Độ phức tạp tính toán và tính hiệu quả của thuật toán

Bởi:

Đại Học Phương Đông

Sự cần thiết phải phân tích thuật toán

Trong khi giải một bài toán chúng ta có thể có một số giải thuật khác nhau, vấn đề là cần phải đánh giá các giải thuật đó để lựa chọn một giải thuật tốt (nhất). Thông thường thì ta sẽ căn cứ vào các tiêu chuẩn sau:

1. Giải thuật đúng đắn.

2. Giải thuật đơn giản.

3. Giải thuật thực hiện nhanh.

Với yêu cầu (1), để kiểm tra tính đúng đắn của giải thuật chúng ta có thể cài đặt giải

thuật đó và cho thực hiện trên máy với một số bộ dữ liệu mẫu rồi lấy kết quả thu

được so sánh với kết quả đã biết. Thực ra thì cách làm này không chắc chắn bởi vì có

thể giải thuật đúng với tất cả các bộ dữ liệu chúng ta đã thử nhưng lại sai với một bộ

dữ liệu nào đó. Vả lại cách làm này chỉ phát hiện ra giải thuật sai chứ chưa chứng minh được là nó đúng. Tính đúng đắn của giải thuật cần phải được chứng minh bằng toán học. Tất nhiên điều này không đơn giản và do vậy chúng ta sẽ không đề cập đến ở đây.

Khi chúng ta viết một chương trình để sử dụng một vài lần thì y ê u cầu (2) là quan trọng nhất. Chúng ta cần một giải thuật dễ viết chương trình để nhanh chóng có được kết quả, thời gian thực hiện chương trình không được đề cao vì dù sao thì chương trình đó cũng chỉ sử dụng một vài lần mà thôi.

Tuy nhiên khi một chương trình được sử dụng nhiều lần thì thì yêu cầu tiết kiệm thời gian thực hiện chương trình lại rất quan trọng đặc biệt đối với những chương trình mà

khi thực hiện cần dữ liệu nhập lớn do đó yêu cầu (3) sẽ được xem xét một cách kĩ càng. Ta gọi nó là hiệu quả thời gian thực hiện của giải thuật.

Thời gian thực hiện của chương trình

Một phương pháp để xác định hiệu quả thời gian thực hiện của một giải thuật là lập trình nó và đo lường thời gian thực hiện của hoạt động trên một máy tính xác định

đối với tập hợp được chọn lọc các dữ liệu vào.

Thời gian thực hiện không chỉ phụ thuộc vào giải thuật mà còn phụ thuộc vào tập

các chỉ thị của máy tính, chất lượng của máy tính và kĩ xảo của người lập trình. Sự

thi hành cũng có thể điều chỉnh để thực hiện tốt trên tập đặc biệt các dữ liệu vào được chọn. Để vượt qua các trở ngại này, các nhà khoa học máy tính đã chấp nhận tính phức tạp của thời gian được tiếp cận như một sự đo lường cơ bản sự thực thi của giải thuật. Thuật ngữ tính hiệu quả sẽ đề cập đến sự đo lường này và đặc biệt

đối với sự phức tạp thời gian trong trường hợp xấu nhất.

Thời gian thực hiện chương trình.

Thời gian thực hiện m ộ t chương t r ì n h là một hàm của kích thước dữ liệu vào, ký hiệu T(n) trong đó n là kích thước (độ lớn) của dữ liệu vào.

Chương trình tính tổng của n số có thời gian thực hiện là T(n) = cn trong đó c là một hằng số.

Thời gian thực hiện chương trình là một hàm không âm, tức là $T(n) \geq 0$ với mọi $n \geq 0$.

Đơn vị đo thời gian thực hiện.

Đơn vị của T(n) không phải là đơn vị đo thời gian bình thường như giờ, phút giây...

mà thường được xác định bởi số các lệnh được thực hiện trong một máy tính lý tưởng.

Khi ta nói thời gian thực hiện của một chương trình là T(n) = Cn thì có nghĩa là chương trình ấy cần Cn chỉ thị thực thi.

