**Sơ đồ chung của phương pháp**

**Bài toán ví dụ**

Giả sử ta có thuật toán α để giải bài toán kích thước dữ liệu với thời gian bị chặn bởi cn2. Xét thuật toánβđểgiải chính bài toán đó bằng cách

* Bước 1 : Chia bài toán cần giải ra thành 3 bài toán con với kích thước n/2
* Bước 2 : Giải 3 bài toán con bằng thuật toán α
* Bước3 : Tổng hợplời giải của 3 bài toán con để thu được lời giải của bài toán

Giả sử bước 3 đượcthực hiện với thời gian d.n

Gọi

*T*

*m*

(

*n*

)

 : thời gian của thuật toánα

*T*

*p*

(

*n*

)

: thời gian của thuật toán β

Khi đó

*T*

*m*

(

*n*

)

 = cn2

*T*

*p*

(

*n*

)

 =3

*T*

*m*

(

*n*

)

 +dn= cn2 + dn

Nên nếu dn<cn2/4 (d<cn/4) thì thuật toán β nhanh hơn α . Điều này luôn đúng với nđủ lớn.

Tuy nhiên ta thấy thuật toán β mới chỉ thay đổi được nhân tử hằng số chưa thay đổiđược bậc nhưng cũng hiệu quả khi n lớn. Do đó, nếu ta tiếp tục chia bài toán con nhỏ nữa tới n0≤ 4 d/c ta sẽ thu được một thuật toán hiệu quả hơn.

Xét thuật toán sau :

ProcedureGamma(n)(\* n kích thước bài toán\*) Begin

If n ≤ n0 Then

Giải bài toán bằng thuật toán α

Else

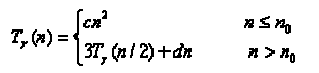
Begin

End

End;

1. Chia bài toán thành ba bài toán con kích thước n/2
2. Giải mỗi bài toán con bằng thuật toán Gamma
3. Tổng hợp lời giải của các bài toán con

Gọi T (n) là thời gian tính của thuật toán trên, và thời gian tổng hợp lời giải của các bài toán con làΟ (n) thì



http://voer.edu.vn/file/25802 Theo định lý thợ ta có

http://voer.edu.vn/file/25810

Thuật toán thu được có thời gian tính là tốt hơn cả thuật toánαvà thuật toánβ .Hiệu quả thu được trong thuật toánγcó được là nhờ ta đã triệt để khai thác hiệu quả việcsử dụng thuật toán β .

Để có được mô tả chi tiết thuật toán chiađể trị chúng ta cần phải xác định :

1. Kích thước tới hạn n0 (Bài toán có kích thước nhỏ hơn n0 sẽ không cần chia nhỏ)
2. Kích thước của mỗi bài toán con trong cách chia
3. Số lượng các bài toán con như vậy
4. Thuật toán tổng hợp lời giải của các bài toán con

Các phần xác định trong 2 và 3 phụ thuộc vào 4. Chia như thế nào để khi tổng hợp có hiệu quả (thường là tuyến tính)

**Sơ đồ thuật toán tổng quát**

Procedure D\_and\_C(n) Begin

If n ( n0 Then

Giải bài toán một cách trực tiếp

Else

1. Chia bài toán thành r bài toán con kích thước n/k
2. For (Mỗi bài toán trong r bài toán con) Do D\_and\_C(n)

End;

1. Tổng hợp lời giải của r bài toán con đểthu được lời giải của bài toán gốc

**Thuật toán tìm kiếm nhị phân**

**Bài toán**: Cho mảng x[1..n] được sắp xếp theo thứ tự không giảm và y. Tìm i sao cho x[i] = y. (Giả thiết i tồn tại).

**Phân tích giải thuật:**

Số y cho trước

* Hoặc là bằng phần tử nằm ở vị trí giữa mảng x
* Hoặc là nằm ở nửa bên trái (y < phần tử ở giữa mảng x )
* Hoặc là nằm ở nửa bên phải (y < phần tử ở giữa mảng x ) Từ nhận xét đó ta có giải thuật sau

Function Bsearch(x[1..n],Start,Finish) Begin

Middle := (Start + Finish)/2;

If (y = x[Middle]) then return middle

Else

If ( y < x[Middle] then return Bsearch(x,Start,Middle-1) Else

End;

Return Bsearch (x,Middle+1,Finish)

Phân tích hiệu quả thuật toán : T(n)

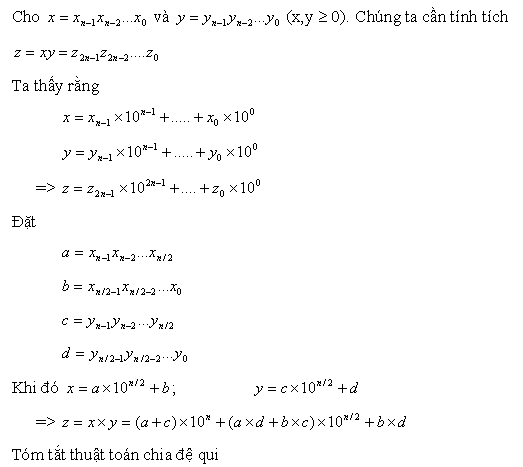
http://voer.edu.vn/file/25794

Theo định lý thợ ta có

http://voer.edu.vn/file/25805

**Phép nhân các số nguyên lớn**

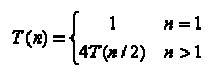
Xét lại vấn đề nhân các số nguyên lớn. Nhớ lại rằng thuật toán cổ điển mà phần lớn chúng ta đều được học ở trường đòi hỏi thời gian tính là Ρ(n2) để nhân các số nguyên m có n chữ số. Chúng ta cũng quen với thuật toán này đến mức có thể còn chẳng bao giờ thắc mắc về tính tối ưu của nó. Liệu chúng ta có thể làm được tốt hơn không? . Một thuật toán được bàn đến gọi là kỹ thuật Chia để trị bao gồm việc rút gọn phép nhân hai số nguyên n chữ số xuống thành bốn phép nhân hai số nguyên n/2 chữ số.



-Việc nhân 2 số nguyên có 1 chữ số có thể thực hiện một cách trực tiếp (neo đệ qui), thời gian thực hiện làO(1)

-Chia : n >1 thì tích củahai số nguyên có n chữ số có thể biểu diễn qua tích 4 số nguyên có n/2 chữ số,thời gian thực hiện là 4. T ( n /2) ( trong đó T(n) là thời gian thực hiện nhân hai số nguyên cón chữ số ).

-Tổng hợp : Cộng và dịch phải, khi đóthời gian thực hiện sẽ làΟ( n ) Khi đó ta có thời gian thực hiệnthuật toán là



Theo định lý thợ ta có độ phức tạp của thuật toán là

*T*

*n*

=

(On2) .Như vậy thuật toán thu được cũng không gặt hái được bất kỳ cải thiện nào so với thuật toán nhân cổ điển mặc dù chúng ta đã khôn ngoan hơn. Để vượt được thuậttoán cổ điển và như vậy mới hoàn toàn thấy rõ được công dụng của phép Chia để trị, chúng ta phải tìm cách rút gọn phép nhân nguyên thuỷ không phải về bốn mà làbaphép nhân hai nửa.

