ĐỆ QUY CÓ NHỚ

I: Các bài toán giải bằng đệ quy

Xét bài toán đếm số đối tượng (hoặc tìm đối tượng tốt nhất, hoặc đếm số đối tượng 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 $x_1,x_2...x_k$ hợp thành thì bài toán có thể giải bằng đệ quy (xét tất cả các khả năng của $x_1,x_2...x_k$)

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. $1 \le k \le n \le 1000$. Do kết quả có thể rất lớn, chỉ cần in ra theo $mod\ 10^9+7$

II: Đệ quy thuần túy

Xem đoạn code sau:

```
{$MODE OBJFPC}

Const nmax=20;
      Base=1000000007;

Type Nghiem=Array [1..nmax] of LongInt;

Var tmp: Nghiem;
    n,k: LongInt;

Function Check(i: LongInt; X: Nghiem): Boolean;
Var j,s: LongInt;
Begin
   s:=0;
   For j:=1 to i do
      If X[j]=1 then
         begin
            Inc(s);
            If s=k then Exit(False);
         end
      else s:=0;
   Result:=True;
End;

Function Tinh(i: LongInt; X: Nghiem): LongInt;
Var
   j: LongInt;
Begin
   If i>n then Exit(1);
   Result:=0;
   For j:=0 to 1 do
      begin
         X[i]:=j;
         If Check(i,X) then Result:=(Result+Tinh(i+1,X)) Mod Base;
      end;
End;

BEGIN
   Read(n,k);
   WriteLn(Tinh(1,tmp));
END.
```

Hàm *Tinh(i,X)* trả về số cách xây dựng phần còn lại của *X* để 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 $O(n*2^n)$

Để ý, tại mỗi bước ta phải gọi hàm *Check()* để kiểm tra có nối được *j* vào sau *X* hay không. Thay vì gọi hàm *Check()* mất O(n), ta có thể vừa đi vừa tính được *s*: Số lượng bit 1 tận cùng của *X* => Chỉ mất O(1) để check. Ngoài ra, ta cũng không cần lưu lại *X* nữa, vì việc lưu chỉ để *Check()* – mà ta đã dùng cách khác. Đoạn code được viết lại:

```
Function Tinh(i: LongInt; s: LongInt): LongInt;
Var
   j: LongInt;
Begin
   If i>n then Exit(1);
   Result:=0;
   For j:=0 to 1 do
         If j=0 then Result:=(Result+Tinh(i+1,0)) Mod Base
         else
            If s+1<k then Result:=(Result+Tinh(i+1,s+1)) Mod Base;
End;
```

Với cách cài trên, độ phức tạp còn $O(2^n)$ (Bỏ qua được hàm Check())

III: Sự trùng lặp các bài toán:

Ta thấy ngay, với mỗi bộ *(i,s)* hàm *Tinh()* trả ra 1 giá trị xác định. Như vậy số bài toán là không quá *iMax*sMax* <=n^2. Trong khi số bộ *(i,X)* là n*2^n => Số bài toán đã giảm xuống rất nhiều!!!

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 ta chỉ giải một *X* rồi áp đặt kết quả cho các *X* khác. VD minh họa:

k=5,n=100. Xét 2 lần gọi hàm: *Tinh(10,000000111)* và *Tinh(10,101100111)*. Rõ ràng số cách xây dựng phần còn lại của *X1* và của *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 *Tinh(i,X1)* và *Tinh(i,X2)* có chung lời giải. Việc có nhiều bài toán có cùng 1 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).

