CHƯƠNG I: GIẢI THUẬT

1.1. Thiết kế và phân tích giải thuật

1.2. Giải thuật đệ qui

1.2.1 Những khái niệm cơ bản

1.2.1.1. Khái niệm về đệ qui

Đệ qui là khái niệm cơ bản trong toán học và khoa học máy tính. Một đối tượng được gọi là đệ qui nếu nó được định nghĩa qua chính nó hoặc một đối tượng khác cùng dạng với chính nó.

Ví dụ: Khi ta đặt hai chiếc gương đối diện nhau, lúc đó chiếc gương thứ nhất chứa ảnh của chiếc gương thứ hai. Chiếc gương thứ hai lại chứa hình ảnh của chiếc gương thứ nhất nên tất nhiên nó lại chứa ảnh của chính nó trong chiếc gương thứ nhất...Các hình ảnh này giống nhau nhưng kích thước nhỏ hơn người ta gọi nó là hình ảnh đệ quy.

Trong toán học ta cũng thường xuyên gặp các định nghĩa đệ qui như định nghĩa giai thừa của một số tự nhiên N:

Nếu N=0 thì N!=1

Nếu N>0 thì N!=N(N-1)!

Chương trình đệ qui là chương trình gọi tới chính nó. Một chương trình đệ qui thì không thể gọi đến chính nó mãi mãi mà phải có điều kiện kết thúc khi chương trình không gọi đến chính nó nữa.

1.2.1.2. Giải thuật đệ qui và thủ tục đệ qui

Trước khi tìm hiểu giải thuật đệ qui ta xem xét khái niệm lời giải đệ qui. Nếu lời giải của bài toán T được thực hiện bằng lời giải của một bài toán T', có dạng giống như T thì đó là một lời giải đệ qui.

Giải thuật đệ qui: là giải thuật mà lời giải của nó được thực hiện bằng lời giải có dạng giống như lời giải ban đầu.

Tính chất của giải thuật đệ qui: Giải thuật đệ qui được gọi lại nhiều lần, mỗi lần gọi thì kích thước lại nhỏ đi và dẫn tới trường hợp đặc biệt. Giải thuật đệ qui bao giờ cũng có một trường hợp đặc biệt dừng đệ qui gọi là trường hợp suy biến.

Thủ tục đệ quy: Thủ tục để thể hiện giải thuật đệ qui gọi là thủ tục đệ qui. Thủ tục đệ qui là điều kiện cần và đủ cho giải thuật đệ qui. Thủ tục đệ qui có các tính chất sau:

- Trong thân của thủ tục có chứa lời gọi tới chính nó.
- Có một trường hợp đặc biệt (trường hợp suy biến) để kết thúc đệ qui.
- Mỗi lần gọi lại thủ tục thì kích thước lại thu nhỏ hơn trước.

1.2.2. Thiết kế giải thuật đệ qui

Định nghĩa hàm đệ qui hay thủ tục đệ qui gồm có hai phần:

- ***Phần neo*** (anchor): Phần này được dùng làm điều kiện kết thúc lời gọi đệ qui, nó được thực thi khi mà công việc quá đơn giản, có thể giải trực tiếp chứ không cần phải nhờ đến một bài toán con nào khác.

Ví dụ: Trong giải thuật tính n! thì phần neo là:

```
If n=0 Then Giaithua:=1;
```

- ***Phần đệ qui***: trong khi bài toán chưa thể giải được bằng phần neo ta xác định bài toán con và gọi đệ qui để giải các bài toán con đó. Phối hợp lời giải của các bài toán con ta sẽ có lời giải của bài toán gốc. Phần này còn được gọi là phần hạ bậc.

Ví dụ: Trong giải thuật tính n! thì phần đệ qui là:

```
If n>1 Then GiaiThua:=n*GiaiThua(n-1);
```

Phần đệ qui thể hiện tính "qui nạp" của lời giải còn phần neo thể hiện tính dừng của lời giải.

Dưới đây là một số ví dụ về giải thật đệ qui:

Ví dụ 1: Hàm tính giai thừa của một số tự nhiên N (N!)

Có thể định nghĩa N! một cách đệ qui như sau:

N!= 1 nếu N=0

N!=N(N-1)! nếu N>0

Qua định nghĩa đệ qui của N! ta có thể thấy ngay phần neo ở đây là

N!=1 nếu N=0

Còn phần đệ qui chính là N!= N(N-1)! nếu N>0

Giải thuật đệ qui được viết dưới dạng hàm như sau:

```
Function Giaithua(n:Integer):longint;
Begin
  If n= 0 Then giaithua:=1
  else    giaithua:=n*giaithua(n-1);
End;
```

Minh hoạ cho trường hợp 3!

Vì n=3 <>0 nên trước hết hàm tính 2! bởi 3! được tính bằng tích của 3*2!. Tương tự để tính 2!, nó lại đi tính 1! bởi 2! được tính bằng 2*1!, cứ tiếp tục ta phải tính 1!=1*0! mà 0! chính là trường hợp N=0 tức là phần neo. Khi đó 0!=1, từ giá trị của 0! ta tính được giá trị của 1!=1*1=1, cứ như thế ta tính được giá trị của 3!=3*2*1=6

3! = 3*2!

2! = 2*1!

1! = 1*0!

0! = 1

Chương trình trên đã thể hiện các chức năng cơ bản của một chương trình đệ qui: nó gọi tới chính nó (với một giá trị đối số nhỏ hơn) và có điều kiện kết thúc để tính trực tiếp kết quả của nó.