Chúng ta minh hoạ quá trình này bằng việc nhân 981 với 1234. Trước tiên chúng ta điền thêm vào toán hạng ngắn hơn một số không vô nghĩa để làm cho haitoán hạng có cùng độ dài, vậy là 981 trở thành 0981. Sau đó tách từng toán hạngthành hai nửa: 0981 cho ra w = 09 và x = 81, còn 1234 thành y = 12 và z= 34.Lưu ý rằng 981 = 102w + x và 1234 = 102y + z. Do đó, tíchcần tìm có thể tính được là 981 x 1234 =(102w + x)( 102y + z) = 104wy + 102(wz + xy) +xz = 1080000 + 127800 + 2754 =1210554

Thủ tục trên đến bốn phép nhân hai nửa:wy, wz, xy và xz.

Để ý điểm mấu chốt ở đây là thực ra thì không cần tính cả wz lẫn xy, mà là tổng của hai số hạng này. Liệu có thể thu được wz + xy với chi phí của một phép nhân mà thôi hay không? Điều này có vẻ nhưkhông thể được cho đến khi chúngta nhớ ra rằngmình cũng cần những giá trị wy và xz để đưa vào công thức trên. Lưu ý về điểmnày, hãy xét tích:

r = (w + x)(y+z) = wy +(wz + xy) + xz

Chỉ sau một phép nhân, chúng ta thu đượctổng của tất cả ba số hạng cần thiết để tính được tích mình mong muốn. Điều này gợi ý một cách tiến hành như sau:

p = wy = 09 \* 12 =108

q = xz = 81 \* 34 =2754

r = (w + x)(y+z) = 90 \*46 = 4140

và cuối cùng

981 x 1234 =104p + 102(r – p – q) + q = 1080000 + 127800 + 2754 =1210554.

Như vậy tích của 981 và 1234 có thể rútgọn về ba phép nhân của hai số có hai chữ số

(09 12, 81 34 và 90 46) cùng với một số nào đó phép dịch chuyển (nhân với luỹ thừa của 10),phép cộng và phép trừ.

Chắc chắn là số các phép cộng – coi phéptrừ như là phép cộng – có nhiều hơn so với thuật toán Chia để trị nguyên thuỷ ởphần trước. Vậy thì có đáng để thực hiện bốn phép cộng nhiều hơn để tiết kiệmmột phép nhân hay không? Câu trả lời là không nếu chúng ta đang nhân số nhỏ như những số trong ví dụ này. Tuy nhiên sẽ là đáng giá nếu các số cần được nhân vớinhau đủ lớn và chúng càng lớn thì lại càng đáng làm như vậy. Khi các số hạng đủlớn, thời gian cần cho các phép cộng và dịch chuyển trở thành bỏ qua được so vớithời gian cần cho chỉ một phép nhân. Như vậy là có lý do để kỳ vọng rằng rút gọn bốn phép nhân về còn ba sẽ giúp chúng ta cắt giảm được 25% thời gian tính toán đòi hỏi cho việc nhân các số lớn. Như chúng ta sẽ thấy, sự tiết kiệm của mình sẽtốt hơn một cách đáng kể.

Để giúp chúng ta hiểu thấu đượcnhững gì mình đạt được, hãy giả thiết rằng có một cài đặt của thuật toán nhân cổđiển đòi hỏi thời gian h(n) = cn2để nhân hai số có n chữ số, với hằng số c phụ thuộc vàocài đặt đó. (ở đây đã có sự đơn giản hoá vì trên thực tế thì thời gian đòi hỏicòn có dạng phức tạp hơn, chẳng hạn như cn2+ bn + a). Tương tự, chog(n) là thời gian mà thuật toán Chia để trị cần để nhân haisố n chữ số,không tính thời gian cần thiết để thực hiện ba phép nhân hai nửa. Nói cách khác,g(n) là thời gian cần thiết cho các phép cộng, dịch chuyển và các phép tính phụthêm khác. Dễ dàng cài đặt các phép tính này sao cho g(n)∈Ρ(n).Hãy tạm thời bỏ qua điều gì sẽ xảy ra nếu n lẻ và nếu các số hạng không có cùngđộ dài.

Nếu từng trong số ba phép nhân hai nửađược thực hiện bằng thuật toán cổ điển, thời gian cần thiết để nhân hai số có nchữ số là: 3h(n/2) + g(n) =3c(n/2)2+ g(n) = cn2+ g(n)= h(n) +g(n).

Vì h(n) rất nhỏ xo với Ρ(n2)và g(n) rất nhỏ xo với Ρ(n),số hạng g(n) là bỏ qua được so với h(n)khi n đủ lớn, có nghĩa là chúng ta tăng được tốc độ lên khoảng 25% so với thuậttoán cổ điển như đã mong đợi. Mặc dù sự cải thiện này là không thể xem thường được nhưng chúng ta vẫn không làm được thay đổi bậc của thời gian cần thiết:thuật toán mới vẫn cần thời gian tính bậc hai.

Để có thể làm được tốt hơn thế, chúng tatrở lại với câu hỏi đặt ra ở đoạn mở đầu: các bài toán con cần được giải như thế nào? Nếu chúng nhỏ thôi thì thuật toán cổ điển có vẫn còn là cách làm tốt nhất. Tuy nhiên, khi những bài toán con cũng đủ lớn, chẳng lẽ sử dụng thuật toánmới của chúng ta một cách đệ quy cũng không hơn gì hay sao? Ý tưởng này tương tựnhư hưởng lợi nhuận từ một tài khoản ngân hàng có gộp vốn lẫnlãi! Nếu chúng talàm như vậy sẽ thu được một thuật toán có thể nhân hai số n chữ số trong mộtthời gian t(n) = 3t(n/2) + g(n) khi nchẵn và đủ lớn. Điều này cũng giống như phép truy toán (đệ quy) ; giải ra tathu được t(n)∈O(nlg3) | n là luỹ thừa của 2. Chúng ta cần phải bằng lòng với lờighi chú tiệm cận có điều kiện vì chưa đề cập đến câu hỏi là nhân các số có độdài là lẻ như thế nào.

Vì lg3 = 1.585 nhỏ hơn 2, thuật toán này có thể nhân hai số nguyên lớn nhanh hơn rất nhiều so với thuật toán nhân cổ điểnvà n càng lớn thì sự cải thiện này càng đáng giá. Mộ cài đặt tốt có thể khôngsử dụng cơ số 10, mà sử dụng cơ số lớn nhất để với cơ số đó phần cứng cho phép nhân trực tiếp hai “chữ số” với nhau.

