Đệ quy có nhớ

# Giải các bài toán bằng đệ quy

Xét bài toán đếm số đối tượng (hoặc tìm đối tượng tốt nhất, hoặc đếm số đốitượng tốt nhất, hoặc liệt kê các đối tượng...) thỏa mãn một số tính chất nào đó. Ta gọi mỗi đối tượng thỏa mãn các tính chất mà bài toán đưa ra là 1 nghiệm của bài toán. Giả sử X là một nghiệm, khi đó, nếu X gồm nhiều thành phần *x1,x2...xk* hợp thành thì bài toán có thể giải bằng đệ quy (xét tất cả các khả năng của *x1,x2...xk*)

Nói chung thì hầu hết các bài toán đều giải được bằng đệ quy. Ví dụ: Bài toán liệt kê các hoán vị của n số (nghiệm X hợp thành từ các thành phần là các số nguyên); bài toán tìm đường đi tốt nhất giữa 2 đỉnh của đồ thị (nghiệm X hợp thành từ các thành phần là các đỉnh); ...

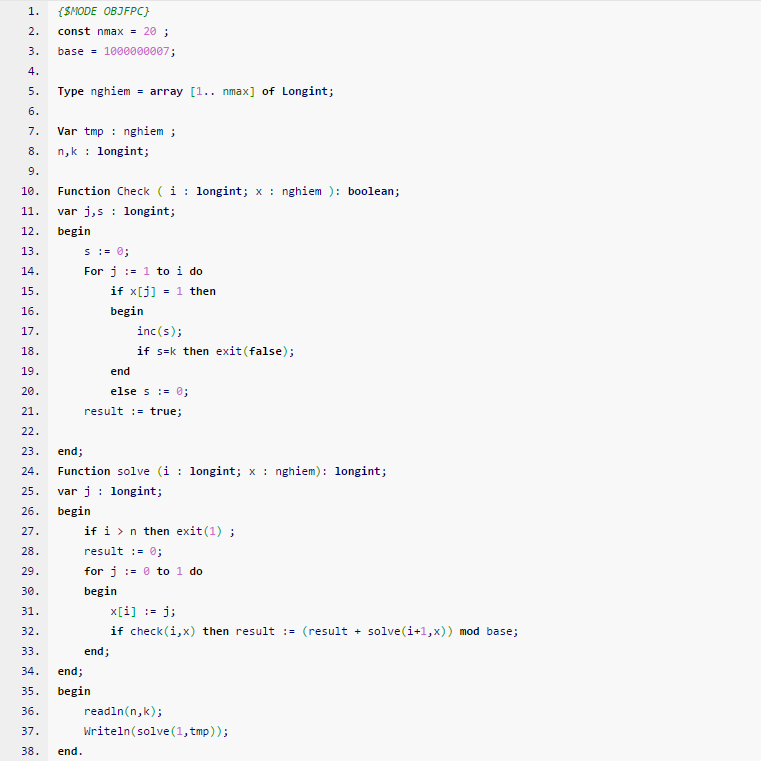
* Bài toán mở đầu: Đếm số lượng xâu nhị phân độ dài n, không có k bit 1 liên tiếp.

1 ≤ k ≤ n ≤ 1000. Do kết quả có thể rất lớn, chỉ cần in ra theo mod 109+7

# Đệ quy thuần túy

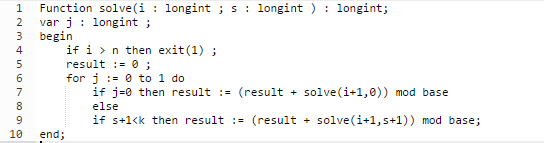
Giả sử đoạn code sau

<http://ideone.com/tls1HN>



Hàm *solve(i,x)* trả về số cách xây dựng phần còn lại để *x* được *x’* là 1 nghiệm của bài toán ( Tức là số nghiệm của bài toán có *i-1*  thành phần đầu tiên là *X[1..i-1]* ). Độ phức tạp của cách cài đặt trên là O(n\*2n)

Để ý rằng hàm *Check()* để kiểm tra nối *j* vào được sau *X*  hay không. Thay vì gọi hàm *Check* mất O(n), ta có thể vừa tính được *s*; Số lượng bit 1 tận cùng của *X* ( Chỉ mất O(1) để check ). Ngoài ra cũng không cần lưu lại X nữa, vì việc lưu chỉ để *Check()*



Với cách cài trên, độ phức tạp còn O(2n)

# Sự trùng lặp của các bài toán

Ta nhận thấy với mỗi bộ (i,s) hàm trả về 1 giá trị xác định. Như vậy số bài toán không quá *iMax \* sMax*  ≤ n2 . Trong khi số bộ (i,X) là n\*2n  =>Số bài toán đã giảm xuống đáng kể.

