Một số vấn đề trong cải tiến hiệu suất

Bởi:

Khoa CNTT ĐHSP KT Hưng Yên

Như đã đề cập ở trên, ta sẽ quan tâm đến các vấn đề để giảm thời gian chạy và chi phí bộ nhớ cho chương trình.

Để minh hoạ cho các luận cứ được nêu về các vấn đề trên, ở đây, ta sẽ chỉ ra các vấn đề quan tâm thông qua các bài toán ví dụ minh hoạ.

Tốc độ xử lý

Trong hầu hết các trường hợp, tốc độ của chương trình là quan trọng như các ứng dụng thời gian thực, ứng dụng về xử lý trên các cơ sở dữ liệu lớn,... Để một ứng dụng có tốc độ nhanh, người lập trình chúng phải quan tâm đến nhiều yếu tố như: thuật toán sử dụng, lựa chọn cấu trúc dữ liệu, tinh chế mã cho chương trình,...

Thuật toán sử dụng

1 . Xác định lại bài toán

Yêu cầu: Trước khi bắt tay vào giải bài toán, hãy tìm hiểu kỹ các yêu cầu mà bài toán đặt ra và tận dụng mọi điều đã biết từ bài toán.

Bài toán minh hoạ: Cho một mảng số nguyên gồm 1.000.000 phần tử; các giá trị nằm trong khoảng từ 0..10 một cách ngẫu nhiên. Hãy sắp xếp để được một mảng có thứ tự giảm dần.

? Giải bài toán tổng quát: là một bài toán sắp xếp; dùng một đoạn chương trình sắp xếp có sẵn của hệ thống hay sử dụng một thuật toán sắp xếp có sẵn như Insert - Sort hay Quick - Sort chẳng hạn. Chi phí về độ phức tạp là o(n2) hay o(nlogn).

? Tuy nhiên, ta đã bỏ qua một tính chất của bài toán đó là các giá trị chỉ nằm trong khoảng 0..10. Sau khi nghiên cứu bài toán ta quyết định sử dụng thuật toán đếm cho việc sắp xếp bài toán.

+ Khởi tạo 10 biến nguyên với giá trị 0.

+ Vcới mỗi giá trị i trong mảng, tăng biến thứ i lên một đơn vị

+ Thực hiện rải giá trị cho mảng ứng với số lần là giá trị của biến thứ i; Như thế, chi phí về độ phức tạp của bài toán là o(n).

2 . Sức mạnh của thuật toán

Yêu cầu: Việc nghiên cứu thuật toán giúp ích rất nhiều cho các nhà lập trình. Các thuật toán có ảnh hưởng quan trọng đến các hệ thống phần mềm và đặc biệt chúng tăng nhanh tốc độ vận hành.

Bài toán minh hoạ: Quay mảng một chiều chứa N phần tử về bên trái I một vị trí. Với N = 8; I = 3 ta được mảng ABCDEFGH sẽ quay thành DEFGHABC.

? Thuật toán 1: Ta có thể giải bài toán bằng cách sao I phần tử đầu tiên của mảng sang một mảng đoạn; dịch chuyển N - I phần tử còn lại của mảng về bên trái I vị trí; sau đó sao I phần tử đầu tiên từ mảng tạm về cuối mảng. Trong trường hợp N và I lớn, như thế việc cần mảng tạm khá tốn bộ nhớ; xét trong trường hợp bộ nhớ của máy không dồi dào thì giải quyết như thế nào?

? Thuật toán 2: Ta cũng có thể viết 1 thủ tục con quay mảng X sang bên trái một vị trí (như thế sẽ giải quyết được vấn đề tốn bộ nhớ vì chỉ cần dùng một biến phụ) và sau đó thực hiện thủ tục trên I lần. Tuy nhiên, thủ tục trên tốn thời gian tỉ lệ với N và như thế chương trình sẽ tỉ lệ với thời gian I*N; do vậy khi N và I lớn thì đây là điều không thể thực hiện được.

? Thuật toán 3: Để ý rằng: khi quay mảng X gồm N phần tử về I vị trí, (giả sử quay sang trái), lúc này phần tử X[i+1] sẽ là phần tử X'[1] (ký hiệu X'[i] là phần tử thứ i của mảng X sau khi quay); X[2i+1] sẽ là phần tử X'[i+1],....và cứ thế tiếp tục. Do đó ta có thuật toán sau:

+ Dịch chuyển X[1] đến 1 biến tạm T

+ Dịch chuyển X[i+1] và X[1];

+ Dịch chuyển X[2+i+1] và X[i+1],....