Một nhân tố quan trọng trong hiệu suất thực tế của cách tiếp cận phép nhân này và của bất kỳthuật toán Chia để trị nào là biết khi nào cần dừng việc phân chia các bài toánvà thay vào đó sử dụng thuật toán cổ điển. Mặc dù cách tiếp cận Chia để trị trởnên có ích khi bài toán cần giải đủ lớn, trên thực tế nó có thể chậm hơn so vớithuật toán cổ điển đối với những bài toán quá nhỏ. Do đó thuật toánChia để trị phải tránh việc thực hiện đệ quy khi kích thước của các bài toán con không phù hợp nữa. Chúng ta sẽ trở lại vấn đề này ở phần sau.

Để đơn giản, một số vấn đề quan trọng đến nay đã bị bỏ qua. Làm thế nào để chúng ta giải quyết được những số có độ dài lẻ?Mặc dù cả hai nửa của số nhân và số bị nhân đều có kích thước n/2, có thể xảy ratrường hợp tổng của chúng bị tràn và có kích thước vượt quá 1. Do đó sẽ không hoàn toàn chính xác khi nói rằng r = (w+x)(y+z) bao hàm phép nhân hai nửa. Điềunày ảnh hưởng tới việc phân tích thời gian chạy như thế nào? Làm thế nào để nhânhai số có kích thước khác nhau? Còn những phép tính số học nào khác với phépnhân mà ta có thể xử lý hiệu quả hơn so với dùng thuật toán cổ điển?

Những số có độ dài lẻ được nhân dễ dàng bằng cách tách chúng càng gần ở giữa càng tốt: một số có n chữ số được táchthành một số có |n/2| chữ số và một số có |n/2| chữ số. Câu hỏi thứ hai còn khắtkhe hơn. Xét nhân 5678 với 6789. Thuật toán của chúng ta tách các số hạng thànhw = 56, x = 78, y = 67 và z = 89. Ba phép nhân hai nửa cần thực hiện là:

p = wy = 56.67

q = xz = 78.89 và

r = (w+x)(y+z) =134.156

Phép nhân thứ ba bao gồm những số 3 chữ số, do vậy nó không thực sự là một nửa so với phép nhân nguyên thuỷ của các sốcó 4 chữ số. Tuy nhiên kích thước của w+x và

y+z không thể vượt quá 1 +|n/2|.

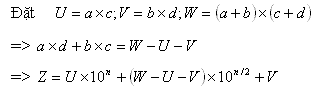
Để đơn giản hoá việc phân tích, cho t(n)là thời gian mà thuật toán này thực hiện trong tìn huống xấu nhất để nhân haisố có kích thước tối đa là n (thay vì chính xác bằng n).

Theo định nghĩa thì t(n) là một hàm không giảm.Khi n đủ lớn thuật toán của chúng ta rút gọn phép nhân hai số có kích thước tối đa n đó về ba phép nhân nhỏ hơn p = wy, q = xz và r = (w+x)(y+z) với kích thước tối đa tương ứng là |n/2|, |n/2| và 1 + |n/2|, thêm vào đó là những thao tác đơn giản chiếmthời gian là O(n). Do đó ở đây tồn tại hằng số dương c saocho: t(n) = t(|n/2|) +t(|n/2|) + t(1+|n/2|) + cn với mọi n đủ lớn. Điều này chính xác là phép đệ quy mà chúng ta đã nghiên cứu cho kết quả giờ đây đã trở nên quen thuộc là t(n)∈O( nlg 3). Do vậy luôn luôn có thể nhân các số n chữ số với thờigian O(nlg3). Phân tích tình huống tồi nhất của thuật toán này chỉ rarằng trên thực tế t(n) Ρ(nlg3),nhưng điều này không được quan tâm lắm vì còn có những thuật toán nhân nhanh hơn.

Quay lại với câu hỏi nhân các số có kích thước khác nhau, giả sử u và v là những số nguyên có kích thước tương ứng là mvà n. Nếu m và n nằm trong khoảng đến hai lần của nhau, tốt nhất là điền vào sốhạng nhỏ hơn những số 0 vô nghĩa để làm cho nó có cùng độ dài như số hạng kia,như chúng ta đã làm khi nhân 981 với 1234. Tuy nhiên cách tiếp cận này khôngđược khuyến khích khi một số hạng lớn hơn số hạng kia rất nhiều. Thậm chí nó có thể tồi hơn là dùng thuật toán nhân cổđiển! Không làm mất đi tính tổng quát, giả sử rằng m≥n.Thuật toán Chia để trị sử dụng điền số và thuật toán cổ điển có thời gian tươngứng là Ρ(nlg3) và Ρ(mn) để tính các tích u và v. Xét thấy rằng hằng số Nn của biểu thức trước có vẻ lớnhơn của biểu thức sau, chúng ta thấy rằng Chia để trị sử dụng điền số là chậmhơn thuật toán cổ điển khi m = nlg(3/2)và như vậy trường hợp đặc biệt khi m = n.Mặc dù vậy rất dễ dàng kết hợp cảhai thuật toán để thu được một thuật toán thực sự tốt hơn. ý tưởng là cắt lát sốhạng dài hơn v thành những đoạn có kích thước m và sử dụng thuật toán Chia đểtrị để nhân u với từng đoạn của v sao cho thuật toán Chia để trị được dùng đểnhân những cặp số hạng có cùng kích thước. Tích cuối cùng của u và v sau đó thuđược dễ dàng bằng các phép cộng và dịch chuyển đơn giản. Thời gian chạy tổngcộng chủ yếu được dùng để thực hiện |n/m| phép nhân các số m chữ số. Vì mỗi phépnhân nhỏ hơn này chiếm thời gian Ρ(mlg3) và vì |n/m|∈Ρ(n/m) ,thời gian chạy tổng cộng để nhân một số n chữ số với một số m chữ số làΡ(nmlg(3/2))khi m = n.

Sau đây là mô hình cải tiến thuật toán nhân số nguyên lớn

Cải tiến để còn lại 3 phép nhân :



Từ đó ta đưa ra thuật toán nhân số nguyên lớn là

Function Karatsuba(x,y,n); Begin

If n = 1 then Return x[0]\*y[0] Else

Begin

a := x[n-1].. . x[n/2];b := x[n/2-1] . . .x[0];

c := y[n-1]. .. y[n/2];d := y[n/2-1] . . .y[0]; U :=Karatsuba(a,c,n/2);

V :=Karasuba(b,d,n/2);

W :=Karatsuba(a+b,c+d,n/2); Return U\*10n+ (W-U-V)\*10n/2+ V end

End;

Phân tích hiệu quả thuật toán : T ( n )

T (1) =1

T ( n ) = 3 T ( n /2) + cn

=>Theo định lý thợ T( n ) =Ρ(nlog 3)

**Một số giải thuật sắp xếp**

Cho T[1..n] là một mảng n phần tử. Vấn đề đặt ra là sắp xếp các phần tử này theo thứ tự tăng. Chúng ta đã có thể giải quyết vấn đề này bằng các phương phápselection sort hay insertion sort hoặc là heapsort

Như chúng ta đã biết thời gian dùng selection sort hay insertion sort để sắp xếp mảng T trong cả hai trường hợp: xấu nhất và trung bình đều vào cỡ n2. Còn heapsort vào khoảng nlogn.

