Chuyên đề:
QUY HOẠCH ĐỘNG CHỮ SỐ

Yêu cầu biết trước:

- Đệ quy
- Quay lui
- Đệ quy có nhớ
- Quy hoạch động

I. ĐẶT VẤN ĐỀ

Có rất nhiều dạng bài toán yêu cầu đếm số lượng các số nguyên k trong phạm vi từ A đến B và thỏa mãn một tính chất cụ thể có thể liên quan đến các chữ số của nó.Thuật toán đơn giản chung cho các bài toán đó là:

- Nhập A, B
- Dem = 0
- Với mỗi k chạy từ A đến B:
    - Nếu k thoả mãn thì tăng Dem
- Xuất Dem

Độ phức tạp: lớn hơn $O(B)$ do có thể việc kiểm tra số k thoả mãn hay không cũng tốn thêm thời gian. Nếu B quá lớn (chẳng hạn như tới 10^{18}) thì không thể chạy được trong thời gian cho phép.

Cách khác là ta sẽ sử dụng kĩ thuật quy hoạch động chữ số.

II. Ý TƯỞNG CHÍNH

Gọi `G(x)` là số lượng các số nguyên như vậy trong phạm vi từ 0 đến x, thì đáp án của bải toán là `G(B)-G(A-1)`. Như vậy, ta cần viết hàm G(x).

Giả sử số x có n chữ số. Chẳng hạn, ta có x= $a_{n-1}a_{n-2}\dots a_2a_1a_0$, trong đó a_i ($0 \le i \le n-1$) cho biết chữ số thứ i tính từ bên phải. Chữ số tận cùng bên trái a_n là chữ số có nghĩa đầu tiên. Khi đó, giả sử số $k \le x$ thì k sẽ có dạng k = $t_{n-1}t_{n-2}\dots t_2t_1t_0$, trong đó có các điều kiện ràng buộc sau:

- $0 \le t_{n-1} \le a_{n-1}$
- $0 \le t_{n-2} \le a_{n-2}$ nếu $t_{n-1} = a_{n-1}$, ngược lại $0 \le t_{n-2} \le 9$.
- $0 \le t_{n-3} \le a_{n-3}$ nếu $t_{n-1} = a_{n-1}$ và$t_{n-2} = a_{n-2}$, ngược lại $0 \le t_{n-3} \le 9$.
- …

Tổng quát: $0 \le t_i \le a_i$ nếu $t_j = a_j$, $\forall j=i+1 \dots n-1$, ngược lại $0 \le t_i \le 9$.

Như vậy, chữ số t_i có thể bị giới hạn ($0 \le t_i \le a_i$) hoặc không bị giới hạn ($0 \le t_i \le 9$).

Điều kiện để t_i bị giới hạn là: $t_{i+1} = a_{i+1}$, $t_{i+2} = a_{i+2}$, …, $t_{n-1} = a_{n-1}$. Hay có thể nói cách khác: t_i bị giới hạn nếu t_{i+1} bị giới hạn và t_{i+1} đạt đến giới hạn (tức là $t_{i+1}=a_{i+1}$).

Hàm chính ở đây là một hàm đệ quy `thu(i,…)`, là hàm quay lui để thử các khả năng của chữ số thứ i (tức là t_i). Với mỗi giá trị của t_i, ta sẽ gọi đệ quy đến `thu(i-1,…)`. Bằng cách gọi `thu(n-1,…)`, ta sẽ sinh ra các số k trong phạm vi từ 0 đến x. Với mỗi số sinh ra, ta kiểm tra nó có thoả mãn tính chất đề bài yêu cầu hay không, nếu có thì tăng kết quả thêm 1.

Với cách này, số trường hợp sinh ra cũng quá lớn. Tuy nhiên, sẽ có nhiều trường hợp 1 hàm cùng tham số giống nhau (gọi là một trạng thái) được gọi nhiều lần. Ta sẽ khắc phục bằng cách dùng một mảng để lưu trạng thái đó. Nếu gặp lại, ta không cần tính lại nữa mà lấy ngay kết quả đã lưu trong mảng (người ta còn gọi đây là kĩ thuật đệ quy có nhớ).

Giá trị của hàm `thu(i,…)` là số lượng số thoả mãn đề bài khi chúng ta đã có các chữ số từ n-1 về i+1, mà các giá trị của các chữ số đó sẽ được đại diện bởi một hoặc nhiều tham số thêm vào (tuỳ thuộc vào từng bài toán).

Mẫu chung cho các hàm như sau:

Khai báo mảng `a[]` để lưu các chữ số của x và biến n là số chữ số.

Khai báo mảng `F[i][…]` để lưu các trạng thái //số chiều tuỳ thuộc vào từng bài toán

Ban đầu, mảng `F[i][…]` sẽ được gán bằng -1 hết, tức là trạng thái đó chưa được tính.

Hàm `thu(i, gh, …)` //Thử các trường hợp cho chữ số thứ i; có giới hạn hay không (gh=true hay false). Các tham số khác tuỳ bài toán.Chữ số thứ i nếu bị giới hạn thì nó chỉ nhận giá trị từ 0..a[i], còn nếu không, nó nhận giá trị từ 0..9.

Hàm `thu(i, gh, …)`

- Nếu i < 0 thì:
  - Nếu số sinh ra thoả mãn điều kiện thì trả về 1;
  - Ngược lại, trả về 0;
- Nếu gh=false và F[i][...] >= 0 thì trả về F[i][...] //*nếu trạng thái này đã được tính trước đó rồi thì lấy kết quả từ mảng lưu kết quả của trạng thái.*
- kq = 0;
- maxc = (gh=true ? a[i] : 9); //*giá trị tối đa mà chữ số thứ i đạt được*
- Với mỗi c chạy từ 0 đến maxc: //*cho chữ số thứ i bằng c*
  - ghm = (gh = true) AND (c = maxc) //*giới hạn của chữ số thứ i-1*
  - kq += `thu(i-1, ghm, …)` //*gọi đệ quy đến chữ số phía sau*
- Nếu gh=false thì F[i][...]=kq; //*lưu kết quả của trang thái để lần sau dùng*
- Trả về kq;

Hàm `G(x)`:

- n=0;
- a[0]=0;
- Trong khi x > 0 thì
  - a[n] = x mod 10; //*tách và lưu chữ số đơn vị của x vào a[n]*

- x = x div 10; *//xoá chữ số đơn vị của x*
- n = n + 1;

- Trả về `thu(n-1, true,...)` *//chữ số thứ n-1 luôn bị giới hạn.*

Hàm `main()` *//chương trình chính*

- Nhập A, B
- Cho F[i][...] = -1 hết //fillchar hoặc memset
- Xuất G(B)-G(A-1)

Đánh giá độ phức tạp thời gian:

Thời gian tiêu tốn nhiều cho việc gọi các trạng thái. Do đó, số trạng thái là tích của số khả năng của các tham số của hàm `thu(i,gh,…)`. Trong mỗi trạng thái, có một vòng lặp chạy tối đa 10 lần. Vậy số lần gọi trạng thái tối đa là 10 * tích của số khả năng của các tham số của hàm `thu(i,gh,…)`.

Sau đây, chúng ta xét một số bài toán cụ thể.

III. MỘT SỐ BÀI TOÁN MẪU

1. Bài toán 1: Số có tổng các chữ số là số nguyên tố

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

Có bao nhiêu số từ A đến B mà tổng các chữ số của nó là số nguyên tố.

Input (tệp TNT.INP)

Hai số A, B ($0 < A \le B \le 10^8$).