Giải thích cho điều này khá đơn giản : Có rất nhiều cấu hình *X* cho cùng một số *s*

Ví dụ 000000 và 110110 đều cho *s = 0*. Và các *X*  cho cùng một *s* thì có thể xem như nhau và tả chỉ cần giải một *X*  rồi áp đặt kết quả cho các *X* khác.

*k = 5, n = 100.*  Xét 2 lần gọi hàm *solve(10,000000111) & solve(10,101100111).* Rõ ràng số cách xây dựng phần còn lại của *X1* và *X2* là bằng nhau, mọi cách xây dựng tiếp cho *X1* đều áp dụng được cho *X2*  và ngược lại. Nói cách khác bài toán *solve(i,X1)* và *solve(i,X2) có chung lời giải.* Việc giả nhiều bài toán có cùng một lời giải như thế gọi là sự trùng lặp các bài toán.

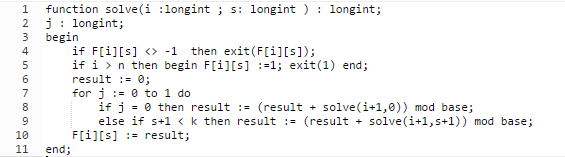
Như vậy với mỗi *X*, ta chỉ quan tâm đến số bit 1 tận cùng của nó *(s),* vì thế *s* đại diện được cho cả xâu *X* (chỉ cần *s,* ta sẽ đủ cơ sở để xây dựng phần còn lại của *X* ). Số (bộ số) *s*  như thế gọi là số đại diện ( bộ đại diện ).

# Mảng nhớ, đệ quy có nhớ

Cả 2 cách viết trên đều có độ phức tạp là O(2n)

Theo cách viết thứ 2, sẽ có rất nhiều lần gọi đến *Solve(i0,s0)*.Mỗi lần gọi, ta có thể phải tính lại, mặc dù kết quả so với lần gọi trước không có gì thay đổi. Ta optimize bằng cách với mỗi bộ *(i,s)* thay vì giải nhiều lần, ta lưu lại lời giải. Kết quả cho *solve(i,s)* sẽ lưu vào mảng *F[iMax, sMax]*

Dễ dàng nhận thấy đây là một tư tưởng có từ rất lâu để optimize bài toán, ví dụ như bài *ByteLandGold*. Thế nhưng áp dụng nó để thay đổi và tính toán chính xác độ phức tạp thuật toán chính là trọng tâm cũng bài học ngày hôm nay.



Ta thấy ngay số bài toán là n2 , mỗi bài toán giải mất O(2) (gọi 2 bài toán con) Và không có bài nào được giải 2 lần => đpt O(2\*n2)

# Các bước giải một bài toán bằng đệ quy có nhớ

1. Tìm *cấu hình nghiệm* của bài toán. Ví dụ ở bài toán trên có cấu hình nghiệm là X = *x1, x2, x3..xn* Với *x­i* là các bit.

Như đã nói, ta sẽ xây dựng nghiệm *X* bằng cách xây dựng từng thành phần *xi* của nó ( theo một thứ tự nào đó )

1. Tìm các ràng buộc nghiệm.
2. Kiểm soát bằng cách ràng buộc đó đều được thõa mãn trong quá trình xây dựng nghiệm bằng cách
   1. Tìm bộ chỉ số mô tả các tính chất ( mà ta quan tâm ) của phần nghiệm đã xây dựng ( ở ví dụ trên, ta quan tâm 2 tính chất là độ dài phần nghiệm đã xây dựng và số bit 1 tận cùng của nó )
   2. Bộ chỉ số *phải đủ cơ sở*  để đảm bảo được việc xây dựng các thành phần tiếp theo của nghiệm sẽ thõa mãn ràng buộ
   3. Thường thì với mỗi ràng buộc, ta sẽ dùng 1 chỉ số để kiểm soát ( ở ví dụ trên, *s*  sẽ kiểm soát rằng độ dài mọi dãy bit 1 liên tiếp đều nhỏ hơn k )
3. Viết hàm đệ quy xây dựng từng thành phần của một nghiệm. Bộ chỉ số để thõa mãn ràng buộc sẽ được đệ quy.
4. Đẩy mảng nhớ vào

# Ví dụ minh họa

*Vd1*

*N13*

Ở đây, ta coi số là một xâu. Gọi n là độ dài xâu B (n ≤ 16). Cho rằng A cũng có độ dài n và có thể có số 0 đứng đầu.