Có một số giải thuật đặc biệt cho bài toán này theo mô hình chia để trị đó làmergesortvà quicksort, chúng ta sẽ lần lượt đi nghiên cứu chúng.

**MergeSort**

Chia để trị tiếp cận tới bài toán này bằng việc tách mảng T thành hai phần màkích thước của chúng sai khác nhau càng ít càng tốt, sắp xếp các phần này bằng cách gọi đệ qui và sau đó trộn chúng lại (chú ý duy trì tính thứ tự). Để làm được điều này chúng ta cần một giải thuật hiệu quả cho việc trộn hai mảng đã được sắp U và V thành một mảng mới T mà kích thước của mảng T bằng tổng kích thước của hai mảng U và V.

Vấn đề này có thể thực hiện tốt hơn nếu ta thêm vào các ô nhớ có sẵn ở cuối của mảng U và V các giá trị đứng canh (giá trị lớn hơn tất cả các giá trị trong U và V) .

Procedure merge(U[1..m+1],V[1..n+1],Ta[1..m+n]);

(\*Trộn 2 mảng U[1..m+1] và V[1..n+1] thành mảng T[1..m+n]); U[m+1],V[n+1] được dùng để chứa các giá trị cầm canh\*)

Begin i:=1;j:=1;

U[m+1]:= ∞ ; V[n+1]:= ∞ ;

For k:=1 to n+m do

If U[i]<V[j] then

Begin

T[k]:=U[i];

i:=i+1;

Else

End

Begin

T[k]:=V[j];

j:=j+1;

End;

End;

Giải thuật sắp xếp trộn sau đây có thể tốt hơn nếu các mảng U và V là các biến toàn cục và xem việc sắp xếp chèn Insert(T) như là giải thuật cơ bản

Procedure mergesort(T[1..n]);

Begin

If n đủ nhỏ then Insert(T)

Else

Begin

Array U[1..1+],V[1..1+; U[1..]:=T[1..];

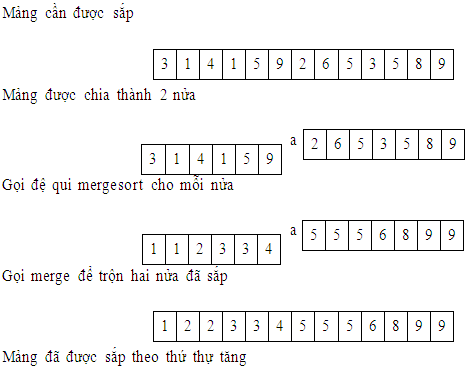
V[1..]:=T[1+..n]; mergesort(U[1..]);

End;

mergesort(V[1..]);

merge(U,V,T);

Hình sau chỉ ra các bước của mergesort.



Giải thuật sắp xếp này minh hoạ tất cả các khía cạnh của chia để trị. Khi số lượng các phần tử cần sắp là nhỏ thì ta thường sử dụng các giải thuật sắp xếp đơn giản.

Khi số phần tử đủ lớn thì ta chia mảng ra 2 phần, tiếp đến trị từng phần một và cuối cùng là kết hợp các lời giải.

Giả sử t(n) là thời gian cần thiết để giải thuật này sắp xếp một mảng n phần tử. Việc tách T thành U và V là tuyến tính. Ta cũng dễ thấy merge(U,V,T) cũng tuyến tính. Do vậy:

t(n)=t([n/2]) + t([n/2]) + g(n)

trong đó g( n ) = O( n )

hay t( n )=2t( n /2)+g( n )

Theo định lý chủ

Tacó: l=2; b=2và k=1

Nên t(n)= Ρ (nlogn), vì bk =l

Như vậy hiệu quả của mergesort tương tự heapsort. Trong thực tế sắp xếp trộn có thể nhanh hơn heapsort một ít nhưng nó cần nhiều hơn bộ nhớ cho các mảng trung gian U và V. Ta nhớ lại heapsort có thể sắp xếp tại chỗ (in-place), và cảm giác nó chỉ sử dụng một ít biến phụ mà thôi. Theo lý thuyết mergesort cũng có thể làm được như vậy tuy nhiên giá thành có tăng một chút ít.

Khi giải bài toán theo thuật giải chia để trị chúng ta hết sức chú ý đến việc tạo ra các bài toán con, nếu không nó có thể tạo ra những thảm hại mà không thể lường trước được. Giải thuật sau đây minh hoạ tính chất quan trọng này khi kích thước bài toán con là hỗn độn:

Procedure badmergesort(T[1..n]);

Begin

If n đủ nhỏ then Insert(T) Else

Begin

Array U[1..n-1, V[1..2]; U[1..n-1]:= T[1..n-1]; V[1]:= T[n]; badmergesort(U[1..n-1]); badergesort(V[1..1]);

merge(U,V,T);

End;

Gọi

|  |
| --- |
| ˆ  t |
|  |

(n)

là thời gian cần để sắp n phần tử với giải thuật badmergesort trên. Rõ ràng là:

|  |
| --- |
| ˆ  t |
|  |

 (n) =

|  |
| --- |
| ˆ  t |
|  |

 (n-1) +

|  |
| --- |
| ˆ  t |
|  |

 (1) +

|  |
| --- |
| ˆ  g |
|  |

 (n), trong đó (n) ∈ Ρ(n).

Sự đệ qui này tạo ra (n) ∈ Ρ(n2), như vậy việc quên cân bằng kích thước của bài toán con đã ảnh hưởng đáng kể đến hiệu quả của việc sử dụng giải thuật chia đểtrị.

**Quicksort**

Giải thuật này được phát minh bởi Hoare, nó thường được hiểu như là tên gọi của nó - sắp xếp nhanh, hơn nữa nó cũng dựa theo nguyên tắc chia để trị. Không giống như mergesort nó quan tâm đến việc giải các bài toán con hơn là sự kết hợp giữa các lời giải của chúng. Bước đầu tiên của giải thuật này là chọn 1 vật trung tâm (pivot) từ các phần tử của mảng cần sắp. Tiếp đến vật trung tâm sẽ ngăn mảng này ra 2 phần: các phần tử lớn hơn vật trung tâm thì được chuyển về bên phải nó, ngược lại thì chuyển về bên trái. Sau đó mỗi phần của mảng được sắp xếp độc lập bằng cách gọi đệ qui giải thuật này. Cuối cùng mảng sẽ được sắp xếp xong. Để cân bằng kích thước của 2 mảng này ta có thể sử dụng phần tử ở giữa (median) như là vật trung tâm . Đáng tiếc là việc tìm phần ở giữa cũng mất 1 thời gian đáng kể. Để giải quyêt điều đó đơn giNn là chúng ta sử dụng 1 phần tử tuỳ ý trong mảng cần sắp như là vậttrung tâm và hi vọng nó làtốt nhất có thể.