Output (tệp TNT.OUT)

Số lượng số tìm được

Ví dụ:

TNT.INP	TNT.OUT	*Giải thích*
7 20	6	*Có 6 số thoả mãn là 7, 11, 12, 14, 16, 20*

Giải:

Gọi G(x) là số các số tự nhiên nhỏ hơn hoặc bằng x mà tổng các chữ số của nó là số nguyên tố.Đáp án của bài toán sẽ là G(B)-G(A-1).

Ta đi tính G(x) với giả thiết rằng x gồm n chữ số a[n-1], a[n-2], …, a[2], a[1], a[0].

Để sinh số nhỏ hơn hoặc bằng x, ta dùng hàm đệ quy `thu(i, gh, tong)` với ý nghĩa: thử chữ số thứ `i` (tức là ta đã có các chữ số từ n-1 về i+1);`gh`=true (hoặc false) nếu chữ số `i` bị giới hạn (hoặc không bị giới hạn);`tong` là tổng các chữ số đã sinh trước đó. Giá trị của hàm là số lượng số cần tìm mà ta có thể sinh tiếp.

Ta dùng mảng F[][] để lưu lại các trạng thái gọi đệ quy: `F[i][tong]` là số lượng số cần tìm của hàm `thu(i, false, tong)`;

Ví dụ: với x=5000000 nếu ta đã sinh được 2 chữ số đầu là 18 thì ta sẽ gọi đến hàm `thu(5, false, 9)`; sau này, nếu sinh 2 chữ số đầu khác là 27 hoặc 36 hoặc 45… thì ta sẽ *gọi lại* trạng thái `thu(5, false, 9)` nên để đỡ tốn thời gian, ta sẽ lưu lại trạng thái bằng phần tử `F[5][9]` và sau đó, nếu gặp lại, ta chỉ cần lấy `F[5][9]` cộng vào kết quả là xong.

Tại sao chúng ta không lưu trạng thái `(i, true, tong)`, tức là trường hợp gh=true? Vì những trường hợp chữ số `i` bị giới hạn chỉ gặp duy nhất 1 lần nên không cần lưu!

Hàm `thu(i, gh, tong)`:

- Nếu i < 0 thì
  - Nếu nguyento[tong]=true thì trả về 1; //có 1 số vừa sinh ra thoả mãn
  - Ngược lại, trả về 0 // số sinh ra không thoả mãn.
- Nếu gh=false và F[i][tong] ≥ 0 thì trả về F[i][tong];
- kq=0;
- maxc = (gh = true ? a[i] : 9)
- Với mỗi c chạy từ 0 đến maxc:
  - ghm = (gh=true) and (c=maxc)
  - kq += `thu(i-1, ghm, tong+c)`;
- Nếu gh=false thì F[i][tong]=kq;
- Trả về kq;

Hàm `G(x)`:

- Tách chữ số x thành mảng các chữ số a[n-1],a[n-2],...,a[2],a[1],a[0].
- Trả về `thu(n-1, true, 0);` //chúng ta bắt đầu sinh từ chữ số đầu tiên (chữ số thứ n-1), chữ số này luôn bị giới hạn và tổng các chữ số đã có là 0.

Hàm `main()`:

- Cho tất cả F[i][j]=-1 hết.
- Nhập A, B
- Xuất G(B)-G(A-1);

Ghi chú:

- *Mảng nguyento[] là sàng số nguyên tố từ 0 đến 72.*
- *Mảng F[][] dùng để tính G(B) và cũng được dùng các giá trị đã có khi tính G(A-1).*

Độ phức tạp:Số trạng tháilà max(i)*max(gh)*max(tong). Mỗi hàm thu() có tối đa 10 lần lặp (c chạy từ 0 đến 9). Vậy số lần gọihàm thu tối đa là 10*max(i)*max(gh)*max(tong)

- Số khả năng của i tối đa là 8;
- Số khả năng của gh là 2;
- Số khả năng của tong tối đa là 8*9 = 72;

Vậy số trạng thái tối đa phải tính là 10*8*2*62 = 9920≈ 10^4.

Code mẫu:

```
#include<bits/stdc++.h>
usingnamespace std;
#define baitoan "tnt"
typedeflonglongint lli;
char a[9],n;
lli F[9][73];
bool nguyento[73];
voidsangnguyento(int  n)
{
int i,j;
    memset(nguyento,true,sizeof(nguyento));
    nguyento[0]=nguyento[1]=false;
for (i=2; i<=n; i++)
    {
if (nguyento[i])
        {
for (j=i*i; j<=n; j+=i) nguyento[j]=false;
        }
    }
}
lli thu(int i, bool gh,int tong)
 {
bool ghm;
if (i <0)
    {
if (nguyento[tong]) return1;
elsereturn0;
    }

if(gh == false&& F[i][tong] >= 0)
return F[i][tong];

    lli kq = 0;
char maxc = (gh ? a[i] : 9);
for (char c = 0; c <= maxc; c++)
    {
        ghm = gh && (c == maxc);
        kq += thu(i-1, ghm, tong+c);
    }
if (gh==false) F[i][tong] = kq;
return kq;
 }

lli G(lli x)
{
    a[0] = 0;
    n = 0;
while (x)
    {
        a[n] = x % 10;
        x /= 10;
        n++;
    }
return thu(n-1,true,0);
}

intmain()
{
    freopen(baitoan".inp", "r", stdin);
    freopen(baitoan".out", "w", stdout);
    memset(F, -1, sizeof(F));
    sangnguyento(72);
```

```
    lli A, B;
    cin >> A >> B;
    cout << G(B) - G(A-1);
return0;
}
```

Từ giờ trở đi, để tránh lặp lại nội dung, giảm thiểu sự rườm rà, tôi chỉ nêu hàm `thu()` và cách gọi nó trong hàm `G()`.

2. Bài toán 2: SQAMOD - Numbers Sum Squares Modulo ZERO

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

Cho số n ($1 \le n \le 10^{10000}$). Tìm số lượng số không âm nhỏ hơn n, có tổng bình phương các chữ số của nó chia hết cho 3.

Input (tệp TBPB3.INP)

Số n

Output (tệp TBPB3.OUT)

Số lượng số tìm được. Chỉ ghi ra số dư của kết quả chia cho 10^9+7.

Ví dụ

TBPB3.INP	TBPB3.INP
9	3
10	4
15	4

Giải:

Phân tích tham số:

Ta cần thêm tham số `tbp` lưu số dư khi chia cho 3 của tổng bình phương của các chữ số đã có trước chữ số `i`.

Hàm `thu(i, gh, tbp)`:với mảng lưu trạng thái`F[i][tbp]`

- Nếu i < 0 thì:
  - Nếu tbp = 0 thì trả về 1;
  - Ngược lại, trả về 0;
- Nếu gh=false và F[i][tbp] ≥ 0 thì trả về F[i][tbp];
- kq=0;
- maxc = (gh = true ? a[i] : 9)
- Với mỗi c chạy từ 0 đến maxc:
  - ghm = (gh=true) and (c=maxc)
  - kq += `thu(i-1, ghm, (tbp+c*c) mod 3)`;
- kq = kq mod 1000000007;
- Nếu gh=false thì F[i][tbp]=kq;

- Trả về kq;

Hàm `G(x)`:

- Tách chữ số x thành mảng các chữ số a[n-1],a[n-2],...,a[2],a[1],a[0].
- Trả về `thu(n-1, true, 0);`

Lưu ý: n rất lớn nên phải lưu bằng xâu (string). Khi đó đáng lẽ ra đáp án là G(n-1), nhưng vì n là string nên không thể trừ 1 được. Cho nên chúng ta có 2 cách làm:

- Viết thêm hàm trừ 1 số lưu trong string cho 1;
- Hoặc đáp án sẽ là G(n) trừ thêm cho 1 nếu tổng bình phương các chữ số của n chia hết cho 3.