Ví dụ 2: Dãy số Fibonacci

Dãy số Fibonacci có nguồn gốc từ một bài toán cổ về việc sinh sản của các cặp thỏ. Bài toán được phát biểu như sau:

1. Giả sử các con thỏ không bao giờ chết.
2. Sau khi ra đời hai tháng, mỗi cặp thỏ mới sẽ sinh ra một cặp thỏ con gồm một con thỏ đực và một con thỏ cái.
3. Cặp thỏ nào đã sinh con rồi thì cứ mỗi tháng tiếp theo chúng lại sinh được một cặp con mới.

Chẳng hạn với n=5 sẽ có:

Tháng thứ 1: có 1 cặp (cặp ban đầu)

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

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

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

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

Yêu cầu đặt ra là tính số cặp thỏ ở một tháng thứ n bất kỳ ký hiệu là F(n)

Ta thấy ngay nếu mỗi cặp thỏ ở tháng thứ (n-1) đều sinh con thì số cặp thỏ ở tháng thứ n là:

$$F(n) = 2*F(n-1)$$

Nhưng theo giả thiết, trong các cặp thỏ ở tháng thứ (n-1) chỉ có những cặp đã có ở tháng thứ (n-2) mới sinh con ở tháng thứ n nên:

$$F(n) = F(n-2) + F(n-1)$$

Vậy ta có thể tính được F(n) theo công thức sau:

F(n) = 1 nếu n<=2

F(n) = F(n-2) + F(n-1) nếu n>2

Dãy số thể hiện F(n) ứng với các giá trị của n =1, 2, 3... có dạng:

1 1 2 3 5 8 13 21 34 ...

Dãy số này được gọi là dãy số Fibonacci.

Sau đây là giải thuật đệ qui được viết dưới dạng hàm:

```
Function Fibo(n:Integer): Integer;
Begin
  If n<=2 Then Fibo:=1
  else Fibo := Fibo(n-2) + Fibo(n-1);
End;
```

Ví dụ 3: Tính A^n với n là số nguyên dương

Trong Pascal để tính a^n có thể dùng hàm exp, tuy nhiên ta có thể tính một cách đệ qui như sau:

$a^n = 1$ nếu n=0

$a^n = a*a^{(n-1)}$ nếu n>0

Sau đây là giải thuật đệ qui được viết dưới dạng hàm:

```
Function Power(a:real; n: word) : real
Begin
        If n=0 Then Power:=1
    Else Power := a*Power(a,n-1);
End;
```

1.2.3. Hiệu lực của đệ qui

Qua các ví dụ ở trên dễ nhận thấy được ưu điểm của đệ qui. Có thể nói đệ qui là một công cụ mạnh để giải các bài toán. Với một số bài toán, bên cạnh giải thuật đệ qui vẫn còn có những giải thuật lặp khá đơn giản và hữu hiệu. Tuy nhiên có rất nhiều bài toán sử dụng giải thuật đệ qui sẽ có hiệu lực cao và thuận lợi hơn so với giải thuật lặp (chẳng hạn giải thuật cho bài toán tháp Hà Nội, giải thuật sắp xếp kiểu phân đoạn (Quick Sort) sẽ dược đề cập tới trong chương III).

Dưới đây là một số ưu nhược điểm của giải thuật đệ qui:

Ưu điểm:

- Sáng sủa, dễ hiểu, nêu bật được bản chất của vấn đề

- Dùng cái hữu hạn để định nghĩa đại lượng vô hạn
- Vẫn có một số thủ tục đệ qui chạy nhanh và hiệu quả hơn (ví dụ như giải thuật Quick Sort)
- Có thủ tục tìm ra đệ qui dễ hơn không đệ qui.
- Hiện nay có nhiều giải thuật chưa có cách giải không đệ qui.

Nhược điểm:

- Tốn bộ nhớ
- Thường chạy chậm hơn so với thủ tục không đệ qui

1.2.4. Đệ qui và qui nạp toán học

Qui nạp toán học chứng minh một tính chất nào đó ứng với số tự nhiên n bằng cách chứng minh tính chất đó đúng với một trường hợp cơ sở (thường ứng với trường hợp n=1) sau đó mở rộng ra chứng minh tính chất đó đúng với n bất kỳ với giả thiết tính chất đó đúng với các số tự nhiên k nhỏ hơn n.

Còn đối với cách giải đệ qui thì dựa trên việc định rõ lời giải cho trường hợp suy biến rồi thiết kế sao cho lời giải của bài toán được suy từ lời giải của bài toán nhỏ hơn cùng loại như thế.

Như vậy qui nạp toán học và đệ quy có cách chứng minh tương tự như nhau.

Khử đệ qui:

Các giải thuật đệ quy sử dụng không gian nhớ lớn dễ xảy ra lỗi tràn khi bài toán có độ sâu đệ quy lớn. Các giải thuật không đệ quy thường thực hiện nhanh hơn giải thuật đệ quy nên một số giải thuật đệ qui thuộc loại tính toán đơn giản có thể được thay thế bởi giải thuật khác không đệ qui. Công việc này được gọi là khử đệ qui.

Ví dụ hàm đệ qui tính n! ở mục trước có thể được khử đệ qui như sau:

```
Function Giaithua(n :Integer):longint;
Var          i:Integer;
             T: longint;
```

```
Begin
        T:=1;
        If n>0 Then
                For i:=1 to n Do T:=T*i;
        Giaithua:=T;
End;
```

Hoặc hàm đệ qui tính dãy số Fibonacci sẽ viết theo cách khác như sau:

```
    Function Fibo(n: Integer):Integer;
    Var
            i,x,y:Integer;
    Begin
            If n= 0 Then Fibo:=0
            else If n=1 Then Fibo:=1
                    else Begin
                            i:=1; y:=0; x:=1;
                            While i< n Do
                                    Begin
                                            inc(i);
                                            x:=x+y;
                                            y:=x-y;
                                    End;
                            End ;
            Fibo:=x;
    End;
```

1.2.5. Một số giải thuật đệ qui

Đệ qui là một phương pháp thiết kế giải thuật đơn giản và hiệu quả nếu biết áp dụng thích hợp. Mục này sẽ trình bày một số giải thuật đệ qui thường gặp. Chúng được chia thành hai nhóm: nhóm giải thuật kiểu "chia để trị" và nhóm giải thuật kiểu "cấu hình tổ hợp".