Việc thiết kế giải thuật ngăn cách mảng bằng vật trung tâm với thời gian tuyến tính không phải là sự thách đố (có thể làm được). Tuy nhiên điều đó là cần thiết để so sánh với các giải thuật sắp xếp khác như là heapsort. Vấn đề đặt ra là mảng con T[i..j] cần được ngăn bởi vật trung tâm p=T[i]. Một cách làm có thể chấp nhận được là:

Duyệt qua từng phần tử của của nó chỉ một lần nhưng bắt đầu từ hai phía (đầu và cuối mảng). Khi đó khởi tạo k=i; l=j+1, k tăng dần cho đến khi T[k] > p, l giảm dần cho đến khi T[l] ( l. Tiếp đến hoán vị T[k] và T[l]. Quá trình này tiếp tục cho đến khi k ( l. Cuối cùng đổi chổ T[i] và T[l] cho nhau và lúc này ta xác định đúng vị trí của phần tử trung tâm.

Procedure Pivot(T[i..j], var l)

(\* Hoán vị các phần tử trong mảng T[i..j] và cuối cùng trả về giá trị l (1≤l≤j)

sao cho T[l]=p ,T[k]≤p với mọi k (i≤k < l) và T[k] > p với mọi k (l < k≤ j), ở đây

p được khởi tạo là T[i] \*) Begin

p:=T[i]; k:=i; l:=j+1;

repeat k:=k+1; until (T[k] > p) or (k≥j);

repeat l:=l-1; until (T[l] < p);

while (k<l) do

begin

End;

Swap(T[k],T[l]); (\* Đổi chỗ T[k] và T[l] \*)

repeat k:=k+1; until (T[k] > p); repeat l:=l-1; until (T[l]≤p); Swap(T[i],T[l]);

end;

Sau đây là giải thuật sắp xếp với tên gọi là Quicksort dùng để sắp xếp mảng T[1..n]:

Procedure Quicksort(T[i..j]); (\* Sắp xếp theo thứ tự không giảm \*) Begin

if n đủ nhỏ then Insert(T[i..j])

else begin

pivot(T[i..j],l);

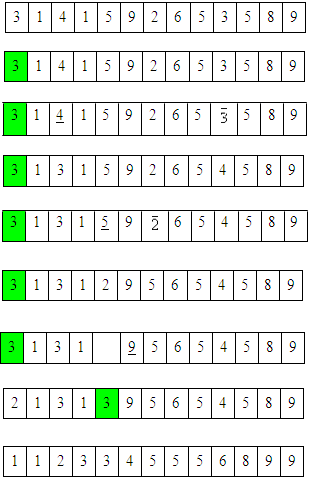
quicksort(T[i..l-1];

quicksort(T[l+1,j];

End;

end;

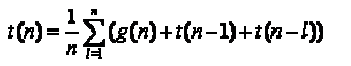
Hình vẽ sau cho thấy sự làm việc của pivot và quicksort.



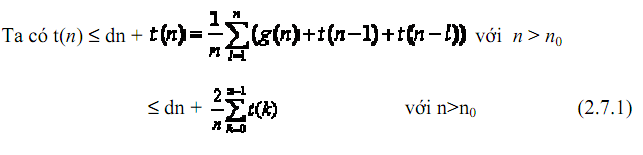
Quicksort sẽ không hiệu quả nếu sử dụng việc gọi đệ quy của các bài toán con mà không chú ý đến sự cân bằng kích thước của chúng. Tình huống xấu nhất là khi T đã được sắp trước mà gọi quicksort và thời gian dùng quicksort để sắp là O( n 2).

Gọi t( n ) là thời gian trung bình dùng quicksort để sắp mảngnphần tử T[1.. n ].llà

giá trị trả về khi gọi pivot(T[1..n],l). Theo pivot() thì l nằm giữa 1 ( n và xác suất là 1/n. Thời gian để tìm vật trung tâm g(n) là tuyến tính. Thời gian để dùng đệ qui để sắp xếp hai mảng con kích thước (l-1) và (n-l) tương ứng là t(n-1) và t(n-l). Như vậy với n đủ lớn ta có:



Hay rõ ràng hơn ta chọnn 0 là giá trị đủ lớn để sử dụng công thức trên. Nghĩa là nếu n n<n 0 thì ta dùng sắp xếp chèn. Khi đó gọi d là hằng số sao cho g( n ) ≤ dn vớin>n 0.



Với công thức như trên quả là khó phân tích độ phức tạp. Ta dự đoán nó tương tự mergesort và hi vọng nó là như vậy tức là vào cỡ O(nlogn). Thật vậy ta có định lý sau:

Định lý: Quicksort cần nlogn thời gian để sắp xếp n phần tử trong trường hợp trung bình.

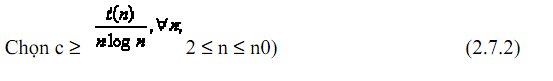
Chứng minh

Gọi t( n ) là thời gian cần thiết để sắp n phần tử trong trường hợp trung bình. a,n 0 là các hằng số giống như công thức 2.7.1

Ta chứng minh t( n ) = cnlogn với mọin≥ 2. với c là hằng số.

Dùng phương pháp qui nạp để chứng minh:

- Với mọinnguyên dương: (2 ≤n≤n) Dễ thấy t( n ) ≤ cnlogn



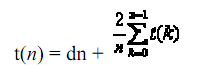
- Bước qui nạp

Ta có

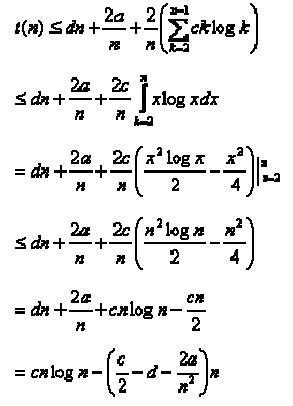
Giả thiết rằng: t(k) ≤ cklogk với mọi 2 ≤ k < n

Ta chỉ ra c sao cho t( n ) ≤ cnlogn

Lấy a = t(0) +t(1)



Theo giả thiết qui nạp : t(k) = cklogk



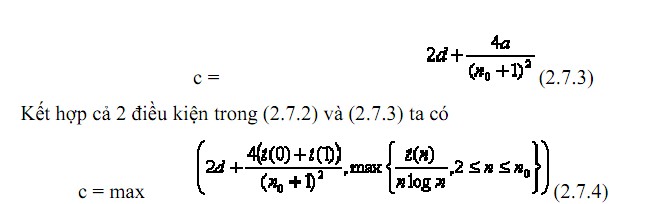
t( n ) = cnlogn với điều kiện là

http://voer.edu.vn/file/25809

,

hay c ≥ 2d + 4a/n2

Từ đó chúng ta chỉ xem xét với những n > n0 thoả mãn:



Hay ta có t( n ) ≤ cnlogn với mọin≥ 2, và như vậy định lý được chứng minh.

