Môn: Tin học

CHUYÊN ĐỀ: QUY HOẠCH ĐỘNG CHỮ SỐ

Tháng 6/2023

MỤC LỤC

I. Mở đầu

Trong quá trình bồi dưỡng học sinh giỏi tôi nhận thấy học sinh đội tuyển của tôi rất ngại trong việc lập trình giải quyết các bài toán Quy hoạch động (QHĐ) chữ số. Một trong những lý do là các phải sử dụng các biến đổi toán học để giải bài toán. Một điểm yếu của nhiều học sinh là lười thực hiện các phép toán hoặc không tìm được thuật giải thích hợp.

QHĐ chữ số là một trong những chủ đề có xuất hiện nhiều trong các kỳ thi HSG Tin học ở Việt Nam. Chuyên đề này tôi đưa ra phương pháp giải quyết các bài toán tin liên quan đến QHĐ chữ số, hướng giải quyết, cảm nhận, có code và test kèm theo. Chuyên đề phù hợp với những học sinh đã làm quen với QHĐ chữ số, rèn luyện kĩ năng giải bài tập từ cơ bản dến nâng cao từ đó có thể giúp các em phát triển tư duy lôgic giải quyết những bài toán QHĐ chữ số nâng cao hơn nữa.

1. Kiến thức cần thiết

❶ Số học

❷ Đệ quy

❸ Quay lui - nhánh cận.

❹ Quy hoạch động

2. Phương pháp giải quyết các bài toán Quy hoạch động chữ số

a. Đặt vấn đề

Có những bài toán yếu cầu đếm số lượng các số nguyên trong một khoản rất lớn và phải thỏa mãn một số điều kiện nào đó liên quan tới chữ số của nó, việc thực hiện trong O(N) là không thể. QHD chữ số là một trong những cách giúp giải quyết những bài toán như vậy.

b. Ý tưởng chính

Gọi L là giới hạn bên trái và R là giới hạn bên phải của bài toán.

Gọi Cal(X) là số lượng số thỏa mãn các điều kiện của bài toán trong khoảng từ 0 đến X.

Kết quả của bài toán là Cal(R) - Cal(L-1).

Ví dụ, ta có X = a[1]a[2]…a[n] với n là số chữ số của X.

Tất cả số có dạng b[1]b[2]…b[n] nhỏ hơn hoặc bằng X nếu:

Với 0 <= b[i] <= a[i] và b[i-1] = a[i-1], b[i-2] = a[i-2]… b[1] = a[1].

Hoặc 0 <= b[i] <= 9, b[i-1] < a[i-1] hoặc b[i-2] < a[i-2]… b[1] < a[1] và b[i-1] <= a[i-1], b[i-2] <= a[i-2]… b[1] <= a[1](Với mọi i <= n)

Khi đó các số này chỉ cần thỏa mãn điều kiện mà đề cho.

c. Cấu trúc chung

Xây dựng hàm Cal(x):

- Phân tích X ra a[1]a[2]…a[n].

- Đặt F[i][…] là số chữ số thỏa mảng các trạng thái của F[i][…](số chiều của mảng sẽ khác nhau với mỗi bài toán) và <= a[1]a[2]…a[i].

- Xây dựng bài toán cơ sở.

- For i theo n và j, k, l… theo các trạng thái của F[i][…] .

F[i][…] += F[i-1][…] nếu các trạng thái của F[i-1][…] có thể tạo nên F[i][…].

Chương trình chính:

- Nhập INPUT.

- Khai báo L là giới hạn bên trái và R là giới hạn bên phải của bài toàn.

- Xuất Cal(R) - Cal(L-1).

II. Các bài tập minh họa

Bài tập trong chuyên đề xin được thiết kế theo trình tự từ dễ đến khó mục tiêu khi đưa vào sử dụng học sinh không cảm thấy bị quá tải đồng thời kích thích các em quyết tâm giải quyết yêu cầu các bài toán khi tiếp cận chuyên đề này.

A. Các bài toán chính (5 bài VIP)

Bài 1. Robot học đếm (cntK)

1) Mô tả bài toán:

Hôm nay các em học sinh lớp lá của trường mẫu giáo Superkid được học chủ đề đếm số. Yêu cầu của cô giáo như sau:

Cho một số nguyên n ($n \leq 10^{15}$) và một số nguyên k ($k \leq n$). Hãy đếm số lượng số $\leq n$ và có thứ tự từ điển $\leq k$.

Với khả năng của mình, các em học sinh dễ dàng tính được đáp án. Tuy nhiên cô giáo đã làm mất tờ giấy ghi đáp án và tìm ra đáp án là điều quá khó với người bình thường nên giờ cô cần bạn giúp.

2) Thuật toán:

Áp dụng QHĐ chữ số để đếm số lượng số $\leq n$ và có thứ tự từ điển $\leq k$ ta chia làm 2 trường hợp:

Trường hợp 1: Độ dài của số cần xây dựng bé hơn độ dài của số n.

Do độ dài của số cần xây dựng < độ dài của n nên số xây dựng chắc chắn < n. Từ đó suy ra ta chỉ cần đếm số lượng số có độ dài là len và có thứ tự từ điển < k.

Ta có f(i, lower): Xây dựng từ trái qua phải đến vị trí I (từ hàng lớn nhất về hàng đơn vị, lower là số có thứ tự từ điển bé hơn k chưa.

Với mỗi giá trị c điền vào vị trí i, ta sẽ cần phải tìm ra giá trị biến nwLow phù hợp.

Kết quả của bài toán sẽ là tổng f(len, 0) + f(len, 1) với $1 \leq len <$ độ dài của số n.

Trường hợp 2: Độ dài của số cần xây dựng bằng độ dài của số n.

Ta cần cố định một chiều là đã < hơn n chưa và đã bé hơn k chưa?

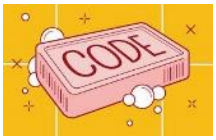
Đặt f(i, lower, lowerK): xây dựng từ trái qua phải đến vị trí I (từ hàng lớn nhất về hàng đơn vị, lower là số có thứ tự từ điển bé hơn n chưa, lowerK là số có thứ tự điển bé hơn K chưa.

Với mỗi giá trị c điền vào vị trí I, ta sẽ cần phải tìm ra giá trị biến nwLow và nwLowK phù hợp.

Kết quả của bài toán nào là f[len][lower][lowerK] với 0 ≤ lower, lowerK < 2.

Kết hợp kết quả của 2 bài toán con trên, ta sẽ ra được kết quả của bài toán cần tìm.