1.2.5.1. Chia để trị

"Chia để trị" là một phương pháp thiết kế giải thuật dựa trên tư tưởng: chia một bài toán thành các bài toán đồng dạng nhưng có kích thước nhỏ hơn (dễ giải hơn). Kết hợp lời giải những bài toán con đó sẽ cho ta lời giải của bài toán ban đầu.

Một số ví dụ sau sẽ cho bạn đọc thấy hiệu quả của phương pháp chia để trị.

a. Tính luỹ thừa nhanh

Để tính hàm luỹ thừa a^n ta có thể dùng phương pháp lặp và sẽ cần n phép nhân, tức là có độ phức tạp giải thuật là O(n).

Để giảm độ phức tạp của giải thuật có thể dùng phương pháp "chia để trị" để tính hàm luỹ thừa. Ta có công thức truy hồi sau:

$$a^n = \begin{cases} 1 : n = 0 \\ (a^k)^2 : n = 2k \\ a(a^k)^2 : n = 2k+1 \end{cases}$$

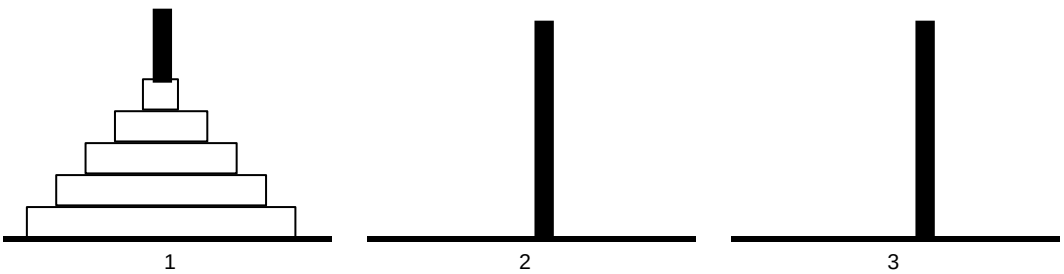
Với công thức trên, để tính a^n ta đi tính a^k với k là n/2. Tương tự, lại chia đôi k để tính a^k. Giải thuật này có độ phức tạp O(logn), việc chứng minh xem như là một bài tập.

Giải thuật được mô tả bằng ngôn ngữ tựa Pascal như sau:

```
function power(a,n);
Begin
  If n = 0 Then return 1;
  k := n div 2;
  If n = 2*k Then return sqr(power(a,k))
  else return a*sqr(power(a,k));
End;
```

b. Bài toán Tháp Hà Nội

Có N đĩa có đường kính khác nhau được đặt chồng lên nhau theo thứ tự giảm dần của đường kính tính từ dưới lên. Có ba cột có thể đặt các đĩa đánh số 1, 2, 3. Chồng đĩa ban đầu được đặt ở cột 1:



Cần chuyển cả chồng đĩa từ cột 1 sang cột 2, theo những quy tắc sau:

- Khi di chuyển một đĩa, phải đặt nó vào một trong ba cột đã cho.
- Mỗi lần chỉ có thể chuyển một đĩa và phải là đĩa ở trên cùng.
- Tại một cột nào đó đĩa mới được chuyển đến sẽ phải đặt lên trên cùng. Đĩa lớn hơn không được phép đặt chồng lên trên đĩa nhỏ.

Chúng ta giải bài toán bằng cách chia bài toán chuyển N đĩa, từ cột 1 sang cột 2 thành ba bài toán đơn giản hơn như sau:

+ Khi n=1 Chuyển đĩa đĩa từ cột 1 sang cột 2 (1->2)

+ Khi n>1 thì:

1. Chuyển N-1 đĩa trên cùng từ cột 1 sang cột 3, sử dụng cột 2 làm trung gian.
2. Chuyển đĩa thứ N từ cột 1 sang cột 2.
3. Chuyển N-1 đĩa từ cột 3 sang cột 2, dùng cột 1 làm trung gian.

Có thể thấy bài toán 1 và 3 tương tự như bài toán ban đầu, chỉ khác ở chỗ kích thước của chúng nhỏ hơn. Áp dụng phương pháp chia để trị có thể dễ dàng giải bài toán trên.

Giải thuật được viết dưới dạng giả ngôn ngữ như sau:

```
Procedure move(n,a,b,c);
{chuyển n đĩa, từ cột a sang cột b, dùng cột c làm trung gian }
Begin
      If n=1 Then move(1,a,b,c)
 Else Begin
                move(n-1,a,c,b);
                move(1,a,b,c);
                move(n-1,c,b,a);
    End;
End;
```

Gọi T(n) là số thao tác chuyển đĩa cần thiết để chuyển xong n đĩa. Độ phức tạp tính toán của giải thuật trên sẽ là:

$$T(n) = T(n-1) + 1 + T(n-1).$$

Bằng các phương pháp giải công thức truy hồi ta có được $T(n) = 2^n-1$. Áp dụng kết quả này với giả thiết rằng mỗi cao tăng phải mất 1 giây để chuyển xong một đĩa từ cọc này sang cọc kia, ta thấy thời gian để chuyển toàn bộ 64 đĩa vàng là $T(64)=2^{64}-1=18446744073709551615$ giây. Thời gian này là gần 60 tỉ năm.

