MỤC LỤC

CHUYÊN ĐỀ

QUY HOẠCH ĐỘNG CHỮ SỐ - DIGIT DP

PHẦN I: MỞ ĐẦU

Quy hoạch động (QHĐ) là một kĩ thuật rất hiệu quả giải nhiều bài toán tin học, đặc biệt là những bài toán tối ưu. Trong các cuộc thi Olympic tin học hiện đại, QHĐ luôn là một trong những chủ đề chính.

QHĐ phân chia bài toán thành các bài toán con, các bài toán con phụ thuộc nhau, mỗi bài toán con có thể tham chiếu tới cùng một số bài toán con ở mức dưới (gọi là các bài toán con gối lên nhau). Vì vậy trong QHĐ, lời giải mỗi bài toán con cần được lưu giá trị trong một bảng phương án để không phải giải lại bài toán con ấy mỗi khi những bài toán con mức trên tham chiếu tới kết quả của nó.

QHĐ thường dùng một trong hai cách tiếp cận

- Tiếp cận từ đáy lên *(Bottom-up)*: Toàn bộ các bài toán con nhỏ nhất cần phải giải trước và sau đó sử dụng chúng để tính kết quả của bài toán lớn hơn. Quá trình cứ tiếp tục như thế cho đến khi bài toán cuối cùng được giải là bài toán đã cho ban đầu.
- Tiếp cận từ đỉnh xuống *(Top-down*): thường dùng với đệ quy gọi giải ngay bài toán ban đầu. Bài toán ban đầu được phân thành các bài toán con, các bài toán con này được chương trình đệ quy phân tích giải và tổ chức ghi nhớ kết quả để phục vụ trường hợp cần tham chiếu tới. Tiếp theo các bài toán con lại được phân chia thành nhiều bài toán con nhỏ hơn nữa, các bài toán này lại được đệ quy phân tích giải và được tổ chức ghi nhớ, quá trình tiếp tục cho đến khi tới thời điểm dừng của đệ quy gặp các bài toán con nhỏ nhất (thấy ngay cách giải chúng, kết quả được gán trực tiếp vào đầu bảng phương án). Khi thực hiện, đệ quy lại làm theo kiểu như bottom-up, nó giải từ những bài toán con nhỏ nhất, dựa trên những ghi nhớ từng bước nó lần ngược về giải bài toán ban đầu. Cách tiếp cận này là sự kết hợp của đệ quy và ghi nhớ.

Trong khuôn khổ chuyên đề "***Quy hoạch động chữ số - Digit DP***" tôi xin trình bày về một lớp các bài toán yêu cầu đếm số lượng số nguyên x nằm giữa hai số nguyên a và b sao cho x thỏa mãn một hoặc một số thuộc tính cụ thể có thể liên quan đến các chữ số của nó với cách tiếp cận *Top-down*

Code và Test của các bài tập trong chuyên đề Thầy cô có thể tham khảo theo đường dẫn sau: https://bit.ly/2GVtXR5

PHẦN II: NỘI DUNG

I. LÝ THUYẾT

Ngay từ tên gọi "Digit DP" ta dễ dàng đoán rằng chúng ta sẽ làm gì đó bằng cách sử dụng các chữ số.

Gọi $G(Num)$ là số lượng số nguyên nằm trong đoạn $[0..Num]$ thỏa mãn yêu cầu đề bài.

Khi đó số lượng số nguyên thuộc đoạn $[a, b]$ thỏa mãn điều kiện bài cho được tính bằng $G(b) - G(a-1)$

Hoặc $G(b) - G(a) + tmp$; trong đó $tmp = 1$ nếu a là số thỏa mãn điều kiện bài cho, $tmp = 0$ nếu a không thỏa mãn điều kiện (thường kiểm tra trực tiếp a).

Ý tưởng chính của Digit DP là trước tiên ta biểu diễn các chữ số của Num dưới dạng một mảng các chữ số hoặc dưới dạng xâu.

Giả sử Num có n chữ số, biểu diễn thập phân là $t_{n-1}t_{n-2} \dots t_1t_0$ trong đó t_i $(0 \le i \le n-1)$ cho biết chữ số thứ i từ bên phải sang trái.

Sau khi biểu diễn số đã cho theo cách này, chúng ta tạo ra các số $X = x_{n-1}x_{n-2} \dots x_1x_0$ nhỏ hơn hoặc bằng Num và đồng thời tính toán bằng cách sử dụng DP, nếu X thỏa mãn tính chất đã cho.

Bắt đầu tạo các số nguyên có số chữ số bằng 1, và lần lượt cho đến khi số chữ số bằng n.

Thực tế chúng ta luôn tạo ra các số nguyên có đúng n chữ số, với những số nguyên mà có số chữ số ít hơn n thì ta coi các chữ số tận cùng bên trái của nó bằng 0.

Nhắc lại các bước giải bài toán bằng đệ quy có nhớ

1. Tìm cấu hình nghiệm của bài toán, giả sử $X = x_{n-1}x_{n-2} \dots x_1x_0$

   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.
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.
   - 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.
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.

II. VÍ DỤ MINH HỌA

Ví dụ 1:

Cho hai số nguyên A và B. Yêu cầu in ra tổng của tất cả các chữ số xuất hiện trong các số nguyên từ A đến B.

Input: Một dòng duy nhất chứa hai số nguyên A, B $(1 \leq A < B \leq 10^{15})$

Output: Một dòng duy nhất chứa một số nguyên là kết quả của bài toán.

Ví dụ:

SUMDG.INP	SUMDG.OUT
5 11	38

Giải thích: Tổng cần tìm là 38 = (5 + 6 + 7 + 8 + 9 + 1 + 0 + 1 + 1)

Hướng dẫn:

Xây dựng hàm $G(Num)$: tổng của tất cả các chữ số xuất hiện trong các số thuộc đoạn $[0, Num]$.

Ta coi Num là 1 xâu độ dài n có biểu diễn thập phân là $t_{n-1}t_{n-2} \dots t_1t_0$

Chúng ta tạo ra các số $X = x_{n-1}x_{n-2} \dots x_1x_0$ nhỏ hơn hoặc bằng Num đồng thời thực hiện tính toán với nhận xét rằng nếu chúng ta đã tính toán câu trả lời cho trạng thái có $n-1$ chữ số là $x_{n-2} \dots x_1x_0$ và chúng ta cần tính câu trả lời cho trạng thái có n chữ số là $x_{n-1}x_{n-2} \dots x_1x_0$ thì rõ ràng chúng ta có thể sử dụng kết quả của trạng thái trước đó thay vì tính toán lại.

Ta sẽ xây dựng X bằng cách xây dựng từng thành phần x_i của nó theo thứ tự từ trái qua phải.

Trạng thái QHĐ của chúng ta sẽ là $(id, smaller, sum)$ trong đó:

1) id : Vị trí ta đang xây dựng

2) $smaller$: bằng $0/1$ với ý nghĩa sau:

- $smaller = 1$: cho biết phần phần nghiệm đã xây dựng có thứ tự từ điển nhỏ hơn hẳn Num. Khi đó chữ số đặt tại vị trí id không bị giới hạn, có thể nhận giá trị từ 0 đến 9.
- $smaller = 0$: cho biết phần nghiệm đã xây dựng là tiền tố của Num (giống hệt phần đầu của Num). Khi đó chữ số đặt tại vị trí id bị giới hạn, chỉ có thể nhận giá trị từ 0 đến $Num[id]$.

Ví dụ : $Num = 3245$ và chúng ta cần tính $G(3245)$

chỉ số	3	2	1	0
chữ số	3	2	4	5

Phạm vi không giới hạn:

Giả sử số nguyên tạo được cho đến bây giờ là: 3 1 * *
(* là chỗ trống, nơi các chữ số được chèn vào để tạo thành số nguyên).
Ta thấy tại vị trí thứ 1, chữ số đặt vào không bị giới hạn phạm vi, có thể nhận các giá trị từ 0 đến 9.
Đối với phạm vi không giới hạn thì $smaller = 1$

Phạm vi bị hạn chế:

Giả sử số nguyên được tạo cho đến bây giờ là: 3 2 * *

Ta thấy tại vị trí thứ 1, chữ số đặt vào bị hạn chế, chỉ có thể nhận các giá trị từ 0 đến 4

Đối với phạm vi bị hạn chế thì $smaller = 0$

3) *sum*: Tham số này sẽ lưu trữ tổng các chữ số trong phần nghiệm đã xây dựng (từ vị trí $id - 1$ đến $n - 1$)

Chuyển trạng thái

Chúng ta xây dựng theo kiểu Top-down.

Bắt đầu ở vị trí chữ số bên trái nhất: $id = n - 1, smaller = 0; sum = 0.$

Giả sử hiện tại ta đang ở trạng thái $(id, smaller, sum)$: Chúng ta sẽ lần lượt điền vào vị trí id một chữ số trong phạm vi từ 0 đến $limit$ ($limit \leq 9$, tùy thuộc vào giá trị $smaller$) và gọi đệ quy để điền chữ số tiếp theo ở vị trí $id - 1$, giá trị sum và $smaller$ được tính lại và truyền đi:

$$sum(mới) = sum(cũ) + chữ\ số\ được\ chọn.$$

Cách tính lại giá trị $smaller$ cho bước gọi đệ quy tiếp?