3) Code tham khảo:




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

vector <int> getDigit(int n)
{
    vector <int> digit;
    while (n)
    {
        digit.push_back(n % 10);
        n /= 10;
    }
    reverse(digit.begin(), digit.end());
    return digit;
}

int f1[22][2];
int count1(int len, vector <int> digitK)
{
    int _len = digitK.size() - 1;
    int maxlen = len;
    memset(f1, 0, sizeof(f1));
    while (digitK.size() < len) digitK.push_back(0);
    for (int c = 1; c < digitK[0]; c ++) f1[0][1] ++;
    f1[0][0] = 1 ;

    for (int i = 0; i < maxlen - 1; i ++) for (int lower = 0; lower
< 2; lower ++)
        for (int c = 0; c < 10; c ++)
            if (f1[i][lower])
            {
                if (!lower && c > digitK[i+1]) continue;
                if (!lower && i >= _len) continue;
                int nwLow = (lower) | (c < digitK[i+1]);
                f1[i+1][nwLow] += f1[i][lower];
            }
    int res = f1[len - 1][0] + f1[len - 1][1];
    return res;
```

```
}

int f2[22][2][2];
int count2(int len, vector <int> digit, vector <int> digitK)
{
    int maxlen = len;
    int _len = digitK.size() - 1;
    memset(f2, 0, sizeof(f2));
    while (digitK.size() < len) digitK.push_back(0);
    for (int c = 1; c < 10; c ++) {
        if (c > digit[0]) continue;
        if (c > digitK[0]) continue;
        int nwLow = c < digit[0];
        int nwLowK = c < digitK[0];
        f2[0][nwLow][nwLowK] ++;
    }
    for (int i = 0; i < maxlen - 1; i ++)
        for (int lower = 0; lower < 2; lower ++)
            for (int lowerK = 0; lowerK < 2; lowerK ++)
                for (int c = 0; c < 10; c ++)
                {
                    if (!lower && c > digit[i+1]) continue;
                    if (!lowerK && c > digitK[i+1]) continue;
                    if (!lowerK && i >= _len) continue;
                    int nwLow = (lower) | (c < digit[i+1]);
                    int nwLowK = (lowerK) | (c < digitK[i+1]);
                    f2[i+1][nwLow][nwLowK] += f2[i][lower][lowerK];
                }
    int res = 0;
    for (int lower = 0; lower < 2; lower ++)
        for (int lowerK = 0; lowerK < 2; lowerK ++)
            res += f2[len - 1][lower][lowerK];
    return res;
}

void process()
{
    int n, k;
    cin >> n >> k;
    vector <int> digit = getDigit(n);
    vector <int> digitK = getDigit(k);

    int res = 0;
    for (int len = 1; len <= digit.size(); len ++) {
        if (len < digit.size()) {
            res += count1(len, digitK);
//            cout << len << " " << count1(len, digitK) << endl;
        }
        else res += count2(len, digit, digitK);
    }
    cout << res;
```

```
}

signed main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

    freopen("cntK.inp", "r", stdin);
    freopen("cntK.out", "w", stdout);

    int ntest = 1;
    while (ntest --) process();
    return 0;
}
```

4) Cảm nhận

Bài toán này mục tiêu nhằm rèn luyện cho học sinh kỹ năng kiểm soát tốt chương trình được viết đồng thời thay đổi cách nhìn của học sinh về QHĐ chữ số bằng cách phân tích ra từng trường hợp riêng biệt để có thể dễ xử lý việc có thứ tự từ điển.

Bài 2. Trật tự (nk05ordr)

1) Mô tả bài toán:

Quỳnh là một blink chính hiệu. Sau khi nghe tin concert Born Pink đặt chân tới Việt Nam Quỳnh rất hào hứng. Thế nhưng do gia cảnh nghèo khó, Quỳnh không đủ tiền để mua vé đi concert. Vào một ngày đẹp trời, Hùng - bạn thân của Quỳnh - đang gặp phải bài toán khó như sau:

Xét các số nguyên từ 1 đến N. Các số này được sắp xếp theo thứ tự từ điển. Ví dụ với N=11, ta có dãy số sau khi sắp xếp là 1, 10, 11, 2, 3, 4, 5, 6, 7, 8, 9.

Ký hiệu $Q_{N,K}$ là vị trí của số K trong dãy được sắp xếp theo cách nói trên. Ví dụ $Q_{11,2} = 4$ Cho các số nguyên K và M. Hãy tìm số nguyên N nhỏ nhất thỏa mãn $Q_{N,K} = M$.

Là bạn thân của Quỳnh nên Hùng quyết định nhờ Quỳnh giúp mình giải quyết bài toán và hứa sẽ mua cho cô vé đi xem Concert. Và điều bất ngờ là Quỳnh cũng không biết làm. Thế nên các bạn hãy giúp Quỳnh giải quyết bài toán để Quỳnh có thể đi xem concert nhé.

Input: Dòng đầu tiên chứa số nguyên T cho biết số bộ test. $T \leq 10$ Mỗi bộ test bao gồm 1 dòng duy nhất chứa 2 số nguyên K và M ($1 \leq K, M \leq 10^9$).

Output: Với mỗi bộ test xuất ra số N, hoặc 0 nếu không tồn tại N.

Input	*Output*
1 2 4	11

2) Thuật toán:

Ví dụ:

N = 10 ⇒ Q[10] = 1 10 2 3 4 5 6 7 8 9

N = 11 ⇒ Q[11] = 1 10 11 2 3 4 5 6 7 8 9

N = 12 ⇒ Q[12] = 1 10 11 12 2 3 4 5 6 7 8 9

Giả sử đặt k = 6, ta có:

N = 10 → m = 7

N = 11 → m = 8

N = 12 → m = 9

Nhận xét:

Khi cố định K, n tăng → m cũng tăng, n giảm → m cũng giảm.

Vậy để giải bài toán ta dùng thuật chặt nhị phân trên số n:

Với mỗi số n, ta đặt ra câu hỏi số thứ k nằm ở vị trí bao nhiêu trong dãy khi được sắp xếp tăng dần theo thứ tự từ điển?

⇔ Có bao nhiêu số ≤ n mà n ≤ k theo thứ tự từ điển? Để giải bài toán có bao nhiêu số ≤ n ≤ k theo thứ tự từ điển, ta thực hiện phương pháp QHĐ chữ số như sau:

Ta chia làm 2 trường hợp:

Trường hợp 1: Độ dài của số cần xây dựng bé hơn độ dài của số n.

Do độ dài của số cần xây dựng < độ dài của n nên số xây dựng chắc chắn < n. Từ đó suy ra ta chỉ cần đếm số lượng số có độ dài là len và có thứ tự từ điển < k.

Ta có f(i, lower): Xây dựng từ trái qua phải đến vị trí I (từ hàng lớn nhất về hàng đơn vị, lower là số có thứ tự từ điển bé hơn k chưa.

Với mỗi giá trị c điền vào vị trí i, ta sẽ cần phải tìm ra giá trị biến nwLow phù hợp.

Kết quả của bài toán sẽ là tổng f(len, 0) + f(len, 1) với 1 ≤ len < độ dài của số n.

Trường hợp 2: Độ dài của số cần xây dựng bằng độ dài của số n.

Ta cần cố định một chiều là đã < hơn n chưa và đã bé hơn k chưa?

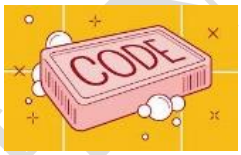
Đặt f(i, lower, lowerK): xây dựng từ trái qua phải đến vị trí I (từ hàng lớn nhất về hàng đơn vị, lower là số có thứ tự từ điển bé hơn n chưa, lowerK là số có thứ tự điển bé hơn K chưa.

Với mỗi giá trị c điền vào vị trí I, ta sẽ cần phải tìm ra giá trị biến nwLow và nwLowK phù hợp.

Kết quả của bài toán nào là f[len][lower][lowerK] với 0 ≤ lower, lowerK < 2.

Kết hợp kết quả của 2 bài toán con trên, ta sẽ ra được kết quả của bài toán cần tìm.

3) Code tham khảo:




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

int m, k;

vector <int> getDigit(int n)
{
    vector <int> digit;
    while (n)
    {
        digit.push_back(n % 10);
        n /= 10;
    }
    reverse(digit.begin(), digit.end());
    return digit;
}

int f1[22][2];
int count1(int len, vector <int> digitK)
{
    int _len = digitK.size() - 1;
    int maxlen = len;
```

```
    memset(f1, 0, sizeof(f1));
    while (digitK.size() < len) digitK.push_back(0);
    for (int c = 1; c < digitK[0]; c ++) f1[0][1] ++;
    f1[0][0] = 1 ;

    for (int i = 0; i < maxlen - 1; i ++) for (int lower = 0;
lower < 2; lower ++)
        for (int c = 0; c < 10; c ++)
            if (f1[i][lower])
            {
                if (!lower && c > digitK[i+1]) continue;
                if (!lower && i >= _len) continue;
                int nwLow = (lower) | (c < digitK[i+1]);
                f1[i+1][nwLow] += f1[i][lower];
            }
    int res = f1[len - 1][0] + f1[len - 1][1];
    return res;
}

int f2[22][2][2];
int count2(int len, vector <int> digit, vector <int> digitK)
{
    int maxlen = len;
    int _len = digitK.size() - 1;
    memset(f2, 0, sizeof(f2));
    while (digitK.size() < len) digitK.push_back(0);
    for (int c = 1; c < 10; c ++) {
        if (c > digit[0]) continue;
        if (c > digitK[0]) continue;
        int nwLow = c < digit[0];
        int nwLowK = c < digitK[0];
        f2[0][nwLow][nwLowK] ++;
    }
    for (int i = 0; i < maxlen - 1; i ++)
        for (int lower = 0; lower < 2; lower ++)
            for (int lowerK = 0; lowerK < 2; lowerK ++)
                for (int c = 0; c < 10; c ++)
                {
                    if (!lower && c > digit[i+1]) continue;
                    if (!lowerK && c > digitK[i+1]) continue;
                    if (!lowerK && i >= _len) continue;
                    int nwLow = (lower) | (c < digit[i+1]);
                    int nwLowK = (lowerK) | (c < digitK[i+1]);
                    f2[i+1][nwLow][nwLowK] +=
f2[i][lower][lowerK];
                }
    int res = 0;
    for (int lower = 0; lower < 2; lower ++)
        for (int lowerK = 0; lowerK < 2; lowerK ++)
            res += f2[len - 1][lower][lowerK];
    return res;
}
```

```
int cal(int n)
{
    if (k > n) return -1;
    vector <int> digit = getDigit(n);
    vector <int> digitK = getDigit(k);
    int res = 0;
    for (int len = 1; len <= digit.size(); len ++) {
        if (len < digit.size()) {
            res += count1(len, digitK);
//            cout << len << " " << count1(len, digitK) <<
endl;
        }
        else res += count2(len, digit, digitK);
    }
    return res;
}
void process()
{
    cin >> k >> m;
    int best = 0;
    int d = 1, c = 1e18;

    while (d <= c)
    {
        int g = (d + c) / 2;
        int cur = cal(g); /// dem xem so m o vi tri bao nhieu
khi n = g
        if (cur >= m)
        {
            if (cur == m) best = g;
            c = g - 1;
        }
        else d = g + 1;
    }
    cout << best << '\n';
}
signed main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    freopen("nk05ordr.inp", "r", stdin);
    freopen("nk05ordr.out", "w", stdout);
    int test = 1;
    cin >> test;
    while (test --) process();
    return 0;
}
```