c. Bài toán nhân hai số lớn

Rất nhiều ứng dụng trong thực tế đòi hỏi phải xử lí các số rất lớn, nằm ngoài khoảng biểu diễn của các kiểu cơ sở của ngôn ngữ lập trình (chẳng hạn việc tìm các số nguyên tố rất lớn trong mã hoá RSA, thống kê dân số, mức thu nhập ...). Để giải quyết các yêu cầu đó, chúng ta phải xây dựng các kiểu số rất lớn và xây dựng các phép toán tương ứng. Trong phần này chỉ xét phép toán nhân đối với hai số rất lớn. Giả thiết cả hai đều có n chữ số và được biểu diễn bằng mảng. Bài toán nhân 2 số lớn phát biểu như sau:

Input. A,B là 2 số nguyên có n chữ số: $A=A_1A_2....A_n$ *và* $B=B_1B_2....B_n$.

Output. C=A.B

Giải thuật tự nhiên (brute-Force) của bài toán nhân 2 số lớn là giải thuật nhân tay ta vẫn thực hiện: lần lượt nhân từng chữ số của số thứ hai với số thứ nhất, dịch kết quả theo vị trí và cộng các kết quả trung gian lại.

Chẳng hạn để nhân A=1981 và B=1234 ta tiến hành như sau:

1981

× 1234

7924

+ 5943

3962

1981

2444554

Giải thuật nhân 2 số lớn kiểu nhân tay được mô tả bằng giả mã:

```
Function Mul(A,B,n)
Begin
    T := 0;
    For i := n Downto 1 Do
      Begin
            D := A* Bi;
            D := D shl (i-1); {dịch D sang trái n-i chữ số, tức là nhân D với 10^(n-i)}
            T := T + D;
      End;
    return T;
End;
```

Dễ dàng tính được độ phức tạp tính toán của giải thuật này là $O(n^2)$. Chúng ta cố gắng giảm bậc của giải thuật với tư tưởng chia để trị. Để đơn giản, giả thiết n=2k và tách A,B dưới dạng XY và UV trong đó X,Y,U,V là các số có k chữ số. Như vậy phép nhân 2 số A,B được tính như sau:

$$A.B = (X.10^k + Y).(U.10^k+V) = XU.10^{2k} + (XV+YU).10^k+YV.$$

Kết quả là bài toán nhân 2 số A.B có 2k chữ số được chia thành 4 bài toán con nhân các số có k chữ số và một số phép cộng, trừ:

Đặt P = XU; Q = YV; R = XV; S=YU.

Ta có: $A.B = P.10^{2k} + (R+S).10^k + Q.$

Như vậy bài toán nhân 2 số A.B có n chữ số được chia thành 4 bài toán con nhân các số n/2 chữ số và một số phép cộng, trừ. Giải thuật được viết dạng giả mã như sau:

```
function Mul(A,B,n)
Begin
  If n=1 Then return A1*B1;
  k = n div 2;
  X :=A(1..k); Y := A(k+1..n);
  U :=B(1..k); V := B(k+1..n);
  P := Mul(X,U,k);
```

```
    Q := Mul(Y,V,k);
    R := Mul(X,V,k);
    S:= Mul(Y,U,k);
    T := P shl 2*k + (R+S) shl k + Q;
    return T
End;
```

Để xác định độ phức tạp của giải thuật ta coi thao tác cơ bản là phép nhân, cộng và dịch trái từng chữ số. Gọi T(n) là số thao tác cơ bản để thực hiện nhân 2 số có n chữ số. Ta có:

$$T(n) = 3T(n/2) + C.n$$

Người ta đã tính được $T(n) = n^{\log_2 3} = n^{1.59}$, tức là có một số cải thiện so với giải thuật nhân tay. (Tuy nhiên khác biệt cũng không rõ rệt và chỉ thể hiện khi n khá lớn nên thông thường trong các bài toán không lớn lắm người ta thường dùng giải thuật đầu tiên).

1.2.5.2. Sinh cấu hình tổ hợp

Có rất nhiều bài toán trong Tin học có yêu cầu: tìm các đối tượng x thoả mãn những điều kiện nhất định trong một tập S các đối tượng cho trước. Bài toán tìm cấu hình tổ hợp là bài toán yêu cầu tìm các đối tượng x có dạng là một vector thoả mãn các điều kiện sau:

1. Đối tượng x gồm n phần tử: $x = (x_1, x_2, \ldots x_n)$.
2. Mỗi phần tử x_i có thể nhận một trong các giá trị rời rạc $a_1, a_2, \ldots a_m$.
3. x thoả mãn các ràng buộc có thể cho bởi hàm logic G(x).

Tuỳ từng trường hợp mà bài toán có thể yêu cầu: tìm một nghiệm, tìm tất cả các nghiệm hoặc đếm số nghiệm. Mỗi đối tượng x như vậy được gọi là một cấu hình tổ hợp.

a) Tổ hợp

Một tổ hợp chập k của n là một tập con k phần tử của tập n phần tử.

Chẳng hạn tập {1,2,3,4} có các tổ hợp chập 2 là: {1,2}, {1,3, {1,4, {2,3}, {2,4}, {3,4}. Vì trong tập hợp các phần tử không phân biệt thứ tự nên tập {1,2} cũng là tập {2,1} và coi chúng chỉ là một tổ hợp.