1. Cấu hình nghiệm: *X = x1x2x3..xn* với *xi [0,9]*

2. Các ràng buộc của nghiệm:

Thứ tự từ điểm xâu X lớn hơn hoặc bằng xâu A

Thứ tự từ điểm xâu X nhỏ hơn hoặc bằng xâu B

X không chứa số 13

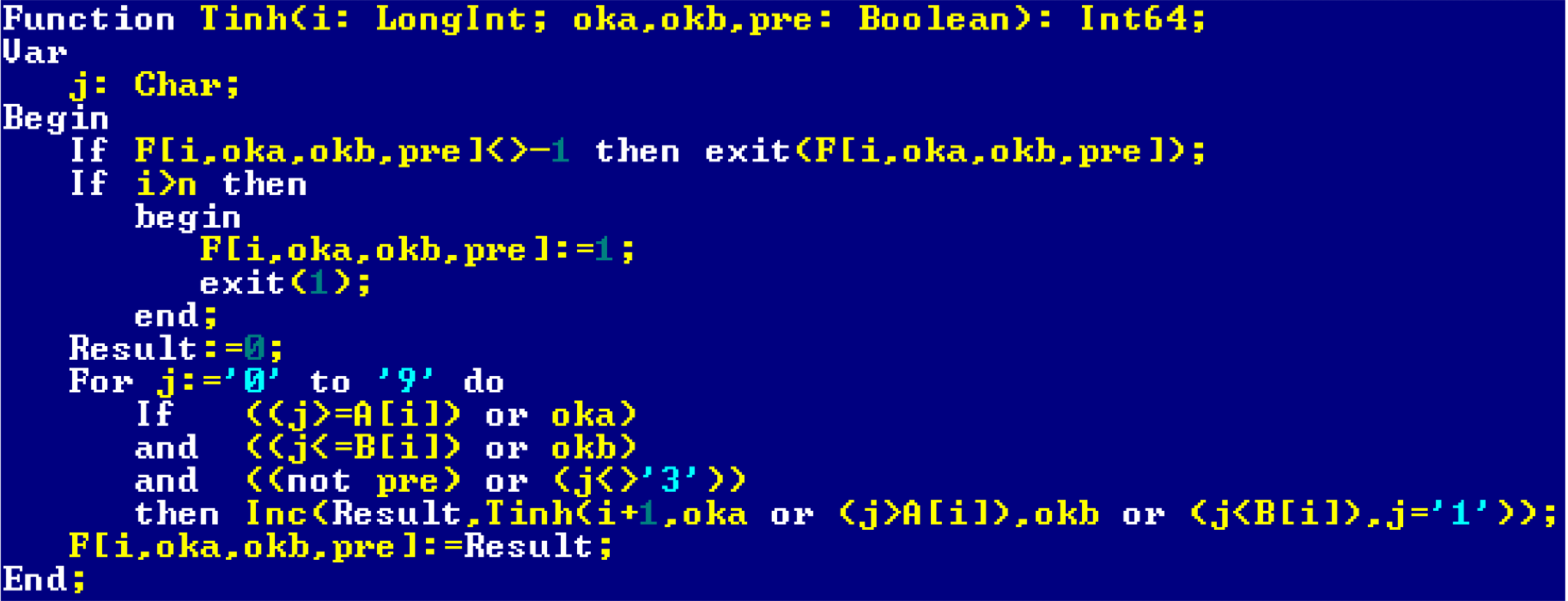
3. (Bộ chỉ số giúp) Kiểm soát ràng buộc:

*i*: Vị trí mà ta đang xây dựng

*oka*: Boolean. Oka=True cho biết phần nghiệm đã xây dựng có thứ tự từ điển lớn hơn hẳn xâu A. oka=False cho biết phần nghiệm đã xây dựng là tiền tố của A (giống hệt phần đầu của A)