Nếu $smaller$ đang là 1, thì trong bước gọi đệ quy tiếp $smaller$ vẫn là 1

Nếu $smaller$ đang là 0 và chữ số điền ở vị trí id là dig thì

Nếu $dig < Num[id]$ thì trong bước gọi đệ quy tiếp $smaller$ là 1

Nếu $dig = Num[id]$ thì trong bước gọi đệ quy tiếp $smaller$ la 0

Tóm lại: $smaller(mới) = smaller(cũ) \ || \ (dig < Num[id])$

Cài đặt:

```
#include <bits/stdc++.h>
#define NAME "SUMDG."
using namespace std;
long long dp[17][2][140];

/* chuyển số n thành xâu s(lưu dưới dạng đảo ngược, chữ số hàng đơn vị của
n tương ứng với s[0]) */
string tostring(long long n)
{
    string s;
    while(n!=0)
```

```
    {
        s+=(n%10)+'0';
        n/=10;
    }
    return s;
}

long long  calc(string s,int idx, int smaller, int sum)
{
    if(idx == -1) //đã xây dựng xong X (X <= s)
        return sum;

    if(dp[idx][smaller][sum]!=-1) //bài toán con này đã được giải rồi
        return dp[idx][smaller][sum]; //lấy luôn kết quả trong mảng nhớ

    //bài toán con chưa được giải → ta đi giải nó
    long long sol=0;
    int limit = (smaller? 9: s[idx]-'0');
    for (int i = 0; i <= limit; i++)
        sol += calc(s,idx - 1, smaller || (i < s[idx]-'0'),sum + i);

    dp[idx][smaller][sum]=sol; //ghi nhớ lại kết quả bài toán con
    return sol;
}

// Tính tổng của tất cả các chữ số xuất hiện trong đoạn [0,x]
long long solve(long long x)
{
    string s = tostring(x);
    int ls = s.length()-1;
    memset(dp,-1,sizeof(dp));
    return calc(s,ls,0,0);
}

int main()
{
    freopen(NAME"inp","r",stdin);
    freopen(NAME"out","w",stdout);
    long long a,b;
    cin>>a>>b;
    cout<<solve(b) - solve(a-1)<<endl;
    return 0;
}
```

Độ phức tạp

Số trạng thái là $id * smaller * sum$, chi phí chuyển trạng thái là 10

Độ phức tạp thời gian $O(10 * id * smaller * sum)$

($smaller = 2$; giới hạn các số nguyên trong bài là 10^{15} nên $max\ id = 16$, sum lớn nhất là 15*9).

Test: ***https://bit.ly/33hPdYG***

Ví dụ 2:

Có bao nhiêu số nguyên x trong đoạn $[A, B]$ mà x có chứa đúng K chữ số d

Input: Một dòng duy nhất chứa bốn số nguyên A, B, d, K ($1 \le A \le B \le 10^{15}$; $0 \le d \le 9$; $0 \le K \le 20$)

Output: Một dòng duy nhất chứa một số nguyên là kết quả của bài toán.

Ví dụ:

DP2.INP	DP2.OUT
11 100 2 1	17

Hướng dẫn:

Xây dựng hàm $G(Num)$: số lượng các số thuộc $[0, Num]$ thỏa mãn đề bài.

Ta coi Num là 1 xâu độ dài n, có biểu diễn thập phân là $t_0 t_1 \dots t_{n-2} t_{n-1}$

Cấu hình nghiệm: $X = x_0 x_1 \dots x_{n-2} x_{n-1}$ với với $x_i \in [0, 9]$

Ta sẽ xây dựng X bằng cách xây dựng từng thành phần x_i của nó theo thứ tự từ trái qua phải.

Trạng thái QHĐ của chúng ta sẽ là $(\boldsymbol{id}, \boldsymbol{cnt}, \boldsymbol{smaller}, \boldsymbol{nonz})$, trong đó:

- $\boldsymbol{id}$ là vị trí chữ số ta đang xây dựng
- $\boldsymbol{cnt}$: Số lượng chữ số d xuất hiện trong phần nghiệm đã xây dựng $X[0..id-1]$
- $\boldsymbol{smaller}$:
  $smaller = 1$: phần nghiệm đã xây dựng $X[0..id-1] < Num[0..id-1]$
  $smaller = 0$: phần nghiệm đã xây dựng $X[0..id-1] = Num[0..id-1]$
- $\boldsymbol{nonz}$: Cho biết tính đến vị trí $id-1$ số X đã có nghĩa hay chưa (hay chỉ chứa toàn số 0). Quy ước = 1: đã có nghĩa, = 0: chỉ chứa toàn 0

Cài đặt:

```
#include <bits/stdc++.h>
using namespace std;
const int M = 20;
long long dp[M][M][2][2];
int d, K;

string tostring(long long n) //chuyển số n thành xâu s
{
    string s;
    while(n!=0)
    {
        s+=(n%10)+'0';
        n/=10;
    }
    reverse(s.begin(), s.end());
    return s;
```

```
}

long long  count(int id, int cnt, int smaller,int nonz, string &num)
{
    if (id == (int)num.length())  //đã xây dựng xong X
    {
        if (cnt == K)
            return 1;
        return 0;
    }

    if (dp[id][cnt][smaller][nonz] != -1)
        return dp[id][cnt][smaller][nonz];

    long long ans = 0;
    int limit = (smaller ? 9 : num[id]-'0');
    for (int dig = 0; dig <= limit; dig++)
    {
        int currCnt = cnt;
        if (dig == d)
        {
            if (d != 0 || (!d && nonz))
                currCnt++;
        }

        int currSmaller = smaller;
        if (dig < num[id]-'0')
            currSmaller = 1;

        ans += count(id + 1, currCnt,currSmaller, nonz || (dig != 0), num);
    }
    return dp[id][cnt][smaller][nonz] = ans;
}


long long  solve(long long x)
{
    string s = tostring(x);
    memset(dp, -1, sizeof(dp));
    return count(0, 0, 0, 0, s);
}

int main()
{
    freopen("dp2.inp","r",stdin);
    freopen("dp2.out","w",stdout);
    long long  L,R;

    cin>>L>>R>>d>>K;
    cout << solve(R) - solve(L - 1) << endl;
    return 0;
}
```

Độ phức tạp

Số trạng thái là $id * cnt * smaller * nonz$,

Chi phí chuyển trạng thái là 10 (xét tất cả các chữ số 0 đến 9 để điền vào vị trí id).

Độ phức tạp thời gian $O(10 * id * cnt * smaller * nonz)$

($smaller = 2$; $nonz = 2$, giới hạn các số nguyên trong bài là 10^{15} nên $max\ id = 16$, $max\ cnt = 15$)

Test: ***https://bit.ly/2FecjHP***

III. BÀI TẬP

Bài 1: N13 *(Nguồn bài: Thầy Đỗ Đức Đông)*

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 (không quá 3000), mỗi dòng chứa 2 số nguyên ($0 \leq A \leq B \leq 10^{15}$)

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

Ví dụ:

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

Hướng dẫn:

Xây dựng hàm $G(Num)$: số lượng các số thuộc $[0, Num]$ thỏa mãn đề bài.

Ta coi Num là 1 xâu độ dài n

- ✓ Cấu hình nghiệm: $X = x_0 x_1 \dots x_{n-2} x_{n-1}$ với với $x_i \in [0, 9]$
- ✓ Các ràng buộc của nghiệm:
  - Thứ tự từ điểm xâu X nhỏ hơn hoặc bằng xâu Num
  - X không chứa số 13

Ta sẽ xây dựng X bằng cách xây dựng từng thành phần x_i của nó theo thứ tự từ trái qua phải.

Trạng thái QHĐ của chúng ta là $(\boldsymbol{id}, \boldsymbol{smaller}, \boldsymbol{pre})$ trong đó :

- ○ $\boldsymbol{id}$: Vị trí mà ta đang xây dựng
- ○ $\boldsymbol{smaller}$
  $smaller = 1$: cho biết phần nghiệm đã xây dựng có thứ tự từ điển nhỏ hơn hẳn xâu Num
  $smaller = 0$: cho biết phần nghiệm đã xây dựng là tiền tố của Num
- ○ $\boldsymbol{pre}$: $0/1$ cho biết $X[i-1]$ có phải bằng 1 hay không?

Cài đặt:

```
#include <bits/stdc++.h>
using namespace std;
const int M = 16;
long long dp[M][2][2];
```

```
string tostring(long long n)
{
    string s;
    while(n!=0)
    {
        s+=(n%10)+'0';
        n/=10;
    }
    reverse(s.begin(), s.end());
    return s;
}

long long  count(int id, int smaller, int pre, string num)
{
    if (id == num.length()) return 1;
    if (dp[id][smaller][pre] != -1) return dp[id][smaller][pre];

    long long ans = 0;
    int limit = (smaller ? 9 : num[id]-'0');

    for (int dig = 0; dig <= limit; dig++)
    {
        int currSmaller = smaller;
        if (dig < num[id]-'0') currSmaller = 1;
        if (dig==3 && pre ==1) continue;
        ans += count(id + 1,currSmaller, (dig == 1), num);
    }
    return dp[id][smaller][pre] = ans;
}
long long  solve(long long x)
{
    string s = tostring(x);
    memset(dp, -1, sizeof(dp));
    return count(0, 0, 0, s);
}
int main()
{
    freopen("n13.inp","r",stdin);
    freopen("n13.out","w",stdout);
    long long  L,R;

    while (cin>>L>>R)
    {
        cout << solve(R) - solve(L - 1) << endl;
    }
    return 0;
}
```

Độ phức tạp

Số trạng thái là $id * smaller * sum$, chi phí chuyển trạng thái là 10

Độ phức tạp thời gian cho mỗi test $O(10 * id * smaller * sum)$

($smaller = 2$; giới hạn các số nguyên trong bài là 10^{15} nên $max\ id = 16$, sum lớn nhất là 15*9).

Độ phức tạp toàn bài cỡ 3000*4000 ~ 10^8

Test: ***https://bit.ly/3k7EhUn***

Bài 2: **Tổng các chữ số** *(Nguồn: COCI 2007)*

Có bao nhiêu số nguyên trong đoạn $[A, B]$ có tổng chữ số của nó bằng K và số nhỏ nhất là số nào?

Input: Một dòng duy nhất chứa ba số nguyên A, B, K $(1 \leq A \leq B \leq 10^{18};\ 1 \leq K \leq 135)$

Output:

- Dòng đầu tiên chứa số lượng số nguyên trong đoạn $[A, B]$ có tổng các chữ số của nó bằng K
- Dòng thứ hai chứa số nguyên nhỏ nhất trong số đó

Dữ liệu vào luôn đảm bảo rằng giá trị nhỏ nhất của dòng đầu là 1.

Ví dụ:

CUDAK.INP	CUDAK.OUT
1 100 10	9 19

Hướng dẫn:

Gọi $f(x)$ là số lượng số nguyên dương trong đoạn [0, x] có tổng các chữ số bằng K.