IV: Mảng nhớ, đệ quy có nhớ

Cả 2 cách viết trên đều có độ phức tạp là O(2^n) (vì đệ quy *n* bước, mỗi bước có 2 khả năng)

Theo cách viết thứ 2, sẽ có rất nhiều lần gọi đến *Tinh(i_0,s_0)*. Mỗi lần gọi, ta lại phải tính lại; mặc dù kết quả so với lần gọi trước không có gì thay đổi! Ta tối ưu thuật toán bằng cách: Với mỗi bộ *(i,s)*, thay vì giải chúng nhiều lần, ta lưu lại lời giải để khi gọi đến lần tiếp theo ta có ngay kết quả mà không cần tính lại. Kết quả cho *Tinh(i,s)* sẽ được lưu vào mảng *F[iMax,sMax]*; gọi là mảng nhớ

```
Function Tinh(i: LongInt; s: LongInt): LongInt;
Var
   j: LongInt;
Begin
   If F[i,s]<>-1 then Exit(F[i,s]); //Bai toan da giai
   If i>n then
      begin
         F[i,s]:=1;
         Exit(1);
      end;
   Result:=0;
   For j:=0 to 1 do
         If j=0 then Result:=(Result+Tinh(i+1,0)) Mod Base
         else
            If s+1<k then Result:=(Result+Tinh(i+1,s+1)) Mod Base;
   F[i,s]:=Result;
End;                                  //Luu lai loi giai bai toan
```

Thấy ngay số bài toán là n^2, mỗi bài toán giải mất O(2) (gọi đến 2 bài toán con). Và không có bài nào được giải 2 lần => ĐPT $O(2*n^2)$ (ĐPT giảm xuống đáng kể ☺)

Cách đẩy mảng nhớ vào đệ quy để khử trùng lặp như thế gọi là đệ quy có mảng nhớ hay đệ quy có nhớ

V: Các bước giải bài toán bằng đệ quy có nhớ

1. Tìm cấu hình nghiệm của bài toán. VD bài toán trên có cấu hình nghiệm là $X=x_1,x_2..x_n$ với x_i là các bit
   Như đã nói, ta sẽ xây dựng nghiệm X bằng cách xây dựng từng thành phần x_i của nó (theo một thứ tự nào đó tùy bài toán)
2. Tìm các ràng buộc của nghiệm. VD bài toán trên ràng buộc nghiệm là không có k bit 1 liên tiếp và độ dài là n.
3. *Kiểm soát* rằng: các ràng buộc đó đều được thỏa mãn trong quá trình xây dựng nghiệm, bằng cách:
   - Tìm bộ chỉ số sẽ mô tả các tính chất (mà ta quan tâm) của phần nghiệm đã xây dựng (Như VD trên, ta quan tâm 2 tính chất là độ dài phần nghiệm đã xây dựng và số bit 1 tận cùng của nó)
   - Bộ chỉ số *phải đủ cơ sở* để đảm bảo việc xây dựng các thành phần tiếp theo của nghiệm sẽ thỏa mãn được các ràng buộc
   - Thường thì với mỗi ràng buộc, ta sẽ dùng 1 chỉ số để kiểm soát (Như VD trên, s sẽ kiểm soát rằng độ dài mọi dãy bit 1 liên tiếp đều nhỏ hơn k)
4. Viết hàm đệ quy để xây dựng từng thành phần một của nghiệm. Bộ chỉ số để thỏa mãn ràng buộc *sẽ được truyền đi*
5. Đẩy mảng nhớ vào

VI: Một số bài toán minh họa

N13

John không hề thích con số 13 vì theo John đó là số không may mắn. Trong một lần phải liệt kê các số tự nhiên từ A đến B, John muốn lọc ra các số mà trong dạng biểu diễn của nó không xuất hiện số 13. Ví dụ số 111539786 không xuất hiện số 13, còn số 113 thì có xuất hiện số 13.

Yêu cầu: Cho A, B hãy xác định số lượng các số nằm trong đoạn [A, B] mà trong dạng biểu diễn của nó không xuất hiện số 13.

Input:

- Gồm nhiều dòng, mỗi dòng chứa 2 số nguyên A, B $(0 \leq A \leq B \leq 10^{15})$

Output:

- Gồm nhiều dòng, mỗi dòng là số lượng tìm được tương ứng với file dữ liệu vào.