.....

+ Quá trình cứ thế tiếp tục; chỉ số được tính theo module của N tức khi vượt quá N sẽ chia lấy dư cho N.

+ Quá trình lặp cho đến khi gặp phần tử X[1] và lúc này dùng giá trị từ biến T và quá trình chấm dứt.

Ví dụ: Với trường hợp N = 8, I = 3 ⇒ mảng X = ABCDEFGH ta có như sau: X = ABCDEFGH; N = 8; I = 3; T = A

+ Dịch chuyển X[i+1]: X[4] → X[1]

+ Dịch chuyển X[2i+1]: X[7] → X[4]

+ Dịch chuyển X[3i+1]: X[10] ⇒ X[2] → X[7]

+ Dịch chuyển X[4i+1]: X[13] ⇒ X[5] → X[10] ⇒ X[2]

....

Quá trình được tóm tắt như sau:

T = A; N = 8; I = 3; X = ABCDEFGH

4 → 1..... DBCDEFGH

7 → 4..... DBCGEFGH

(10)2 → 7 . DBCGEFBH

5 → 2..... DECGEFBH

8 → 5..... DECGHFBH

(11)3 → 8.. DECGHFBC

6 → 3(11)... DEFGHFBC

(9)1 → 6 ... DEFGHABC T => Dừng

Đây là thuật toán có chi phí vùng nhớ không lớn và thời gian chạy chấp nhận được.

3 . Các kỹ thuật thiết kế thuật toán v à tinh chế thuật toán.

Yêu cầu: Thực hiện theo các nguyên tắc sau:

? Lưu trữ các trạng thái cần thiết để tránh tính lại,

? Tiền xử lý thông tin để đưa vào các cấu trúc dữ liệu,

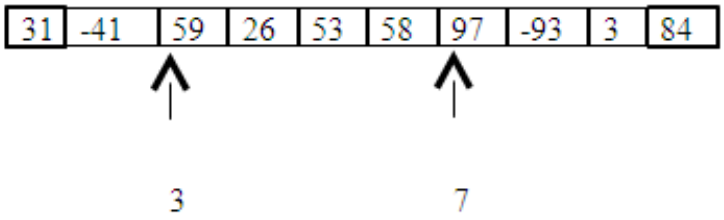
? Sử dụng các thuật toán thích hợp,

? Chỉ ra được cận dưới của thuật toán,

? Sử dụng các kết quả được tích luỹ,...

Bài toán minh hoạ: Cho vector X chứa N số thực X[1], X[2], ...,X[N]. Gọi vector con của X là vector mà phần tử của nó là các phần tử liên tiếp trong X. Tổng của một vector được tính là tổng các phần tử của vector đó. Tính tổng lớn nhất trong các vector con, tức tìm L,U∈1..N để tổng X[i], i∈L..U là lớn nhất.

Để đơn giản, vectơ con có tổng lớn nhất được gọi tắt là vectơ lớn nhất. Ví dụ, nếu vectơ đầu vào có dạng:



Lúc này, kết quả của bài toán là tổng của vectơ X[3..7]. Bài toán rất đơn giản khi tất cả các số là số dương - khi đó kết quả chính là bản thân vectơ X. Vấn đề sẽ phức tạp hơn khi có thêm các số âm. Chúng ta nhận xét rằng nếu tất cả đều là số âm thì kết quả là bằng 0 (đó chính là tổng của vectơ rỗng).

? Thuật toán 1: Một chương trình có thể viết ngay được là xét tất cả các cặp số nguyên L và U thoả mãn 1<= L<= U<= N; đối với mỗi cặp như vậy ta tính tổng của vectơ con X[L..U] và so sánh tổng này với giá trị lớn nhất hiện có. Ta có giã mã cho thuật toán 1 như sau:

MaxSoFar := 0; For L := 1 to N do

For U := L to N do

Begin

Sum := 0;

For I := L to U do

End;

Sum := Sum + X[I];

/ Sum chứa tổng của X[L .. U] */*

MaxSoFar := Max(MaxSoFar, Sum);

Chương trình này ngắn và dễ hiểu, tuy nhiên điều không may là nó chạy rất chậm. Độ phức tạp của chương trình là o(n3).