Khi đó câu trả lời đầu tiên của bài toán là $f(B) - f(A - 1)$.

Để tính $f(x)$ làm tương tự như Ví dụ 1, chỉ cần thay đoạn code:

```
if(id == -1)
    {
        return sum;
    }
```

Thành đoạn code:

```
if(id == -1)
    {
        if(sum == K)
            return 1;
        return 0;
    }
```

Để trả lời câu hỏi thứ hai ta cần tìm số nhỏ nhất $x \in [A, B]$ có $f(x) > f(A-1)$. Việc này có thể thực hiện bằng tìm kiếm nhị phân.

Cài đặt:

```
#include <bits/stdc++.h>
#define NAME "cudak."
using namespace std;
int K;
long long a, b;
long long dp[19][2][163];
string tostring(long long n)
{
    string s;
    while(n!=0)
    {
        s+=(n%10)+'0';
        n/=10;
    }
    return s;
}

long long  calc(string &s,int id, int smaller, int sum)
{
    if(id == -1) //da xay dung xong x (x<=s)
    {
        if(sum == K)
            return 1;
        return 0;
    }

    if(dp[id][smaller][sum]!=-1)
        return dp[id][smaller][sum];

    long long sol = 0;
    int limit = smaller? 9: s[id]-'0';


    for(int i=0 ; i<=limit ; i++)
    {
        int currSmaller = smaller;
        if (i < s[id]-'0')
            currSmaller = 1;
        sol += calc(s, id - 1, currSmaller, sum + i);
    }

    dp[id][smaller][sum] = sol;
    return sol;
}
```

```
long long solve(long long x)
{
    string s=tostring(x);
    int ls = s.length()-1;
    memset(dp,-1,sizeof(dp));
    return calc(s,ls,0,0);
}

int main()
{
    freopen(NAME"inp","r",stdin);
    freopen(NAME"out","w",stdout);

    cin>>a>>b>>K;
    long long a1 = solve(a-1);
    cout<<solve(b) - a1 <<endl;

    long long dau,cuoi,kq;
    dau = a, cuoi = b;
    while (dau <= cuoi)
    {
        long long giua = (dau + cuoi)/2;
        long long tmp = solve(giua);
        if (tmp == a1)
            dau = giua + 1;
        else
        {
            kq = giua;
            cuoi = giua-1;
        }
    }
    cout<<kq;
    return 0;
}
```

Độ phức tạp:

Số trạng thái *là* $id * smaller * sum,$ chi phí chuyển trạng thái là 10

Tổng độ phức tạp thời gian tính hàm $f(x)$ là $O(10 * id * smaller * sum)$

$max\ id\ = 19, sum = 18 * 9, smaller = 2 -> 10^5$

ĐPT toàn bài $O(\log(10^{15})*10^5) \sim 5.10^6$

Test: https://bit.ly/32l31Cv

Bài 3: Số dễ chịu

Một số nguyên dương được gọi là số dễ chịu nếu các chữ số của nó xuất hiện theo trình tự không giảm, ví dụ 1111, 123, 88999, . . . là những số dễ chịu.

Yêu cầu: Cho 2 số nguyên a và b ($0 < a \leq b \leq 10^{100}$). Hãy xác định số lượng số dễ chịu trong đoạn $[a, b]$. Kết quả có thể rất lớn vì vậy chỉ cần đưa ra theo mô đun 10^9+7.

Input*:*

- Dòng đầu tiên chứa số nguyên a,
- Dòng thứ 2 chứa số nguyên b.

Output: Một số nguyên là số số lượng số dễ chịu tìm được theo mô đun 10^9+7.

Ví dụ*:*

PLEASANT.INP	PLEASANT.OUT
1 100	54

Hướng dẫn:

Cũng tương tự như các bài toán trước, ta xây dựng hàm $G(Num)$: số lượng các số thuộc $[0, Num]$ thỏa mãn đề bài.

Khi đó số lượng số thuộc đoạn $[a, b]$ thỏa mãn điều kiện bài là:

$$res = G(b) - G(a-1)$$

Tuy nhiên bài cho giá trị a, b lớn, vượt quá giới hạn 64 bít. Để tránh việc phải tính $a-1$ ta có thể tính res theo cách khác:

$$res = G(b) - G(a) + tmp\ ;$$

trong đó $tmp = 1$ nếu a là số thỏa mãn điều kiện bài cho, $tmp = 0$ nếu a không thỏa mãn điều kiện (thường kiểm tra trực tiếp a).

Việc tích lũy kết quả được thực hiện theo mô đun 10^9+7 vì vậy không cần xử lý số lớn.

Ta coi Num là 1 xâu độ dài n

- ✓ Cấu hình nghiệm: $X = x_0 x_1 \ldots x_{n-2} x_{n-1}$ với với $x_i \in [0, 9]$
- ✓ Các ràng buộc của nghiệm:
  - Thứ tự từ điểm xâu X nhỏ hơn hoặc bằng xâu Num
  - $x_{i-1} \leq x_i$ với $\forall i \in [1..n-1]$

Ta sẽ xây dựng X bằng cách xây dựng từng thành phần x_i của nó theo thứ tự từ trái qua phải.

Trạng thái QHĐ của chúng ta là $(\boldsymbol{id}, \boldsymbol{smaller}, \boldsymbol{last})$ trong đó :

- ***id*:** Vị trí mà ta đang xây dựng

- ***smaller***
  $smaller = 1$: cho biết phần nghiệm đã xây dựng có thứ tự từ điển nhỏ hơn hẳn xâu Num
  $smaller = 0$: cho biết phần nghiệm đã xây dựng là tiền tố của Num
- ***last*** **:** lưu giá trị đã được điền ở bước trước : $last = X[i-1]$

Cài đặt:

```
#include <bits/stdc++.h>
#define task "PLEASANT"
#define ll long long
#define FOR( i, l, r) for(int i = l; i <= r; ++i)
#define FOD( i, l, r) for(int i = l; i >= r; --i)
#define endl '\n'

using namespace std;
int const N = 102;
int const mod = 1e9 + 7;
int dp[N][2][10];
string a, b;

int calc(int id, int smaller, int last, string s)
{
    if(id == (int) s.size())
        return 1;

    if(dp[id][smaller][last] != -1)
        return dp[id][smaller][last];

    int ans = 0;
    int limit = smaller? 9: s[id] - '0';

    FOR( i, last, limit) ans = (ans + calc( id + 1, (smaller || i < limit),
i, s)) % mod;
    dp[id][smaller][last]=ans;
    return ans;
}

void inp()
{
    cin >> a >> b;
}

void solve()
{
    memset(dp, -1, sizeof(dp));
    int res = calc( 0, 0, 0, b);
    memset(dp, -1, sizeof(dp));
    res = (res - calc( 0, 0, 0, a) + mod) % mod;

    bool check = 1;
    FOR( i, 1, (int) (a.size() - 1))
    if(a[i - 1] > a[i])
```

```
    {
        check = 0;
        break;
    }
    res = (res + check) % mod;
    cout << res;
}

int main()
{
    freopen(task".inp","r",stdin);
    freopen(task".out","w",stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    inp();
    solve();
    return 0;
}
```

Độ phức tạp

Số trạng thái là $id * smaller * last$, chi phí chuyển trạng thái là 10

Tổng độ phức tạp thời gian là $O(10 * id * smaller * last)$

($a, b \leq 10^{100}$ thì max của id là101; $smaller = 2, last = 10$)

Test : https://bit.ly/3m91cQS

Bài 4: Số đặc biệt *(Đề thi liên tỉnh của Lê Quý Đôn – Đà Nẵng)*

Trong lúc chờ Tân loại bỏ bớt các hình dạng không hợp lý cho trò chơi oẳn tù tì, Lương và Định viết ra một con số đặc biệt: 2941999. Sau đó, hai bạn đã rủ Ngọc cùng ngồi giết thời gian bằng một trò chơi thú vị, ba bạn cùng cộng bình phương các chữ số của con số đặc biệt lại và lấy kết quả đó thay thế cho con số hiện tại: $2^2 + 9^2 + 4^2 + 1^2 + 9^2 + 9^2 + 9^2 = 345$. Họ kiên trì lặp lại bước trên: $345 \rightarrow 50 \rightarrow 25 \rightarrow 29 \rightarrow 85 \rightarrow 89 \rightarrow 145 \rightarrow 42$ … cho tới khi nào được kết quả là số 1 thì dừng lại, vì cả ba đều rất ghét con số này (gợi lên sự lẻ loi của những chàng trai FA). Với tài năng toán học thiên bẩm, Ngọc nhận ra rằng nếu xuất phát từ con số 2941999 như trên thì không bao giờ biến đổi được nó về số 1 bằng cách lặp đi lặp lại bước cộng bình phương các chữ số. Lương và Định cũng công nhận điều này và hai bạn nhờ Ngọc xác định giúp xem có bao nhiêu con số đặc biệt như vậy (không thể biến đổi về số 1) trong đoạn các số tự nhiên từ L đến R.

Yêu cầu: Với từng cặp số tự nhiên L và R ($1 \leq L \leq R \leq 10^{18}$), hãy giúp Ngọc xác định số lượng số đặc biệt nằm trong đoạn $[L, R]$.

Input: Vào từ file văn bản PEARL.INP:

- Dòng đầu chứa số nguyên dương T là số lượng câu hỏi của Lương và Định.
- Tiếp đến là T dòng, mỗi dòng chứa hai số tự nhiên L và R biểu thị câu hỏi tương ứng.

Output: Ghi ra file văn bản PEARL.OUT gồm T dòng, mỗi dòng chứa một số nguyên duy nhất là câu trả lời cho câu hỏi tương ứng.

Ví dụ:

PEARL.INP	PEARL.OUT
1 2941999 2942002	3

Ràng buộc:

- Có 20% số test ứng với 20% số điểm của bài có $T \leq 30$ và $R - L \leq 10^6$.
- Có 20% số test khác ứng với 20% số điểm của bài có $T \leq 100$ và $1 \leq L \leq R \leq 10^9$
- 60% số test còn lại ứng với 60% số điểm của bài có $T \leq 100$.