Độ phức tạp: số lần lặp tối đa là 10 * 10000 * 2 * 3.

Code mẫu

```
#include<bits/stdc++.h>
usingnamespace std;
#define baitoan "tbpb3"
typedeflonglongint lli;
char a[20],n;
lli F[20][3];
int loai=0;
lli thu(int i, bool gh,int tbp)
 {
bool ghm;
if (i <0)
    {
if (tbp == 0) return1;
elsereturn0;
    }

if(gh == false&& F[i][tbp] >= 0) return F[i][tbp];

    lli kq = 0;
char maxc = (gh ? a[i] : 9);
for (char c = 0; c <= maxc; c++)
    {
        ghm = gh && (c == maxc);
        kq += thu(i-1, ghm, (tbp+c*c)%3);
    }
if (gh == false) F[i][tbp] = kq;
return kq;
 }

lli G(string x)
{
int t=0;
    n=x.length();
for (int i=0; i<n; i++) a[n-1-i]=x[i]-48;
for (int i=0; i<n; i++) t+=a[i]*a[i];
if (t%3==0) loai=1;
return thu(n-1,true,0);
}

intmain()
{
    freopen(baitoan".inp", "r", stdin);
    freopen(baitoan".out", "w", stdout);
    memset(F, -1, sizeof(F));
    string A;
```

```
    cin >> A ;
    cout << G(A)-loai;
return0;
}
```

3. Bài toán 3: Investigation

http://lightoj.com/volume_showproblem.php?problem=1068

https://toph.co/p/m-beautiful-numbers

Một số nguyên chia hết cho 3 thì tổng các chữ số của nó cũng chia hết cho 3.

Ví dụ: 3702 ⋮ 3 và 3+7+0+2 = 12 ⋮ 3. Tính chất này cũng đúng đối với số 9.

Trong bài toán này, chúng ta sẽ dùng tính chất đó cho các số nguyên khác.

Input (tệp CHIAHET.INP)

Ba số nguyên dương**A, B** và **K (1 ≤ A ≤ B < 2^{31}và 0 < K < 10000)**.

Output(tệp CHIAHET.OUT)

Số lượng số nguyên trong phạm vi từ A đến B mà chia hết cho K, đồng thời, tổng các chữ số của nó cũng chia hết cho K.

Ví dụ:

CHIAHET.INP	CHIAHET.OUT
1 20 2	5

CHIAHET.INP	CHIAHET.OUT
1 1000 4	64

Giải:

Phân tích tham số:

Khi sinh đến chữ số i, ta cần biết tổng các chữ số của nó và số dư của phần số đã sinh rồi khi chia cho K. Như vậy, ta cần thêm 2 tham số:

- `sum` : là tổng các chữ số đã sinh được. (`sum` ≤ 90)
- `sodu` : là số dư của phần số đã sinh được chia cho K.

Hàm đệ quy `thu(i, gh, sum, sodu)` với mảng lưu `F[i][sum][sodu]`.

- Nếu i<0 thì
  - Nếu sum mod K =0 và sodu=0 thì trả về 1;
  - Ngược lại, trả về 0
- Nếu gh=false và F[i][sum][sodu] ≥ 0 thì trả về F[i][sum][sodu]
- kq=0;
- maxc = (gh = true ? a[i] : 9)
- Với mỗi c chạy từ 0 đến maxc:
  - $ghm = (gh = true)\ AND\ (c = maxc)$
  - $kq = kq + thu(i-1, ghm, sum+c, (10 * sodu + c)\ mod\ K)$;
- Nếu $gh = false$ thì gán $F[i][sum][sodu] = kq$

- Trả về *kq*;

Hàm G(x)

- Tách các chữ số của x;
- Trả về thu(n-1, true, 0,0);

Độ phức tạp: số trạng thái tối đa là 10 * 2 * 90 * K.

Code mẫu

```
#include<bits/stdc++.h>
usingnamespace std;
#define baitoan "chiahet"
typedeflonglongint lli;
char a[10],n;
lli F[11][91][10000];
int K;
lli thu(int i, bool gh,int sum, int sodu)
 {
bool ghm;
if (i <0)
    {
if (sum%K==0&& sodu==0) return1;else return0;
    }
if(gh == false&& F[i][sum][sodu] >= 0)
return F[i][sum][sodu];

    lli kq = 0;
char maxc = (gh ? a[i] : 9);
for (char c = 0; c <= maxc; c++)
    {
        ghm = gh && (c == maxc);
        kq += thu(i-1, ghm, sum+c, (10*sodu+c)%K);
    }
if (gh==false) F[i][sum][sodu] = kq;
return kq;
 }
lli G(lli x)
{
    a[0] = 0;    n = 0;
while (x)
    {
        a[n] = x % 10;        x /= 10;        n++;
    }
return thu(n-1,true,0,0);
}
intmain()
{
    freopen(baitoan".inp", "r", stdin);
    freopen(baitoan".out", "w", stdout);
    memset(F, -1, sizeof(F));
    lli A, B;
    cin >> A >> B;
    cout << G(B) - G(A-1);
return0;
}
```

Bài toán 4: Ra-One Numbers

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

Số Ra-One là số mà hiệu của tổng các chữ số ở vị trí chẵn và tổng các chữ số ở vị trí lẻ là bằng 1. Ví dụ số 234563 là soos Ra-One, vì (2+4+6) - (3+5+3) = 1.

Còn số 123456 không phải số Ra-One, vì (1+3+5) - (2+4+6) = -4 ≠ 1

Tìm số lượng số Ra-One từ A đến B.

Input (tệp RAONE.INP)

Hai số A, B.

Output (tệp RAONE.OUT)

Số lượng số Ra-One tìm được.

Ví dụ:

RAONE.INP	RAONE.OUT
1 10	1

RAONE.INP	RAONE.OUT
10 100	9

Giải thích:

VD1: Chỉ có 1 số Ra-One duy nhất là 10

VD2: Các số Ra-One là 10, 21, 32, 43, 54, 65, 76, 87, 98.

Giới hạn: $1 \le A \le B \le 10^8$.

Giải:

Phân tích tham số:

Ta cần biết hiệu đã có khi xét tới chữ số i, nên ta sẽ dùng thêm 1 tham số `hieu`.

Nếu chữ số thứ i bằng c thì hieu = `hieu` + c (nếu i chẵn) ngược lại hieu=`hieu`-c;

Ta có thể dùng mảng hằng hs[0..1] = {+1, -1}. Khi đó, hieu=hieu+c*hs[i mod 2];

Hàm `thu(i, gh, hieu)` với mảng lưu kết quả `F[i][hieu]`.

