hơn thì chuyển n đĩa từ cọc A sang cọc C nay là chuyển $n-1$ đĩa từ cọc A sang cọc B và bước này thì giải thuật lại là:

- Chuyển $n-2$ đĩa từ cọc A sang cọc C.
- Chuyển 1 đĩa từ cọc A sang cọc B.
- Chuyển $n-2$ đĩa từ cọc B sang cọc C.

và cũng như cho tới khi trình bày suy biến xảy ra, đó là trình bày hình ảnh với bài toán chuyển 1 đĩa.

Vậy thì các điểm cần đưa quy trong giải thuật đã được xác định và ta có thể viết giải thuật đệ quy của bài toán “Tháp Hà Nội” như sau:

```
Void Chuyen(int n, char A, char B, char C)
{
    if (n==1) { printf("\nchuyển đĩa từ %c sang %c ",A,C); return } ;
    else
    {
        Chuyen(n-1, A, C, B);
        printf("\nchuyển đĩa từ %c sang %c ",A,C);
        Chuyen(n-1, B, A, C);
    }
}
```

4. HƯỚNG DẪN CÀI ĐẶT QUY

Qua các ví dụ trên ta có thể thấy: đệ quy là một công cụ để giải quyết các bài toán. Có những bài toán, bên cạnh giải thuật đệ quy vẫn có những giải thuật lặp khá đơn giản và hiệu quả. Chính hơn giải thuật lặp tính $n!$ có thể viết:

```
Int Factorial(n)
{
    if ((n==0) || (n==1))  gt= 1;
    else
    {
        gt=1;
    }
}
```

```

        for ( i=2;i<=n;i++)
            gt:= gt*i;
    }
    return gt;
}

```

Hãy ta xét giải thuật tính số Fibonacci thứ n:

```

int Fibonacci(n)
{
    If (n<=2) Fibonacci= 1;
    else {
        Fib1 = 1; Fib2 = 1;
        for (i:=3;i<= n;i++)
        {
            Fibn= Fib1 + Fib2;
            Fib1= Fib2;
            Fib2= Fibn;
        }
        Fibonacci = Fibn;
    }
}

```

Tuy vậy, để quy nạp có vai trò xứng đáng của nó. Có những bài toán vì cần nghĩ ra giải thuật để quy nạp nên liên hệ nhiều so với giải thuật lặp và có những giải thuật để quy nạp sẽ có hiệu suất cao, chẳng hạn giải thuật sắp xếp kiểu phân hoạch (Quick Sort) hoặc các giải thuật duyệt cây như phân mà ta sẽ có dịp xét trong các chương tiếp theo của môn học này.

Một điều nữa cần nói thêm là: về mặt định nghĩa, công cụ để quy đã cho phép xác định một tập vô hạn các đối tượng bằng một phát biểu hữu hạn. Ta sẽ thấy vai trò của công cụ này trong định nghĩa văn phạm, định nghĩa cú pháp ngôn ngữ, định nghĩa một số cấu trúc dữ liệu v.v...

Chú thích: khi thay các giải thuật để quy bằng các giải thuật lặp thông thường ta gọi là không để quy. Tuy nhiên có những bài toán vì cần không để quy thông thường để

giải (ví dụ: giải thuật tính $n!$, tính số fibonacci ...), nhưng có những bài toán việc khó để quy là một phép tính (ví dụ: bài toán tháp Hà Nội, giải thuật sắp xếp phân hoạch...).