Bài toán đặt ra là **hãy xác định tất cả các tổ hợp chập k của tập n phần tử**. Để đơn giản chỉ xét bài toán tìm các tổ hợp của tập các số nguyên từ 1 đến n. Đối với một tập hữu hạn bất kì, bằng cách đánh số thứ tự của các phần tử ta cũng đưa được về bài toán đối với tập các số nguyên từ 1 đến n.

Nghiệm cần tìm của bài toán tìm các tổ hợp chập k của n phần tử phải thoả mãn các điều kiện sau:

1. Là một vector x =(x_1,x_2,…x_k)
2. x_i lấy giá trị trong tập {1,2,…n}
3. Ràng buộc: x_i<x_{i+1} với mọi giá trị i từ 1 đến k-1.

Có ràng buộc 3 là vì tập hợp không phân biệt thứ tự phần tử nên ta sắp xếp các phần tử theo thứ tự tăng dần.

b) Chỉnh hợp lặp

Chỉnh hợp lặp chập k của n là một dãy k thành phần, mỗi thành phần là một phần tử của tập n phần tử, có xét đến thứ tự và không yêu cầu các thành phần khác nhau.

Một ví dụ dễ thấy nhất của chỉnh hợp lặp là các dãy nhị phân. Một dãy nhị phân độ dài m là một chỉnh hợp lặp chập m của tập 2 phần tử {0,1}. Chẳng hạn 101 là một dãy nhị phân độ dài 3. Ngoài ra ta còn có 7 dãy nhị phân độ dài 3 nữa là 000, 001, 010, 011, 100, 110, 111. Vì có xét thứ tự nên dãy 101 và dãy 011 là 2 dãy khác nhau.

Như vậy, bài toán xác định tất cả các chỉnh hợp lặp chập k của tập n phần tử yêu cầu tìm các nghiệm như sau:

1. Là một vector x =(x_1,x_2,…x_k)
2. x_i lấy giá trị trong tập {1,2,…n}

3. Không có ràng buộc nào giữa các thành phần.

Cũng như bài toán tìm tổ hợp, chỉ xét đối với tập n số nguyên từ 1 đến n. Nếu tập hợp cần tìm chỉnh hợp không phải là tập các số nguyên từ 1 đến n thì ta có thể đánh số các phần tử của tập đó để đưa về tập các số nguyên từ 1 đến n.

c) Chỉnh hợp không lặp

Khác với chỉnh hợp lặp là các thành phần được phép lặp lại, tức là có thể giống nhau, chỉnh hợp không lặp chập k của tập n phần tử cũng là một dãy k thành phần lấy từ tập n phần tử có xét thứ tự nhưng các thành phần không được phép giống nhau.

Chẳng hạn có n người, một cách chọn ra k người để xếp thành một hàng là một chỉnh hợp không lặp chập k của n.

Một trường hợp đặc biệt của chỉnh hợp không lặp là hoán vị. Hoán vị của một tập n phần tử là một chỉnh hợp không lặp chập n. Nói một cách trực quan thì hoán vị của tập n phần tử là phép thay đổi vị trí của các phần tử (do đó mới gọi là hoán vị).

Nghiệm của bài toán **tìm các chỉnh hợp không lặp chập k của tập n số nguyên từ 1 đến n** là các vector x thoả mãn các điều kiện:

1. x có k thành phần: $x = (x_1,x_2,\ldots x_k)$
2. Các giá trị x_i lấy trong tập {1,2,..n}
3. Ràng buộc: các giá trị x_i từng đôi một khác nhau, tức là $x_i \neq x_j$ với mọi $i \neq j$.

Trên đây là một số bài toán tìm cấu hình tổ hợp cơ bản. Những bài toán dạng này rất phổ biến. Giải thuật tổng quát để sinh các cấu hình tổ hợp thường được gọi là "duyệt", "vét cạn" hay "quay lui" dựa trên ý tưởng: xây dựng lần lượt từng thành phần của cấu hình, mỗi thành phần đều duyệt hết tất

cả các khả năng có thể, nếu không còn khả năng nào thì quay lại để xét các thành phần trước đó.

Giải thuật này có thể viết như sau:

```
Procedure search;
Begin
   try(1);
End;
Procedure try(i);
Var j;
Begin
   For j := 1 to m Do
      If <chọn được a[j]> Then
                Begin
         x[i] := a[j];       <ghi nhận trạng thái mới>;
         If i=n Then print(x)
         else try(i+1);       <trả lại trạng thái cũ>;
       End;
End;
```

Áp dụng giải thuật tổng quát đó cho bài toán tìm tất các tổ hợp k phần tử của các số từ 1 đến n, ta có giải thuật đệ qui như sau:

```
Procedure try(i);
Var j;
Begin
   For j := x[i–1]+1 to n–k+i Do
Begin
      x[i] := j;
      If i=k Then print(x)else try(i+1);
           End;
End;
```

Giải thuật tương tự dùng sinh các hoán vị như sau:

```
Procedure Try(i);
Var j;
Begin
   For j := 1 to n Do
      If d[j]=0 Then
                  Begin
         x[i] := j; d[j] := 1;
```

```
        If i=n Then Print(x)
        else Try(i+1);
        d[i] := 0;
        End;
End;
```

Ý tưởng cơ bản của giải thuật sinh hoán vị là dùng một mảng để đánh dấu các phần tử đã chọn. Do đó mỗi phần tử trong một hoán vị chỉ xuất hiện đúng một lần.