- Nếu i < 0 thì
  - Nếu hieu=1 thì trả về 1;
  - Ngược lại, trả về 0
- Nếu gh=false và F[i][hieu]≥0 thì trả về F[i][hieu];
- kq=0;
- maxc = (gh = true ? a[i] : 9)
- Với mỗi c chạy từ 0 đến maxc:
  - ghm = (gh=true) and (c=maxc)
  - kq += `thu(i-1, ghm, hieu + hs[(i+1) mod 2]*c);`
- Nếu gh=false thì F[i][hieu]=kq;
- Trả về kq;

Hàm G(x):

- Tách x
- Trả về `thu(n-1, true, 0)`;

Lưu ý: *Chỉ số của các chữ số theo mô tả đề bài bắt đâu từ 1, tức là chữ số đơn vị là chữ số ở vị trí lẻ. Trong khi theo phân tích thuật toán từ trước đến giờ thì i đếm từ 0 (tức là chữ số đơn vị là chữ số chẵn) nên trong thuật toán có ghi (i+1) mod 2.*

Độ phức tạp: tối đa 10 * 8 * 2 * 73.

Code mẫu:

```
#include<bits/stdc++.h>
usingnamespace std;
#define baitoan "raone"
typedeflonglongint lli;
char a[9],n;
lli F[9][74];
bool snt[74];
int hs[2] = {-1, 1};
lli thu(int i, bool gh,int hieu)
 {
bool ghm;
if (i <0)
    {
if (hieu==1) return1;
elsereturn0;
    }

if(gh == false&& F[i][hieu+37] >= 0)
return F[i][hieu+37];

    lli kq = 0;
char maxc = (gh ? a[i] : 9);
for (char c = 0; c <= maxc; c++)
    {
        ghm = gh && (c == maxc);
        kq += thu(i-1, ghm, hieu+hs[i%2]*c);
    }
if (gh==false) F[i][hieu+37] = kq;
return kq;
 }

lli G(lli x)
{
    a[0] = 0;
    n = 0;
while (x)
    {
        a[n] = x % 10;
        x /= 10;
        n++;
    }
return thu(n-1,true,0);
}

intmain()
{
    freopen(baitoan".inp", "r", stdin);
    freopen(baitoan".out", "w", stdout);
    memset(F, -1, sizeof(F));
    lli A, B;
    cin >> A >> B;
    cout << G(B) - G(A-1);
```

```
return0;
}
```

Bài toán 5: LUCIFER NUMBER

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

Một số là Lucifer nếu hiệu giữa tổng các chữ số ở vị trí chẵn và tổng các chữ số ở vị trí lẻ là một số nguyên tố.

Ví dụ số 20314210 là số Lucifer. Vì (1+4+3+2)-(0+2+1+0)=10-3 = 7 là số nguyên tố.

Tìm số lượng số Lucifer trong phạm vi từ A đến B.

Input (Tệp LUCIFER.INP)

Hai số nguyên A,B.

Output (Tệp LUCIFER.OUT)

Số lượng số Lucifer trong phạm vi từ A đến B.

Ví dụ:

LUCIFER.INP	LUCIFER.OUT
150 200	16
100 150	3
50 100	18

Giới hạn: $0 \le A \le B \le 10^9$.

Phân tích tham số:

Ta cần biết hiệu đã có khi xét tới chữ số i, nên ta sẽ dùng thêm 1 tham số `hieu`.

Nếu chữ số thứ i bằng c thì hieu mới = hieu + c (nếu i chẵn) ngược lại hieu mới = hieu-c;

Ta có thể dùng mảng hằng hs[0..1] = {+1, -1}. Khi đó, hieu=hieu+c*hs[i mod 2];

Hàm `thu(i, gh, hieu)`

- Nếu i < 0 thì
  - Nếu nguyento[hieu]=true thì trả về 1;
  - Ngược lại, trả về 0
- Nếu gh=false và F[i][hieu]≥ 0 thì trả về F[i][hieu];
- kq=0;
- maxc = (gh = true ? a[i] : 9)
- Với mỗi c chạy từ 0 đến maxc:
  - ghm = (gh=true) and (c=maxc)
  - kq += thu(i-1, ghm, hieu + hs[(i+1) mod 2]*c);
- Nếu gh=false thì F[i][hieu]=kq;
- Trả về kq;

Hàm G(x):

- Tách x
- Trả về `thu(n-1, true, 0);`

Độ phức tạp: tối đa 10 * 10 * 2 * 83 lần gọi trạng thái.

Code mẫu:

```
#include<bits/stdc++.h>
usingnamespace std;
#define baitoan "lucifer"
typedeflonglongint lli;
char a[10],n;
lli F[10][84];
bool nguyento[84];
int hs[2] = {-1, 1};
voidsangnguyento(int  n)
{
int i,j;
    memset(nguyento,true,sizeof(nguyento));
    nguyento[0]=nguyento[1]=false;
for (i=2; i<=n; i++)
    {
if (nguyento[i])
        {
for (j=i*i; j<=n; j+=i) nguyento[j]=false;
        }
    }
}
lli thu(int i, bool gh,int hieu)
 {
bool ghm;
if (i <0)
    {
if (hieu>1&& nguyento[hieu]) return1;
elsereturn0;
    }

if(gh == false&& F[i][hieu+45] >= 0)
return F[i][hieu+45];

    lli kq = 0;
char maxc = (gh ? a[i] : 9);
for (char c = 0; c <= maxc; c++)
    {
        ghm = gh && (c == maxc);
        kq += thu(i-1, ghm, hieu+hs[i%2]*c);
    }
if (gh==false) F[i][hieu+45] = kq;
return kq;
 }

lli G(lli x)
{
    a[0] = 0;
    n = 0;
while (x)
    {
        a[n] = x % 10;
        x /= 10;
        n++;
    }
return thu(n-1,true,0);
}
```

```
intmain()
{
    freopen(baitoan".inp", "r", stdin);
    freopen(baitoan".out", "w", stdout);
    sangnguyento(83);
    memset(F, -1, sizeof(F));
    lli A, B;
    cin >> A >> B;
    cout << G(B) - G(A-1);
return0;
}
```

6. Bài toán 6: Số bất thường

Một số được coi là bất thường, nếu tổng các chữ số và tổng bình phương các chữ số (trong hệ thập phân) của nó nguyên tố cùng nhau.Ví dụ: số 23, số 41 là các số bất thường.

Bờm rất thích thú với định nghĩa số bất thường này và Bờm muốn nhờ các bạn xác định số lượng số bất thường trong đoạn [L,R]

Input: Tệp văn bản SBT.INP gồm hai số nguyên L và R ($1 \le L, R \le 10^{18}$).

Output:Tệp văn bản SBT.OUT gồm 1 số nguyên là kết quả cần tìm.

Ví dụ:

SBT.INP	SBT.OUT
10 11	1

SBT.INP	SBT.OUT
100 150	19

Subtask 1(40%): $1 \le L, R \le 10^6$

Subtask 2(30%): $1 \le L, R \le 10^9$

Subtask 3(30%): $1 \le L, R \le 10^{18}$

Giải:

Phân tích tham số:

Khi sinh xong 1 số, vì không lưu số đó, ta cần biết tổng các chữ số và tổng bình phương các chữ số của nó, nên ta cần thêm 2 tham số cho hàm `thu()`:

- `sum` là tổng các chữ số đã sinh ra.
- `tbp` là tổng bình phương các chữ số đã sinh ra.

Ta dùng mảng `F[i][sum][tbp]` để lưu trạng thái.

Hàm `thu(i, gh, sum, tbp)`:

- Nếu i < 0 thì
  - Nếu UCLN(sum, tbp)=1 thì trả về 1;
  - Ngược lại, trả về 0.
- Nếu gh=false và F[i][sum][tbp]≥0 thì trả về F[i][sum][tbp];
- kq=0;
- maxc = (gh = true ? a[i] : 9)

- Với mỗi c chạy từ 0 đến maxc:
    - ghm = (gh=true) and (c=maxc)
    - kq += thu(i-1, ghm, sum+c, tbp+c*c);
- Nếu gh=false thì F[i][sum][tbp]=kq;
- Trả về kq;

Hàm `G(x)`:

- Tách x
- Trả về `thu(n-1, true, 0, 0)`;

Ghi chú: UCLN(p, q) là hàm trả về ước chung lớn nhất của p và q.