*okb*: Boolean. Okb=True cho biết phần nghiệm đã xây dựng có thứ tự từ điển nhỏ hơn hẳn xâu B. okb=False cho biết phần nghiệm đã xây dựng là tiền tố của B

*pre*: Boolean cho biết X[i-1] có phải bằng 1 hay không



*Vd2*

*PFNum* (code lâu lắm rồi đừng chửi nếu sai)

Ở đây, ta coi số là một xâu. Gọi n là độ dài xâu B (n ≤ 19). Cho rằng A cũng có độ dài n và có thể có số 0 đứng đầu

1. Cấu hình nghiệm: *X = x1x2x3..xn* với *xi [0,9]*

2. Các ràng buộc của nghiệm:

Thứ tự từ điểm xâu X lớn hơn hoặc bằng xâu A

Thứ tự từ điểm xâu X nhỏ hơn hoặc bằng xâu B

Xâu X không chứa xâu đối xứng nào

3. (Bộ chỉ số giúp) Kiểm soát ràng buộc:

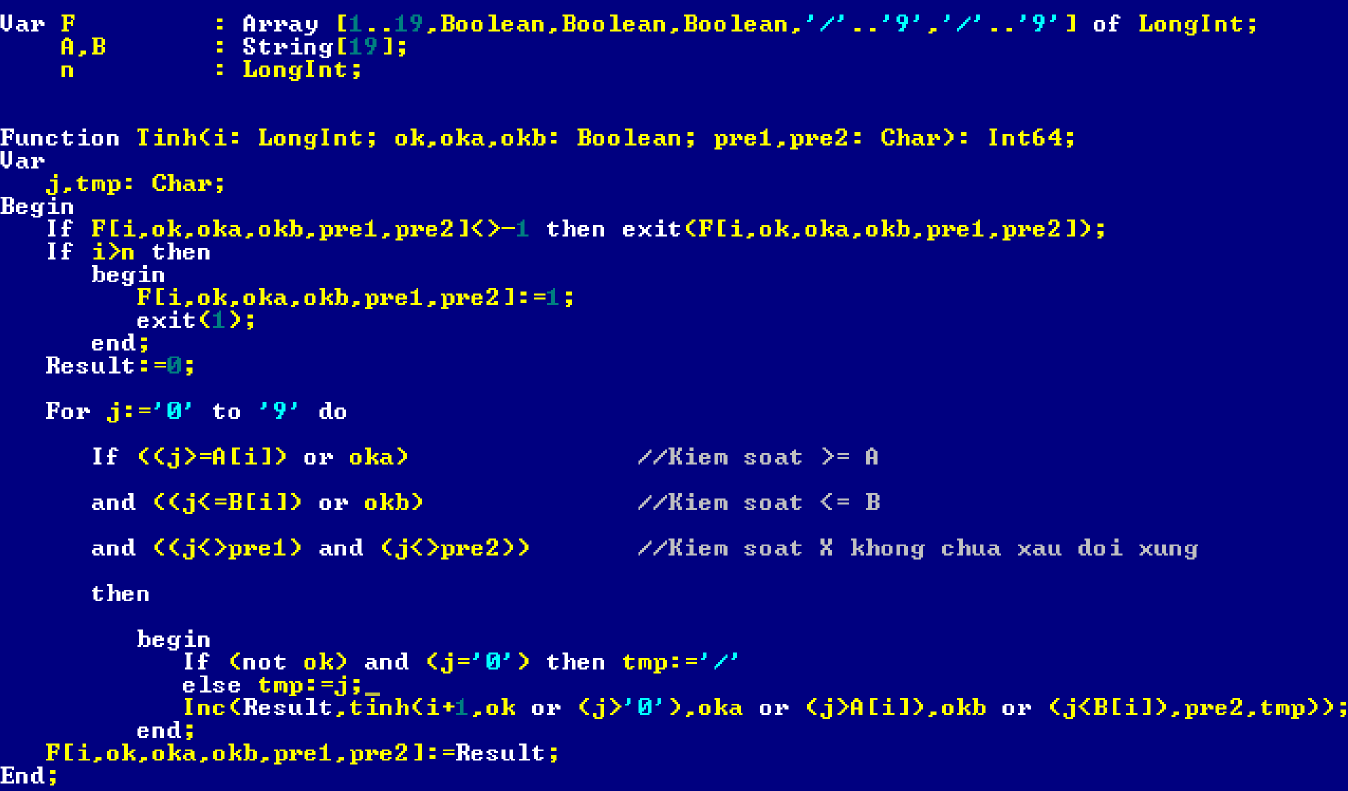
*i*: Vị trí mà ta đang xây dựng

*ok*: Boolean cho biết tính đến i-1, số X đã có nghĩa chưa (hay chỉ chứa toàn số 0)