Hướng dẫn:

Mấu chốt của bài toán là nhận ra với mỗi số nguyên dương không vượt quá 10^{18}, sau phép biến đổi đầu tiên sẽ trở thành một số ≤ 9^2 *18 = 1458

Vì vậy đầu tiên chúng ta tìm tập hợp A chứa tất cả các số Pearl từ 1 đến 1458. Sau đó tính xem trong đoạn $[L, R]$ có bao nhiêu số có tổng bình phương các chữ số bằng một số thuộc A.

Sub 1: Kiểm tra từng số. Độ phức tạp $O(n * MAXD)$ (ở đây $MAXD$ là số chữ số tối đa)

Sub 2: Một số có không quá 9 chữ số có thể tách ra như sau: $abcde|xyzt$ (tách thành 2 phần, phần đầu 5 chữ số, phần sau 4 chữ số).

Gọi $S(n)$ là tổng bình phương các chữ số của n.

Ta cần $S(abcde) + S(xyzt) = S(m)$ với $m \in A$

Hay $S(abcde) = S(m) - S(xyzt)$

Vì chỉ có tối đa 1458 số m và 10^4 số $xyzt$ nên ta có thể xét hết tất cả các cặp $(m, xyzt)$ và lưu lại giá trị ở vế phải. Cuối cùng, ứng với mỗi $abcde$ có thể dễ dàng tính được đáp số.

ĐPT : O(|A| * 10^4 + 10^5)

Sub 3: Bài này có thể giải quyết bằng quy hoạch động theo chữ số.

Xây dựng hàm $G(Num)$: số lượng số Pearl trong đoạn $[0, Num]$

Theo như phân tích ở trên $G(Num)$ = số lượng số trong đoạn [0, Num] mà có tổng bình phương các chữ số bằng một số thuộc A

Ta coi Num là 1 xâu độ dài n

✓ Cấu hình nghiệm: $X = x_0 x_1 \dots x_{n-2} x_{n-1}$ với với $x_i \in [0, 9]$

✓ Các ràng buộc của nghiệm:
- Thứ tự từ điểm xâu X nhỏ hơn hoặc bằng xâu Num
- $x_0^2 + x_1^2 + \cdots + x_{n-1}^2 = k \in A$

Ta sẽ xây dựng X bằng cách xây dựng từng thành phần x_i của nó theo thứ tự từ trái qua phải.

Trạng thái QHĐ của chúng ta là ($\boldsymbol{id}, \boldsymbol{sum}, \boldsymbol{smaller}$) trong đó :
- $\boldsymbol{id}$: Vị trí mà ta đang xây dựng
- $\boldsymbol{sum}$: tổng bình phương các chữ số trong phần nghiệm đã xây dựng
- $\boldsymbol{smaller}$
  $smaller = 1$: cho biết phần nghiệm đã xây dựng có thứ tự từ điển nhỏ hơn hẳn xâu Num
  $smaller = 0$: cho biết phần nghiệm đã xây dựng là tiền tố của Num

Duyệt lần lượt từng số k trong tập A, tính số lượng số trong đoạn $[0, Num]$ mà có tổng bình phương các chữ số bằng k.

Cài đặt:

```
#include <bits/stdc++.h>
using namespace std;
#define TASK "pearl"
#define REP(i, a, b) for(int i = (a); i < (b); i++)
#define FORU(i, a, b) for(int i = (a); i <= (b); i++)
#define FORD(i, a, b) for(int i = (a); i >= (b); i--)

const int MAXD = 19; // 10^18
const int MAXS = 9*9*18; //999999999999999999

int digit[MAXD+3];
long long dp[MAXD+3][MAXS+3][2];

/* Chuyển n thành 1 mảng các chữ số digit, chữ số hàng đơn vị của n tương
ứng với digit[0] */
void parse(long long n)
{
    FORU(i, 1, MAXD) digit[i] = n % 10, n /= 10;
}

int sumDigit(int n)  //Tính tổng bình phương các chữ số của n
{
    int ret = 0;
    while(n)
        ret += (n % 10)*(n % 10), n/=10;
    return ret;
}

long long count(int id, int sum, bool smaller)
{
```

```
    if(id == 0)
        return !sum;
    if(dp[id][sum][smaller] != -1)
        return dp[id][sum][smaller];
    long long ret = 0;
    int upper = smaller ? digit[id] : 9;
    FORU(i, 0, upper)
    {
        bool newsmaller = smaller && (i == digit[id]);
        if(sum < i*i)
            break;
        ret += count(id - 1, sum - i*i, newsmaller);
    }
    return (dp[id][sum][smaller] = ret);
}

bool isPearl[MAXS+3];
bool Have[MAXS+3];
int Stack[MAXS+3];

void checkPearl(int n, bool& r)
{
    int Top = 0;
    while(1)
    {
        if(n == 1)
        {
            r = 0;
            break;
        }
        if(Have[n])
        {
            r = 1;
            break;
        }
        Have[n] = 1, Stack[++Top] = n;
        n = sumDigit(n);
    }
    FORU(i, 1, Top) Have[Stack[i]] = 0;
}

long long get(long long n)
{
    if(n == 0)
        return 0;
    parse(n);
    memset(dp, -1, sizeof(dp));
    long long cnt = 0;
    FORU(i, 1, MAXS)
    if(isPearl[i])
```

```
        cnt += count(MAXD, i, 1);
    return cnt;
}

//Tìm các số Pearl trong doan [1..1458], đánh dấu trong mảng isPearl
void init()
{
    FORU(i, 1, MAXS)
    {
        checkPearl(i, isPearl[i]);
    }
}
int main()
{
    freopen(TASK".inp","r",stdin);
    freopen(TASK".out","w",stdout);

    init();
    int TC;
    cin >> TC;
    while(TC--)
    {
        long long L, R;
        cin >> L >> R;
        cout << get(R) - get(L - 1) << endl;
    }
}
```

Test: ***https://bit.ly/3hq3JTl***

Bài 5: Lại là bài toán đếm *(Nguồn bài : Contest Training Phạm Văn Hạnh)*

Các học trò của Giáo sư X đều biết rằng, mỗi khi làm đề thi mà bí bài thì Giáo sư sẽ cho bài toán truyền thống: Cho hai số A và B, hãy tính tổng của hai số đó.

Lần này Giáo sư muốn đổi phong cách một chút bằng cách phát biểu lại đề chút cho hứng thú: Cho một số nguyên dương C, hãy tìm hai số nguyên dương A và B sao cho $C = A + B$.

Tuy nhiên, sợ bài như này khó quá, các thí sinh không làm được nên Giáo sư phát biểu lại đề và thêm một số ràng buộc như sau:

Cho một số nguyên dương C có n chữ số, hãy đếm xem có bao nhiêu số nguyên dương A và B sao cho:

- A và B là những số nguyên dương có n chữ số (không được bắt đầu bằng chữ số 0)
- $A + B = C$
- A, B phải là những số đẹp. Một số gọi là đẹp nếu không có hai chữ số cạnh nhau mà giống hệt nhau. Ví dụ: 1221 không phải là số đẹp nhưng 1212 lại là một số đẹp.

Yêu cầu: Cho số C, hãy đếm xem có bao nhiêu cặp số nguyên dương A và B thỏa mãn yêu cầu của Giáo sư. Vì đáp án rất lớn nên chỉ cần đưa ra phần dư đáp số cho $10^9 + 7$.

Input: Vào từ file văn bản APLUSB.INP gồm một dòng duy nhất chứa một số nguyên C (Số chữ số của C tối đa là 10.000 chữ số)

Output: Ghi ra file văn bản APLUSB.OUT một số nguyên duy nhất là số lượng cặp A, B tìm được theo yêu cầu đề bài.

Ví dụ:

APLUSB.INP	APLUSB.OUT
23	2
100	0

C=23, có 4 cách phân tích là:

1. 10 + 13
2. 11 + 12
3. 12 + 11
4. 13 + 10

Tuy nhiên chỉ có 2 cách 1 và 4 được chấp nhận vì trong hai cách còn lại có số 11 có 2 chữ số liên tiếp giống nhau.

Giới hạn:

- *25% số test có* $C < 1000$
- *25% số test khác có* $C < 10^6$
- *50% số test còn lại không có ràng buộc gì thêm*

Hướng dẫn:

Gọi n là số chữ số của C $(n \leq 10000)$.

Cấu hình nghiệm: $X = \begin{cases} A = a_0\, a_1 \dots a_{n-1} \\ B = b_0 b_1 \dots b_{n-1} \end{cases}$ $với\ a_i,\ b_i \in [0..9]$

Ràng buộc nghiệm:

- A, B có n chữ số $(a_0 \neq 0, b_0 \neq 0)$
- Tổng $A + B = C$
- $a_i \neq a_{i+1},\ b_i \neq b_{i+1}$ $với\ \forall i: 0..n-2$

Ta sẽ xây dựng X bằng cách xây dựng từng thành phần a_i, b_i của nó theo thứ tự từ phải qua trái.

Trạng thái QHĐ của chúng ta là $(\boldsymbol{id}, \boldsymbol{lasta}, \boldsymbol{lastb}, \boldsymbol{carry})$

- **i*d*:** Vị trí mà ta đang xây dựng
- $\boldsymbol{lasta}, \boldsymbol{lastb}$**:** chữ số vừa được điền ở bước trước
- $\boldsymbol{carry}$**:** là phần NHỚ (liệu chữ số thứ $id + 1$ của $A + B$ có nhớ (ví dụ 5+5=10 → có nhớ), ta phải kiểm soát điều kiện $(a_i + b_i + carry)\ mod\ 10 = C_i$

Đặc biệt phải lưu ý trong trường hợp khi $id = 0$ thì chữ số được điền vào phải khác không để đảm bảo điều kiện A , B có n chữ số.