Độ phức tạp: tối đa có 10 * 18 * 2 * 162 * 1458 lần gọi hàm thu().

Code mẫu:

```
#include<bits/stdc++.h>
usingnamespace std;
#define baitoan "sbt"
typedeflonglongint lli;
char a[19],n;
lli F[19][163][1459];

lli thu(int i, bool gh,int sum, int tbp)
 {
bool ghm;
if (i <0)
    {
if (__gcd(sum,tbp)==1) return1;
elsereturn0;
    }

if(gh == false&& F[i][sum][tbp] >= 0)
return F[i][sum][tbp];

    lli kq = 0;
char maxc = (gh ? a[i] : 9);
for (char c = 0; c <= maxc; c++)
    {
        ghm = gh && (c == maxc);
        kq += thu(i-1, ghm, sum+c, tbp+c*c);
    }
if (gh==false) F[i][sum][tbp] = kq;
return kq;
 }

lli G(lli x)
{
    a[0] = 0;
    n = 0;
while (x)
    {
        a[n] = x % 10;
        x /= 10;
        n++;
    }
return thu(n-1,true,0,0);
}

intmain()
```

```
{
    freopen(baitoan".inp", "r", stdin);
    freopen(baitoan".out", "w", stdout);
    memset(F, -1, sizeof(F));
    lli A, B;
    cin >> A >> B;
    cout << G(B) - G(A-1);
return0;
}
```

7. Bài toán 7: Số tăng-giảm xen kẽ

http://lequydon.ntucoder.net/Problem/Details/5841/

Một số $a_1a_2...a_n$ là một số tăng-giảm xen kẽ nếu:

$a_1 < a_2$; $a_2 > a_3$; $a_3 < a_4$; ... hoặc $a_1 > a_2$; $a_2 < a_3$; $a_3 > a_4$; ...

Trong các số tự nhiên từ L đến R, có bao nhiêu số tăng-giảm xen kẽ?

Input (tệp XENKE.INP)

2 số tự nhiên L, R ($1 \le L \le R \le 10^{100000}$).

Output (tệp XENKE.OUT)

In ra số lượng số tăng giảm trong đoạn từ L đến R. Vì đáp số có thể hơi lớn nên các bạn chỉ cần in ra số dư của đáp số khi cho 10^9+7.

Ví dụ

XENKE.INP	XENKE.OUT
8 15	7

XENKE.INP	XENKE.OUT
1998 2004	0

Giải thích

Ở ví dụ 1, các số từ 8 đến 15 đều thoả mãn trừ số 11.

Ở ví dụ 2, các số từ 1998 đến 2004 có 2 chữ số ở giữa bằng nhau nên không số nào thỏa mãn.

Giải:

Vì khi xét chữ số thứ i, ta cần biết nó lớn hơn hay nhỏ hơn chữ số trước đó, nên ta cần thêm 2 tham số:

- `dig` : là chữ số liền trước đó (chữ số thứ i+1)
- `tang` : cho biết a[i]>a[i+1] (tang=true) hay a[i]<a[i+1] (tang=false)

Nếu chữ số thứ i vô nghĩa thì không xét tăng hay giảm được. Vậy, ta cần thêm tham số nghia cho biết đã có chữ số có nghĩa hay chưa (true hoặc false)

Mảng `F[i][dig][tang][nghia]` để lưu trạng thái.

Hàm `thu(i, gh, dig, tang, nghia)`

- Nếu i<0 thì trả về 1
- Nếu gh=false và F[i][dig][tang][nghia]≥0 thì trả về F[i][dig][tang][nghia]
- kq=0
- maxc = (gh=true ? a[i] : 9)
- Nếu nghia=true thì

- Nếu tang=true thì
  - Với mỗi c chạy từ dig+1 đến maxc:
    - ghm = (gh=true) and (c=maxc)
    - kq += thu(i-1, ghm, c, false,true)
- Ngược lại (tang = false)
  - Với mỗi c chạy từ 0 đến min(dig-1, maxc):
    - ghm = (gh=true) and (c=maxc)
    - kq += thu(i-1, ghm, c, true, c ≠ 0 OR nghia)

- Ngược lại (nghia=false) thì:
  - ghm= gh and (maxc=0);
  - kq += thu(i-1, ghm, 0, true, false);
  - Với mỗi c chạy từ 1 đến maxc:
    - ghm = (gh=true) and (c=maxc)
    - kq += thu(i-1, ghm, c, true, true)
    - Nếu i>0 thì kq += thu(i-1, ghm, c, false, true);
- Nếu gh=false thì F[i][dig][tang][nghia]=kq
- Trả về kq

Hàm `G(x)`:

- Tách chữ số của x;
- Trả về `thu(n-1,true,-1,true) + thu(n-1,true,10,false)`

Độ phức tạp thời gian: Số trạng thái tối đa $10 * 10^5 * 10 * 2 * 2$.

Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
#define baitoan "xenke"
typedef long long int lli;
char a[100001],n;
lli F[100001][10][2][2];

lli thu(int i, bool gh,char dig, bool tang, bool nghia)
 {

    bool ghm;
    if (i < 0)
    {
        return 1;
    }
    if(gh == false && F[i][dig][tang][nghia] >= 0)
        return F[i][dig][tang][nghia];
    lli kq = 0;
    char maxc = (gh ? a[i] : 9);
    if (nghia)
      {
            if (tang)
            {
                  for (char c = dig+1; c <= maxc; c++)
```