4) Cảm nhận

Bài 2 là một bài toán được phát triển từ các tư duy đã có ở bài 1. Để có thể dễ dàng giải bài 2 bằng phương pháp QHĐ chữ số, ta phải cần nền tản của bài 1

để làm gốc, từ đó tư duy nhìn nhận được các tính chất của bài toán và kết hợp tư duy thêm bước chặt nhị phân để có thể giải quyết được bài toán.

Bài 3. SELF-DIVISIBLE

1) Mô tả bài toán:

Nam là một học sinh của trường tiểu học Sakura. Hôm nay Nam được cô giáo dạy buổi học đếm đầu tiên cậu bắt đầu hứng thú với các bài toán đếm và tự tìm hiểu thêm những bài toán cho mình. Nam tìm thấy một bài toàn như sau: Đếm số lượng số tự chia hết có đúng N chữ số (một số tự nhiên chia hết cho tất cả các chữ số của nó được gọi là số tự chia hết).

Ví dụ: số 324 là số chia hết cho 3, 2 và 4.

Một số ví dụ khác về số tự chia hết là: 5, 12, 784, 8736.

Số 102 không phải là số tự chia hết vì nó không chia hết cho 0.

Do giới hạn quá lớn nên Nam mất rất nhiều thời gian để giải được bài này. Sau nhiều ngày ngồi đếm cuối cùng cậu cũng đã tìm ra kết quả nhưng do phải đếm nhiều ngày liền nên có thể có sai sót trong quá trình đếm. Vì vậy Nam cần bạn giúp đỡ để tìm ra kết quả chính xác.

Yêu cầu: Đếm số lượng số tự chia hết có đúng N chữ số. Vì kết quả có thể rất lớn, bạn chỉ cần in ra phần dư khi chia cho 10007.

Input: Một dòng duy nhất ghi số N ($N \leq 500$).

Output: Phần dư khi chia số lượng số tự chia hết có đúng N chữ số cho 10007.

Input	Output
3	56

2) Thuật toán:

Nhận xét:

Dễ dàng nhận ra, bất kì số nào có chứa chữ số 0 thì không phải là số tự chia hết. Đối với mỗi số ta chỉ cần những thông tin sau số đó bao gồm những chữ số nào, phần dư của số đó khi chia dư 7, 8 , 9 và số đó kết thúc bằng số mấy.