Cài đặt:

```
#include <bits/stdc++.h>
using namespace std;
#define For(i,a,b) for(int i=a;i<=b;i++)
#define ll long long
#define TASK "APLUSB"
const ll oo = 1e15+7 ;
const ll mod = 1e9+7 ;
const int N = 1e4+7 ;

ll f[N][11][11][2] ;
string s ;
int n ;
ll calc(int id,int lasta,int lastb,int carry)
{
    if(id < 0) return (carry == 0) ;
    if(f[id][lasta][lastb][carry] != -1) return f[id][lasta][lastb][carry];
    ll tmp = 0 ;
    int k = 0 ;
    if(id == 0) k = 1 ;
    For(u,k,9)
    {
        For(v,k,9)
        {
            int sum = u + v + carry ;
            if(sum % 10 == s[id] - '0' && u != lasta && v != lastb)
                tmp = (tmp+calc(id-1,u,v,sum/10))%mod ;
        }
    }
    return f[id][lasta][lastb][carry] = tmp % mod ;
}
int main()
{
    freopen(TASK".INP","r",stdin) ;
    freopen(TASK".OUT","w",stdout) ;
    ios_base::sync_with_stdio(0) ;
    cin.tie(NULL) ;
    cout.tie(NULL) ;
    cin >> s ;
    n = s.size() - 1 ;
    memset(f,-1,sizeof(f)) ;
    cout << calc(n,10,10,0)%mod;
    return 0;
}
```

Độ phức tạp

Số trạng thái : $id * lasta * lastb * carry$, chi phí chuyển trạng thái là 10*10

Độ phức tạp toàn bài : $O(id * lasta * lastb * carry * 10 * 10)$

$(max\ id = 10000, lasta = 10, lastb = 10, carry = 2)$

Test: ***https://bit.ly/3ipBnda***

<u>Bài 6</u>: NUM68 *(Nguồn bài: Thầy Đỗ Đức Đông)*

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 (không quá 50 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.

Ví dụ:

NUM68.INP	NUM68. OUT
10 19	2 4

Hướng dẫn:

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.

Cấu hình nghiệm: $X = \begin{cases} A = a_0\, a_1 \dots a_{n-1} \\ B = b_0 b_1 \dots b_{n-1} \end{cases}$ $với\ a_i,\ b_i \in [0..9]$

Ràng buộc 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

Ta sẽ xây dựng X bằng cách xây dựng từng thành phần a_i, b_i của nó theo thứ tự từ phải qua trái.

Nhận xét: Ràng buộc 2 có thể bỏ qua, khi đó mỗi nghiệm (A, B) sẽ được đếm 2 lần, (trừ bộ $A = B$). Do vậy sau khi tính, giả sử kết quả là Res thì ta xử lí như sau:

Nếu Res chẵn: khi đó tất cả các nghiệm đều được tính 2 lần → $output = Res/2$

Nếu Res lẻ: có 1 nghiệm được tính 1 lần → $output = (Res + 1)/2$

Trạng thái QHĐ của chúng ta là $(\boldsymbol{id}, \boldsymbol{carry}, \boldsymbol{ok}, \boldsymbol{lead1}, \boldsymbol{lead2})$

- **$\boldsymbol{id}$:** Vị trí mà ta đang xây dựng
- ***carry*:** là phần NHỚ (liệu chữ số thứ $id + 1$ của $A + B$ có nhớ (ví dụ 5+8=13 → có nhớ), ta phải kiểm soát điều kiện $(a_i + b_i + carry)\ mod\ 10 = N_i$
- ***ok*:** phần nghiệm xây dựng đã ch
-
- ứa 6 hoặc 8 chưa?
- $\boldsymbol{lead1}, \boldsymbol{lead2}$**:** nhận giá trị 1/0, cho biết tính đến vị trí $id + 1$ số A, B đã có nghĩa hay chưa (hay chỉ chứa toàn số 0). Quy ước =1 (đã có nghĩa); =0 (chứa toàn 0)

Cài đặt cách 1:

```
#define FOR(i, l, r) for(int i = l; i <= r; i++)
#define task "NUM68"
#define int long long
using namespace std;

string n;
int dp[20][2][2][2][2];

int calc(int id, int carry, int ok, int lead1, int lead2, string &digit)
{
    if(id < 0)
    {
        if(carry)
            return 0;
        if(ok && lead1 && lead2) return 1;
        else return 0;
    }
    if(dp[id][carry][ok][lead1][lead2] != -1)
        return dp[id][carry][ok][lead1][lead2];
    int &ret = dp[id][carry][ok][lead1][lead2];
    ret = 0;
    FOR(i, 0, 9) FOR(j, 0, 9)
    {
        if((i + j + carry)%10 != digit[id]-'0')
            continue;
        ret = ret + calc(id - 1, (i + j + carry)/10, (ok | ((i == 6 ) || (i
== 8) || (j == 6 ) || (j == 8))), (lead1 | (i != 0)), (lead2 | (j != 0)),
digit);
    }
    return ret;
}

main()
{
    freopen(task".inp", "r", stdin);
    freopen(task".out", "w", stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
```

```
    while(cin >> n)
    {
       int len = n.size() - 1;
        memset(dp, -1, sizeof(dp));
        cout << (calc(len, 0, 0, 0, 0, n) + 1)/2 << '\n';
    }
    return 0;
}
```

Độ phức tạp

Số trạng thái : $id * carry * ok * lead1 * lead2$), chi phí chuyển trạng thái là 10*10

ĐPT cho mỗi test là : $O(10 * 10 * id * carry * ok * lead1 * lead2)$

(id lớn nhất =19, carry=2, ok=2, lead1=2, lead2= 2. Do vậy $10 * 10 * id * carry * ok * lead1 * lead2 \sim 30000$)

Cải tiến:

Ràng buộc 1 có thể bỏ qua, ta không cần dùng biến $\boldsymbol{lead1}$, $\boldsymbol{lead2}$.

Khi đó ta có thể đếm thừa bộ $(0, N)$. Sau khi tính, ta kiểm tra nếu N chứa 6 hoặc 8 thì ta giảm Res đi 1 đơn vị

Cài đặt cách 2

```
#include <bits/stdc++.h>
#define FOR(i, l, r) for(int i = l; i <= r; i++)
#define task "NUM68"
#define int long long
using namespace std;

string n;
int dp[20][2][2];

int calc(int id, int carry, int ok, string &digit)
{
    if(id < 0)
    {
        if(carry == 0 && ok)
            return 1;
        return 0;
    }
    if(dp[id][carry][ok] != -1)
        return dp[id][carry][ok];
    int &ret = dp[id][carry][ok];
    ret = 0;
    FOR(i, 0, 9) FOR(j, 0, 9)
    {
        if((i + j + carry)%10 != digit[id]-'0')
            continue;
        ret = ret + calc(id - 1, (i + j + carry)/10, (ok | ((i == 6 ) || (i
== 8) || (j == 6 ) || (j == 8))), digit);
```

```
    }
    return ret;
}
main()
{
    freopen(task".inp", "r", stdin);
    freopen(task".out", "w", stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    while(cin >> n)
    {
        memset(dp, -1, sizeof(dp));
        int len = n.size() - 1;
        int Res = (calc(len, 0, 0,n) + 1)/2;
        for (int i =0; i<n.size(); i++)
            if (n[i] == '6' || n[i] =='8')
            {
                Res --;
                break;
            }
        cout << Res << '\n';
    }
         return 0;
}
```

Test: https://bit.ly/32lGhSI

Bài 7: 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

Hướng dẫn:

Xây dựng hàm $G(Num)$: số lượng các số thuộc $[0, Num]$ thỏa mãn đề bài.

Ta coi Num là 1 xâu độ dài n

✓ Cấu hình nghiệm: $X = x_0x_1 \dots x_{n-2}x_{n-1}$ với với $x_i \in [0, 9]$
✓ Các ràng buộc của nghiệm:
- Thứ tự từ điển xâu X nhỏ hơn hoặc bằng xâu Num
- X không chứa xâu đối xứng nào

Ta sẽ xây dựng X bằng cách xây dựng từng thành phần x_i của nó theo thứ tự từ trái qua phải.

Trạng thái QHĐ của chúng ta là $(\boldsymbol{id}, \boldsymbol{smaller}, \boldsymbol{nonz}, \boldsymbol{pre1}, \boldsymbol{pre2})$ trong đó :

- $\boldsymbol{id}$: Vị trí mà ta đang xây dựng
- $\boldsymbol{smaller}$
  $smaller = 1$: cho biết phần nghiệm đã xây dựng có thứ tự từ điển nhỏ hơn hẳn xâu Num
  $smaller = 0$: cho biết phần nghiệm đã xây dựng là tiền tố của Num
- $\boldsymbol{nonz}$: cho biết tính đến vị trí $\boldsymbol{id - 1}$, số X đã có nghĩa chưa (hay chỉ chứa toàn số 0)
- $\boldsymbol{pre1}, \boldsymbol{pre2}$: cho biết 2 kí tự ngay trước đó đã được xây dựng ($pre1 = X[id - 2]$; $pre2 = X[id - 1], quy$ ước $X[-2] = X[-1] = 10$)

  Nhận 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). Do đó ta chỉ lần lưu trữ 2 kí tự ngay trước đó đã được xây dựng để kiểm soát ràng buộc này

Cài đặt:

```
#include <bits/stdc++.h>
#define task "PFNum"
#define ll long long
#define FOR( i, l, r) for(int i = l; i <= r; ++i)
#define FOD( i, l, r) for(int i = l; i >= r; --i)
#define endl '\n'
using namespace std;
int const N = 20;
ll dp[N][2][2][11][11];
ll a, b;

string tostring(ll n)
{
    string s;
    while(n!=0)
    {
        s+=(n%10)+'0';
        n/=10;
    }
    reverse(s.begin(), s.end());
    return s;
}
ll calc(int id, int smaller, int ok, int pre1, int pre2, string s)
```

```
{
    if(id == (int) s.size())
        return 1;
    if(dp[id][smaller][ok][pre1][pre2]!= -1)
        return dp[id][smaller][ok][pre1][pre2];

    ll ans = 0;
    int limit = smaller? 9: s[id] - '0';

    FOR( i, 0, limit)
    {
        if (i != pre1 && i != pre2)
        {
            int tmp;
            if (!ok && i == 0)
                tmp = 10;
            else
                tmp = i;
            ans = ans + calc( id + 1, (smaller || i < limit),ok || (i > 0),
pre2, tmp, s);
        }
    }
    dp[id][smaller][ok][pre1][pre2]=ans;
    return ans;
}
int  solve()
{string s1, s2;
    s1 = tostring(b);
    memset(dp, -1, sizeof(dp));
    ll res1 = calc(0, 0, 0, 10, 10, s1);

    s2 = tostring(a-1);
    memset(dp, -1, sizeof(dp));
    ll res2 = calc(0, 0, 0, 10, 10, s2);
    cout<<res1 - res2;
}

int main()
{
    freopen(task".inp","r",stdin);
    freopen(task".out","w",stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> a >> b;
    solve();
    return 0;
}
```