Thời gian thực hiện trong trường hợp xấu nhất.

Nói chung thì thời gian thực hiện chương trình không chỉ phụ thuộc vào kích thước mà còn phụ thuộc vào tính chất của dữ liệu vào. Nghĩa là dữ liệu vào có cùng kích thước

nhưng thời gian thực hiện chương trình có thể khác nhau. Chẳng hạn chương trình sắp xếp dãy số nguyên tăng dần, khi ta cho vào dãy có thứ tự thì thời gian thực hiện khác với khi ta cho vào dãy chưa có thứ tự, hoặc khi ta cho vào một dãy đã có thứ tự tăng thì thời gian thực hiện cũng khác so với khi ta cho vào một dãy đã có thứ tự giảm.

Vì vậy thường ta coi T(n) là thời gian thực hiện chương trình trong trường hợp xấu nhất trên dữ liệu vào có kích thước n, tức là: T(n) là thời gian lớn nhất để thực hiện chương trình đối với mọi dữ liệu vào có cùng kích thước n.

Tỷ suất tăng và Độ phức tạp của giải thuật

Tỷ suất tăng

Ta nói rằng hàm không âm T(n) có tỷ suất tăng (growth rate) f(n) nếu tồn tại các hằng số C và N0 sao cho $T(n) \leq Cf(n)$ với mọi $n \geq N0$.

Ta có thể c h ứ ng minh đư ợ c rằng "Cho m ộ t hàm không âm T(n) b ấ t kỳ, ta luôn tìm đư ợ c t ỷ s u ất tăng f (n) c ủa nó".

Giả sử T(0) = 1, T(1) = 4 và tổng quát $T(n) = (n+1)_2$. Đặt N0 = 1 và C = 4 thì với mọi $n \geq 1$ chúng ta dễ dàng chứng minh được rằng $T(n) = (n+1)_2 \leq 4n_2$ với mọi $n \geq 1$, tức là tỷ suất tăng của T(n) là n_2.

Tỷ suất tăng của hàm $T(n) = 3n_3 + 2n_2$ là n_3. Thực vậy, cho N0 = 0 và C = 5 ta dễ dàng chứng minh rằng với mọi $n \geq 0$ thì $3n_2 + 2n_2 \leq 5n_3$

Khái niệm độ phức tạp của giải thuật

Giả sử ta có hai giải thuật P1 và P2 với thời gian thực hiện tương ứng là $T1(n) = 100n_2$ (với tỷ suất tăng là n_2) và $T2(n) = 5n_3$ (với tỷ suất tăng là n_3). Giải thuật nào sẽ thực hiện nhanh hơn? Câu trả lời phụ thuộc vào kích thước dữ liệu vào. Với $n < 20$ thì P2 sẽ nhanh hơn P1 ($T2<T1$), do hệ số của $5n_3$ nhỏ hơn hệ số của $100n_2$ ($5<100$). Nhưng khi $n > 20$ thì ngược lại do số mũ của $100n_2$ nhỏ hơn số mũ của $5n_3$ ($2<3$). Ở đây chúng ta chỉ nên quan tâm đến trường hợp $n>20$ vì khi $n<20$ thì thời gian thực hiện của cả P1 và P2 đều không lớn và sự khác biệt giữa T1 và T2 là không đáng kể.

Như vậy một cách hợp lý là ta xét tỷ suất tăng của hàm thời gian thực hiện chương trình thay vì xét chính bản thân thời gian thực hiện.