Như vậy quicksort có thể sắp xếp 1 mảng n phần tử khác nhau trong trường hợp trung bình là O( nlogn ). Câu hỏi đặt ra là liệu có thể sửa đổi quicksort để nó sắp xếp với thời gian O( nlogn ) trong trương hợp xấu nhất hay không. Câu trả lời là có thể!!!. Tuy nhiên nếu việc tìm phần tử ở giữa của T[i..j] là tuyến tính và lấy nó làm vật trung tâm (pivot) (Finding the median) thì quicksort cũng cần O( n 2) để sắp xếpnphần tử trong trường hợp xấu nhất (khi tất cả các phần tử của mảng cần sắp là bằng nhau). Giải thuật sau đây có thể khắc phục được vấn đề này:

Procedure Pivotbis(T[1..n],p; var k,l);

p - là vật trung tâm

T[1..n] chia ra 3 phần:

T[1..k]: gồm các phần tử < p T[k+1..l-1]: gồm các phần tử = p T[l..n]: gồm các phần tử > p

k,l là các giá trị trả về của Pivotbis(T[i..j],T[i],k,l)

Sau khi ngăn cách mảng T[1..n] bằng việc gọi Pivotbis(T[i..j],T[i],k,l), phần còn lại có thể gọi đệ qui quicksort cho T[1..k] và T[l+1..n].

Với cách sửa đổi này việc sắp mảng trên là tuyến tính. Thú vị hơn là quicksort có thể sắp với thời gian là O( nlogn ) trong trường hợp xấu nhất nếu phần tử ở giữa chọn làm vật trung tâm là tuyến tính. Tuy nhiên chúng ta đưa ra vấn đề này chỉ có tính lý thuyết bởi vì để cải tiến nó thì sẽ tăng sự phức tạp của giải thuật này ở các hằng số Nn (hide constant), thà vậy thì dùng heapsort còn hơn!.

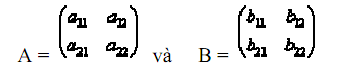
**Bài toán nhân ma trận**

**Bài toán**: Cho hai ma trận A, B với kích thước n\*n, ta có ma trận C chứa kết quả của phép nhân hai ma trận A và B. Thuật toán nhân ma trận cổ điển như công thức dưới đây:

http://voer.edu.vn/file/25798

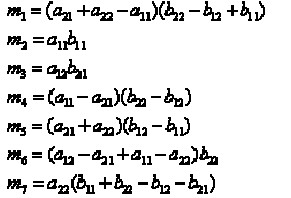
phân tích thuật toán

Với mảng một chiều (kích thước n phần tử), ma trận C được tính trong thời gian O(n), giả sử rằng phép cộng vô hướng và phép nhân là các phép tính cơ bản (có thời gian tính là hằng số). Với mảng hai chiểu (kích thước n\*n) thì thời gian để tính phép nhân ma trận AB là O( n 3)

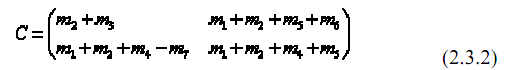


Đến cuối những năm 1960, Strassen đưa ra một giải pháp cải tiến thuật toán trên, nó có tính đột phá trong lịch sử của thuật toán chia để trị, thậm chí gây ngạc nhiên không kém thuật toán nhân số nguyên lớn được phát minh ở thập kỷ trước. ý tưởng cơ bản của thuật toán Strassen cũng tương tự như thuật toán trên. Đầu tiên ta chứng minh rằng phép nhân hai ma trận với kích thước 2\*2 có thể thực hiện được bằng cách sử dụng ít hơn 8 phép nhân vô hướng như thuật toán cổ điển bắt buộc. Ta xét phép nhân hai ma trận A, B như sau:

Ta có các biểu thức sau, mỗi biểu thức chỉ có một phép nhân:



Ta có ma trận C là tích của hai ma trận A và B là:



Do đó ta thấy rằng, có thể nhân hai ma trận kích thước 2\*2 bằng cách chỉ sử dụng 7 phép nhân vô hướng. Nhìn thoáng qua thấy rằng thuật toán này không có gì thú vị lắm, nó sử dụng một số lượng lớn các phép cộng và phép trừ, trong khi thuật toán cổ điển chỉ cần 4 phép cộng.

Bây giờ nếu ta thay thế các phần tử trong A và B bằng các ma trận có kích thước n\*n, thì để thực hiện phép nhân hai ma trận A và B ta phải thực hiện 7 phép nhân hai ma trận với kích thước n\*n và cũng từng đấy phép cộng, trừ của hai ma trận n\*n. Nếu ta làm việc với các ma trận lớn thì phép cộng sẽ nhanh hơn rất nhiều so với phép nhân, việc tiết kiệm 1 phép nhân sẽ lợi hơn nhiều so với việc thực hiện các phép cộng cơ bản.

Gọi t(n) là thời gian cần thiết để nhân hai ma trận kích thước n\*n bằng cách sử dụng đệ quy hai phương trình 2.3.1 và 2.3.2. Giả sử rằng n là luỹ thừa bậc 2. Do thời gian để tình phép cộng, trừ ma trận là

Do đó ta thấy rằng, có thể nhân hai ma trận kích thước 2\*2 bằng cách chỉ sử dụng 7 phép nhân vô hướng. Nhìn thoáng qua thấy rằng thuật toán này không có gì thú vị lắm, nó sử dụng một số lượng lớn các phép cộng và phép trừ, trong khi thuật toán cổ điển chỉ cần 4 phép cộng.

Bây giờ nếu ta thay thế các phần tử trong A và B bằng các ma trận có kích thước n\*n, thì để thực hiện phép nhân hai ma trận A và B ta phải thực hiện 7 phép nhân hai ma trận với kích thước n\*n và cũng từng đấy phép cộng, trừ của hai ma trận n\*n. Nếu ta làm việc với các ma trận lớn thì phép cộng sẽ nhanh hơn rất nhiều so với phép nhân, việc tiết kiệm 1 phép nhân sẽ lợi hơn nhiều so với việc thực hiện các phép cộng cơ bản.

Gọi t(n) là thời gian cần thiết để nhân hai ma trận kích thước n\*n bằng cách sử dụng đệ quy hai phương trình 2.3.1 và 2.3.2. Giả sử rằng n là luỹ thừa bậc 2. Do thời gian để tình phép cộng, trừ ma trận là, do đót ( n ) = 7 t ( n /2) +dn 2, điều này là một ví dụ minh hoạ cho chúng ta trong việc phân tích tổng quát thuật toán chia để trị. áp dụng dụng định lý thợ ta có . Đối với các trường hợp ma trận vuông nhưng kích thước không phải là luỹ thừa bậc 2 thì giải quyết vấn đề bằng cách thêm các dòng và các cột sao cho kích thước ma trận mới gấp đôi kích thước ma trận cũ và gán giá trị cho các phần tử mới thêm là 0. Điều này không làm ảnh hưởng đến thời http://voer.edu.vn/file/25808 gian tính toán. Do lg7<2,81 nên có thể thực hiện phép nhân hai ma trận kích thước n\*n trong thời gian với điều kiện các phép tính vô hướng là phép tính cơ bản.