N13.INP	N13.OUT
1 13 100 1000	12 882

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

1. Cấu hình nghiệm: $X = x_1 x_2 ... x_n$ với $x_i \in [0,9]$
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
4. 5.

```
Function Tinh(i: LongInt; oka,okb,pre: Boolean): Int64;
Var
   j: Char;
Begin
   If F[i,oka,okb,pre]<>-1 then exit(F[i,oka,okb,pre]);
   If i>n then
      begin
         F[i,oka,okb,pre]:=1;
         exit(1);
      end;
   Result:=0;
   For j:='0' to '9' do
      If   ((j>=A[i]) or oka)
      and  ((j<=B[i]) or okb)
      and  ((not pre) or (j<>'3'))
      then Inc(Result,Tinh(i+1,oka or (j>A[i]),okb or (j<B[i]),j='1'));
   F[i,oka,okb,pre]:=Result;
End;
```

(VD khá đơn giản, không đọc tiếp nếu chưa hiểu)

PFNum

Một xâu được gọi là xâu đối xứng nếu đọc từ trái qua phải cũng bằng việc đọc từ phải qua trái. Một số nguyên n được gọi là số PFNum nếu trong biểu diễn của nó không chứa xâu đối xứng có độ dài lớn hơn 1.

Ví dụ, số 16276 là số PFNum, còn số 17276 không là số PFNum.

Yêu cầu: Cho đoạn số nguyên $[A, B]$, hãy đếm số lượng số PFNum trong $[A, B]$.

Input

- Gồm một dòng chứa hai số nguyên A, B $(A, B \leq 10^{18})$.

Output

- Gồm một dòng chứa một số nguyên là số lượng số PFNum trong $[A, B]$.

PFNum.inp	PFNum.out
123 321	153

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

1. Cấu hình nghiệm: $X = x_1 x_2 ... x_n$ với $x_i \in [0,9]$
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*)

4. 5.

```
Var F          : Array [1..19,Boolean,Boolean,Boolean,'/'..'9','/'..'9'] of LongInt;
    A,B        : String[19];
    n          : LongInt;

Function Tinh(i: LongInt; ok,oka,okb: Boolean; pre1,pre2: Char): Int64;
Var
   j,tmp: Char;
Begin
   If F[i,ok,oka,okb,pre1,pre2]<>-1 then exit(F[i,ok,oka,okb,pre1,pre2]);
   If i>n then
      begin
         F[i,ok,oka,okb,pre1,pre2]:=1;
         exit(1);
      end;
   Result:=0;

   For j:='0' to '9' do

      If ((j>=A[i]) or oka)                //Kiem soat >= A

      and ((j<=B[i]) or okb)               //Kiem soat <= B

      and ((j<>pre1) and (j<>pre2))        //Kiem soat X khong chua xau doi xung

      then

         begin
            If (not ok) and (j='0') then tmp:='/'
            else tmp:=j;
            Inc(Result,tinh(i+1,ok or (j>'0'),oka or (j>A[i]),okb or (j<B[i]),pre2,tmp));
         end;
   F[i,ok,oka,okb,pre1,pre2]:=Result;
End;
```

NUM68

Một số nguyên dương N $(N > 1)$ luôn có thể biểu diễn dưới dạng tổng hai số nguyên dương A và B $(N = A + B;\ A \leq B)$. Trong bài toán này chúng ta sẽ quan tâm đến các cách biểu diễn một số nguyên N thành tổng hai số nguyên dương A và B thỏa mãn tính chất: trong biểu diễn của A hoặc B phải chứa chữ số 6 hoặc chữ số 8.

Ví dụ: $N = 10$, có tất cả 5 cách biểu diễn nhưng chỉ có 2 cách biểu diễn thỏa mãn là: 2+8; 4+6.

Yêu cầu: Cho số nguyên dương N $(N > 1)$, hãy đếm số cách cách biểu diễn N thành tổng hai số nguyên dương A và B thỏa mãn tính chất: trong biểu diễn của A hoặc B phải chứa chữ số 6 hoặc chữ số 8.

Input

- Gồm nhiều dòng, mỗi dòng tương ứng với một số nguyên N $(1 < N \leq 10^{18})$.