```
                  {
                        ghm = gh && (c == maxc);
                        kq += thu(i-1, ghm, c, false,(c!=0)|nghia);
                  }
            }
            else
            {
                  for (char c = 0; c <= min(dig-1,maxc+0); c++)
                  {
                        ghm = gh && (c == maxc);
                        kq += thu(i-1, ghm, c, true, (c!=0)|nghia);
                  }
            }
      }
    else
      {
            ghm = gh && (0 == maxc);
            kq += thu(i-1, ghm, 0, true, false);
            for (char c = 1; c <= maxc; c++)
            {
                  ghm = gh && (c == maxc);
                  kq += thu(i-1, ghm, c, true, true);
                  if (i>0)
                  kq += thu(i-1, ghm, c, false, true);
            }
      }
    kq %= 1000000007;
    if (gh==false) F[i][dig][tang][nghia] = kq;
    return kq;
 }
lli G(string x)
{
    //if (x=="0") return 1;
    n=x.length();
    for (int i=0; i<n; i++) a[n-1-i]=x[i]-48;
    if (n==1) return x[0]-47;
    return thu(n-1,true,-1,true,false);
}
void giam1(string &s)
{
      int i = s.length()-1;
      while (s[i]==48)
      {
            s[i]='9';   i--;
      }
      s[i]--;
      while(s[0]==48 && s.length()>0) s.erase(0,1);
}
int main()
{
    freopen(baitoan".inp", "r", stdin);
    freopen(baitoan".out", "w", stdout);
    memset(F, -1, sizeof(F));
    string A, B;    cin >> A >> B;
    giam1(A);    cout << G(B) - G(A);
    return 0;
}
```

8. Bài toán 8: Số gồm các chữ số phân biệt

http://ntucoder.net/Problem/Details/4492

Cho hai số nguyên L và R, yêu cầu tính xem trong đoạn [L,R] có bao nhiêu số thỏa mãn:

- Không có các chữ số 0 vô nghĩa ở đầu

- Các chữ số của số đó hoàn toàn phân biệt.

Input (tệp CSPB.INP)

Hai số nguyên L và R ($1 \le L \le R \le 10^{18}$).

Output(tệp CSPB.OUT)

Kết quả tìm được.

Ví dụ

CSPB.INP	CSPB.OUT
1 11	10

Giải:

Phân tích tham số:

Khi xét chữ số i, ta cần biết:

- Những chữ số nào đã dùng rồi để tránh. Ta sẽ dùng 10 bit của một số nguyên (tham số `tt`) để đánh dấu các chữ số đã dùng: chữ số nào đã được dùng rồi thì bit đó được bật (bằng 1). Số nguyên `tt` đó có giá trị tối đa là $2^{10} = 1024$.
- Đã có chữ số có nghĩa trước đó hay chưa (dùng tham số `nghia`=true/false). Nếu chưa có thì bây giờ chữ số i nếu cho bằng 0 thì cũng là chữ số vô nghĩa. Còn nếu đã có chữ số có nghĩa rồi thì nếu giờ cho chữ số i bằng 0 là chữ số có nghĩa, nên cần đánh dấu đã dùng chữ số 0 để chuyển sang gọi đệ quy cho chữ số tiếp theo.

Kiểm tra chữ số c đã dùng chưa, hay bit c bằng 1 hay 0: tt and $2^c = 1$.

Đánh dấu chữ số c đã dùng, hay bật bit c: tt = tt or 2^c.

Mảng `F[i][tt][nghia]` để lưu trạng thái

Hàm `thu(i, gh, tt, nghia)`

- Nếu i<0 thì trả về 1
- Nếu gh=false và F[i][tt][nghia]≥0 thì trả về F[i][tt][nghia]
- kq=0
- Nếu gh = true thì
  - Nếu nghia=true thì
    - Nếu chưa dùng chữ số 0 thì kq+=thu(i-1,a[i]==0,tt|1,true);
  - Ngược lại
    - kq+=thu(i-1,a[i]==0,tt,false);
  - Với mỗi c chạy từ 1 đến a[i]:
    - Nếu chưa dùng chữ số c thì kq+=thu(i-1,c==a[i],tt|(1<<c),true);
- Ngược lại (gh=false)
  - Nếu nghia=true thì
    - Nếu chưa dùng chữ số 0 thì kq+=thu(i-1,false,tt|1,true);
  - Ngược lại

        - kq+=thu(i-1,false,tt,false);
    - Với mỗi c chạy từ 1 đến 9:
        - Nếu chưa dùng chữ số c thì kq+=thu(i-1,false,tt|(1<<c),true);
- Nếu gh=false thì F[i][tt][nghia]=kq
- Trả về kq

Hàm `G(x)`:

- Tách x;
- Trả về `thu(n,true,0,false)`

Độ phức tạp: tối đa 10 * 18*1024*2*2 trạng thái

Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
#define baitoan "cspb"
typedef long long int lli;
char a[19],n;
lli F[19][1025][2];

lli thu(int i, bool gh,int tt, bool nghia)
 {
    bool ghm;
    if (i < 0)
    {
        return 1;
    }

    if(gh == false && F[i][tt][nghia] >= 0)
        return F[i][tt][nghia];

    lli kq = 0;
    if (gh)
      {
            if (nghia)
            {
                  ghm=a[i]==0;
                  if ((tt & 1) ==0) kq+= thu(i-1,ghm, tt|1, true);
            }
            else
            {
                  ghm=a[i]==0;
                  kq+=thu(i-1,ghm,tt,false);
            }
            for(char c=1; c<=a[i]; c++)
            {
                  ghm=a[i]==c;
                  if ((tt & (1 << c))==0) kq+= thu(i-1,ghm, tt|(1<<c), true);
            }
      }
      else
      {
            if (nghia)
            {
                  if ((tt & 1)==0) kq+= thu(i-1,false, tt|1, true);
            }
            else
            {
                  kq+=thu(i-1,false,tt,false);
```

```
            }
            for(char c=1; c<=9; c++)
            {
                  if ((tt & (1 << c))==0) kq+= thu(i-1,false, tt|(1<<c),
true);
            }
      }
    if (gh==false) F[i][tt][nghia] = kq;
    return kq;
 }

lli G(lli x)
{
    a[0] = 0;
    n = 0;
    while (x)
    {
        a[n] = x % 10;
        x /= 10;
        n++;
    }
    return thu(n-1,true,0,false);
}

int main()
{
    freopen(baitoan".inp", "r", stdin);
    freopen(baitoan".out", "w", stdout);
    memset(F, -1, sizeof(F));
    lli A, B;
    cin >> A >> B ;
    cout << G(B) - G(A-1);
    return 0;
}
```

9. Bài toán 9: Playing with digits

https://www.hackerearth.com/problem/algorithm/playing-with-digits-4e25844f/

Xác định số lượng số nguyên từ a đến b thoả mãn các điều kiện

- Tổng của các chữ số là số nguyên tố
- Chia hết cho K

Input (tệp TNTCHK.INP)

a, b, K

Output (tệp TNTCHK.OUT)

Kết quả bài toán

Ví dụ:

TNTCHK.INP	**TNTCHK.OUT**	Giải thích
5 86 4	**7**	7 số đó là 12 16 20 32 52 56 76

Giới hạn:

- $1 \le a \le b \le 2*10^{10}$.
- $1 \le K \le 4000$.

Giải:

Phân tích tham số:

Khi sinh đến chữ số i, ta cần biết tổng các chữ số của nó (để kiểm tra số nguyên tố) và số dư của phần số đã sinh rồi khi chia cho K. Như vậy, ta cần thêm 2 tham số:

- `sum` : là tổng các chữ số đã sinh được. (`sum` ≤ 92)
- `sodu` : là số dư của phần số đã sinh được chia cho K.

Hàm đệ quy `thu(i, gh, sum, sodu)` với mảng lưu trạng thái `F[i][sum][sodu]`.

- Nếu i<0 thì
  - Nếu nguyento[sum]=true và sodu=0 thì trả về 1;
  - Ngược lại, trả về 0
- Nếu gh=false và F[i][sum][sodu] ≥ 0 thì trả về F[i][sum][sodu]
- kq=0;
- maxc = (gh = true ? a[i] : 9)
- Với mỗi c chạy từ 0 đến maxc:
  - ghm = (gh=true) AND (c=maxc)
  - kq = kq + thu(i-1, ghm, sum+c, (10*sodu+c) mod K);
- Nếu gh=false thì gán F[i][sum][sodu]=kq
- Trả về kq;

Hàm `G(x)`:

- Tách các chữ số của x;
- Trả về `thu(n-1, true, 0, 0)`;

Độ phức tạp: số lần gọi thu() tối đa là 10 * 11 * 2 * 92 * 4000.