Độ phức tạp

Số trạng thái là $id * smaller * nonz * pre1 * pre2$, chi phí chuyển trạng thái là 10

Tổng độ phức tạp thời gian là $O(10 * id * smaller * nonz * pre1 * pre2)$

***Test:* https://bit.ly/3mgUE2Q**

Bài 8: Chef và số đặc biệt *(https://www.codechef.com/problems/WORKCHEF)*

Chef tham gia vào một kỳ thi toán. Nhiệm vụ chính của kỳ thi là tính số lượng số K-đặc biệt trong khoảng $[L, R]$ (tức là $L, L+1, L+2, \ldots, R$). Một số X được gọi là K-đặc biệt nếu trong biểu diễn thập phân của nó chứa ít nhất ít nhất K chữ số khác nhau, và X chia hết cho những chữ số đó. Chú ý rằng không có số nguyên dương nào chia hết cho 0.

Ví dụ, số 48 là 1 và 2 đặc biệt, bởi nó chia hết cho các chữ số 4 và 8.

Chef nhờ bạn giúp anh ta. Có Q câu hỏi dạng L, R, K. Với mỗi câu, bạn cần phải trả lời "Có bao nhiêu số K-đặc biệt trong đoạn $[L, R]$?".

Input:

- Dòng đầu tiên chứa một số nguyên Q – số lương câu hỏi Chef nhận được.
- Q dòng tiếp theo là các câu hỏi, dòng thứ i biểu diễn câu hỏi thứ i bằng 3 số nguyên L, R, K.

Output: Với mỗi câu hỏi, in ra một dòng chứa số nguyên là đáp án câu hỏi đó.

Ví dụ:

Input	Output
3 48 48 1 48 48 2 5 15 1	1 1 11

Ràng buộc và Subtask:

Subtask 1 : [10 điểm] : $1 \le Q \le 3, 1 \le L \le R \le 10^6, 0 \le K \le 9$

Subtask 2 : [20 điểm] : $1 \le Q \le 3, 1 \le L \le R \le 10^9, 0 \le K \le 9$

Subtask 3 : [70 điểm] : $1 \le Q \le 3, 1 \le L \le R \le 10^{18}, 0 \le K \le 9$

Hướng dẫn:

Giả sử chúng ta có 1 số nguyên X.

Gọi $lcm(a_1, a_2, \ldots, a_k)$ là bội chung nhỏ nhất của các số $a_1, a_2, \ldots, a_k$

Nếu ta biết được phần dư của X khi chia cho $lcm(a_1, a_2, \ldots, a_k)$, thì ta dễ dàng tìm được phần dư của X khi chia cho từng số $a_1, a_2, \ldots, a_k$

Ta viết số X dưới dạng :

$$x = p * lcm(a_1, a_2, \ldots, a_k) + q, \qquad với\ \ 0 \le q < lcm(a_1, a_2, \ldots, a_k)$$

Khi đó $x\ mod\ a_i = q\ mod\ a_i, \qquad với\ \ ai \mid lcm(a_1, a_2, \ldots, a_k)$

Quay trở lại bài toán ban đầu. Ta cần kiểm tra tính chia hết của X cho các số từ 2 đến 9 ta làm như sau ;

Tính $LCM(2,3,4,5,6,7,8,9) = 2\text{^}3 * 3\text{^}2 * 5 * 7 = 2520$.

Đặt $modv = X \% 2520$

Nếu $modv$ chia hết cho 2 thì X cũng chia hết cho 2. Tương tự với các giá trị 3, 4, 5, 6, 7, 8, 9

Cải tiến:

Nhận xét nếu chữ số tận cùng của X là 0 hoặc 5 thì X chia hết cho 5

Ta chỉ cần tính $LCM(2,3,4,6,7,8,9) = 2\text{^}3 * 3\text{^}2 * 7 = 504$.

Đặt $modv = X \% 504$

Nếu $modv$ chia hết cho 2 thì X cũng chia hết cho 2. Tương tự với các giá trị 3, 4, 6, 7, 8, 9

Để giải quyết bài toán, ta xét 2 trường hợp:

Nếu $K = 0$ thì số lượng số K đặc biệt trong đoạn $[L, R]$ là $R - L + 1$

Nếu $K >= 1$: dùng thuật toán Digit DP

Xây dựng hàm $G(Num)$: số lượng các số K đặc biệt trong đoạn $[0, Num]$

Ta coi Num là 1 mảng độ dài n, mỗi phần tử trong mảng lưu trữ một chữ số của Num

- ✓ Cấu hình nghiệm: $X = x_0 x_1 \dots x_{n-2} x_{n-1}$ với với $x_i \in [0,9]$
- ✓ Các ràng buộc của nghiệm:
  - Thứ tự từ điển X nhỏ hơn hoặc bằng Num
  - X chứa ít nhất K chữ số khác nhau, giả sử là $d_0, d_1, \dots, d_K$ $(1 \le d_i \le 9)$
  - X chia hết cho các số $d_0, d_1, \dots, d_K$

Để kiểm tra xem số X có chứa ít nhất K chữ số phân biệt hay không ta tạo một mặt nạ bit có độ dài 9, bit thứ i $(i = 0..8)$ lưu trữ sự tồn tại của chữ số $i + 1$ trong X. (không cần lưu trữ sự tồn tại của chữ số 0)

Ta sẽ xây dựng X bằng cách xây dựng từng thành phần x_i của nó theo thứ tự từ trái qua phải.

- ❖ **Cách 1:**

Trạng thái QHĐ của chúng ta là $(\boldsymbol{id}, \boldsymbol{smaller}, \boldsymbol{rem}, \boldsymbol{mask})$ trong đó :

- ***id*:** Vị trí mà ta đang xây dựng
- ***smaller***
  $smaller = 1$: cho biết phần nghiệm đã xây dựng có thứ tự từ điển nhỏ hơn hẳn Num
  $smaller = 0$: cho biết phần nghiệm đã xây dựng là tiền tố của Num
- ***rem*:** phần dư của $X[0..id-1]$ khi chia cho 2520

- ***mask***: mặt nạ bít độ dài 9, bít thứ i (i: 0..8) lưu trữ sự tồn tại của chữ số $i + 1$ trong phần nghiệm đã xây dựng với quy ước bít thứ i bằng 1 có nghĩa là chữ số $i + 1$ đã xuất hiện

Cài đặt cách 1:

```
#include <bits/stdc++.h>

using namespace std;

const int LCM = 2520;
const int MAXDIG = 20;
long long memo[MAXDIG][2][LCM][(1 << 9) + 5];
vector<int> dig;
int K;

long long dp(int id, int smaller, int rem, int mask)
{
    long long &res = memo[id][smaller][rem][mask];
    if (res != -1)
    {
        return res;
    }

    res = 0;
    if (id == dig.size())  //Đã xây dựng xong X
    {
        int cnt = 0;
        for (int d = 1; d < 10; d++)
        {
            if (mask & (1 << (d - 1))) //Nếu chữ số d xuất hiện trong X
            {
                if (rem % d == 0) // Nếu X chia hết cho d
                {
                    cnt++;
                }
            }
        }
        if (cnt >= K)
        {
            res = 1;
        }
    }
    else  //Xây dựng tiếp X
    {
        int limit = smaller? 9: dig[id];
        for (int d = 0; d <=limit; d++)
        {
            int newRem = (rem * 10 + d) % LCM;
            int newMask = mask;
            //ghi nhận d xuất hiện trong X, không cần ghi nhận chữ số 0
            if (d != 0)
            {
                newMask = (mask | (1 << (d - 1)));
```

```
            }
            res += dp(id + 1, smaller || (d < limit), newRem, newMask);
        }
    }
    return res;
}

long long solve(long long n)
{
    dig.clear();
    if (n == 0)
    {
        dig.push_back(n);
    }
    while (n)
    {
        dig.push_back(n % 10);
        n /= 10;
    }
    reverse(dig.begin(), dig.end());
    memset(memo, -1, sizeof(memo));
    return dp(0, 0, 0, 0);
}

int main()
{
    int Q;
    //freopen("workchef.inp","r",stdin);
    //freopen("workchef.out","w",stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    scanf("%d", &Q);
    while (Q--)
    {
        long long L, R;
        scanf("%lld %lld %d", &L, &R, &K);
        long long ans = solve(R) - solve(L - 1);
        printf("%lld\n", ans);
    }
    return 0;
}
```

Độ phức tạp:
Số trạng thái là: số chữ số * smaller * rem * mask ~ 19 * 2 * 2520 * 2^9
Chi phí chuyển trạng thái là 10
→ Cách này bị TLE, chỉ được 30/100 điểm

- **Cách 2:**

Trạng thái QHĐ của chúng ta là ($\boldsymbol{id}, \boldsymbol{smaller}, \boldsymbol{mask}, \boldsymbol{modv}, \boldsymbol{last}$) trong đó :

- ***id*:** Vị trí mà ta đang xây dựng
- ***smaller***
  $smaller = 1$: cho biết phần nghiệm đã xây dựng có thứ tự từ điển nhỏ hơn hẳn xâu Num
  $smaller = 0$: cho biết phần nghiệm đã xây dựng là tiền tố của Num
- ***mask*:** mặt nạ bít độ dài 9, bít thứ i (i: 0..8) lưu trữ sự tồn tại của chữ số $i + 1$ trong phần nghiệm đã xây dựng với quy ước bít thứ i bằng 1 có nghĩa là chữ số $i + 1$ đã xuất hiện
- ***modv*:** phần dư của $X[0..id - 1]$ khi chia cho 504
- ***last***: lưu giá trị đã được điền ở bước trước: $last = X[id - 1]$