Cho mộ t hàm T(n), T(n) g ọ i là có độ phức t ạ p f(n) n ế u t ồn t ạ i các hằng C, N 0 sao cho $T(n) \leq Cf(n)$ v ớ i m ọ i $n \geq N 0$ (tức là T(n) có t ỷ suấ t t ăng là f(n)) và kí h i ệu T(n) là O(f(n)) (đọc là "ô c ủ a f(n)")

$T(n)= (n+1)_2$ có tỷ suất tăng là n_2 nên $T(n)= (n+1)_2$ là $O(n_2)$

Chú ý: O(C.f(n))=O(f(n)) với C là hằng số. Đặc biệt O(C) = O(1)

Nói cách khác độ phức tạp tính toán của giải thuật là một hàm chặn trên của hàm thời gian. Vì hằng nhân tử C trong hàm chặn trên không có ý nghĩa nên ta có thể bỏ qua vì vậy hàm thể hiện độ phức tạp có các dạng thường gặp sau: log2n, n, nlog2n, n_2, n_3, 2_n, n!, n_n. Ba hàm cuối cùng ta gọi là dạng hàm mũ, các hàm khác gọi là hàm đa thức. Một giải thuật mà thời gian thực hiện có độ phức tạp là một hàm đa thức thì chấp nhận được tức là có thể cài đặt để thực hiện, còn các giải thuật có độ phức tạp hàm mũ thì phải tìm cách cải tiến giải thuật.

Vì ký hiệu log2n thường có mặt trong độ phức tạp nên trong khuôn khổ tài liệu này, ta sẽ dùng logn thay thế cho log2n với mục đích duy nhất là để cho gọn trong cách viết.

Khi nói đến độ phức tạp của giải thuật là ta muốn nói đến hiệu quả của thời gian thực hiện của chương trình nên ta có thể xem việc xác định thời gian thực hiên của chương trình chính là xác định độ phức tạp của giải thuật.

Cách tính Độ phức tạp

Cách tính độ phức tạp của một giải thuật bất kỳ là một vấn đề không đơn giản. Tuy nhiên ta có thể tuân theo một số nguyên tắc sau:

Qui tắc cộng

Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2; và

T1(n)=O(f(n)), T2(n)=O(g(n)) thì thời gian thực hiện của đoạn hai chương trình đó

nối tiếpnhaulà T(n)=O(max(f(n),g(n)))

Lệnh gán x:=15 tốn một hằng thời gian hay O(1), Lệnh đọc dữ liệu

READ(x) tốn một hằng thời gian hay O(1). Vậy thời gian thực hiện cả hai lệnh trên nối tiếp nhau là O(max(1,1))=O(1)

Qui tắc nhân

Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1và P2 và T1(n) = O(f(n)), T2(n) = O(g(n)) thì thời gian thực hiện của đoạn hai đoạn chương trình đó **lồngnhau**là T(n) = O(f(n).g(n))

Qui tắc tổng quát để phân tích một chương trình

Thời gian thực hiện của mỗi lệnh gán, READ, WRITE là O(1).

Thời gian thực hiện của một chuỗi tuần tự các lệnh được xác định bằng qui tắc cộng. Như vậy thời gian này là thời gian thi hành một lệnh nào đó lâu nhất trong chuỗi lệnh.

Thời gian thực hiện cấu trúc IF là thời gian lớn nhất thực hiện lệnh sau THEN hoặc sau ELSE và thời gian kiểm tra điều kiện. Thường thời gian kiểm tra điều kiện là O(1).

Thời gian thực hiện vòng lặp là tổng (trên tất cả các lần lặp) thời gian thực hiện thân vòng lặp. Nếu thời gian thực hiện thân vòng lặp không đổi thì thời gian thực hiện vòng lặp là tích của số lần lặp với thời gian thực hiện thân vòng lặp.

Tính thời gian thực hiện của thủ tục sắp xếp "nổi bọt"

```
PROCEDURE Bubble(VAR a: ARRAY[1..n] OF integer); VAR
i,j,temp: Integer; BEGIN {1} FOR i:=1 TO n-1 DO {2} FOR
j:=n DOWNTO i+1 DO {3} IF a[j-1]>a[j]THEN BEGIN{hoán vị
a[i], a[j]} {4} temp := a[j-1]; {5} a[j-1] := a[j]; {6}
a[j] := temp; END; END;
```

Về giải thuật sắp xếp nổi bọt, chúng ta sẽ bàn kĩ hơn trong chương 2. Ở đây, chúng ta chỉ quan tâm đến độ phức tạp của giải thuật.