Giải thuật sinh cấu hình tổ hợp có thể dùng để giải các bài toán tối ưu tổ hợp. Đó là các bài toán yêu cầu tìm nghiệm dưới dạng một cấu hình tổ hợp và cho giá trị tối ưu với một hàm mục tiêu nào đó. Các bài toán tối ưu tổ hợp có thể phát biểu cách tổng quát như sau:

Bài toán: Tìm các đối tượng x thoả mãn các điều sau:

1. Đối tượng x gồm n thành phần: $x = (x_1,x_2,\ldots x_n)$.
2. Mỗi phần tử x_i có thể nhận một trong các giá trị rời rạc $a_1, a_2, \ldots a_m$.
3. x thoả mãn các ràng buộc có thể cho bởi hàm logic G(x).
4. x làm cực đại/cực tiểu hàm mục tiêu $F(x) \rightarrow \max/\min$.

Bài toán tối ưu tổ hợp điển hình là bài toán Hamilton: Cho n thành phố, chi phí đi lại từ thành phố i đến thành phố j là C[i,j]. Hãy tìm một lộ trình xuất phát từ một thành phố, đi qua n thành phố rồi quay lại thành phố ban đầu sao cho tổng chi phí là thấp nhất.

Nghiệm của bài toán sẽ là một hoán vị của n thành phố sao cho luôn có đường đi giữa 2 thành phố liên tiếp và từ thành phố cuối cùng về thành phố xuất phát. Hàm mục tiêu ở đây là tổng chi phí vì cần lộ trình có tổng chi phí là min.

Giải thuật "vét cạn" để giải bài toán tối ưu tổ hợp như sau:

```
Procedure search;
Begin
   try(1);
End;
```

```
Procedure try(i);
Var j;
Begin
   For j := 1 to m Do
      If <chọn được a[j]> Then
                  Begin
          x[i] := a[j];       <ghi nhận trạng thái mới>;
          If i=n Then Update(x)
          else If <đi tiếp được> Then try(i+1);
          <trả lại trạng thái cũ>;
       End;
End;
```

```
Procedure Update(x)
Begin
  If x <tốt hơn> best Then best := x; {best là nghiệm tối ưu}
End;
```

1.3. Một số giải thuật thường dùng trong các bài toán khoa học, kinh tế, tài chính

1.3.1. Tính tổng và tích các phần tử của một dãy

Trong các bài toán khoa học hay kinh tế thường gặp các yêu cầu tính tổng một dãy số cho trước.

Ví dụ 1: Cần tính tổng số tiền thuế phải nộp của n doanh nghiệp, trong đó sô thuế mỗi doanh nghiệp phải nộp là a_i (i=1..n)

Ví dụ 2: Tính tổng số điện sử dụng cả năm của một hộ gia đình. Biết mỗi tháng hộ đó sử dụng hết a_i số điện (i=1..12).

Ví dụ 3: Tính tổng tiền lương của các cán bộ trong một đơn vị công tác.

Ví dụ 4: Tính tổng các khoản thu, tổng các khoản chi trong năm của một đơn vị

...

Các bài toán trên đều qui về tính tổng các số. Do đó ta có thể phát biểu bài toán tổng quát như sau:

Bài toán:

Cho dãy số A gồm n phần tử a_1, a_2,… a_n. Hãy tính tổng $S=a_1+a_2+…+a_n$.

Giải thuật:

Ta tính tổng S bằng phương pháp lặp. Đầu tiên cho S bằng 0, sau đó lần lượt cộng vào S các phần tử a_1, a_2,… a_n. Giải thuật được diễn đạt bằng ngôn ngữ tựa Pascal như sau:

```
Function Sum(a,n);
Begin
  S := 0;
  For i:=1 to n Do S:=S+a[i];
  return S;
End;
```

Tương tự như bài toán tính tổng các phần tử của một dãy, bài toán tính tích các phần tử của một dãy như sau:

Bài toán:

Cho dãy số A gồm n phần tử a_1, a_2,… a_n. Hãy tính tích $P=a_1.a_2…a_n$.

Giải thuật:

Ta sẽ tính P bằng phương pháp lặp. Đầu tiên ta cho P bằng 1. Sau đó lần lượt nhân vào P các phần tử a_1, a_2,… a_n. Kết thúc quá trình lặp ta sẽ được P là tích của n phần tử của dãy.

Giải thuật được diễn đạt bằng ngôn ngữ tựa Pascal như sau:

```
Function Product(a,n);
Begin
  P := 1;
  For i:=1 to n Do P:=P*a[i];
  return P;
End;
```

1.3.2. Tìm phần tử lớn nhất và nhỏ nhất của một dãy

Xác định phần tử lớn nhất và nhỏ nhất của một dãy cũng là công việc rất hay gặp trong các vấn đề khoa học hay kinh tế.

Ví dụ 1: Có n hộ kinh doanh phải nộp thuế, số thuế mỗi hộ phải nộp là a_i. Cần xác định xem hộ nào nộp thuế nhiều nhất và hộ nào nộp ít nhất?.

Ví dụ 2: Trong một tháng mỗi hộ gia đình sử dụng diện hết a_i số điện. Cần xác định xem hộ nào sử dụng nhiều nhất và hộ sử dụng ít nhất?.

Ví dụ 3: Có n nhà đầu tư vào một khu vực kinh tế. Xác định xem ai có vốn đầu tư cao nhất, ai có vốn đầu tư thấp nhất.

Ví dụ 4: Trong các khách hàng vay vốn, khách hàng nào có kết quả kinh doanh hiệu quả nhất, khách hàng nào có kết quả kinh doanh kém hiệu quả nhất..

Các bài toán trên đều qui về tìm số lớn nhất (Max) và số nhỏ nhất (Min) Có thể phát biểu bài toán tổng quát như sau:

Bài toán:

Cho dãy số A gồm n phần tử a_1, a_2,… a_n. Hãy tìm phần tử max là giá trị lớn nhất của dãy.