Cài đặt cách 2:

```
#include <bits/stdc++.h>
#include <string>
#include <cstdlib>

using namespace std;
char s[33];
int len;
long long dp[20][512][504][2];

long long solve(int K,int id=0,int mask=0,int modv=0,int smaller=0,int
last=0)
{
    long long &res= dp[id][mask][modv][smaller];

    if(res!=-1)
        return res;

    if(id==len)
    {
        long long cnt=0;
        for(int j=0; j<9; j++)
            if(mask&(1<<j))
            {
                if(j!=4)
                    cnt+= ( modv%(j+1)==0);
                else
                    cnt+=(last==5 || last==0);
            }
        return cnt>=K;

    }
    res = 0;

    int limit = smaller ? 9 : s[id]-'0' ;

    for(int i=0; i<=limit; i++)
    {
        int new_mask = mask ;
        if(i)
```

```
        {
            new_mask |= (1 << (i-1)) ;
        }
        res += solve(K,id+1,new_mask,(modv*10+i)%504,smaller||i<(s[id]-
'0'),i);
    }
    return res ;
}


int main(void)
{
    std::ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int t,k;
    long long l,r,ans;
    cin>>t;
    while(t--)
    {
        ans=0;
        cin>>l>>r>>k;
        l--;
        if(k==0)
        {
            cout<<r-l<<'\n';
            continue;
        }

        sprintf(s,"%lld",r);
        len=strlen(s);

        memset(dp, -1, sizeof(dp));

        ans+=solve(k);

        sprintf(s,"%lld",l);
        len=strlen(s);
        memset(dp, -1, sizeof(dp));

        ans-=solve(k);

        cout<<ans<<endl;
    }
    return 0;
}
```

Độ phức tạp

Nếu $id < length$, số trạng thái là số chữ số * 2^9 * 504 *2, chi phí chuyển là 10

Nếu $id = length$, nó có 10 cái $last$ tương ứng, mỗi cái $last$ phần check kết quả lại mất 10. Như vậy nó có 2^9 *504 * 2* 10 * 10

Tổng độ phức tạp toàn bài cỡ 2^9 * 504 * 2 * 10 * (số chữ số + 10)

Cách này nhanh hơn so với cách 1, đủ để AC bài toán.

***Link chấm bài**:* https://www.codechef.com/problems/WORKCHEF

Bài 9: SDNum

Cho hai số nguyên dương N và K. Yêu cầu đếm xem có bao nhiêu số nguyên dương X thỏa mãn: X có N chữ số và có ít nhất một hậu tố khác 0 của X (xét trong dạng biểu diễn thập phân) chia hết cho K

Ví dụ:

$N = 1, K = 2$: có 4 số thỏa mãn là {2, 4, 6, 8}

$N = 2, K = 3$: có 48 số thỏa mãn là {13, 23, 30, 33, 60, 90, 21, 51,… }

Input: Gồm một dòng chứa hai số nguyên N, K ($N \leq 1000, K \leq 3000$)

Output: Gồm một dòng chứa một số nguyên là số lượng số SDNum trong $[A, B]$. Kết quả có thể rất lớn vì vậy chỉ cần đưa ra theo mô đun 10^9+7.

SDNum.inp	SDNum.out
2 3	48

Hướng dẫn:

Xét tất cả các số nguyên có N chữ số và đối với mỗi số nguyên, kiểm tra xem có hậu tố nào của số đó chia hết cho K hay không, nếu có thì tăng số lượng các số đó lên 1. Cách làm này chỉ phù hợp với giá trị N nhỏ.

Ta nhận thấy bài toán này có thể được giải một cách đệ quy, trong đó ở mỗi bước chúng ta có thể chọn các chữ số làm hậu tố cho số N chữ số.

✓ Cấu hình nghiệm: $X = x_{n-1}x_{n-2} \dots x_1x_0$ với với $x_i \in [0, 9]$

✓ Các ràng buộc của nghiệm:

- $x_{n-1} > 0$
- Có ít nhất một hậu tố khác 0 của X chia hết cho K

Ta sẽ xây dựng X bằng cách xây dựng từng thành phần x_i của nó theo thứ tự từ phải qua trái.

Trạng thái QHĐ của ta là ($\boldsymbol{id}$, $\boldsymbol{rem}, \boldsymbol{nonz}$) trong đó :

- $\boldsymbol{id}$: Vị trí mà ta đang xây dựng

- ***rem*:** phần dư của hậu tố (chính là phần nghiệm đã xây dựng $x_{id-1} \dots x_1 x_0$) khi chia cho K
- ***nonz*:** Cho biết tính đến vị trí $id - 1$ số X đã có nghĩa hay chưa (hay chỉ chứa toàn số 0). Quy ước nonz = 1: đã có nghĩa; =0: chỉ chứa toàn 0.

❖ Trường hợp cơ sở cho bài toán này:

TH1: là ở vị trí id bất kì nào đó mà có $rem = 0$ và $nonz = 1$:

- Nếu $id = n$: ta được 1 số thỏa mãn
- Nếu $id < n$: tất cả các vị trí từ id đến $n - 1$ ta đều có thể đặt tùy ý các chữ số từ 0 đến 9 (riêng vị trí $n - 1$ đặt các chữ số từ 1 đến 9). Khi đó ta tính luôn được số lượng số có n chữ số và nhân$x_{id-1} \dots x_1 x_0$ làm hậu tố là $9 * 10^{n-id-1}$

TH2: $id = n$ và $rem \neq 0$: số vừa xây dựng xong không thỏa mãn.

❖ Trường hợp đệ quy:

Tại mỗi bước đệ quy, chúng ta tăng độ dài hậu tố lên một và lần lượt đặt tất cả các số nguyên từ 0 đến 9 vào vị trí đó. Tính lại rem và $nonz$ cho phù hợp và chuyển sang bước tiếp theo.

Cài đặt:

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;

ll mod = 1000000007;
ll dp[1005][3005][2];
ll powers[1005];
int n, k;

void init()
{
    ll st = 1;
    for (int i = 0; i <= n; i++)
    {
        powers[i] = st;
        st *= 10;
        st %= mod;
    }

}
ll calc(int id, int rem, int nonz)
{
    // Base case
    if (rem == 0 && nonz)
    {
        if (id != n)
            return (powers[n - id - 1] * 9) % mod;
        else
```

```
            return 1;
    }

    if (id == n)
        return 0;

    if (dp[id][rem][nonz] != -1)
        return dp[id][rem][nonz];

    ll count = 0;
    for (int i = 0; i < 10; i++)
    {
        count = (count + (calc(id + 1, (rem + (i *powers[id]) % k) % k,
nonz || (i>0)))) % mod;
    }

    return dp[id][rem][nonz] = count;
}

int main()
{
    freopen("SDNum.inp","r", stdin);
    freopen("SDNum.out","w",stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin>>n>>k;
    init();
    memset(dp, -1, sizeof(dp));
    cout << calc(0, 0, 0);
    return 0;
}
```

Độ phức tạp:

Số lượng trạng thái: $id * rem * nonz$

Chi phí chuyển trạng thái là 10

Tổng độ phức tạp cỡ O(20*N*K)

Test: https://bit.ly/2FucyOy

Bài 10: NUMTSN-369 Numbers *(https://www.spoj.com/problems/NUMTSN/)*

Một số nguyên dương K được gọi là số 369 nếu thỏa mãn các điều kiện sau:

1. Số lượng chữ số 3, 6, 9 xuất hiện trong biểu diễn thập phân của K phải bằng nhau
2. Số lượng chữ số 3 phải lớn hơn hoặc bằng 1

Ví dụ 12369, 383676989, 396 là các số 369; các số 213, 342143, 111 thì không phải là số 369

Yêu cầu: Cho hai số nguyên dương A, B $(A \leq B)$, tính số lượng số 369 trong đoạn $[A, B]$. Kết quả đưa ra theo modulo 1000000007

Input:

- Dòng đầu tiên chứa số nguyên dương T $(T \leq 100)$ - là số test
- T dòng tiếp theo, mỗi dòng chứa hai số nguyên A, B $(1 \leq A \leq B \leq 10^{50})$

Output: Gồm T dòng, là kết quả tương ứng của từng test

Ví dụ:

NUMTSN.INP	NUMTSN.OUT
3 121 4325 432 4356 4234 4325667	60 58 207159

***Hướng dẫn**:*

Xây dựng hàm $G(Num)$: số lượng các số 369 trong đoạn $[0, Num]$

Ta coi Num là 1 xâu độ dài n: $t_0 t_1 \dots t_{n-2} t_{n-1}$ (t_0 là chữ số hàng cao nhất, t_{n-1} là chữ số hàng đơn vị)

- ✓ Cấu hình nghiệm: $X = x_0 x_1 \dots x_{n-2} x_{n-1}$ với với $x_i \in [0, 9]$
- ✓ Các ràng buộc của nghiệm:
  - Thứ tự từ điển xâu X nhỏ hơn hoặc bằng xâu Num
  - Số lượng chữ số 3, 6, 9 xuất hiện trong X phải bằng nhau và lớn hơn hoặc bằng 1

Ta sẽ xây dựng X bằng cách xây dựng từng thành phần x_i của nó theo thứ tự từ trái qua phải.

Trạng thái QHĐ của ta là ($\boldsymbol{id}, \boldsymbol{smaller}, \boldsymbol{cn3}, \boldsymbol{cn6}, \boldsymbol{cn9}$) trong đó :

- $\boldsymbol{id}$**:** Vị trí mà ta đang xây dựng
- $\boldsymbol{smaller}$
  $smaller = 1$: cho biết phần nghiệm đã xây dựng có thứ tự từ điển nhỏ hơn hẳn xâu Num
  $smaller = 0$: cho biết phần nghiệm đã xây dựng là tiền tố của Num
- $\boldsymbol{cn3}, \boldsymbol{cn6}, \boldsymbol{cn9}$**:** tương ứng là số lượng chữ số 3, 6, 9 xuất hiện trong phần nghiệm đã xây dựng.

Nếu ta dùng mảng nhớ là dp[50] [2] [50] [50] [50] cho ($\boldsymbol{id}, \boldsymbol{smaller}, \boldsymbol{cn3}, \boldsymbol{cn6}, \boldsymbol{cn9}$) thì chắc chắn sẽ TLE.

Vì $X \leq 10^{50}$ và số lượng chữ số 3, 6, 9 xuất hiện trong X phải bằng nhau nên ta suy ra mỗi số 3, 6, 9 không thể xuất hiện nhiều hơn 16 lần (3*17 = 51). Do vậy ta chỉ cần mảng nhớ là dp[50] [2] [18] [18] [18]

Cài đặt cách 1 :