Code mẫu:

```
#include<bits/stdc++.h>
using namespace std;
#define baitoan "tntchk"
typedef long long int lli;
char a[12],n;
lli F[12][100][4001];
int K;
bool nguyento[94];
void sangnguyento(int  n)
{
    int i,j;
    memset(nguyento,true,sizeof(nguyento));
    nguyento[0]=nguyento[1]=false;
    for (i=2; i<=n; i++)
    {
        if (nguyento[i])
        {
            for (j=i*i; j<=n; j+=i) nguyento[j]=false;
        }
    }
}
lli thu(int i, bool gh,int sum, int sodu)
 {
```

```
    bool ghm;
    if (i < 0)
    {
        if (nguyento[sum] && sodu==0) return 1;        else return 0;
    }

    if(gh == false && F[i][sum][sodu] >= 0)
        return F[i][sum][sodu];

    lli kq = 0;
    char maxc = (gh ? a[i] : 9);
    for (char c = 0; c <= maxc; c++)
    {
        ghm = gh && (c == maxc);
        kq += thu(i-1, ghm, sum+c, (10*sodu+c)%K);
    }
    if (gh==false) F[i][sum][sodu] = kq;
    return kq;
 }

lli G(lli x)
{
    a[0] = 0;
    n = 0;
    while (x)
    {
        a[n] = x % 10;
        x /= 10;
        n++;
    }
    return thu(n-1,true,0,0);
}

int main()
{
    freopen(baitoan".inp", "r", stdin);
    freopen(baitoan".out", "w", stdout);
    memset(F, -1, sizeof(F));
    sangnguyento(93);
    lli A, B;
    cin >> A >> B >> K;
    cout << G(B) - G(A-1);
    return 0;
}
```

10. Bài toán 10: Tổng các chữ số

https://www.spoj.com/problems/CPCRC1C/

Cho 2 số nguyên dương A, B (A≤B). Tính tổng các chữ số có mặt trong các số nguyên từ A đến B.

Input (tệp TONGCS.INP)

Hai số A, B.

Output (tệp TONGCS.OUT)

Tổng tìm được.

Ràng buộc: $1 \leq A < B \leq 10^{18}$.

Ví dụ:

TONGCS.INP	TONGCS.OUT
5 11	38

Giải thích: 38 = 5 + 6 + 7 + 8 + 9 + 1+0 + 1+1.

Giải:

Phân tích tham số:

Ta cần một tham số `sum` là tổng các chữ số đã có.

Hàm `thu(i, gh, sum)` với mảng nhớ `F[i][sum]`

- Nếu i<0 thì trả về sum
- Nếu gh=false và F[i][sum]≥0 thì trả về F[i][sum]
- kq=0;
- maxc = (gh = true ? a[i] : 9)
- Với mỗi c chạy từ 0 đến maxc:
  - ghm = (gh=true) AND (c=maxc)
  - kq = kq + `thu(i-1, ghm, sum+c)`
- Nếu gh=false thì gán F[i][sum]=kq
- Trả về kq;

Hàm `G(x)`

- Tách các chữ số của x;
- Trả về `thu(n, true, 0)`;

Ghi chú: bài này không phải là đếm số mà là tổng các chữ số của các số… Vẫn dùng được, đáp số vẫn là `G(B)-G(A-1)`.

Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
#define baitoan "tongcs"
typedef long long int lli;
char a[19],n;
lli F[19][163];

lli thu(int i, bool gh,int sum)
 {
    bool ghm;
    if (i < 0)
    {
        return sum;
    }

    if(gh == false && F[i][sum] >= 0)
        return F[i][sum];

    lli kq = 0;
    char maxc = (gh ? a[i] : 9);
    for (char c = 0; c <= maxc; c++)
    {
        ghm = gh && (c == maxc);
        kq += thu(i-1, ghm, sum+c);
    }
    if (gh==false) F[i][sum] = kq;
    return kq;
 }

lli G(lli x)
{
    a[0] = 0;
    n = 0;
    while (x)
    {
        a[n] = x % 10;
        x /= 10;
        n++;
    }
    return thu(n-1,true,0);
}