*oka*: Boolean. oka=True cho biết phần nghiệm đã xây dựng có thứ tự từ điển lớn hơn hẳn xâu A. oka=False cho biết phần nghiệm đã xây dựng là tiền tố của A (giống hết phần đầu của A)

*okb*: Boolean. okb=True cho biết phần nghiệm đã xây dựng có thứ tự từ điển nhỏ hơn hẳn xâu B. okb=False cho biết phần nghiệm đã xây dựng là tiền tố của B

Về ràng buộc thứ 3, ta thấy chỉ cần X không chứa xâu đối xứng độ dài 2 hoặc độ dài 3 thì sẽ không chứa xâu đối xứng. Thật vậy, vì nếu X chứa xâu đối xứng độ dài ≥ 4, thì nó sẽ chứa xâu độ dài 2 hoặc 3 (bằng cách xóa đều 2 bên xâu đó để còn lại 2 hoặc 3 ký tự). => Cách thỏa ràng buộc: *pre1,pre2* : cho biết 2 ký tự ngay trước đó đã xây dựng ( *pre1 là X[i-2], pre2 là X[i-1]*, quy ước *X[-1]=X[0]=-1* )



*Vd3*

*Num68*(chắc t k cần nói lại ☺)

Gọi n là số chứ số của N (n ≤ 19). Cho rằng A,B cũng có n chữ số và có thể có số 0 đứng đầu.