Output

- Gồm nhiều dòng, mỗi dòng là kết quả tương ứng với dữ liệu vào.

NUM68.INP	NUM68.OUT
10 19	2 4

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

1. Cấu hình nghiệm: $X = \begin{cases} A = a_1 a_2 ... a_n \\ B = b_1 b_2 ... b_n \end{cases}$ với $a_i, b_i \in [0,9]$
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 x_n về x_1, dùng biến *nho* (nhớ) để kiểm soát $(a_i+b_i+nho)\ mod\ 10=n_i$
   - Dùng biến *ok*: Boolean: Phần nghiệm đã xây dựng có chứa 6 hoặc 8?

4. 5.

```
Function Tinh(i,nho: ll; ok: Boolean): Int64;
Var
   j,k: ll;
Begin
   If i=0 then exit(Ord(ok and (nho=0)));   //nghiem da xay dung thoa man
   If F[i,nho,ok]<>-1 then exit(F[i,nho,ok]);

   Result:=0;
   For j:=0 to 9 do
      For k:=0 to 9 do
         If (j+k+nho) mod 10=key[i] then
            Inc(Result,tinh(i-1,(j+k+nho) div 10,ok or (j=6) or (j=8) or (k=6) or (k=8)));
   F[i,nho,ok]:=Result;
End;


Procedure Process;
Begin
   FillChar(F,SizeOf(F),255);
   Res:=Tinh(n,0,false);
   If Res mod 2=0 then                    //Tat ca cac nghiem deu duoc tinh 2 lan
      Res:=Res div 2
   else                                   //Co 1 nghiem duoc tinh 1 lan
      Res:=(Res+1) div 2;

   For n:=1 to n do
      If (key[n]=8) or (key[n]=6) then    //Bo (0,N) bi loai
         begin
            Dec(Res);
            break;
         end;
End;
```

ESTR

Xâu A gọi là xuất hiện trong B nếu A là một đoạn con gồm các phần tử liên tiếp của B. Cho xâu S chỉ chứa các chữ cái latin hoa. Đếm số xâu độ dài n, chỉ chứa các ký tự latin hoa và S xuất hiện trong đó, in ra phần dư khi chia cho 10^9+7

ESTR.inp

- Dòng đầu chứa số nguyên dương n
- Dòng thứ 2 chứa xâu S

ESTR.out

- Một số nguyên duy nhất là kết quả bài toán

ESTR.inp	ESTR.out
5 THAI	52

- Length(S), n ≤ 1000

1. Cấu hình nghiệm: $X = x_1 x_2 ... x_n$ với $x_i \in ['A','Z']$
2. Các ràng buộc của nghiệm:
   - Xâu *S* xuất hiện trong *X*
3. Bộ chỉ số mô tả phần nghiệm đã xây dựng, giúp kiểm soát ràng buộc:

- *i*: Vị trí mà ta đang xây dựng
- *ok*: Boolean: Phần nghiệm xây dựng được đã chứa /chưa chứa xâu S
- *Match:* Cho biết vị trí trên xâu S mà phần X xây dựng đã khớp được

4. 5. (Hãy đọc code và "phiêu" :v)

```
Procedure Init;
Var
   i: LongInt;
   j: Char;
   Tmp: String;
Begin
   S:=S+'.';
   For i:=0 to Length(S)-1 do
      For j:='A' to 'Z' do
         begin
            Tmp:=Copy(S,1,i)+j;
            While Tmp<>Copy(S,1,Length(Tmp)) do Delete(Tmp,1,1);
            Next[i,j]:=Length(Tmp);
         end;
   Delete(S,Length(S),1);
   FillChar(F,SizeOf(F),255);
End;

Function Tinh(i: LongInt; ok: Boolean; Match: LongInt): LongInt;
Var
   j: Char;
Begin
   If F[i,ok,Match]<>-1 then Exit(F[i,ok,Match]);
   If (i>n) then
      begin
         F[i,ok,Match]:=Ord(ok);
         Exit(F[i,ok,Match]);
      end;
   Result:=0;

   For j:='A' to 'Z' do
      Result:=(Result+Tinh(
                           i+1
                           ,ok or (Next[Match,j]=Length(S))
                           ,Next[Match,j])
                     ) Mod Base;

   F[i,ok,Match]:=Result;
End;
```

(Mình cố tình code đoạn Init $O(n^3)$. Các bạn tự cải tiến!)