int main()
{
freopen(baitoan".inp", "r", stdin);
freopen(baitoan".out", "w", stdout);
    memset(F, -1, sizeof(F));
    lli A, B;
    cin >> A >> B ;
    cout << G(B) - G(A-1);
    return 0;
}
```

Sau đây, tôi giới thiệu thêm một số bài toán dùng để học sinh tự giải.

IV. CÁC BÀI TOÁN TỰ LUYỆN

1. Counting Digits:

https://vn.spoj.com/problems/MDIGITS/vn

Cho hai số nguyên a, b. Viết tất cả các số nằm giữa a, b (tính cả 2 số này).

Tính xem mỗi chữ số 0, 1, .., 9 mỗi số xuất hiện bao nhiêu lần.

Ví dụ, nếu a = 1024 và b = 1032, dãy sẽ là

1024 1025 1026 1027 1028 1029 1030 1031 1032

và có 10 chữ số 0, 10 chữ số 1, 7 chữ số 2, 3 chữ số 3, 1 chữ số 4, 1 chữ số 5, 1 chữ số 6, 1 chữ số 7, 1 chữ số 8, 1 chữ số 9.

2. Cool Numbers

https://vn.spoj.com/problems/COOLNUMS/

CácCool Numbers là các số mà các chữ số của nó chia được thành 2 phần có tổng bằng nhau. Ví dụ, 23450 và 91125 là các Cool Numbers; còn 567 và 34523 không phải là Cool Numbers.

Đếm số lượng Cool Numbers trong phạm vi từ A đến B.

3. Đếm số có đúng K chữ số khác 0 và tổng các chữ số là lẻ và không lặp lại.

https://www.geeksforgeeks.org/count-numbers-with-exactly-k-non-zero-digits-and-distinct-odd-digit-sum/

Đếm các số nguyên từ 0 đến N ($N \leq 10^{18}$) thoả mãn các tính chất:

- Có đúng K chữ số khác 0;
- Tổng các chữ số là số lẻ;
- Tổng của các chữ số là duy nhất (không lặp lại).

Ví dụ 1: Input : N = 10, K = 1. Output : 5.

5 số đó là 1, 3, 5, 7 và 9, số 10 có tổng các chữ số là 1+0 = 1, trùng với số 1 nên không tính.

Ví dụ 2: Input : N = 100, K = 2. Output : 8

4. Số đối xứng:

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

http://lightoj.com/volume_showproblem.php?problem=1205

Số đối xứng là số nếu viết ngược thì cũng là chính nó, ví dụ các số: 16561, 11, 7 là các số đối xứng.

Đếm số đối xứng trong phạm vi từ A đến B.

Ví dụ:

Input	Output
1 100	18

5. Số có các chữ số con tăng dài nhất

https://toph.co/p/lids

Các chữ số của một số N có thể xoá bớt một số chữ số để tạo thành dãy chữ số tăng dài nhất, gọi là LIDS (longest increasing digit sub sequence) của số N. Ví dụ Độ dài LIDS của 122334 là 4, LIDS của 451263 là 3.

Trong các số nguyên từ A đến B, các bạn trả lời 2 câu hỏi sau:

1. Độ dài lớn nhất của LIDS của các số đó là bao nhiêu?
2. Có tất cả bao nhiêu cách tạo ra LIDS có độ dài lớn nhất đó.

Ví dụ:

Input	Output
117120	2 7

Input	Output
15432 15432	2 4

Giải thích:

Với VD1:

- Độ dài LIDS của số 117 là 2; có 2 LIDS độ dài 2 đều là {1, 7}
- Độ dài LIDS của số 118 là 2;có 2 LIDS độ dài 2 đều là {1, 8}
- Độ dài LIDS của số 119 là 2; có 2 LIDS độ dài 2 đều là {1, 9}
- Độ dài LIDS của số 120 là 2; có 1 LIDS độ dài 2 là {1, 2}

Vậy Độ dài lớn nhất là 2 và có tất cả 2+2+2+1 = 7 cách tạo LIDS độ dài 2.

Với VD2:

Phạm vi đã cho chỉ có 1 số duy nhất 15432. LIDS dài nhất là 2,và có 4 cách tạo LIDS độ dài 2 là {1, 5}, {1, 4}, {1, 3} và {1, 2}.

Giới hạn:

Subtask 1: $1 \le A \le B \le 10^3$.

Subtask 2: $1 \le A \le B \le 10^9$ và $B-A \le 10^3$.

Subtask 3: $1 \le A \le B \le 10^9$.

6. 369 Numbers

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

Một số được gọi là số 369 nếu có ít nhất một chữ số 3 và số chữ số 3 bằng số chữ số 6 bằng số chữ số 9.

Tìm số lượng số 369 trong phạm vi từ A đến B, chỉ in ra kết quả lấy dư khi chia cho 1000000007.

Ví dụ:

Input	Output
121 4325	60

Input	Output
432 4356	58

Giới hạn: $1 \le A \le B \le 10^{50}$.

7. Counting Strings

http://gautamdegitdp.blogspot.com/2016/09/counting-strings.html

Cho một xâu s bao gồm các chữ cái in hoa từ 'A' đến 'Z'. Bạn cần tìm ra có bao nhiêu xâu t có độ dài bằng với xâu s – cũng bao gồm các chữ in hoa từ 'A' đến 'Z' và thỏa mãn các điều kiện sau:

• Xâu t lớn hơn xâu s

• Xâu đảo ngược của t lớn hơn xâu đảo ngược của s

Tìm số lượng các xâu t như vậy. Bởi vì kết quả có thể rất lớn, hãy in ra phần dư của nó với $10^9 + 7$.

Input: Dữ liệu vào chứa một xâu s duy nhất.

Output:In ra một dòng duy nhất bao gồm một số nguyên tương ứng là phần dư của kết quả với $10^9 + 7$.

Ràng buộc: $1 \leq$ độ dài của S $\leq 10^5$

Ví dụ

Input	Output
ZAZ	25

Input	Output
XYZ	5

Giải thích

VD1: Để tạo ra xâu t thỏa mãn thì cần thay chữ cái A trong xâu S bởi bất cứ một chữ cái nào khác. Ví dụ thay 'A' bằng 'B', ta được xâu t = ZBZ.Chú ý rằng ZBZ có thứ tự theo từ điển lớn hơn là ZAZ. Đảo ngược xâu t (tức là: ZBZ) thì nó vẫn có thứ tự theo từ điển lớn hơn xâu s khi đảo ngược (tức là: ZAZ)

VD2: Có 5 xâu hợp lệ : YYZ, ZYZ, XZZ, YZZ, ZZZ.

8. Benny And The Broken Odometer

https://www.hackerearth.com/problem/algorithm/benny-and-the-broken-odometer/

Một ngày đẹp trời, Benny quyết định tính số km mà cô ấy đi được bằng xe đạp. Cô ấy mua một công-tơ-mét (loại đồng hồ đo số km đi được) và gắn vào chiếc xe đạp. Nhưng cái công-tơ-mét bị hỏng, nó không thể hiển thị chữ số 3. Nói một cách khác, nó không hiện những số có chữ số 3.

Ví dụ, sau số 1299, cái công-tơ-mét sẽ hiện số 1400.

Benny đã đạp xe rất nhiều và cô ấy muốn biết số km mà cô đã đi. Bạn được cho biết con số N mà Benny thấy trên công-tơ-mét. Nhiệm vụ của bạn là xác định khoảng cách thực tế mà Benny đã đi.

Giới hạn:$50 \leq N < 10^9$.

Ví dụ

INPUT	OUTPUT
5	4
14	12
76	59
67	51
40	27

Giải thích:

VD1: công-tơ-mét bỏ qua số 3, các số mà nó hiển thị là 1,2,4,5. Vậy Benny đã đi 4 km.

VD2: công-tơ-mét bỏ qua số 3, 13; các số mà nó hiển thị là 1,2,4,5,6,7,8,9,10, 11, 12, 14. Vậy Benny đã đi 12 km.

9. Số Whirligig

https://vn.spoj.com/problems/MZVRK/vn/

Số "whirligig" của 1 số là số thu được bằng cách xóa tất cả các chữ số nằm bên trái của số 1 ở bên phải phải nhất của số đó trong biểu diễn nhị phân.

Ví dụ, whirligig của 6 (110_2) là 2 (10_2), và whirligig của 40(101000_2) là 8 (1000_2).

Tính tổng tất cả các số whirligig của các số nằm trong khoảng [A,B].

Input

Gồm hai số nguyên A,B, ($1 \le A \le B \le 10^{15}$).

Output

Ghi ra tổng tìm được.

Ví dụ:

INPUT	OUTPUT
176 177	17
5 9	13
25 28	8

10. Counting Lucky Numbers

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

Tìm xem có bao nhiêu số từ a đến b mà trong biểu diễn nhị phân của nó, số lượng chữ số 1 là một số may mắn.

Một số là may mắn nếu trong biểu diễn thập phân của nó chỉ chứa chữ số 4 và 7, ví dụ, các số 4, 7, 47, 77 là các số may mắn, trong khi 14, 41 không phải.

Giới hạn $0 \le a \le b \le 10^{19}$.

Input

- T là số bộ test($T \le 10^5$)
- T dòng tiếp theo, mỗi dòng là 2 số a, b($a \le b$).

Output

Kết quả của mỗi bộ test.

Ví dụ:

INPUT	OUTPUT
2 15 15 63 63	1 0

Ngoài ra còn rất nhiều bài toán có thể được giải bằng kĩ thuật quy hoạch động chữ số được liệt kê thêm ở phụ lục 2.

V. KẾT LUẬN

Kĩ thuật quy hoạch động chữ số là một kĩ thuật hiệu quả để giải lớp bài toán liên quan đến tính chất của các chữ số.

Ngoài dạng toán đếm các số từ a đến b thoả mãn các điều kiện nào đó về các chữ số của nó, chúng ta còn có thể vận dụng kĩ thuật quy hoạch động chữ số vào các dạng toán: đếm số xâu thoả mãn điều kiện nào đó giữa các kí tự của nó, tìm số thứ k trong phạm vi từ 0 đến n thoả mãn các điều kiện nào đó về các chữ số của nó (Ví dụ bài số 6 trong phụ lục 2). Tôi sẽ thu thập thêm các bài tập cùng dạng để viết các chuyên đề khác.

Thông qua chuyên đề này, tôi mong muốn đóng góp thêm một tài liệu thiết thực trong việc bồi dưỡng học sinh giỏi. Tôi mong nhận được sự đóng góp ý kiến của quý thầy cô để tài liệu về chuyên đề này hoàn thiện hơn.

PHỤ LỤC 1.
CÁC HÀM VÀ PHÉP TOÁN SỬ DỤNG TRONG THUẬT TOÁN

1. Hàm UCLN: có thể dùng hàm có sẵn __gcd() trong C++
2. `mod` là phép chia lấy dư (VD: 9 `mod`3 = 0, tương đương với % trong NNLT C/C++)
3. `div`là phép chia lấy thương (VD: 7 `div`3 = 2, tương đương với / trong C++)
4. Bật bit c của số tt: `c or (1 shl c)` hoặc `tt | (1<<c)`
5. Trả về bit c của số tt: `tt and (1 shl c)` hoặc `tt & (1<<c)`