Xét qua tất cả các chữ số mà số đó có. Nếu số đó có chữ số 1 thì ta không cần quan tâm thì bất kì số nào cũng chia hết cho 1. Với số 2 thì ta cần xét phần dư của nó với 8 có chia hết cho 2 không nếu có thì số đó chia hết cho 2 ngược lại thì

không, số 4 và 8 cũng tương tự với số 2. Số 3 thì ta cần xét phần dư của nó với 9 có chia hết cho 3 không nếu có thì số đó chia hết cho 3 ngược lại thì không, số 9 cũng tương tự với số 3. Với số 5 ta chỉ cần xét xem số cuối cùng có phải là 0 hay 5 không nếu phải thì số đó chia hết cho 5 ngược lại thì không. Còn với số 6 ta xét xem số đó có chia hết cho 2 và 3 không với phương pháp đã nếu trên nếu có thì số đó chia hết cho 6 ngược lại thì không.

Với suy luận trên ta có thể lập công thức QHD để giải bài toán này.

3) Code tham khảo:


```
#include "bits/stdc++.h"
using namespace std;
int BIT(int x) {
    return (1 << x);
}
int OR(int x, int y) {
    return (x | y);
}
const int N = 505, mod = 10007;
int n;
int g[(1 << 9)][7][8][9];
int f[(1 << 9)][7][8][9];
bool check(int m, int num, int m7, int m8, int m9) {
    for (int j = 2 ; j <= 9 ; j++)
        if (m & BIT(j-1)) {
            if ((j == 2 || j == 4 || j == 8) && m8 % j) return
0;
            if ((j == 3 || j == 9)  && m9 % j) return 0;
            if (j == 6 && (m8 % 2 || m9 % 3)) return 0;
            if (j % 5 == 0 && num != 5) return 0;
            if (j % 7 == 0 && m7 % j) return 0;
        }
    return 1;
}
signed main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
freopen("SELF-DIVISIBLE.inp", "r", stdin);
freopen("SELF-DIVISIBLE.out", "w", stdout);
    cin >> n;
    g[0][0][0][0] = 1;
    for (int i = 1 ; i < n ; i++) {
        for (int m = 0 ; m < (1 << 9) ; m++)
            for (int num = 1 ; num <= 9 ; num++)
                for (int se = 0 ; se < 7 ; se++)
                    for (int ei = 0 ; ei < 8 ; ei++)
```

```
                        for (int ni = 0 ; ni < 9 ; ni++) {
                            int nm = OR(m, BIT(num-1));
                            int m7 = (se*10+num)%7;
                            int m8 = (ei*10+num)%8;
                            int m9 = (ni*10+num)%9;
                            f[nm][m7][m8][m9] +=
g[m][se][ei][ni];
                            f[nm][m7][m8][m9] %= mod;
                        }
        for (int m = 0 ; m < (1 << 9) ; m++)
            for (int se = 0 ; se < 7 ; se++)
                for (int ei = 0 ; ei < 8 ; ei++)
                    for (int ni = 0 ; ni < 9 ; ni++) {
                        g[m][se][ei][ni] = f[m][se][ei][ni];
                        f[m][se][ei][ni] = 0;
                    }
    }
    int res = 0;
    for (int m = 0 ; m < (1 << 9) ; m++) {
        for (int num = 1 ; num <= 9 ; num++) {
            for (int se = 0 ; se < 7 ; se++)
                for (int ei = 0 ; ei < 8 ; ei++)
                    for (int ni = 0 ; ni < 9 ; ni++) {
                        int m7 = (se*10+num)%7;
                        int m8 = (ei*10+num)%8;
                        int m9 = (ni*10+num)%9;
                        int nm = OR(m, BIT(num-1));
                        if (check(nm, num, m7, m8, m9)) {
                            res += g[m][se][ei][ni];
                            res %= mod;
                        }
                    }
        }
    }
    cout << res;
    return 0;
 }
```

4) Cảm nhận

Bạn sẽ phải sử dụng các biến đổi toán học để giải bài này. Một điểm yếu của nhiều học sinh là lười thực hiện các phép toán, những bài toán như này sẽ luyện cho học sinh tư duy toán học.

Bài 4. Soluongso

1) Mô tả bài toán:

Hôm nay vào giờ học ngoại khóa môn Tin học của lớp 10 Tin trường THPT Nguyễn Huệ, cô giáo đã ra một bài toán vô cùng hóc búa và yêu cầu những học sinh của mình phải giải xong thì mới được về nhà. Bài toán nội dung như sau:

Cho 3 số nguyên dương T, X, Y. Đếm số lượng các số có tích của nó với tổng các chữ số của nó nằm trong khoảng [X, Y] cho trước.

Các bạn lớp 10 Tin nhờ bạn giúp giải bài toán trên để các bạn có thể sớm được về nhà.

INPUT:

Dòng đầu ghi số nguyên dương T là số test (T<21).

T dòng sau mỗi dòng ghi cặp số X, Y ($0 < X \leq Y < 10^9$)

OUTPUT:

Ghi trên T dòng, mỗi dòng là số lượng số tìm được.

2) Thuật toán:

Gọi x là số hợp lệ, S là tích các chữ số của x. Theo đề bài ta có bất đẳng thức: $L \leq x * s \leq R$

Nhận thấy: s nhỏ ($s \leq 19 * 9$) suy ra ta có thể cố định s bằng 1 for

→ Ta được bất đẳng thức mới $L / s \leq x \leq R$. Lúc này bài toán trở về đếm số lượng số có tổng các chữ số bằng s trong đoạn l, r. Để giải bài toán trên ta sẽ biến đổi thành:

(đếm số lượng số có tổng các chữ số bằng s trong đoạn từ 1 → r) - (số có tổng các chữ số bằng s trong đoạn từ 1 → l – 1)

Gọi dp[i][sum][positive][lower] là:

- Các số trong đoạn từ numdigit → i (từ hàng lớn nhất về hàng đơn vị)
- Đã xây dựng được một dãy có tổng các chữ số bằng sum
- Postive : Đã có một số khác 0 xuất hiện trước đó hay chưa (kiểm tra số 0 có nghĩa)
- Lower : Đã bé hơn số x cho chưa

Với mỗi số c mới điền vào vị trí I – 1, ta sẽ xây dựng các biến nwPos, nwLow, nwSum sao cho phù hợp.