Ta thấy toàn bộ chương trình chỉ gồm một lệnh lặp {1}, lồng trong lệnh {1} là lệnh

{2}, lồng trong lệnh {2} là lệnh {3} và lồng trong lệnh {3} là 3 lệnh nối tiếp nhau

{4}, {5} và {6}. Chúng ta sẽ tiến hành tính độ phức tạp theo thứ tự từ trong ra.

Trước hết, cả ba lệnh gán {4}, {5} và {6} đều tốn O(1) thời gian, việc so sánh a[j-1] >

a[j] cũng tốn O(1) thời gian, do đó lệnh {3} tốn O(1) thời gian.

Vòng lặp {2} thực hiện (n-i) lần, mỗi lần O(1) do đó vòng lặp {2} tốn O((n-i).1) = O(n-i).Vòng lặp {1} lặp có I chạy từ 1 đến n-1nên thời gian thực hiện của vòng lặp

{1} và cũng là độ phức tạp của giải thuật là:

$$T(n) = \sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2} = O(n^2).$$

Chú ý: Trong trường hợp vòng lặp không xác định được số lần lặp thì chúng ta phải lấy số lần lặp trong trường hợp xấu nhất.

Tìm kiếm tuần tự. Hàm tìm kiếm Search nhận vào một mảng a có n số nguyên và một số nguyên x, hàm sẽ trả về giá trị logic TRUE nếu tồn tại một phần tử a[i] = x, ngược lại hàm trả về FALSE.

Giải thuật tìm kiếm tuần tự là lần lượt so sánh x với các phần tử của mảng a, bắt đầu từ a[1], nếu tồn tại a[i] = x thì dừng và trả về TRUE, ngược lại nếu tất cả các phần tử của a đều khác X thì trả về FALSE.

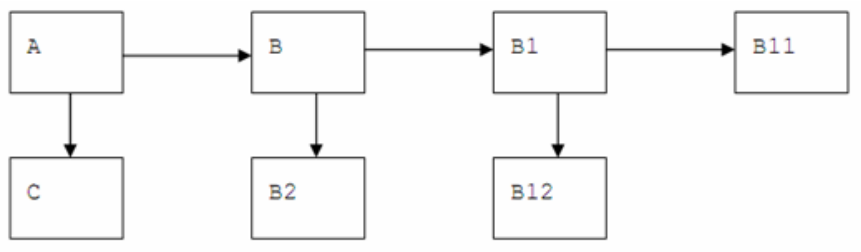
```
FUNCTION Search(a:ARRAY[1..n] OF
Integer;x:Integer):Boolean; VAR i:Integer; Found:Boolean;
BEGIN {1} i:=1; {2} Found:=FALSE; {3} WHILE(i<=n)AND (not
Found) DO {4} IF A[i]=X THEN Found:=TRUE ELSE i:=i+1; {5}
Search:=Found; END;
```

Ta thấy các lệnh {1}, {2}, {3} và {5} nối tiếp nhau, do đó độ phức tạp của hàm Search chính là độ phức tạp lớn nhất trong 4 lệnh này. Dễ dàng thấy rằng ba lệnh {1}, {2} và {5} đều có độ phức tạp O(1) do đó độ phức tạp của hàm Search chính là độ phức tạp của lệnh {3}. Lồng trong lệnh {3} là lệnh {4}. Lệnh {4} có độ phức tạp O(1). Trong trường hợp xấu nhất (tất cả các phần tử của mảng a đều khác x) thì vòng lặp {3} thực hiện n lần, vậy ta có T(n) = O(n).

Độ phức tạp của chương trình có gọi chương trình con không đệ quy

Nếu chúng ta có một chương trình với các chương trình con không đệ quy, để tính thời gian thực hiện của chương trình, trước hết chúng ta tính thời gian thực hiện của các chương trình con không gọi các chương trình con khác. Sau đó chúng ta tính thời gian thực hiện của các chương trình con chỉ gọi các chương trình con mà thời gian thực hiện của chúng đã được tính. Chúng ta tiếp tục quá trình đánh giá thời gian thực hiện của mỗi chương trình con sau khi thời gian thực hiện của tất cả các chương trình con mà nó gọi đã được đánh giá. Cuối cùng ta tính thời gian cho chương trình chính.