? Thuật toán 2: Đối với thuật toán 1, đa số người lập trình cho rằng có thể viết chương trình chạy nhanh hơn. Có hai cách như vậy. Các cách này đều có độ phức tạp o(n2). Thuật toán thứ nhất tính nhanh các tổng của vectơ con bằng cách sử dụng hệ thức: Tổng của X[L..U] = Tổng của X[L..U-1] + X[U]; Ta có thuật toán 2 như sau:

MaxSoFar := 0.0; For L := 1 to N do Begin

Sum := 0.0;

For U := L to N do

Begin

End;

End;

Sum := Sum + X[U];

/ Sum chứa tổng của X[L .. U] */*

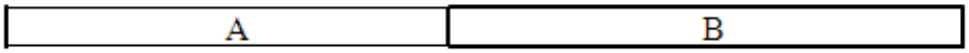
MaxSoFar := Max(MaxSoFar, Sum);

End

Các lệnh trong vòng lặp thứ nhất thực hiện N lần. Với mỗi lần thực hiện các lệnh trong vòng lặp thứ nhất, các lệnh trong vòng lặp thứ hai thực hiện nhiều nhất là N lần. Vậy ta có độ phức tạp là o(n2).

? Thuật toán 3: Thuật toán chia để trị: "Để giải bài toán kích thước N, chúng ta giải một cách đệ quy hai bài toán con kích thước khoảng N/2, kết hợp lời giải của chúng để tạo ra lời giải của toàn bộ bài toán".

Trong trường hợp này bài toán của ta là xử lý vectơ độ dài N, do đó một cách tự nhiên là chia vectơ này thành hai vectơ con có độ dài gần bằng nhau. Chúng ta gọi hai vectơ này là A và B.



Sau đó, bằng đệ quy chúng ta tìm vectơ con lớn nhất trong A và B gọi là MA và MB.

Để ý rằng kết quả bài toán là giá trị lớn nhất trong hai tổng của vectơ MA và MB. Kết quả của bài toán có thể là tổng của vectơ MC chứa đồng thời các thành phần của A và B. Ta gọi là vectơ con như vậy là vectơ vượt biên.

Như vậy thuật toán chi để trị sẽ tính MA,MB bằng đệ quy và tính MC bằng phương pháp khác, kết quả bài toán này là giá trị lớn nhất trong ba tổng của ba vectơ này. Các mô tả trên là gần đủ để viết chương trình. Chúng ta còn phải mô tả cách quản lý các vectơ nhỏ và cách tính vectơ MC. Phần đầu tiên rất dễ: Đối với vectơ chỉ chứa một phần tử, vectơ con lớn nhất hoặc là chính nó hoặc là vectơ rỗng trong trường hợp phần tử của vectơ đó là số âm, và vectơ con lớn nhất của vectơ rỗng cũng là vectơ rỗng. Để tính MC, chúng ta nhận xét rằng thành phần của vectơ MC nằm trong vectơ A là vectơ con lớn nhất trong tất cả các vectơ con của vectơ A, bắt đầu từ biên của A và B. Tương tự như thế đối với thành phần của vectơ MC nằm trong vectơ B. Kết hợp tất cả các yếu tố này, chúng ta có thuật toán 3, được

gọi bởi lệnh:

Answer := MaxSum(1,N); Recursive Function MaxSum(L,U)

Begin

if L > U then return 0; /* *vectơ rỗng* */

if L = U then return Max(0.0, X[L]); /**vectơmột phần tử* */

M := (L+U) div 2

/* *A là v ectơ X[L.. M], B là vectơ X[M+1.. U]* * /

/* *Tìm giá trị lớn nhất của tổng c á c thành phần b ên trái (trong vectơ A)*

của v e ctơ vượt biên */

```
Sum := 0; MaxToLeft := 0;

For I := M downto L do

Begin

End;

Sum := Sum + X[I]

MaxToLeft := Max(MaxToLeft, Sum)
```

/ Tìm giá trị lớn nhất của tổng c á c thành phần b ên phải (trong vectơ*

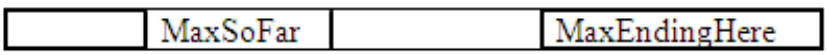
*B) của v e ctơ vượt b i ên */*

```
Sum := 0; MaxToRight := 0;

for I := M +1 to U do

Begin

Sum := Sum + X[I]

MaxToRight := Max(MaxToRight, Sum)

End;
```

Thuật toán thực hiện o(n) công việc trong mỗi mức đệ quy, và có tất cả là o(logn)mức đệ quy. Nên chương trình này giải quyết bài toán với độ phức tạp o(nlogn).