Kết quả trả về: dp[1][curSum][1][0] + dp[1][curSum][1][1]

3) Code tham khảo:


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

```
typedef long long ll;

const int numDigit = 20;

int digit[numDigit + 55];
int dp[numDigit + 55][222][2][2];

int cal(int x, int curSum)
{
    for (int i = 1; i <= numDigit; i ++){
        digit[i] = x % 10;
        x /= 10;
    }
    memset(dp, 0, sizeof(dp));
    dp[numDigit + 1][0][0][0] = 1;
    for (int i = numDigit + 1; i >= 2; i --)
        for (int sum = 0; sum <= 200; sum ++)
            for (int postive = 0; postive < 2; postive ++)
                for (int lower = 0; lower < 2; lower ++) if
(dp[i][sum][postive][lower])
                    for (int c = 0; c < 10; c ++)
                    {
                        if (!lower && c > digit[i-1]) break;
                        int nwLower = (lower) || (c < digit[i-
1]);
                        int nwPostive = (postive) || (c > 0);
                        if (sum + c <= 200)
                            dp[i-1][sum +
c][nwPostive][nwLower] += dp[i][sum][postive][lower];
                    }
    return dp[1][curSum][1][0] + dp[1][curSum][1][1];
}

void process()
{
    int l,r;
    cin >> l >> r;
    int res = 0;
    for (int curSum = 1; curSum <= 19 * 9; curSum ++)
    {
        int x = l / curSum + (l % curSum != 0);
        int y = r / curSum;
        if (x != 0)
            res += cal(y, curSum) - cal(x - 1, curSum);
    }
    cout << res << "\n";
}
signed main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    #define TASK "soluongso"
```

```
    freopen(TASK".inp", "r", stdin);
    freopen(TASK".out", "w", stdout);
    int test;
    cin >> test;
    while (test --)
    {
        process();
    }
    return 0;
}
```

4) Cảm nhận

Bài 4 là một bài khá đặc biệt. Để có thể tư duy ra được lời giải ngoài việc phải quy hoạch động chữ số như bình thường, học sinh cần phải thực hiện nháp, biến đổi các bất đẳng thức để có thể biến đổi bài toán khó ban đầu thành một bài toán quen thuộc với quy hoạch động chữ số.

Bài 5. Dãy chữ số (Pdigit)

1) Mô tả bài toán:

Vào ngày nghỉ cuối tuần An được bố dẫn đi vào cửa hàng mua đồ chơi. Sau khi nhìn thấy mẫu mô hình Lego Batman siêu xịn An đòi bố mua. Bố An ra một điều kiện nếu An giải được bài toán thì bố mua cho An. Bài toán bố đặt ra cho An như sau: "Đếm số lượng số nguyên dương có tổng các chữ số là số nguyên tố". Là một học sinh giỏi toán nên An có thể giải được bài toán trên ngay tức thì. Tuy nhiên bố An lại nâng cấp bài toán lên một tầm cao mới: "đếm xem có bao nhiêu chữ số 0, chữ số 1, … chữ số 9 có trong các số đó". Lúc này bài toán lại trở nên quá khó với An. Bạn có thể giúp An hoàn thành bài toán.

Xét các số nguyên dương có tổng các chữ số là số nguyên tố: 2, 3, 5, 7, 11, 12, 14, 16, 20, 21, 23

Viết liên tiếp các số nguyên dương trong đoạn [A, B] có tổng các chữ số là số nguyên tố ta được một dãy các chữ số.

Ví dụ: A = 10, B = 25 ta được dãy 1112141620212325

Người ta muốn khảo sát xem trong dãy có bao nhiêu chữ số 0, chữ số 1, …, bao nhiêu chữ số 9.

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

Output: Gồm một dòng chứa 10 số là số lượng chữ số 0, chữ số 1, …, bao nhiêu chữ số 9 (các số cách nhau một dấu cách)

Ví dụ:

Input	Output
1 5	0 0 1 1 0 1 0 0 0 0
123 456	26 51 62 61 43 21 19 22 23 23

2) Thuật toán:

Để đếm số lượng số phải sử dụng là bao nhiêu trong đoạn từ a → b các số có tổng là nguyên tố ta thực hiện đếm với từng chữ số thuộc đoạn từ 1 → b (cntR) - cho từng chữ số trong đoạn từ 1 → a - 1(cntL)

Bài toán con: Đếm số lượng chữ số cần dùng trong trong đoạn từ 1 → x sao cho tổng các chữ số được xây dựng là một số nguyên tố.

Đặt dp[i][sum][positive][lower] lần lượt là:

- I: Xét các số từ bên trái về I (từ hang cao nhất về hang đơn vị)
- Sum: tổng các chữ số mà số đã xây dựng được là sum
- Posive: Số đã xây dựng đã dương hay chưa (dùng để kiểm tra số 0 có nghĩa)
- Lower: số đã xây dựng đã bé hơn x hay chưa.
- c: số điền vào vị trí i sẽ là số nào.

Với mỗi trạng thái, ta cần tìm một trạng thái mới (i+1, sum + c,nwPos, nwLow, c) sao cho phù hợp với trạng thái cũ để thực hiện tối ưu.

Ta sẽ thực hiện lấy kết quả ở trạng thái cuối là với:

- i = numDigit
- sum là một số nguyên tố
- pos = 1
- $0 \leq lower \leq 1$

Với mỗi trạng thái hợp lệ, ta sẽ cộng vào mảng cnt cho biết số lượng số c có thể sử dụng để tạo nên trạng thái này.

3) Code tham khảo:


```
#include <bits/stdc++.h>
#define sz(v) (v).size()
#define ALL(v) (v).begin(), (v).end()
```

```
#define int long long
using namespace std;

const int numDigit = 16;
const int maxSum = 160;

int dp[numDigit+5][maxSum+5][2][2][10];
int cntDigit[numDigit+5][maxSum+5][2][2];
int cntR[numDigit+5], cntL[numDigit+5];
int digits[numDigit+5];
int ans[numDigit+5];

void add(int a[], int b[])
{
    for (int i = 0; i < 10; i ++)
        a[i] += b[i];
}

bool IsPrime(int x)
{
    if (x < 2) return 0;
    for (int i = 2; i * i <= x; i ++){
        if (x % i == 0) return 0;
    }
    return 1;
}

void process(int n, int cnt[])
{
    vector <int> save;
    memset(digits, 0, sizeof(digits));

    while (n > 0){
        save.push_back(n % 10);
        n /= 10;
    }

    while (save.size() < numDigit)
        save.push_back(0);
    save.push_back(0);

    reverse(ALL(save));

    for (int i = 0; i < save.size(); i ++)
        digits[i] = save[i];

//    for (int i = 1; i <= numDigit; i ++)
//        cout << digits[i] << " ";
//    cout << endl;

    /// DP
    memset(cntDigit, 0, sizeof(cntDigit));
```

```
    memset(dp, 0, sizeof(dp));

    cntDigit[0][0][0][0] = 1;
    for (int i = 0; i < numDigit; i ++)
    {
        for (int sum = 0; sum < maxSum; sum ++)
            for (int lower = 0; lower <= 1; lower ++)
                for (int postive = 0; postive <= 1; postive
++)
                    for (int digit = 0; digit <= 9; digit ++)
                    {

                        if (!lower && digit > digits[i+1])
continue;
                        int newSum = sum + digit;
                        int newLower = (lower) || (digit <
digits[i+1]);
                        int newPostive = (postive) || (digit >
0);


cntDigit[i+1][newSum][newLower][newPostive] +=
cntDigit[i][sum][lower][postive];
                        if (newPostive == 0 && digit == 0)
continue;

add(dp[i+1][newSum][newLower][newPostive],
dp[i][sum][lower][postive]);

dp[i+1][newSum][newLower][newPostive][digit] +=
cntDigit[i][sum][lower][postive];
                    }
    }
//    for (int i = 1; i <= numDigit; i ++)
//        cout << i << " " << cntDigit[i][0][0][0] << endl;

    for (int digit = 0; digit < 10; digit ++) cnt[digit] = 0;
    for (int sum = 2; sum <= maxSum; sum ++) if (IsPrime(sum))
        for (int digit = 0; digit < 10; digit ++)
            for (int lower = 0; lower <= 1; lower ++){
                cnt[digit] +=
dp[numDigit][sum][lower][1][digit];
            }
}

signed main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    freopen("pdigit.inp", "r", stdin);
    freopen("pdigit.out", "w", stdout);
    int l,r;
```

```
    cin >> l >> r;
//    assert(r < (int)1e3);


    process(r, cntR);
    process(l-1, cntL);

    for (int i = 0; i <= 9; i ++)
        cout << cntR[i] - cntL[i] << " ";
    return 0;
}
```