Cùng với phát minh của Strassen, có một số nhà nghiên cứu cố gắng tìm kiếm thuật toán để xác định được hằng số ω, khi đó độ phức tạp tính toán phép nhân hai ma trận kích thước n\*n là . Để thực hiện được điều này, việc đầu tiên phải tiến hành là nhân hai ma trận kích thước 2\*2 với 6 phép nhân cơ bản. Nhưng vào năm 1971 Hopcroft và Kerr đã chứng minh điều này là không thể vì phép nhân hai ma trận không có tính chất giao hoán. Việc tiếp theo phải thực hiện là tìm cách nào để nhân hai ma trận 3\*3 với nhiều nhất chỉ 21 phép nhân cơ bản. Nếu thực hiện được việc này có thể sử dụng thuật toán này để suy ra thuật toán đệ quy nhân hai ma trận n\*n với thời gian , nhanh hơn thuật toán của Strassen vì . Không may mắn là điều này không thể thực hiện. Trong suốt một thập kỷ trước khi Pan phát hiện ra cách để nhân hai ma trân kích thước 70\*70 với 143640 phép nhân cơ bản – so sánh với 343000 phép nhân nếu sử dụng thuật toán cổ điển và quả thực là bé hơn một chút so với lg7. Phát hiện này được gọi là cuộc chiến tranh của số thập phân (decimal war). Nhiều thuật toán, mà trong đó hiệu suất tiệm cận cao, được tìm ra sau đó. Ví dụ như cuối năm 1979, phép nhân ma trận có thời gian tính toán là . Hãy tưởng tượng rằng, ngay sau đó tháng 1 năm 1980 thời gian tính của phép nhân ma trận là . Biểu thức tiệm cận thời gian tính toán tồi nhất của thuật toán nhân ma trận kích thước n\*n được Coppersmith và Winograd phát minh ra năm 1986 là . Tại vì các hằng số liên quan bị Nn nên không một thuật toán nào được tìm ra sau thuật toán của Strassen được nghiên cứu và sử dụng.

**Giới thiệu về khoa học mật mã**

Trong phần trước chúng ta đã thấy việc rút gọn số mũ trong số các phép nhân cần thiết để tínhankhông tiết kiệm được một cách đáng kể thời gian tính. Tuy nhiên, có những ứng dụng trong đó các phép nhân được coi là có chi phí ngang nhau. Đó là trường hợp khi chúng ta quan tâm đến số học đồng dư, tức là đến các phép toánanchia đồng dư cho một số nguyênz . Chúng ta đã biết rằngxmodzsẽ cho phần dư của phép chia nguyênxchoz . Ví dụ, 25 mod 7 = 4 vì 25 = 3 x 7 + 4. Nếuxvàylà các số nguyên nằm giữa 0 vàz- 1, và nếuzlà một số nguyên có kích thướcm , thì phép tính nhân đồng dưxymodzsẽ bao gồm một phép nhân thường của hai số nguyên có kích thước tối đam , mà kết quả là một số nguyên có kích thước tối đa 2 m , và tiếp theo là một phép chia tích của hai số đó choz , cũng là một số nguyên có kích thước m , kết quả mà chúng ta quan tâm sẽ là phần dư của phép chia này. Như vậy thời gian cần cho mỗi phép nhân đồng dư phần nào không bị ảnh hưởng bởi giá trị của hai sốxvày . dụng những phân tích trong phần trước với một số thay đổi cần thiết cho phép chúng ta kết luận rằng thuật toán trên chỉ cần một số lượng phép nhân đồng dư vào cỡ (log n) để tínhanmodz . Phân tích một cách chính xác hơn ta thấy số lượng phép nhân đồng dư bằng số Bít trong phép triển khai nhị phân củan , cộng với số Bít 1 trong số những Bít này; tức là vào khoảng 3/2 lgn cho các giá trị đặc trưng củan . Thuật toán tương ứngexposeqđòi hỏin– 1 phép nhân cho tất cả các giá trị củan . Để rõ hơn về điều này, hãy giả dụa ,nvàzlà những số có 200 chữ số và với các số có độ lớn như vậy có thể thực hiện phép nhân đồng dư trong một phần ngàn giây. Thuật toán expomod sẽ tính anmodz trong thời gian ít hơn một giây. Trong khi thuật toánexposeqcần thời giản vào khoảng 10179 lần tuổi của vũ trụ để tínhan .

Trong thực tế, liệu chúng ta có cần tính những lũy thừa đồng dư lớn như vậy hay không? Khoa học mật mã hiện đại, môn khoa học và nghệ thuật của những giao tiếp bí

mật trên những kênh không an toàn, phụ thuộc chủ yếu vào những phép tính này.

Chúng ta hãy xem xét trường hợp sau: Hai người là anh A và chị B trao đổi các thông

điệp với nhau. Giả sử anh A muốn gửi thông điệp cá nhânmcho chị B, nhưng kênh liên lạc lại không an toàn, hay nói một cách khác là dễ dàng bị nghe trộm. Để ngăn không cho những người khác đọc được thông điệp, anh A chuyển nó thành dạng mật mãc , sau đó gửi cho chị B. Việc chuyển đổi này là kết quả của một thuật toán chuyển mã. Đầu ra của nó không những chỉ phụ thuộc vào thông điệpmmà còn phụ thuộc vào

tham biếnk . Chúng ta gọi tham biến này làkhóa . Theo phương pháp kinh điển thì khóa chính là thông tin bí mật được thoả thuận giữa anh A và chị B trước khi họ trao đổi thông điệp với nhau. Nhờ có khoáknên khi nhận được thông điệp (đã mật mã hoá)c , chị B có thể tạo lại thông điệpm . Những hệ thống bí mật kiểu như vậy đều dựa trên nguyên tắc là dù kẻ nghe trộm lấy được thông điệpc , nhưng không biết khóa k thì cũng không thể đọc được nội dung thông điệpm .

Phương pháp mật mã hoá này được sử dụng với những thành công nhất định trong lịch sử phát triển của nó. Việc đòi hỏi các bên tham gia phải thỏa thuận những thông tin bí mật trước khi tham gia truyền thông có thể chấp nhận được trong giới ngoại giao và quân sự, nhưng khó có thể chấp nhận đối với những người dân bình thường. Trong thời đại siêu tốc điện tử, người ta có nhu cầu truyền thông riêng tư với nhau mà không cần có sự sắp đặt từ trước. Liệu anh A và chị B có thể trao đổi thông tin với nhau một cách bí mật với sự có mặt của những người khác mà không cần thỏa thuận từ trước những quy ước bí mật với nhau? Thế hệ mật mã hóa với khóa công khai đã được khai sinh với ý tưởng của Diffie, Hellman và Merkle về sự khả thi vấn đề trên trong những năm giữa thập kỷ 70 thế kỷ 20. Tiếp theo đây chúng tôi sẽ trình bày một giải pháp độc đáo được Rivest, Shamir và Adleman đưa ra ít năm sau. Giải pháp nàyn ngày nay được biết dưới cái tên hệ thống mật mã RSA, viết tắt từ tên của những người đã phát minh ra nó.