? Thuật toán 4: Thuật toán quét: Giả sử rằng chúng ta đã giải bài toán cho vectơ X[1..I-1]; làm thế nào để mở rộng kết quả này cho bài toán với vectơ X[1..I]? Lý luận tương tự như trong thuật toán "chia để trị": tổng lớn nhất trong vectơ X[1..I-1]

(gọi là MaxSoFar), hoặc tổng lớn nhất trong tất cả các tổng của vectơ con kết thúc tại I (gọi là MaxEndingHere).



Nếu chúng ta tính MaxEndingHere bằng cách tương tự như trong thuật toán

3, thì ta chỉ có một thuật toán bình phương (có độ phức tạp o(n2). Để làm nhanh hơn, chúng ta nhận xét điều như sau: vectơ con lớn nhất kết thúc tại vị trí I là vectơ con lớn nhất kết thúc tại vị trí I-1 được bổ sung thêm phần tử X[I] ở cuối hoặc là vectơ rỗng trong trường hợp tổng của vectơ nhận được là số âm. Ta có thuật toán 4 như sau:

MaxSoFar = 0; MaxEndingHere = 0; For I := 1 to N do Begin

/ Bất biến: MaxEndingHerevà MaxSoFar là đúng đối với* n *X[1..I-1]*/*

End;

MaxEndingHere := Max(MaxEndingHere + X[I],0); MaxSoFar := Max(MaxSoFar, MaxEndingHere);

Chương trình này có thời gian chạy là o(n). Vì vậy thuật toán này được gọi là thuật toán tuyến tính.

Như vậy, khi xây dựng ứng dụng, việc sử dụng các thuật toán phù hợp làm giảm thời gian chạy chương trình một cách đáng kể.

Lựa chọn cấu trúc dữ liệu

Song song với thuật toán, việc chọn lựa cấu trúc dữ liệu ảnh hưởng lớn đến hiệu suất chương trình và nó tác động đến bản thân thuật toán bởi cấu trúc dữ liệu gắn bó mật thiết với thuật toán.

Việc chọn đúng đắn cấu trúc dữ liệu làm giảm không gian bộ nhớ, giảm thời gian chạy, tăng tính chuyển đặc và dễ bảo trì, đặc biệt là các cấu trúc dữ liệu cao cấp, mặc dầu chúng không thường được dùng nhưng khi cần thiết thì không thể thiếu chúng được.

Ta sẽ gặp lại việc chọn lựa cấu trúc dữ liệu trong phần 5.6.2. ở sau, trong phần xét về không gian bộ nhớ chương trình.

Tinh chế mã

Thông thường, để tăng tính hiệu quả của chương trình, người ta thường bàn về các tiếp cận bậc cao như: định nghĩa bài toán, cấu trúc hệ thống, thiết kế thuật toán và chọn cấu trúc dữ liệu.

Tuy nhiên, các tiếp cận bậc thấp như tinh chế mã mà nó thường được thực hiện ở những phần tốn kém của chương trình để cải tiến hiệu suất. Mặc dù đây là phương pháp không phải lúc nào cũng cần thiết nhưng đôi lúc nó tạo ra khác biệt lớn trong hiệu suất của chương trình.

Các phương pháp thường dùng của tinh chế mã.

+ Tính trước các giá trị,

+ Thay tương đương,

+ Dùng biến trung gian thích hợp, không tính lại các hằng trong vòng lặp.

Bài toán: Cho một chuỗi gồm 1 triệu ký tự. Hãy phân loại mỗi ký tự theo 4 kiểu sau: kiểu chữ in, kiểu chữ hoa, kiểu số hay là các kiểu "khác".

? Lời giải mà ta thường làm: là thực hiện các so sánh đối với mỗi ký tự. Như vậy, trong bảng mã ASCII, để xác định mỗi ký tự thuộc loại nào phải mất rất nhiều lần so sánh; và đây chính là điểm "nóng" của chương trình.

? Tinh chế mã: Ở đây, nếu ta xem mỗi ký tự như là một chỉ số của mảng mà thành phần của nó là các kiểu ký tự. Như vậy, kiểu ký tự C là mảng [C] và để xác định kiểu của một ký tự, ta chỉ cần truy cập đến một mảng đơn giản thay vì phải thực hiện các chuỗi so sánh phức tạp.

Như vậy, khi thực hiện tinh chế mã, cần xác định ở đâu là điểm "nóng" của chương trình và hãy tập trung vào điểm nóng. Hơn nữa, ta đã biết rất nhiều phương pháp để cải tiến hiệu suất của chương trình và hãy dùng đến phương pháp này sau cùng và đôi khi, phương pháp này còn được dùng để làm giảm không gian chiếm bởi chương trình.