4) Cảm nhận

Pdigit là một bài toán khá khó đòi hỏi khả năng lập trình của người code phải vững. Khác với các bài quy hoạch động thông thường chỉ trả về một giá trị là kết quả của một biểu thức gì đó. Bài pdigit đòi hỏi khi QHĐ hàm QHĐ của ta phải trả về một mảng các số là số lượng số cần dùng. Không chỉ thế, việc phải truyền vào các mảng tham chiếu sẽ làm cho bài toán thêm phần rắc rối.

B. Các bài toán mở rộng

Bài 6. Atcoder_dp_s

(https://oj.vnoi.info/problem/atcoder_dp_s)

1) Mô tả bài toán:

Cho 2 số nguyên K và D. Hãy đếm số lượng số nguyên (modulo 10^9+7) trong phạm vi từ 1 đến K (tính cả 1 và K) thoả mãn: Tổng các chữ số trong biểu diễn thập phân của số đó là bội của D.

Input:

- Dòng thứ nhất chứa số nguyên K ($1 \leq K \leq 10^{10000}$)
- Dòng thứ hai chứa số nguyên D ($1 \leq D \leq 100$)

Output:

In ra số lượng số nguyên thoả mãn điều kiện, modulo $10^9 + 7$

Ví dụ:

Input	Output
30 4	6
1000000009 1	2
9876543210987654321058	635270834

2) Thuật toán:

Xây dựng các số nhị phân từ trái sang phải (trái = chữ số to nhất). Cần duy trì các thông tin:

- Số hiện tại có < N không
- Chữ số 0 đã có nghĩa hay chưa
- Phần dư của tổng các chữ số đó cho d

f(i, cnt, positive, lower): số cách xây dựng từ 1 → i, lower = true nếu số hiện tại đã < N, positive = true nếu đã xuất hiện 1 số lớn hơn 0 trước đó và số 0 này là số 0 có nghĩa

Kết quả: f(1, 0, 1, 0) + f(1, 0, 1, 1)

3) Code tham khảo:

Code tham khảo có thể chấm bài trên: https://oj.vnoi.info/problem/atcoder_dp_s

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

const int numDigit = 1e4+55;
const int MOD = 1e9+7;

int digit[numDigit + 55], dp[numDigit + 55][111][2][2];
int P[numDigit + 55];

void add(int &a, const int &b)
{
    a += b;
    if (a >= MOD) a -= MOD;
}

int cal(string s, int maxsum)
{
    reverse(s.begin(), s.end());
    for (int i = 0; i < s.size(); i ++) digit[i+1] = s[i] -
'0';

    P[0] = 1;
    for (int i = 1; i <= numDigit; i ++)
        P[i] = (P[i-1] * 10) % maxsum;
    dp[numDigit + 1][0][0][0] = 1;
    for (int i = numDigit + 1; i >= 2; i --)
    {
        for (int sum = 0; sum < maxsum; sum ++)
            for (int postive = 0; postive < 2; postive ++)
                for (int lower = 0; lower < 2; lower ++) if
(dp[i][sum][postive][lower])
                    for (int c = 0; c < 10; c ++)
                    {
                        if (!lower && c > digit[i-1]) break;
                        int nwPos = (postive) || (c > 0);
                        int nwLow = (lower) || (c < digit[i-
1]);
```

```
                            add(dp[i - 1][(sum + c) %
maxsum][nwPos][nwLow], dp[i][sum][postive][lower]);
                        }
    }

    int res = 0;
    for (int lower = 0; lower < 2; lower ++)
        add(res, dp[1][0][1][lower]);
    return res;
}

signed main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

    freopen("atcoder_dp_s.inp", "r", stdin);
    freopen("atcoder_dp_s.out", "w", stdout);

    string s;
    cin >> s;
    int d;
    cin >> d;
    cout << cal(s, d);

  return 0;
}
```

4) Cảm nhận

Bài 6 là một bài khá cơ bản. Để giải được bài này ta cần phải biết qua được quy hoạch động digit, đồng thời phải để ý được là không nên lưu hết tất cả các bội của D mà chỉ nên lưu lấy phần dư của nó để tối ưu bộ nhớ cũng như tối ưu thời gian.

Bài 7. Số nhị phân có nghĩa – Binary

https://oj.vnoi.info/problem/binary

1) Mô tả bài toán:

Cho số nguyên không âm N($N<2^{31}$). Hãy xác định xem trong phạm vi từ 0 tới N có bao nhiêu số mà trong dạng biểu diễn nhị phân của nó có đúng K chữ số 0 có nghĩa.

Ví dụ: N = 18, K = 3 có 3 số:

1. 8 =1000
2. 17 =10001
3. 18 = 10010

Input: Gồm một số dòng, mỗi dòng chứa hai số nguyên N và K cách nhau một dấu cách.

Output: Ứng với mỗi bộ N, K ở input đưa ra số lượng tìm được.

Input	Output
18 3 8 1	3 4

2) Thuật toán:

Xây dựng các số nhị phân từ trái sang phải (trái = chữ số to nhất). Cần duy trì các thông tin:

- Số hiện tại có < N không
- Số chữ số 0 có nghĩa

f(i, positive, lower, k) = số cách xây dựng từ 1 → i, lower = true nếu số hiện tại đã < N, positive = true nếu đã xuất hiện 1 số lớn hơn 0 trước đó và số 0 này là số 0 có nghĩa.

3) Code tham khảo:

Code tham khảo có thể chấm bài trên: https://oj.vnoi.info/problem/binary

```
#include <bits/stdc++.h>
#define BIT(x,i) (((x) >> (i)) & 1)
#define MASK(x) ((1) << (x))
#define int long long
using namespace std;

const int numDigit = 32;

int digit[numDigit + 55];
int dp[numDigit + 55][numDigit + 55][2][2];

void process(int x, int k)
{
    for (int i = 0; i < numDigit; i ++) digit[i] = BIT(x, i);