Giả sử ta có một hệ thống các chương trình gọi nhau theo sơ đồ sau:



Hình 2- 1: Sơ đồ gọi thực hiện các chương trình con không đệ quy

Chương trình A gọi hai chương trình con là B và C, chương trình B gọi hai chương trình con là B1 và B2, chương trình B1 gọi hai chương trình con là B11 và B12.

Để tính thời gian thực hiện của A, ta tính theo các bước sau:

1. Tính thời gian thực hiện của C, B2, B11 và B12. Vì các chương trình con này không gọi chương trình con nào cả.

2. Tính thời gian thực hiện của B1. Vì B1 gọi B11 và B12 mà thời gian thực hiện của B11 và B12 đã được tính ở bước 1.

3. Tính thời gian thực hiện của B. Vì B gọi B1 và B2 mà thời gian thực hiện của B1 đã được tính ở bước 2 và thời gian thực hiện của B2 đã được tính ở bước 1.

4. Tính thời gian thực hiện của A. Vì A gọi B và C mà thời gian thực hiện của B

đã được tính ở bước 3 và thời gian thực hiện của C đã được tính ở bước 1.

Ta có thể viết lại chương trình sắp xếp bubble như sau: Trước hết chúng ta viết thủ tục Swap để thực hiện việc hoàn đổi hai phần tử cho nhau, sau đó trong thủ

tục Bubble, khi cần ta sẽ gọi đến thủ tục Swap này.

```
PROCEDURE Swap (VAR x, y: Integer); VAR temp: Integer;
BEGIN END; temp := x; x := y; y := temp; PROCEDURE Bubble
(VAR a: ARRAY[1..n] OF integer); VAR i,j :Integer; BEGIN
{1} FOR i:=1 TO n-1 DO {2} FOR j:=n DOWNTO i+1 DO {3} IF
a[j-1]>a[j] THEN Swap(a[j-1], a[j]); END;
```

Trong cách viết trên, chương trình Bubble gọi chương trình con Swap, do đó để tính

thời gian thực hiện của Bubble, trước hết ta cần tính thời gian thực hiện của Swap.

Dễ thấy thời gian thực hiện của Swap là O(1) vì nó chỉ bao gồm 3 lệnh gán. Trong

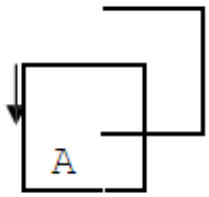
Bubble, lệnh {3} gọi Swap nên chỉ tốn O(1), lệnh {2} thực hiện n-i lần, mỗi lần tốn

O(1) nên tốn O(n-i). Lệnh {1} thực hiện n-1 lần nên:

$$T(n) = \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2} = O(n^2).$$

Phân tích các chương trình Đệ quy

Với các chương trình có gọi các chương trình con đệ quy, ta không thể áp dụng cách tính như vừa trình bày trong mục 1.5.4 bởi vì một chương trình đệ quy sẽ gọi chính bản thân nó. Có thể thấy hình ảnh chương trình đệ quy A như sau:



Hình 2- 2: Sơ đồ chương trình con A đệ quy

Với phương pháp tính độ phức tạp đã trình bày trong mục 1.5.4 thì không thể thực hiện được. Bởi vì nếu theo phương pháp đó thì, để tính thời gian thực hiên của chương trình A, ta phải tính thời gian thực hiện của chương trình A và cái vòng luẩn quẩn ấy không thể kết thúc được.

Với các chương trình đệ quy, trước hết ta cần thành lập các phương trình đệ quy, sau đó giải phương trình đệ quy, nghiệm của phương trình đệ quy sẽ là thời gian thực hiện của chương trình đệ quy.