VII: Một vài bộ chỉ số và thứ tự xây dựng giúp thỏa 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:
   - Nếu thứ tự xây dựng nghiệm như nhau, bộ chỉ số là *bộ các bộ* trên
   - Nếu khác nhau thứ tự xây dựng, tìm 1 thứ tự hợp lý, thay đổi một số bộ

- VD1: Khi buộc phải xây dựng nghiệm theo chiều ngược mà phải kiểm soát ràng buộc về thứ tự từ điển, ta dùng biến *ok*: boolean… (bạn đọc tự nghĩ, bài toán VD: vuive9 Đỗ Đức Đông)
- VD2: Khi xây dựng nghiệm theo chiều xuôi mà phải kiểm soát tổng / hiệu / tích, ta dùng biến *PhaiNho*: LongInt để buộc bước tính tiếp theo nhớ một lượng bằng *PhaiNho*
- ...

Trong thực tế sẽ có rất nhiều loại ràng buộc khác nhau. Cách duy nhất là làm nhiều bài tập ➔ quen :3

VIII: Bài tập vận dụng

Các bài tập của Thầy Đỗ Đức Đông

➔ https://www.mediafire.com/?ccp2e11c9ib69m8

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. Đếm số xâu đẹp độ dài n nhận S làm xâu con, in ra phần dư khi chia cho 10^9+7

FSTR.inp

- Dòng đầu chứa số nguyên dương n
- Dòng thứ 2 chứa xâu S

FSTR.out

- Một số nguyên duy nhất là kết quả bài toán

FSTR.inp	FSTR.out
4 thai	1

- Length(S), n ≤ 100

Hướng dẫn: Liên hệ: Thai9cdb@gmail.com

IX: 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

Bài toán mở đầu:

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*. Xem mã nguồn:

```
Function Tinh(i,k: LongInt; pre: Char): Int64;
Var
   j: Char;
Begin
   If (i>n) then Exit(Ord(k=Length(S)));
   If F[i,k,pre]<>-1 then Exit(F[i,k,pre]);
   Result:=0;
   For j:='A' to 'Z' do
      If Abs(Ord(j)-Ord(pre))<>1 then
         If k=Length(S) then Inc(Result,Tinh(i+1,k,j))
         else Inc(Result,Tinh(i+1,k+Ord(j=S[k+1]),j));
   F[i,k,pre]:=Result;
End;

Procedure Lankq(i,k: LongInt; pre: Char);
Var
   j: Char;
   tmp: Int64;
Begin
   If i>n then Exit;
   For j:='A' to 'Z' do
      If Abs(Ord(j)-Ord(pre))<>1 then
         begin
            If k=Length(S) then tmp:=Tinh(i+1,k,j)
            else tmp:=Tinh(i+1,k+Ord(j=S[k+1]),j);

            {*  tmp la so xau nghiem co i thanh phan
                dau tien la X[1],X[2], ... ,X[i-1],j  *}

            If tmp<m then Dec(m,tmp) //j < X[i]
            else Break;              //j = X[i]
         end;
   X[i]:=j;
   If k=Length(S) then Lankq(i+1,k,j)
   else Lankq(i+1,k+Ord(j=S[k+1]),j);
End;
```

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 $X[i]=j_0$. Khi đó, ta đếm tất cả xâu nghiệm có dạng $X[1],X[2]...X[i-1],j$ với $j<j_0$ $\{t_{j0}\}$. 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 j_0 bằng cách *for j* từ 'A' đến 'Z', cho đến ký tự *j* đầu tiên có $t_j >= m$ thì $j=j_0$