//    for (int i = 0; i < numDigit; i ++) cout << digit[i] <<
" ";
//    cout << endl;

    if (k > numDigit) {
```

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

    memset(dp, 0, sizeof(dp));
    dp[numDigit][0][0][0] = 1; /// dp[i][cnt][postive][lower]

    for (int i = numDigit; i >= 1; i --)
    {
        for (int cnt = 0; cnt <= k; cnt ++)
            for (int postive = 0; postive < 2; postive ++)
                for (int lower = 0; lower < 2; lower ++) if
(dp[i][cnt][postive][lower])
                    for (int c = 0; c < 2; c ++)
                    {
                        if (!lower && c > digit[i-1])
continue;
//                          cout << i << " " << cnt << " " <<
postive << " " << lower << endl;
                        int nwPos = (postive) || (c > 0);
                        int nwLow = (lower) || (c < digit[i-
1]);
                        dp[i-1][cnt + ((!nwPos) ? 0 : (c ==
0))][nwPos][nwLow] += dp[i][cnt][postive][lower];
                    }
    }

    cout << dp[0][k][1][1] + dp[0][k][1][0] + (k == 1) <<
endl;
}

signed main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

    freopen("binary.inp", "r", stdin);
    freopen("binary.out", "w", stdout);

    int n,k;
    while (cin >> n >> k)
    {
        process(n , k);
    }

    return 0;
}
```

4) Cảm nhận

Để giải được bài toán này khá cơ bản. Để giải được bài này, ngoài việc phải biết qua một chút cơ bản về QHĐ chữ số ta cần phải có thêm một nhận xét là nếu mà x < n thì biểu diễn nhị phân của x cũng phải < hơn n

Bài 8. Đếm số - Counting

https://lqdoj.edu.vn/problem/cses2220

1) Mô tả bài toán:

Cho 2 số nguyên a, b. Nhiệm vụ của bạn là đếm số lượng số nguyên giữa a và b mà trong đó không có hai chữ số liền kề nào giống nhau.

Input: Dòng đầu vào duy nhất có hai số nguyên a và b ($0 \le a \le b \le 10^{18}$).

Output: In một số nguyên duy nhất thỏa yêu cầu bài toán.

Input	Input
123 321	171

2) Thuật toán:

Xây dựng các số nhị phân từ trái sang phải (trái = chữ số to nhất). Cần duy trì các thông tin:

- Số hiện tại có < N không
- Chữ số 0 đã có nghĩa hay chưa
- Chữ số cuối cùng của số đó là gì

f(i, positive, lower, last): Số cách xây dựng từ 1 → i, lower = true nếu số hiện tại đã < N, positive = true nếu đã xuất hiện 1 số lớn hơn 0 trước đó và số 0 này là số 0 có nghĩa, last: chữ số cuối cùng đã sử dụng để điền vào ô thứ i

Lưu ý ở đây là chữ số đã điền vào ô thứ i sẽ phải khác với chữ số phải điền vào ô thứ i + 1

3) Code tham khảo:

Code tham khảo có thể chấm bài trên https://lqdoj.edu.vn/problem/cses2220

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

const int numDigit = 20;

int dp[numDigit+55][2][2][11]; /// dp[i][postive][lower]
int digit[numDigit+55];

void add(int &a, const int &b) {
```

```
    a += b;
}

int cal(int x)
{
    if (x < 0) return 0;
    memset(digit, 0, sizeof(digit));
    int i = 1;
    while (x > 0)
    {
        digit[i] = x % 10;
        i ++;
        x /= 10;
    }
//    for (int i = 1; i <= numDigit; i ++)
//        cout << digit[i] << " ";
//    cout << endl;
    memset(dp, 0, sizeof(dp));

    dp[numDigit][0][0][0] = 1;
    for (int i = numDigit; i >= 1; i --)
    {
        for (int postive = 0; postive < 2; postive ++)
            for (int lower = 0; lower < 2; lower ++)
                for (int c1 = 0; c1 < 10; c1 ++)
                    for (int c2 = 0; c2 < 10; c2 ++)
                    {
                        if (!lower && c2 > digit[i - 1])
continue;
                        int nwPos = (postive || c2 > 0);
                        int nwLow = (lower || c2 < digit[i-
1]);
                        if (c1 == c2 && postive) continue;
                        add(dp[i - 1][nwPos][nwLow][c2],
dp[i][postive][lower][c1]);
                    }
    }
    int best = 0;
    for (int postive = 0; postive < 2; postive ++)
        for (int lower = 0; lower < 2; lower ++)
            for (int c = 0; c <= 9; c ++)
                add(best, dp[1][postive][lower][c]);
    return best;
}

signed main()
{
ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    freopen("counting.inp", "r", stdin);
    freopen("counting.out", "w", stdout);
    int l,r;
    cin >> l >> r;
```

```
    cout << cal(r) - cal(l - 1);

    return 0;
}
```

4) Cảm nhận

Đây là một bài quy hoạch động chữ số cơ bản. Ta chỉ cần quan tâm thêm hai điều kiện là dãy đã xét số tận cùng đã điền là gì và số phía trước đã điền là gì.

Bài 9. Đếm số

(https://oj.vnoi.info/problem/demso)

1) Mô tả bài toán:

Với một số tự nhiên được viết trong hệ cơ số 10, ta định nghĩa vị trí xấu là vị trí mà chữ số tại đó với chữ số kề sau nó có độ chênh lệch không quá K. Nếu một số có không quá D vị trí xấu thì đó là số đẹp.

Hãy đếm số lượng số đẹp trong đoạn từ A đến B.

Input:

Gồm một dòng duy nhất là 4 số nguyên A, B, D, K ($1 \leq A \leq B \leq 10^{15}$).

Output:

Gồm một dòng duy nhất là số lượng số đếm được.

Input	Output
1 13 1 0	10

2) Thuật toán:

Nhận xét:

Để đếm số lượng số đẹp trong khoảng từ A → B, ta đếm số lượng số đẹp trong khoảng từ 1 đến B – số lượng số đẹp trong khoảng từ 1 → A – 1.

Để đếm số lượng số đẹp trong khoảng từ 1 đến X, ta làm như sau:

Đặt dp[i][cnt][positive][lower][c] : xét từ hàng numDigit đến vị trí i, có bao nhiêu vị trí xấu, positive: số đã xây dựng đã dương chưa, lower số đã xây dựng đã bé hơn số X chưa và số điền ở vị trí i là c.

Với dp[i][cnt][postive][lower][last] ta sẽ dùng nó đi tối ưu hóa cho dp[i-1][nwCnt][nwPos][nwLow][c] với :

- nwCnt = cnt + (postive && abs(c – last) ≤ d)
- nwPos = (postive) || (c > 0)
- nwLow = (lower) || (c < digit[i-1])

Lưu ý: Trường hợp dãy xây dựng vẫn chưa bé hơn X nhưng số điền vào mà lớn hơn X sẽ không hợp lệ vì dãy đã xây dựng sẽ lớn hơn X.

Kết quả sẽ là tổng của các dp[1][k][1][lower][c] với (0 ≤ k ≤ K) (0 ≤ lower ≤ 1) (0 ≤ c ≤ 9)

3) Code tham khảo:

Code tham khảo có thể chấm bài trên: https://oj.vnoi.info/problem/demso

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

const int numDigit = 16;

int digit[numDigit + 55];
int dp[numDigit + 55][numDigit + 55][2][2][10];
/// dp[i][cnt][postive][lower][c]

int cal(int x, int d, int maxk)
{
    memset(digit, 0, sizeof(digit));
    for (int i = 1; i <= numDigit; i ++)
    {
        digit[i] = x % 10;
        x /= 10;
    }

//    for (int i = 1; i <= numDigit; i ++) cout << digit[i] << " ";

    memset(dp, 0, sizeof(dp));
    dp[numDigit + 1][0][0][0][0] = 1;

    for (int i = numDigit + 1; i >= 2; i --)
    {
        for (int cnt = 0; cnt <= maxk; cnt ++)
            for (int postive = 0; postive < 2; postive ++)
```

```
                for (int lower = 0; lower < 2; lower ++)
                    for (int last = 0; last < 10; last ++) if
(dp[i][cnt][postive][lower][last])
                        for (int c = 0; c < 10; c ++)
                        {
                            if (!lower && c > digit[i-1])
continue;
                            int nwPos = (postive) || (c > 0);
                            int nwLow = (lower) || (c <
digit[i-1]);
                            dp[i-1][cnt + (postive && abs(c -
last) <= d)][nwPos][nwLow][c] +=
dp[i][cnt][postive][lower][last];
                        }
    }

    int res = 0;
    for (int k = 0; k <= maxk; k ++)
    {
        for (int lower = 0; lower < 2; lower ++) for (int c =
0; c < 10; c ++)
            res += dp[1][k][1][lower][c];
    }
    return res;

}

signed main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

    freopen("demso.inp", "r", stdin);
    freopen("demso.out", "w", stdout);

      int l, r, d, k;
    cin >> l >> r >> d >> k;
    cout << cal(r, d, k) - cal(l - 1, d, k);

    return 0;
}
```