Không gian bộ nhớ

Trong những ngày đầu của kỹ thuật máy tính, các nhà lập trình bị hạn chế bởi những bộ nhớ nhỏ; Ngày nay vấn đề này không còn là điểm "nóng" nữa. Tuy vậy, khi thiết kế chương trình không phải lúc nào ta cũng có đủ bộ nhớ để sử dụng bởi nhiều lý do khác nhau.

Không gian dữ liệu

Nguyên tắc để làm giảm không gian lưu trữ dữ liệu.

+ Đảm bảo tính đơn giản,

+ Trong một số trường hợp đừng lưu trữ, hãy tính lại khi cần thiết,

+ Đặc biệt, việc nghiên cứu kỹ các cấu trúc dữ liệu, (thường là cấu trúc dữ liệu thưa thớt) sẽ làm giảm nhiều không gian cần thiết để lưu trữ các thông tin cho trước,

+ Nén dữ liệu sau đó giải nén khi dùng,

+ Sử dụng các nguyên tắc cấp phát bộ nhớ: chẳng hạn như cấp phát bộ nhớ động,...

Xét bài toán: Trên một bản đồ chứa 2.000 điểm (bảng đồ quân sự) được đánh số từ 1 đến 2.000. Một vị trí trên bảng đồ được xác định bằng cặp tọa độ (x,y) với x là

số nguyên nằm trong khoảng 1...200; y là số nguyên nằm trong khoảng 1...150. Chương trình dùng cặp (x,y) để xác định điểm nào (nếu có) được chọn trong 2000 điểm đã cho.

? Không gian lưu trữ 1: Một cách hiển nhiên để lưu trữ bản đồ trên là dùng mảng 2

chiều 200 x 150 số nguyên; ứng với mỗi x.y sẽ chứa một giá trị trong khoảng từ

1...2.000 hay sẽ chứa giá trị 0.

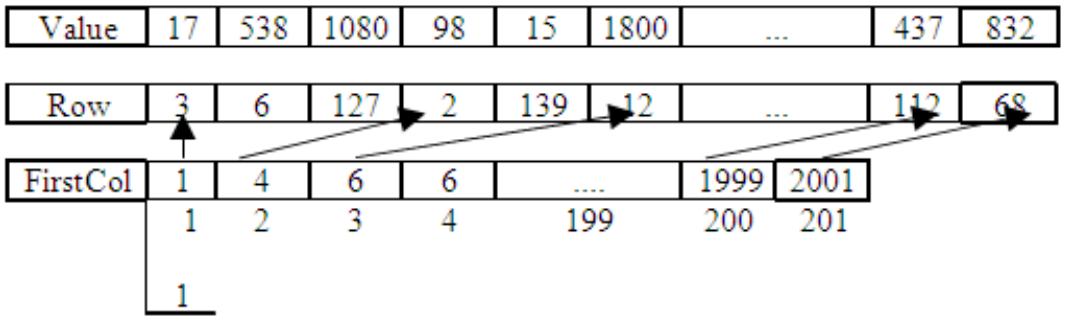
Việc dùng bảng này, thời gian truy cập nhanh; nhưng nó chiếm đến 200 x

150 = 30.000 ô nhớ; và giả sử để lưu trữ dữ liệu (ở đây là 1...2.000) cần 2 byte thì ta cần 60.000B bộ nhớ.

Tuy nhiên, trong mảng trên thì đa số là giá trị 0 (giá trị không dùng). Do vậy, nếu ta dùng cái nào chỉ lưu trữ các giá trị cần thiết (ở đây là 2.000 giá trị) thì việc chiếm bộ nhớ sẽ giảm đáng kể.

? Không gian lưu trữ 2: Thay vì dùng một mảng 2 chiều ở trên, ta sử dụng 3 mảng 1

chiều như sau:



Mảng value dùng để chứa các giá trị (ở đây là 1...2.000) theo từng cột, như vậy cần 2.000 ô nhớ; Mảng Row là mảng để chứa hàng tương ứng với giá trị ở mảng value; như thế cần 2.000 ô nhớ. Mảng FirstCol là mảng để chứa số cột hiện có - tương ứng với giá trị ở mảng value; như thế cần 200 ô nhớ; thêm 1 ô nhớ đầu tiên để đánh dấu, nên tổng cộng cần 201 ô.