1. Cấu hình nghiệm: X = A = *a1a2a3..an* với *ai [0,9] ; B* = *b1b2b3..bn* với *bi [0,9]*

2. Ràng buộc của nghiệm:

A,B khác 0

Thứ tự từ điển của A nhỏ hơn hoặc bằng của B

Tổng A+B=N

A hoặc B chứa 6 hoặc 8

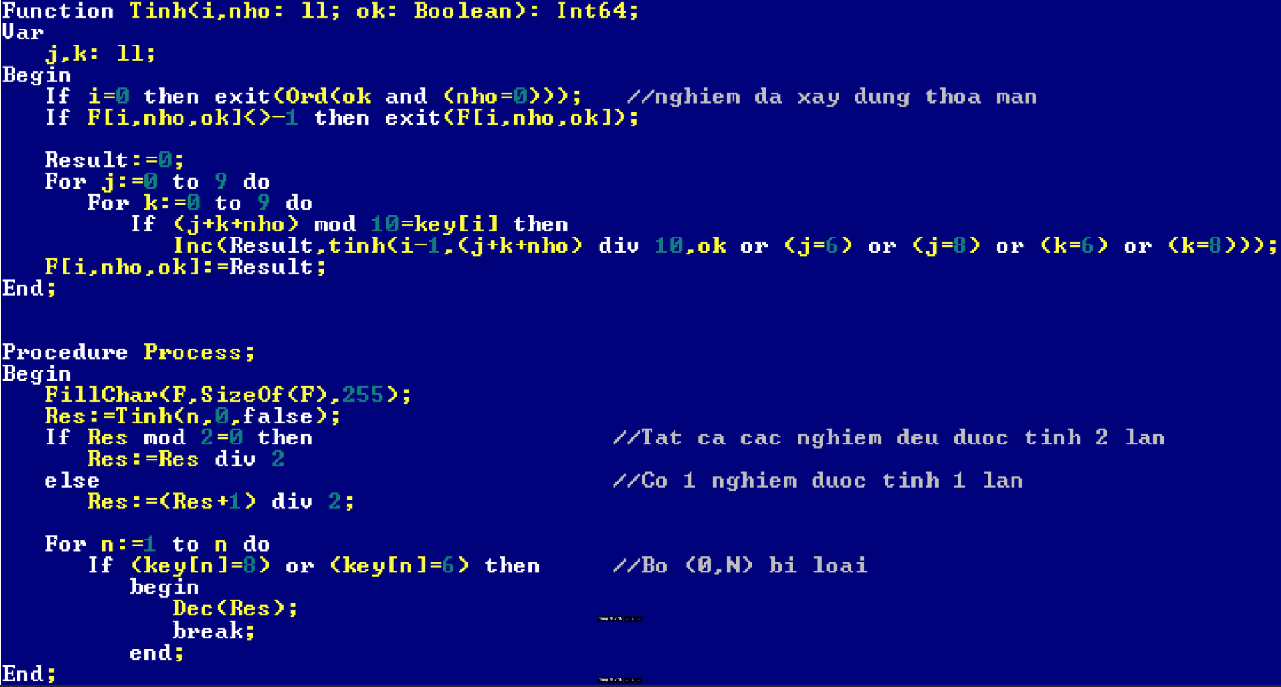
3. Cách thỏa ràng buộc

Ràng buộc 1 có thể bỏ qua, khi đó *có thể* ta sẽ đếm thừa bộ (0,N)

Ràng buộc 2 có thể bỏ qua, khi đó mỗi nghiệm (A,B) sẽ được đếm 2 lần (trừ bộ A=B) => Sau khi tính, ta xử lí kết quả 1 chút

Ta sẽ xây dựng từ *xn* về *x1*, dùng biến *nho* (nhớ) để kiểm soát (*ai+bi+nho*) *mod 10=ni*

Dùng biến *ok*: Boolean: Phần nghiệm đã xây dựng có chứa 6 hoặc 8?



# Một số bộ chỉ số và thứ tự xây dựng giúp thõa mãn ràng buộc

1. Thứ tự từ điển lớn hơn/nhỏ hơn 1 xâu cho trước: Xây dựng nghiệm theo chiều xuôi, sử dụng biến ok: boolean
2. Tổng/hiệu/tích bằng một số cho trước: Xây dựng nghiệm theo chiều ngược, sử dụng biến *nho*: LongInt
3. Xâu đối xứng: Xây dựng nghiệm từ 2 đầu lại
4. Hợp thành từ các ràng buộc trên:
   1. Nếu thứ tự xây dựng nghiệm như nhau, bộ chỉ số là *bộ các bộ* trên
   2. Nếu khác nhau thứ tự xây dựng, tìm 1 thứ tự hợp lý, thay đổi một số bộ.

# Bài toán thứ tự từ điển

Phát biểu: Cho một tập các đối tượng so sánh được, tìm thứ của một phần tử *X* cho

trước / tìm phần tử có thứ tự *k* cho trước

FSTR’

A gọi là xâu con của B nếu ta có thể thu được A bằng cách xóa bớt 1 số ký tự của B (có thể không xóa ký tự nào)

2 ký tự gọi là liên tiếp nếu mã Ascii của chúng hơn kém nhau 1 đơn vị. Một xâu gọi là xâu đẹp nếu nó chỉ chứa các ký tự latin thường và không chứa 2 ký tự gần nhau nào là liên tiếp nhau.

Cho xâu S chỉ chứa các ký tự latin thường. Ta viết các xâu đẹp độ dài n nhận S làm xâu con lên bảng theo thứ tự từ điển, in ra xâu thứ m trên bảng

**Giải:**

Ta nói xâu nghiệm là xâu đẹp độ dài *n* nhận *S* làm xâu con. Gọi *X* là xâu cần tìm

(xâu nghiệm thứ *m*). Ta sẽ xây dựng từng vị trí của *X* và kiểm soát được có đúng *m-1*

xâu nghiệm nhỏ hơn *X*.

Hàm *Tinh(i,k,Pre)* trả về số cách xây dựng tiếp từ i để thu được xâu nghiệm nếu như phần nghiệm đã xây dựng khớp đến vị trí *k* trên xâu *S* và ký tự *i-1* là *Pre*