Xét hai số nguyên tố có 1 trăm chữ sốpvàqđược chọn một cách ngẫu nhiên bởi chị B. Gọizlà tích củapvàq . Chị B có thể tínhzmột cách hiệu quả từpvàq . Cho dù chúng ta sở hữu những máy tính hiện đại bậc nhất, cũng không có cách nào tính đượcpvàqtừzsử dụng những thuật toán đã biết, ngay cả khi chúng ta sử dụng hết cả thời gian của vũ trụ. Gọi ϕ là tích ( p- 1)( q- 1). Gọinlà một số nguyên nằm giữa 1 vàz– 1 được chọn một cách ngẫu nhiên bởi chị B sao cho n và ϕ nguyên tố cùng nhau. (Chị B không cần kiểm tra một cách cụ thể xemncó thỏa mãn điều kiện trên hay không vì chị sẽ nhanh chóng biết được điều đó). Trong số học chúng ta đã biết là chỉ có duy nhất một số nguyên s nằm giữa 1 và z– 1 thỏa mãn điều kiện nsmod = 1. Ngoài ra có thể dễ dàng tính rastừnvà ϕ (xem lời giải bài tập 7.31) đồng thời sự tồn

tại củascòn cho phép kiểm tra xemnvà ϕ có nguyên tố cùng nhau hay không. Nếuskhông tồn tại, chị B cần phải chọn một giá trị ngẫu nhiên khác chon ; xác xuất thành công của mỗi lần chọn đều như nhau. Mấu chốt trong giải pháp được trình bày ở đây là định lý:axmodz=atrong đó 0 ≤a<zvàxmod ϕ = 1.

Để có thể cho phép anh A hoặc ai đó truyền thông riêng tư với mình, chị B phổ biến cho tất cả sự lựa chọn của chị về giá trị củaxvàn , nhưng giữ bí mậts . Gọimlà một thông điệp mà anh A định gửi cho chị B. Sử dụng bộ mã chuNn chẳng hạn như ASCII, anh A chuyển thông điệp của mình thành một chuỗi Bít, được hiểu như là một sốa . Để đơn giản chúng ta giả thiết rằng 0 ≤a<z– 1; trong trường hợp a ≥z- 1 anh A có thể chia thông điệpmcủa mình thành từng đoạn có kích thước phù hợp. Tiếp đó anh A sử dụng thuật toánexpomodđể tínhc=anmodz , rồi gửi thông điệp đã được mã hóac cho chị B thông qua một kênh không an toàn. Nhờ có khóa s , Chị B giải mã và nhận được sốa , và qua đó là thông điệp mcủa anh A, việc này được thực hiện bởi một phép gọi hàmexpomod ( c ,s ,z ). Điều này thực hiện được vì:csmodz= ( anmodz ) s modz= ( an ) s mod z =ansmodz=a

Bây giờ chúng ta sẽ xét đến hành động của kẻ nghe trộm. Giả sử anh ta nghe được mọi trao đổi giữa anh A và chị B, như vậy anh ta sẽ biếtz ,nvàc . Mục đích của anh ta là xác định giá trị sốamà anh A gửi cho chị B, sốalà số duy nhất nằm giữa 0 vàz– 1 thỏa mãn điều kiệnc=anmodz . Để tính ra a chưa có một thuật toán hiệu quả nào được biết đến: phép tính lũy thừa đồng dư có thể thực hiện một cách hiệu quả với thuật toánexpomodnhưng điều ngược lại hình như không thể thực hiện được. Phương pháp tốt nhất được biết đến hiện nay là: phân tíchzra thành hai thừa số p và q , ( p- 1)( q- 1), tính s và tính a=csmod. Đó chính là cách chị B đã làm. Mỗi bước nêu trên đều khả thi, trừ bước đầu tiên: phân tích một số có 200 chữ số ra thừa số là một việc vượt quá khả năng cho phép của công nghệ hiện nay. Bởi vậy lợi thế của chị B trong việc giải mã chính là ở chỗ chị ta là người duy nhất biết các thừa số củaz , là những số cần thiết để tính ra ϕ vàs . Chị ta biết được không phải do có tài phân tíchzra thừa số mà do chị ta tính z từ các thừa số được chị ta chọn.

Hiện tại độ chắc chắn của sơ đồ mật mã hóa này vẫn chưa được xác định trên phương diện toán học: Không có chứng minh toán học nào khẳng định việc phân tích ra thừa số là không thể được và cũng không có chứng minh nào khẳng định rằng để phá mã cần phải phân tích ra thừa số. Xét trên phương diện khác thì chúng ta có thể có

những thuật toán có khả năng phân tích thừa số một cách hiệu quả, nhưng chúng lại đòi hỏi những máy tính lượng tử, mà để tạo ra những máy tính đó, thì lại nằm ngoài mtầm của công nghệ hiện tại. Hệ thống mật mã hóa mà chúng tôi mô tả ở trên vẫn được coi là một trong những phát minh sáng giá nhất trong lịch sử khoa học mật mã.

**Bài tập chương**

Cho mảng số liệu sau:

10, 4, -5, 7, -45, 14, 30, -2, 50

Hãy minh họa các bước của thuật toán để tìm mảng con lớn nhất.

Dành cho độc giả

Cho dãy số liệu

80, 12, 47, 16, 7, 56, 14, 19, 100

Hãy minh họa các bước của thuật toán MergeSort, QuickSort để sắp xếp dãy khóa trên theo thứ tự tăng dần.

Dành cho độc giả

Thiết kế thuật toán nhân 2 số nguyên dương, sử dụng thuật toán chia để trị, trong

đó mỗi số nguyên dương được chia làm ba phần, và tích của hai số đó sẽ tìm được sau 5 phép nhân số này với độ xấp xỉ n/3. Phân tích độ phức tạp tính toán trong

thuật toán thu được

Dành cho độc giả

Sử dụng các kĩ thuật đánh giá độ phức tạp trong chương 1 để đánh giá độ phức tạp của các thuật toán trong chương 2.

Dành cho độc giả

Xét ma trận

F =

|  |  |
| --- | --- |
| 0 | 1 |
| 1 | 1 |

Hãy tính thử kết quả của phép nhân ma trận này với vector (i, j). Trong đó i, j là hai số

nguyên. Có gì đặc biệt khi i, j là hai số hạng liên tiếp của dãy Fibonaci? Từ đó xây dựng thuật toán chia để trị để tính các phần tử của dãy số Fibonaci và phân tích độ phức tạp tính toán của thuật toán trong 2 trường hợp

a. Coi mỗi phép toán số học là phép toán cơ bản đòi hỏi một đơn vị thời gian

b. Thời gian tính tích của hai số nguyên có độ dài s và q là : O(sqa-1) (s ³ q).

Lưu ý là độ dài của số Fn cỡ O(n) với số thực a > 1.

Dành cho độc giả