Các điểm trong cột I được biểu diễn bằng các phần tử trong mảng Row và Value giữa các vị trí FirstinCol [I] và FirstinCol [I+1]-1. Ở đây, FirstinCol [201] được xác định (mặc dù chúng ta chỉ có 200 cột) để biểu thức I+1 hợp lệ. Theo hình trên, chúng ta có 3 điểm trong cột thứ nhất điểm 17 ở vị trí (1,3), điểm 538 ở vị trí

(1,6,), điểm 1053 ở vị trí (1,127). Có hai điểm trong cột 2 (điểm 98 ở vị trí (2,2), điểm 15 ở vị trí (2,139)), cột 3 không có điểm nào, và có 2 điểm trong cột 200. Để

xác định điểm nào (trong số 2000 điểm) được lưu giữ tại vị trí (I,J), chúng ta dùng giãi mã sau:

```
For K := FirstinCol[I] to FirstinCol[I+1] do

If Row[K] = J then

/* Tìm thấy ở vị t rí */

Return Value[K]

/* Không có đi ể m nào tại vị trí (I , J) */

Else

Return 0;
```

Phương pháp này dùng ít không gian hơn nhiều so với phương pháp trước đó. Ở đây chúng ta dùng hai mảng 2000 phần tử và một mảng 201 phần tử (như vậy tất cả là 4201 từ 16 bit thay vì 30000 từ trong phương pháp trước đó). Mặc dù nó hơi chậm hơn (một lần truy nhập phải mất 150 lần so sánh trong trường hợp tồi nhất, nhưng trung bình chỉ cần 6 lần), nhưng chương trình chạy tốt và người dùng không gặp phải vấn đề gì.

Lời giải này minh hoạ một số điểm tổng quát về cấu trúc dữ liệu. Vấn đề ở đây rất cổ điển: việc biểu diễn thưa thớt (tức là mảng trong đó hầu hết các phần tử có cùng một giá trị, thường là giá trị 0). Lời giải trên có ý tưởng rất đơn giản và dễ cài đặt bằng mảng. Chú ý rằng ở đây không có mảng LastinCol đi cùng với mảng FirstinCol, bởi vì chúng ta sử dụng một điều là điểm cuối cùng trong một cột chính là điểm đứng trước điểm đầu tiên của cột tiếp theo. Đay là một ví dụ tầm thường của nguyên tắc tính lại thay vì lưu trữ. Tương tự, chúng ta không có mảng Col đi cùng với mảng Row vì chúng ta chỉ truy nhập mảng Row thông qua mảng FirstinCol, do đó chúng ta luôn biết cột hiện thời là cột nào.

Nhiều kỹ thuật sử dụng cấu trúc dữ liệu khác cũng có thể làm giảm không gian. Trong thực tế, chúng ta tiết kiệm không gian bằng cách thay thế mảng 3 chiều thành mảng 2 chiều. Nếu chúng ta sử dụng một khoá được lưu trữ như là chỉ số của mảng, thì chúng

ta không cần phải lưu trữ bản thân khoá này; thay vào đó, chúng ta chỉ cần lưu trữ các thuộc tính thích hợp của nó, chẳng hạn như số lần xuất hiện của nó. Thêm vào đó, các ứng dụng của kỹ thuật đánh chỉ số bằng khoá đã được vận dụng. Trong ví dụ về ma trận thưa thớt trên đây, việc đánh chỉ số bằng khoá thông qua mảng FirstinCol cho phép chúng ta giải quyết vấn đề mà không cần dùng đến mảng Col.

Không gian chương trình

Trong một số chương trình, đôi lúc thì kích thước của chính bản thân nó là vấn đề. Hãy định nghĩa các chương trình con hay sử dụng các bộ thông dịch chuyên dụng để làm cho chương trình đơn giản, trong sáng hơn làm cho nó rõ ràng hơn và dễ bảo trì.

Lựa chọn hệ thống và phần cứng

Nên lựa chọn các ngôn ngữ lập trình phù hợp với ứng dụng của bạn. Đôi lúc cần hãy thay thế các chương trình con viết trên ngôn ngữ khác để có tốc độ lớn hơn.

Trong xu thế phát triển của phần cứng hiện nay, cần phải tận dụng thế mạnh của phần cứng để có hiệu suất của chương trình cao, mặc dù điều này làm hạn chế tính phổ cập của nó nhưng hiện nay yêu cầu về phần cứng cao là chấp nhận được.