Giải thuật:

Ta sẽ tìm max bằng phương pháp so sánh 2 số một số nào lớn hơn lấy số đó so sánh với số tiếp theo. Đầu tiên ta cho max bằng a_1. Sau đó ta lần lượt so sánh max với các phần tử a_2,…,a_n. Nếu max nhỏ hơn phần tử nào thì gán max cho phần tử đó. Kết thúc quá trình lặp sẽ có max là giá trị lớn nhất của dãy.

Giải thuật được diễn đạt bằng ngôn ngữ tựa Pascal như sau:

```
Function Max(a,n);
Begin
  max := a1;
  For i:=2 to n Do If max < ai Then max := ai;
```

```
  return max;
End;
```

Trong giải thuật trên, nếu thay dấu "<" bằng dấu ">" thì ta được giải thuật tìm giá trị nhỏ nhất của một dãy:

```
Function Min(a,n);
Begin
  min := a1;
  For i:=2 to n Do If min > ai Then min := ai;
  return min;
End;
```

1.3.3. Tính giá trị trung bình

Ví dụ 1: Trong một tháng có n hộ kinh doanh phải nộp thuế, số thuế mỗi hộ phải nộp là a_i. Tính số tiền thuế bình quân mỗi hộ phải là nộp bao nhiêu?.

Ví dụ 2: Một hộ gia đình sử dụng điện mỗi tháng hết a_i số điện. Tính số điện sử dụng bình quân trong một tháng của hộ gia đình này trong năm.

Ví dụ 3: Có n nhà đầu tư vào một khu vực kinh tế với số tiền là a_i đồng (i=1..n). Tính bình quân vốn đầu tư của mỗi nhà kinh tế là bao nhiêu.

...

Các bài toán trên đều qui về tính giá trị trung bình cộng của một dãy số. Do đó có thể phát biểu bài toán tổng quát như sau:

Bài toán:

Cho các phần tử a_1, a_2,… a_n của một dãy và các tần số tương ứng f_1, f_2,… f_n. Hãy tính giá trị trung bình của chúng theo công thức sau:

$$\overline{a} = \frac{\sum_{i=1}^{n} a_i f_i}{\sum_{i=1}^{n} f_i}$$

Giải thuật:

Sử dụng giải thuật tính tổng ở phần 1.3.1 và công thức trên, sẽ tính được giá trị trung bình. Giải thuật cụ thể được trình bày bằng ngôn ngữ tựa Pascal như sau.

```
Function Mean(a,n);
Begin
   ts := 0; ms := 0;
   For i:=1 to n Do
         Begin
               ts := ts + ai*fi;
               ms := ms + fi;
  End;
   return (ts/ms);
End;
```

1.3.4. Tìm số nguyên tố và phân tích thành thừa số nguyên tố

Một số nguyên dương được gọi là số nguyên tố nếu chúng không có ước số nào khác ngoài 1 và chính nó. Mọi số nguyên dương đều có thể phân tích một cách duy nhất dưới dạng tích của một hay nhiều số nguyên tố.

Rất nhiều bài toán trong thực tế (chẳng hạn như mật mã) cần đến các số nguyên tố và kết quả phân tích thành thừa số nguyên tố của các số. Người ta đã phát triển nhiều giải thuật khác nhau để thực hiện công việc này. Tuy nhiên giáo trình chỉ trình bày những giải thuật cơ bản nhất.

Các giải thuật được trình bày trong phần này dựa trên nhận xét sau: một số mà là hợp số thì sẽ có một ước số nhỏ hơn hoặc bằng căn bậc hai của nó. (Giả sử n=a*b và a≤b, thế thì rõ ràng $a^2 \le n$).

Vậy một số nguyên lớn hơn 1 sẽ là số nguyên tố nếu nó không có ước số nào nhỏ hơn hoặc bằng căn bậc hai của nó. Dựa trên nhận xét đó, ta xây dựng giải thuật kiểm tra số nguyên tố như sau:

```
Function ngto(n);
Begin
  If n<2 Then return false;
  For i := 2 to round(sqrt(n)) Do
```

```
    If n mod i = 0 Then return false;
  Return true;
End;
```

Cũng dựa trên nhận xét đó, ta có thể phân tích một số thành thừa số nguyên tố bằng cách chia nó cho 2,3,4,… cho đến hết thì thôi.

```
Function ptnt(n)
Begin
  t := n; i := 2;
  Repeat
    If t mod i = 0 Then
                  Begin
              write(i);
                   t := t div i;
                  End
         else i := i + 1;
  Until t=1;
End;
```

1.3.5. Tính các hàm theo công thức lặp

Trong toán học có rất nhiều hàm được định nghĩa theo các công thức lặp như:

- Hàm luỹ thừa: a^n=a*a*a…*a (n lần)
- Hàm giai thừa: n! = 1*2*…*n
- Hàm mũ: $e^x = 1 + x + \frac{x^2}{2!} + ... \frac{x^n}{n!} + ...$
- Hàm lượng giác: $\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + ...$

  $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + ...$

Nhìn chung người ta thường tính các hàm này bằng phương pháp lặp. Chẳng hạn để tính $e^x = 1 + x + \frac{x^2}{2!} + ... \frac{x^n}{n!} + ...$

đặt $p_n = \frac{x^n}{n!}$, $s_n = 1 + x + \frac{x^2}{2!} + ... \frac{x^n}{n!}$ ta có công thức truy hồi:

$s_n=s_{n-1}+p_n$

$p_n=p_{n-1}*x/n$

Từ công thức trên, xây dựng giải thuật tính hàm e^x gần đúng như sau:

```
Function exp(x,n)
Begin
 s := 1; p := 1;
 For i := 1 to n Do
        Begin
              p := p*x/i;
              s := s + p;
 End;
 return s;
End;
```

Dựa trên giải thuật này, bạn đọc có thể tự xây dựng giải thuật để tính các hàm sin, cos theo phương pháp lặp như trên. Phần này coi như là bài tập dành cho bạn đọc.