4) Cảm nhận

Để giải được bài toán này thì cũng khá đơn giản, ta chỉ cần quan tâm thêm một điều kiện là chữ số tận cùng của dãy đã điền hiện tại đang là số gì? Từ đó ta có thể quản lý được số lượng vị trí xấu có trong số mà ta đã điền.

Bài 10. 23adiv2

(https://lqdoj.edu.vn/problem/23adiv2)

Thời gian chạy: 2 giây

1) Mô tả bài toán:

Hàm F(a) với a là một số nguyên dương được định nghĩa đệ quy như sau:

- Nếu a < 10, F(a) = a
- Nếu a > 9, F(a) = F (tổng các chữ số của a)

Ví dụ: F(91) = F(10) = 1

An cần đếm số lượng số x trong đoạn [l, r] mà F(x) = 9

Input:

+ Dòng đầu tiên chứa 1 số nguyên dương Q là lượng câu hỏi.
+ Q dòng tiếp theo chứa, mỗi dòng chứa 1 cặp số nguyên dương l, r biểu thị một đoạn.

Output:

Q dòng, mỗi dòng là số lượng số x tương ứng.

Input	Output
2 1 2 1 9	0 1

2) Thuật toán:

Nhận xét:

Để hàm F(x) = 9 thì ta có tổng các chữ số của x sẽ là bội của 9. Từ đó suy ra bài toán trở thành đếm số lượng số có tổng các chữ số là bội của 9 trong đoạn từ l đến r.

⇔ Số lượng các số có tổng các chữ số là bội của 9 trong đoạn từ 1 -> r - số lượng các số có tổng các chữ số là bội của 9 trong đoạn từ 1 -> l – 1

Để số lượng các số có tổng các chữ số là bội của 9 trong đoạn từ 1 -> n, ta làm như sau:

Xây dựng các số nhị phân từ trái sang phải (trái = chữ số to nhất). Cần duy trì các thông tin:

- Số hiện tại có < N không
- Chữ số 0 đã có nghĩa hay chưa
- Phần dư của tổng các chữ số đó cho 9

f(i, cnt, positive, lower): Số cách xây dựng từ 1 → i, lower = true nếu số hiện tại đã < N, positive = true nếu đã xuất hiện 1 số lớn hơn 0 trước đó và số 0 này là số 0 có nghĩa.

Kết quả: f(1, 0, 1, 0) + f(1, 0, 1, 1)

3) Code tham khảo:

Code tham khảo có thể chấm bài trên: https://lqdoj.edu.vn/problem/23adiv2

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

const int numDigit = 10;

int digit[numDigit + 55];
int dp[numDigit + 55][9][2][2];

int cal(int x)
{
    memset(digit, 0, sizeof(digit));
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= numDigit; i ++) {
        digit[i] = x % 10;
        x /= 10;
    }
    dp[numDigit + 1][0][0][0] = 1;
    for (int i = numDigit + 1; i >= 2; i --)
    {
        for (int sum = 0; sum < 9; sum ++)
            for (int postive = 0; postive < 2; postive ++)
                for (int lower = 0; lower < 2; lower ++) if
(dp[i][sum][postive][lower])
                    for (int c = 0; c < 10; c ++)
                    {
                        if (!lower && c > digit[i-1])
continue;
                        int nwLower = (lower) || (c < digit[i-
1]);
                        int nwPos = (postive) || (c > 0);
                        dp[i-1][(sum + c) % 9][nwPos][nwLower]
+= dp[i][sum][postive][lower];
                    }
```

```
    }

    return dp[1][0][1][0] + dp[1][0][1][1];


}

signed main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    freopen("23adiv2.inp", "r", stdin);
    freopen("23adiv2.out", "w", stdout);
    int test;
    cin >> test;
    while (test --)
    {
        int l, r;
        cin >> l >> r;
        cout << cal(r) - cal(l - 1) << endl;
    }
    return 0;
}
```

4) Cảm nhận:

Đây là một bài khá đặc biệt, để có thể giải được bài toán ta cần có thêm một nhận xét là F(x) = 9 suy ra tổng các chữ số của nó phải là bội của 9. Từ nhận xét này ta có thể thực hiện QHĐ chữ số để tìm ra được đáp án của bài toán.

III. Kết luận

Chuyên đề này giúp học sinh áp dụng được kiến thức quy hoạch động chữ số giải các bài tập lập trình giản và dần nâng cao hơn. Tuy nhiên, một bài toán thường có rất nhiều cách giải quyết khác nhau. Tác giả rất mong thầy cô góp ý, chia sẻ những kiến thức có liên quan đến chuyên đề này để tác giả có thể nghiên cứu hoàn chỉnh, mở rộng chuyên đề ở tầm vĩ mô.

Tác giả xin chân thành cảm ơn!

IV. Nguồn tham khảo

Tài liệu giáo khóa chuyên tin: Quyển 1. Quyển 2, Quyển 3.

Bài tập trên các trang: lqdoj, oj.vnoi, codeforces.