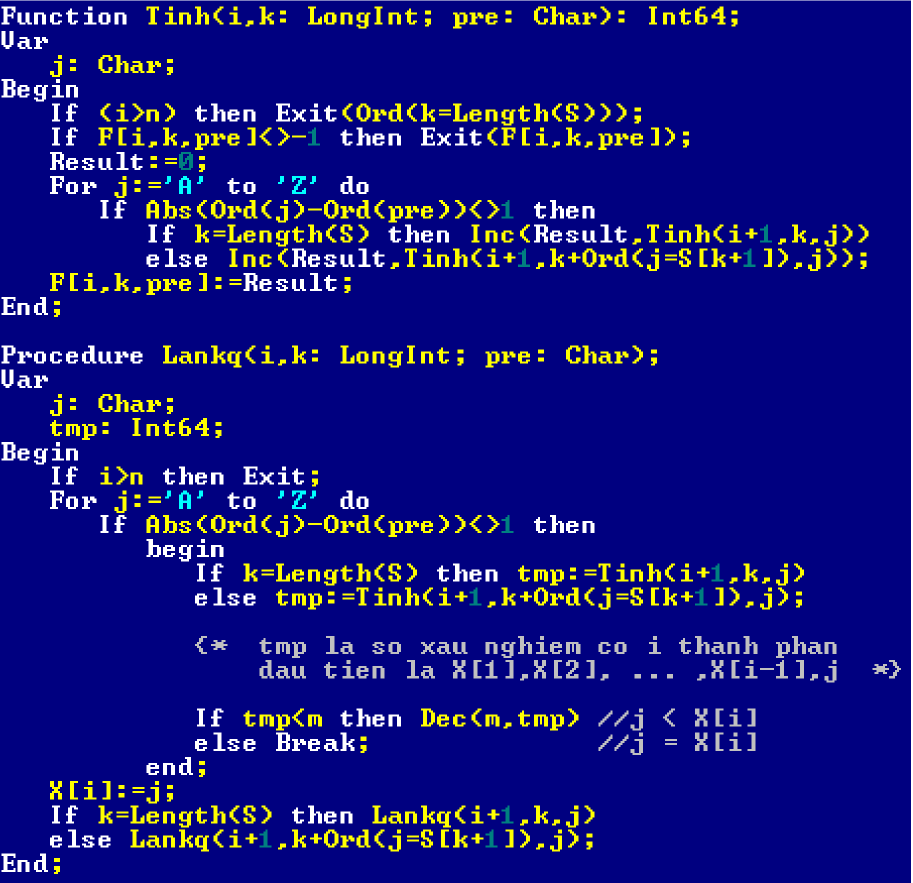
Hàm *Lankq(i,k,Pre)* sẽ xây dựng vị trí thứ *i* của *X*; *k* và *Pre* là 2 đặc tính của phần *X* đã xây dựng, nó được truyền đi để mô tả *X* mà không cần *For* lại

Giả sử *X[i]=j0*. Khi đó, ta đếm tất cả xâu nghiệm có dạng *X[1],X[2]...X[i-1]*, j với *j<j0* { tj0 }. Tất cả những xâu nghiệm như thế đều có thứ tự từ điển nhỏ hơn *X*, nên để kiểm soát số lượng của chúng nhỏ hơn *m*, ta sẽ tìm ra *j0* bằng cách for j từ ‘ A ’ đến ‘ Z ’, cho đến ký tự *j* đầu tiên có *tj >= m* thì *j=j0*

Đánh giá đô phức tạp:

Hàm Tinh() mất O( n\*Length(S)\*26\*26 )

Hàm Lankq() mất O(n\*26)



# Bài toán thứ tự từ điển – bài toán đếm

*Bản chất bài toán thứ tự từ điển chính là bài toán đếm*: Để tìm thứ tự từ điển của nghiệm *X* nào đó, ta đếm số nghiệm nhỏ hơn *X*.

*Bài toán thứ tự từ điển trong bài toán đếm*: Một lớp các bài toán đếm có dính lứu đến thứ tự từ điển, như cách mà tôi đề cập, chúng ta thường dùng biến ok: Boolean để kiểm soát. VD: Bài PFNUM (ở mục VI), ta dùng oka,okb để kiểm soát thứ tự từ điển xâu X. Thay vào đó, ta đưa bài toán về 2 bài toán con: Viết tất cả số PFNUM lên bảng, tìm thứ tự xâu B và xâu A-1 (kết quả sẽ là hiệu của chúng) => Hàm Tinh() nhanh gấp 4 lần.

Đặc biệt, khi có nhiều truy vấn, như cách làm cũ ta phải FillChar lại mảng F rồi tính lại, VD bài N13 (ở mục VI). Còn nếu dùng cách “thứ tự từ điển”: Ta viết tất cả các nghiệm lên bảng, với mỗi truy vấn [A,B], ta tìm thứ tự của B, của A-1 (kết quả là hiệu của chúng). Độ phức tạp cho hàm Tinh() giảm đi 4 lần, và mỗi truy vấn trả lời trong O(n\*10)

# Quan hệ giữa quy hoạch động và đệ quy có nhớ

*Mọi bài toán giải được bằng quy hoạch động đều giải được bằng đệ quy có nhớ.*