1.3.6. Nhân ma trận

Phép nhân ma trận được định nghĩa như sau: Cho ma trận A kích thước m×n, ma trận B kích thước n×p, ma trận C kích thước m×p là ma trận kết quả của phép nhân ma trận A và B với các phần tử được tính theo công thức sau:

$$Cij = \sum_{k=1}^{n} A_{ik} * B_{kj}$$

Giải thuật sau sẽ tính tích 2 ma trận theo công thức trên:

```
Function nhan(a,b,c,m,n,p);
Begin
 For i := 1 to m Do
  For j := 1 to p Do
        Begin
    c[i,j]:=0;
        For k:=1 to n Do c[i,j]:= c[i,j] + a[i,k]*b[k,j];
    End;
End;
```

Câu hỏi và bài tập chương 1

1. Trình bày các bước cơ bản khi giải một bài toán trong tin học.
2. Hãy thu thập các chứng từ tài chính như: hoá đơn bán hàng, hoá đơn nhập hàng, báo cáo cuối năm; chứng từ thu, chứng từ chi, bảng lương. bảng quyết toán cuối năm...

   + Xác định trên các chứng từ đó những dữ liệu cơ sở.

   + Xác định dữ liệu nào là thông tin vào, dữ liệu nào là thông tin ra
3. Trình bày khái niệm cấu trúc dữ liệu, giải thuật. Cho một số ví dụ về cấu trúc dữ liệu mà bạn biết.
4. Mối liên hệ giữa cấu trúc dữ liệu và giải thuật. Cho ví dụ minh hoạ
5. Hãy trình bày về các phương pháp mô tả giải thuật.
6. Mô tả giải thuật sắp xếp dãy số X_1, X_2 ... X_n theo trình tự giảm dần
7. Tính thời gian thực hiện các đoạn chương trình sau:

a. Tính tổng:

```
Sum:=0;
For i:=1 to n Do
 Begin
        Readln(x);
        Sum:=Sum+x;
 End;
```

b. Tính tích hai ma trận vuông cấp n:

```
For i:=1 to n Do
For j:=1 to n Do
 Begin
        C[i,j]:=0;
        For k:=1 to n Do
        C[i,j]:=C[i,j] + A[i,k]*B[k,j];
 End;
```

8. Hãy trình bày tư tưởng của phương pháp module bài toán? Tại sao lại phải module hoá bài toán?.
9. Cho thủ tục sau:

```
Procedure Mystery(n:Integer);
```

```
Var     i,j,k: Integer;
Begin
        For i:=1 to n-1 Do
                For j:=i+1 to n Do
                        For k:=1 to j Do S:=S+1;
End;
```

Hãy chỉ ra thời gian thực hiện các lệnh trong thủ tục trên là:

T(n)= $O(1)*\frac{(n-1)*n*(n+1)}{3}$. Từ đó suy ra rằng độ phức tạp của thủ tục này là $O(n^3)$.

10. Hãy chỉ ra thời gian thực hiện các lệnh trong thủ tục dưới đây là:

T(n)= (n+1)*((n Div 2) +1). Từ đó suy ra độ phức tạp của thủ tục này là $O(n^2)$

```
Procedure Veryodd(n: Integer);
Var i,j,x,y: Integer;
Begin
        For i:=1 to n Do
                If odd(i) Then
                        Begin
                                For j:=i to n Do x:=x+1;
                                For j:=1 to i Do y:=y+1;
                        End;
End.
```

11. Cho trước khai báo sau đây:

Type cardinal=1..maxInt;

Các hàm đệ quy dưới đây làm công việc gì?

a.
```
Function F(n:cardinal):Cardinal;
Begin
        If n=0 Then F:=0
        Else F:=n+F(n-1);
End;
```

b.
```
Function F(x:real, n:Cardinal):Real;
Begin
        If n=0 Then F:=1
        Else F:=x*F(x,n-1);
End;
```

c. Function f(n:cardinal):Cardinal;
Begin
    If n<1 Then F:=0
    Else F:=1+F(n Div 2);
End;

12. Hãy viết lại các hàm trong bài tập 11 dưới dạng không đệ qui
13. Hãy viết hàm đệ qui nhận chuỗi ký tự và trả lại chuỗi ký tự đảo ngược lại với chuỗi ban đầu.
14. Hãy viết thủ tục đệ qui và không đệ qui hiển thị các chữ số của một số nguyên theo thứ tự ngược lại với các chữ số ban đầu.
15. Một số được gọi là đối xứng nếu biểu diễn thập phân của nó là đối xứng (Ví dụ: 121, 356653 là các số đối xứng). Viết hàm đệ qui xác định một số nguyên cho trước có đối xứng hay không?
16. Viết giải thật đệ qui và không đệ qui tính tổ hợp chập k của n phần tử.

TÀI LIỆU THAM KHẢO

1. Robert Bedgenich - Cẩm nang thuật toán – Nhà xuất bản khoa học kỹ thuật - 1998
2. Henning Mittelbach - Lập trình bằng Turbo Pascal – Nhà xuất bản khoa học kỹ thuật - 1996
3. Đỗ Xuân Lôi - Cấu trúc dữ liệu và giải thuật – Nhà xuất bản thống kê - 1999
4. Lê Minh Trung – Bài tập cấu trúc dữ liệu và giải thuật – Nhà xuất bản thống kê - 2003