Thành lập phương trình đệ quy

Phương trình đệ quy là một phương trình biểu diễn mối liên hệ giữa T(n) và T(k), trong đó T(n) là thời gian thực hiện chương trình với kích thước dữ liệu nhập là n, T(k) thời gian thực hiện chương trình với kích thước dữ liệu nhập là k, với k < n. Để thành lập được phương trình đệ quy, ta phải căn cứ vào chương trình đệ quy.

Thông thường một chương trình đệ quy để giải bài toán kích thước n, phải có ít nhất một trường hợp dừng ứng với một n cụ thể và lời gọi đệ quy để giải bài toán kích thước k (k<n).

Để thành lập phương trình đệ quy, ta gọi T(n) là thời gian để giải bài toán kích thước n, ta có T(k) là thời gian để giải bài toán kích thước k. Khi đệ quy dừng, ta phải xem xét khi đó chương trình làm gì và tốn hết bao nhiêu thời gian, chẳng hạn thời gian này là c(n). Khi đệ quy chưa dừng thì phải xét xem có bao nhiêu lời gọi đệ quy với kích thước k ta sẽ có bấy nhiêu T(k). Ngoài ra ta còn phải xem xét đến thời gian để phân chia bài toán và tổng hợp các lời giải, chẳng hạn thời gian này là d(n).

Dạng tổng quát của một phương trình đệ quy sẽ là:

$$T(n) = \begin{cases} C(n) \\ F(T(k)) + d(n) \end{cases}$$

Trong đó C(n) là thời gian thực hiện chương trình ứng với trường hợp đệ quy dừng. F(T(k)) là một đa thức của các T(k). d(n) là thời gian để phân chia bài toán và tổng hợp các kết quả.

Xét hàm tính giai thừa viết bằng giải thuật đệ quy như sau:

```
FUNCTION Giai_thua(n:Integer): Integer; BEGIN END; IF n=0
then Giai_thua :=1 ELSE Giai_thua := n* Giai_thua(n-1);
```

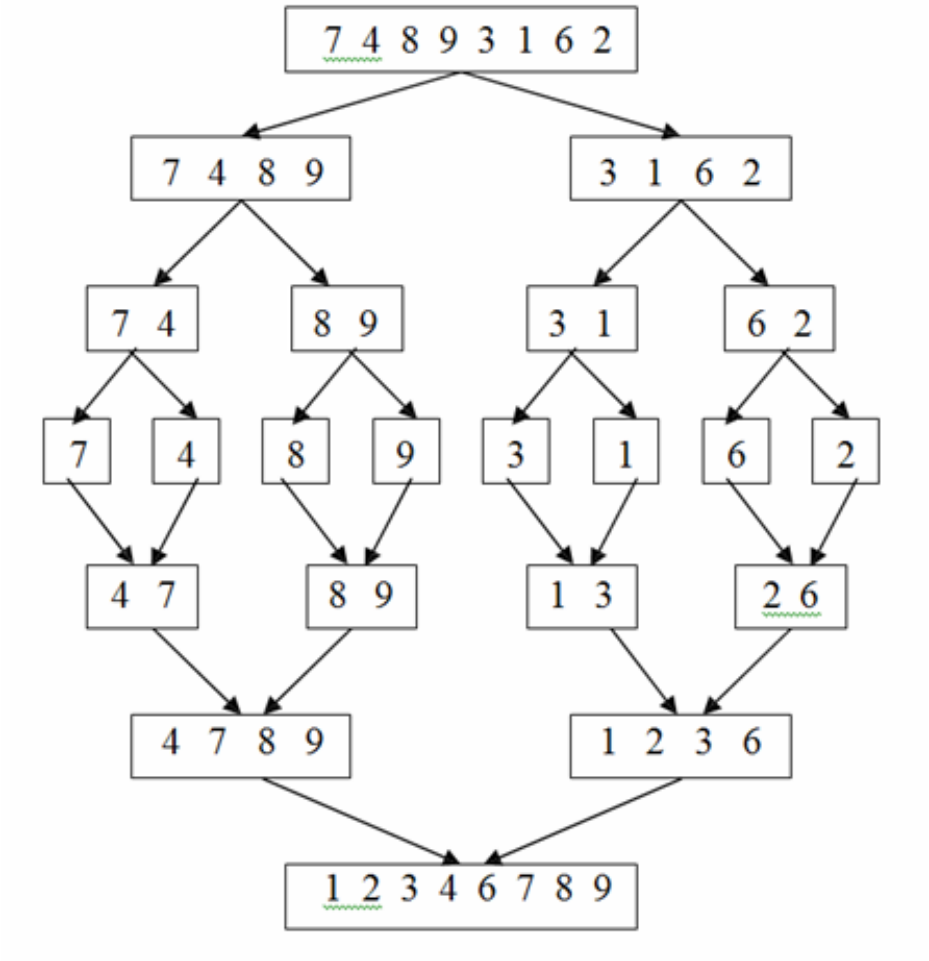
Gọi T(n) là thời gian thực hiện việc tính n giai thừa, thì T(n-1) là thời gian thực hiện việc tính n-1 giai thừa. Trong trường hợp n = 0 thì chương trình chỉ thực hiện một lệnh gán Giai_thua:=1, nên tốn O(1), do đó ta có T(0) = C1. Trong trường hợp n>0 chương trình phải gọi đệ quy Giai_thua(n-1), việc gọi đệ quy này tốn T(n-1), sau khi có kết quả của việc gọi đệ quy, chương trình phải nhân kết quả đó với n và gán cho Giai_thua. Thời gian để thực hiện phép nhân và phép gán là một hằng C_2. Vậy ta có