```
#include <bits/stdc++.h>
using namespace std;
#define MOD 1000000007
long long  DP[52][2][18][18][18];
string num, a, b;
long long t;

long long  calc(int id,int smaller, int cn3, int cn6, int cn9 )
{
    if(cn3 >= 17 || cn6 >= 17 || cn9 >= 17)
        return 0;
    if(id == (int)num.size())
    {
        if(cn3==cn6 && cn6==cn9 && cn3>=1 )
            return 1;
        return 0;
    }
    long long &ret = DP[id][smaller][cn3][cn6][cn9];
    if (ret != -1)
        return ret % MOD;

    long long  ans=0;
    int limit = smaller ? 9: num[id] - '0' ;

    for(int i=0; i<=limit; i++)
    {
        ans+= (calc(id+1,smaller || (i < limit),(i == 3) + cn3, (i == 6) +
cn6, (i == 9) + cn9)%MOD);
        ans%=MOD;
    }

    return ret =ans;
}

long long  solve(string x)
{
    num=x;
    memset(DP,-1,sizeof(DP));
    return calc(0,0,0,0,0);
}
int main()
{

    freopen("NUMTSN.inp","r", stdin);
    freopen("NUMTSN.out","w",stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
```

```
    cin>>t;
    while(t--)
    {
        cin>>a>>b;
        long long ans=solve(b) - solve(a);
        int sz = a.length(), c3 = 0, c6 = 0, c9 = 0;
        for(int i = 0; i < sz; i++)
        {
            if(a[i] == '3')
                c3++;
            if(a[i] == '6')
                c6++;
            if(a[i] == '9')
                c9++;
        }
        if(c3 == c6 and c6 == c9 and c3 != 0)
            ans++;
        cout<<(ans + MOD) % MOD<<"\n";
    }
    return 0;
}
```

Độ phức tạp :

Với mỗi test : số trạng thái là $50 * 2 * 18^3$, chi phí chuyển trạng thái là 10 → độ phức tạp mỗi test cỡ O(6.10^6)

Số lượng test T<=100

Độ phức tạp toàn bài là O(T * $6*10^6$) ~ O(6. 10^8) vẫn chưa đủ nhanh để AC bài này.

Cải tiến :

Theo cách làm trên, mỗi test ta đều $memset(dp, -1)$ và tính toán riêng cho từng test.

Một câu hỏi đặt ra là liệu ta có thể sử dụng lại mảng dp ở các test trước để tính toán cho test hiện tại hay không ? Câu trả lời là có!

Thay vì lưu trữ Num dưới dạng 1 xâu $t_0t_1 \dots t_{n-2}t_{n-1}$ với t_0 là chữ số hàng cao nhất, t_{n-1} là chữ số hàng đơn vị rồi xây dựng nghiệm $X = x_0x_1 \dots x_{n-2}x_{n-1}$ bằng cách xây dựng từng thành phần x_i của nó theo thứ tự từ **trái qua phải**.

Thì ta lưu trữ Num dưới dạng 1 xâu: $t_0t_1 \dots t_{n-2}t_{n-1}$ với t_0 là chữ số hàng đơn vị, t_{n-1} là chữ số hàng cao nhất rồi xây dựng nghiệm $X = x_0x_1 \dots x_{n-2}x_{n-1}$ bằng cách xây dựng từng thành phần x_i của nó theo thứ tự từ **phải qua trái**

Mảng nhớ chỉ cần dp[50][18] [18] [18]: chỉ trong trường hợp smaller =1 ta mới động đến mảng nhớ

$dp[id][cn3][cn6][cn9]$: nghĩa là còn id chữ số cần điền (id chữ số có hàng bé nhất), số lượng chữ số 3, 6, 9 trong phần đã điền tương ứng là $cn3, cn6, cn9$ thì có bao nhiêu cách điền thỏa mãn.

Mảng dp ta có thể sử dụng lại được, không phải tính lại mỗi lần, các chữ số ở đầu cụ thể như thế nào không quan trọng, chỉ quan trọng là phần đã điền nó đã bé hơn thôi (tức là $smaller = 1$) → các số đằng sau chọn thoải mái.

Cài đặt cách 2:

```
#include <bits/stdc++.h>
using namespace std;
#define MOD 1000000007
#define mem(a,b) memset(a, b, sizeof(a) )
#define ll long long int

int mx;
string v;
ll dp[50][18][18][18];

ll calc(int id, bool smaller, int cn3, int cn6, int cn9)
{
    if(cn3 >= 18 || cn6 >= 18 || cn9 >= 18)
        return 0;
    if(id == -1)
    {
        return cn3 == cn6 && cn6 == cn9 && (cn9 != 0);
    }

    ll &ret = dp[id][cn3][cn6][cn9];
    if(smaller && ret != -1)
        return ret % MOD;

    ll ans = 0;

    int limit = smaller ? 9: v[id] - '0' ;

    for(int i = 0; i <= limit; i++)
    {
        ans += calc(id - 1, smaller || (i < limit), (i == 3) + cn3, (i == 6)
+ cn6, (i == 9) + cn9) % MOD;
        ans %= MOD;
    }

    if (smaller)
        ret = ans;
    return ans;

}

ll process(string x)
{
    v = x;
    mx = v.size();
```

```
    reverse(v.begin(), v.end());
    return calc(mx - 1, 0, 0, 0, 0);
}
int main()
{
    int t;
    freopen("NUMTSN.inp","r", stdin);
    freopen("NUMTSN.out","w",stdout);
    string a,b;
    cin>>t;
    mem(dp, -1);
    while(t--)
    {
        cin>>a>>b;
        ll ans = process(b) - process(a);
        int sz = a.length(), c3 = 0, c6 = 0, c9 = 0;
        for(int i = 0; i < sz; i++)
        {
            if(a[i] == '3')
                c3++;
            if(a[i] == '6')
                c6++;
            if(a[i] == '9')
                c9++;
        }
        if(c3 == c6 && c6 == c9 && c3 != 0)
            ans++;
        cout<< (ans + MOD) % MOD<<"\n";
    }
    return 0;
}
```

Độ phức tạp

Số trạng thái: 50 * 18^3, chi phí chuyển trạng thái 10

Số lượng test không quan trọng, nó chỉ phụ thuộc vào tổng của độ dài các cái xâu trong các test, ta gọi là sum(|a|+|b|)

ĐPT toàn bài là: 50*18^3*10+sum(|a|+|b|)

Link chấm bài *: https://www.spoj.com/problems/NUMTSN/*

III. BÀI TẬP TỰ LUYỆN

https://codeforces.com/problemset/problem/628/D

https://www.codechef.com/problems/DGTCNT

https://www.spoj.com/problems/CPCRC1D/

https://www.spoj.com/problems/GONE/

https://www.spoj.com/problems/LUCIFER/

https://www.spoj.com/problems/RAONE/

https://vjudge.net/problem/LightOJ-1205

https://codeforces.com/gym/100886/problem/G

PHẦN III: KẾT LUẬN

Chuyên đề đã trình bày một lớp các bài toán bài toán yêu cầu đếm số lượng số nguyên x nằm giữa hai số nguyên a và b sao cho x thỏa mãn một hoặc một số thuộc tính cụ thể có thể liên quan đến các chữ số của nó. Giới hạn của a, b lớn, thường là 10^{15}, 10^{18},.., thậm chí có thể lên tới 10000 chữ số, ta không thể duyệt từng số để kiểm tra điều kiện. Sử dụng phương pháp Quy hoạch động, tiếp cận theo hướng Top-down đã giúp chúng ta giải quyết được lớp bài toán này một cách khá hiệu quả. Những bài tập trong chuyên đề được tôi sưu tầm và lựa chọn ở nhiều nguồn tài liệu khác nhau. Chuyên đề đã được tôi sử dụng trong quá trình giảng dạy học sinh đội tuyển. Kết quả thu được là học sinh có cái nhìn tổng quan về dạng bài, nắm vững những bài cơ bản, từ đó làm nền tảng cho việc giải quyết những bài toán khó hơn.

Do thời gian hạn chế và kiến thức còn chưa được sâu, rộng nên chắc chắn chuyên đề còn nhiều thiếu sót. Tôi rất mong quý thầy cô đồng nghiệp đóng góp ý kiến để chuyên đề hoàn thiện hơn.

Tôi xin chân thành cảm ơn !

TÀI LIỆU THAM KHẢO

[1]. Trần Đỗ Hùng (Chủ biên), Đỗ Đức Đông, Lê Sĩ Quang, *"Chuyên đề bồi dưỡng HSG Tin học THPT – Bài tập Quy hoạch động"*, NXB Giáo dục, 2007.

[2]. Bài tập của Thầy Đỗ Đức Đông.

[3]. Tài liệu tham khảo của các đồng nghiệp khác.

[4]. Các trang web :

https://www.geeksforgeeks.org/

http://codeforces.com/

https://www.codechef.com

https://www.spoj.com

http://vn.spoj.com/

https://vjudge.net/

Người viết chuyên đề

Bùi Thu Hiền – Giáo viên Trường THPT Chuyên Thái Bình

ĐT : 0989382988

Email : thuhienctb@gmail.com