Đánh giá đô phức tạp:

- Hàm Tinh() mất O(n*Length(S)*26*26)
- Hàm Lankq() mất O(n*26)

X: 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*. Như VD trên, nếu đề bài là cho xâu *X*, tìm thứ tự của nó trên bảng thì ta giải như sau:

```
Procedure Lankq(i,k: LongInt; pre: Char);
Var
   j: Char;
Begin
   If i>n then Exit;
   For j:='A' to Chr(Ord(X[i])-1) do
      If Abs(Ord(j)-Ord(pre))<>1 then
         If k=Length(S) then Inc(Res,Tinh(i+1,k,j))
         else Inc(Res,Tinh(i+1,k+Ord(j=S[k+1]),j));

   If k=Length(S) then Lankq(i+1,k,X[i])
   else Lankq(i+1,k+Ord(j=S[k+1]),X[i]);
End;
```

Kết quả là *Res+1*

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)

Bài tập vận dụng:

FNUMBER

Số nguyên dương n được gọi là FNUMBER nếu n chia hết cho 3 nhưng không chia hết cho 9 và ở dạng biểu diễn thập phân của n không có 2 chữ số nào có tổng chia hết cho 5. VD 240 là một FNUMBER, 231 không là một FNUMBER.

Cho trước 2 số nguyên dương A,B. Đếm số lượng FNUMBER thuộc [A;B]. Do kết quả có thể rất lớn, chỉ cần in ra theo modun 10^9+7.

Dữ liệu vào từ tệp FNUMBER.INP:

- Dòng 1 chứa số Q: số lượng testcase.
- Q dòng tiếp theo, mỗi dòng chứa 2 số A,B.

Dữ liệu xuất ra tệp FNUMBER.OUT:

- Gồm Q dòng là kết quả cho Q testcase.

FNUMBER.INP	FNUMBER.OUT
3 100 110 5 2345 0 100000	1 209 3616
1 0 1000000000000000000	698089379

Giới hạn:

- Subtask 1: $Q \le 10^6$. $0 \le A \le B \le 10^5$
- Subtask 2: $Q=1$. $0 \le A \le B \le 10^{18}$
- Subtask 3: $Q \le 100$. $0 \le A \le B \le 10^{10000}$

Hướng dẫn, bộ test: Thai9cdb@gmail.com

XI: 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. VD bài QBMAX có thể viết như sau:

```
Function Best(i,j: LongInt): LongInt;
Begin
   If (i=0) or (i>n) then Exit(-oo);
   If (j>m) then Exit(0);
   If F[i,j]<>-1 then Exit(F[i,j]);
   Result:=a[i,j]+Max(Best(i-1,j+1),Max(Best(i,j+1),Best(i+1,j+1)));
   F[i,j]:=Result;
End;
```

Điều ngược lại không đúng! Trong nhiều trường hợp, đệ quy có nhớ chạy nhanh hơn quy hoạch động (vì có thể nó không giải hết tất cả các bài toán con). Một số bài toán, tìm ra công thức quy hoạch động, cơ sở quy hoạch động nhưng thứ tự giải các bài toán rất khó để thiết kế vòng *for*; khi đó, đệ quy có nhớ là 1 giải pháp tuyệt vời! Cũng có 1 số bài toán, số bài toán con rất lớn khiến mảng lưu trữ lời giải không đủ, do đó dùng QHĐ là không khả thi; đệ quy có nhớ cho phép giảm bớt không gian mảng lưu trữ - chịu thiệt một ít trong độ phức tạp tính toán

Tuy vậy, QHĐ cũng không mất đi chỗ đứng của mình, vì nó cài đặt khá đơn giản, trực quan, và trong nhiều trường hợp có thể bạn sẽ không muốn truy vết trong đệ quy có nhớ! Do đó cần linh hoạt trong sử dụng QHĐ và đệ quy có nhớ, cần chọn cách làm trực quan dễ hiểu, dễ cài đặt!

Thanks!

Question?