$$T(n) = \begin{cases} C_1 & \text{nêu } n = 0 \\ T(n-1) + C_2 & \text{nêu } n > 0 \end{cases}$$

Đây là phương trình đệ quy để tính thời gian thực hiện của chương trình đệ quy Giai_thua.

Chúng ta xét thủ tục MergeSort một cách phác thảo như sau:

```
FUNCTION MergeSort (L:List; n:Integer):List; VAR
L1,L2:List; BEGIN IF n=1 THEN RETURN(L) ELSE BEGIN Chia
đôi L thành L1 và L2, với độ dài n/2;
RETURN(Merge(MergeSort(L1,n/2),MergeSort(L2,n/2))); END;
END;
```



Hình 2- 3: Minh hoạ sắp xếp trộn

Chẳng hạn để sắp xếp danh sách L gồm 8 phần tử 7, 4, 8, 9, 3, 1, 6, 2 ta có mô hình minh họa của MergeSort như sau:

Hàm MergeSort nhận một danh sách có độ dài n và trả về một danh sách đã được sắp xếp. Thủ tục Merge nhận hai danh sách đã được sắp L1 và L2 mỗi danh sách có độ dài n/2 trộn chúng lại với nhau để được một danh sách gồm n phần tử có thứ tự. Giải thuật chi tiết của Merge ta sẽ bàn sau, chúng ta chỉ để ý rằng thời gian để Merge các danh sách có độ dài n/2 là O(n). Gọi T(n) là thời gian thực hiện MergeSort một danh sách n phần tử thì T(n/2) là thời gian thực hiện MergeSort một danh sách n/2 phần tử. Khi L có độ dài 1 (n = 1) thì chương trình chỉ làm một việc duy nhất là return(L), việc này tốn O(1) = C1 thời gian. Trong trường hợp n > 1, chương trình phải thực hiện gọi đệ quy MerSort hai lần cho L1 và L2 với độ dài n/2 do đó thời gian để gọi hai lần đệ quy này là 2T(n/2). Ngoài ra còn phải tốn thời gian cho việc chia danh sách L thành hai nửa bằng nhau và trộn hai danh sách kết quả (Merge). Người ta xác đinh được thời gian để chia danh sách và Merge là O(n) = C_2n. Vậy ta có phương trình đệ quy như sau:

$$T(n) = \begin{cases} C_1 & \text{nêu } n = 1 \\ 2T(\frac{n}{2}) + C_2\ n & \text{nêu } n > 1 \end{cases}$$

Các hàm tiến triển khác

Trong trường hợp hàm tiến triển không phải là một hàm nhân thì chúng ta không

thể áp dụng các công thức ứng với ba trường hợp nói trên mà chúng ta phải tính trực tiếp nghiệm riêng, sau đó so sánh với nghiệm thuần nhất để lấy nghiệm lớn nhất trong hai nghiệm đó làm nghiệm của phương trình.

Vídụ2-17: Giải phương trình đệ quy sau: T(1) = 1

T(n) = 2T(n/2) + nlogn

Phương trình đã cho thuộc dạng phương trình tổng quát nhưng d(n) = nlogn không phải là một hàm nhân.

b Ta có nghiệm thuần nhất = nlog a = nlog2 = n

Do d(n) = nlogn không phải là hàm nhân nên ta phải tính nghiệm riêng bằng cách xét trực tiếp

$$\text{Nghiệm riêng} = \sum_{j=0}^{k-1} a^j d(b^{k-j}) = \sum_{j=0}^{k-1} 2^j 2^{k-j} \log 2^{k-j} = 2k \sum_{j=0}^{k-1} (k - j) = 2^k \frac{k(k+1)}{2} = O(2^k k^2)$$

Theo giả thiết trong phương trình tổng quát thì n = bk nên k = logbn, ở đây do b=2

nên 2_k=n và k=logn, chúng ta có nghiệm riêng là O(nlog2n), nghiệm này lớn hơn

nghiệm thuần nhất do đó T(n) = O(nlog2n).

Bài tập chương

Tính thời gian thực hiện của các đoạn chương trình sau:

a) Tính tổng của các số

{1} Sum := 0;

{2} for i:=1 to n do begin

{3} readln(x);

{4} Sum := Sum + x;

end;

b) Tính tích hai ma trận vuông cấp n C = A*B:

{1} for i := 1 to n do

{2} for j := 1 to n do begin

{3} c[i,j] := 0;

{4} for k := 1 to n do

{5} c[i,j] := c[i,j] + a[i,k] * b[k,j];

end;

Dành cho độc giả

Giải các phương trình đệ quy sau với T(1) = 1 và

a) $T(n) = 3T(n/2) + n$ b) $T(n) = 3T(n/2) + n_2$ c) $T(n) = 8T(n/2) + n_3$

Dành cho độc giả

Giải các phương trình đệ quy sau với T(1) = 1 và a) $T(n) = 4T(n/3) + n$

b) $T(n) = 4T(n/3) + n_2$.

c) $T(n) = 9T(n/3) + n_2$.

Dành cho độc giả

Giải các phương trình đệ quy sau với T(1) = 1 và a) $T(n) = T(n/2) + 1$

b) $T(n) = 2T(n/2) + \log n$ c) $T(n) = 2T(n/2) + n$

d) $T(n) = 2T(n/2) + n_2$

Dành cho độc giả

Giải các phương trình đệ quy sau bằng phương pháp đoán nghiệm:

a) T(1) = 2 và T(n) = 2T(n-1) + 1 với n > 1

b) T(1) = 1 và T(n) = 2T(n-1) + n với n > 1

Dành cho độc giả

Cho một mảng n số nguyên được sắp thứ tự tăng. Viết hàm tìm một số nguyên trong mảng đó theo phương pháp **tìmkiếmnhị phân**, nếu tìm thấy thì trả về TRUE, ngược lại trả về FALSE. Sử dụng hai kĩ thuật là đệ quy và vòng lặp. Với mỗi kĩ thuật hãy viết một hàm tìm và tính thời gian thực hiện của hàm đó.

Dành cho độc giả

Tính thời gian thực hiện của giải thuật đệ quy giải bài toán Tháp Hà nội với n tầng?

Dành cho độc giả

Xét công thức truy toán để tính số tổ hợp chập k của n như sau:

a) Viết một hàm đệ quy để tính số tổ hợp chập k của n.

b) Tính thời gian thực hiện của giải thuật nói trên.

Dành cho độc giả