CHUYÊN ĐỀ ỨNG DỤNG CỦA PHÉP NHÂN MA TRẬN TRONG GIẢI CÁC BÀI TOÁN TIN HỌC

Nguyễn Thị Hồng Nhớ- THPT Biên Hòa – Hà Nam

PHẦN I: MỞ ĐẦU

Trong Tin học, khi lập trình một bài toán thì vấn đề đặt ra cho mỗi người lập trình không phải chỉ là cho ra đáp số đúng, mà vấn đề quan trọng hơn là phải đưa ra đáp số đúng trong thời gian ngắn nhất. Thông thường, để đạt được độ phức tạp thuật toán như mong muốn, cách làm thường là tìm ra một thuật toán ban đầu làm cơ sở, rồi từ đó dùng các kỹ năng để giảm độ phức tạp của thuật toán. Đối với nhiều bài toán có công thức truy hồi nhưng với kích thước dữ liệu rất lớn, chẳng hạn 10^{18} ta không thể làm thuật toán quy hoạch động thông thường mà cần phải kết hợp thêm một kĩ thuật nữa. Một kĩ thuật khá thông dụng, hay được sử dụng trong các kì thi học sinh giỏi quốc gia, quốc tế là ***nhân ma trận***.

Hiện nay, kĩ thuật nhân ma trận áp dụng nhiều trong các kì thi học sinh giỏi môn tin học, nhưng tài liệu viết một cách chi tiết, hệ thống về kĩ thuật này thì chưa có. Điều này làm cho việc nghiên cứu, giảng dạy và học kĩ thuật này khá khó khăn. Với kinh nghiệm bồi dưỡng học sinh giỏi tỉnh, quốc gia tôi thấy các bài toán quy hoạch động thông thường thì học sinh có thể làm được, nhưng những bài toán quy hoạch động có dữ liệu rất lớn thì học sinh thường ít khi được 100% số điểm, đó là một điều rất đáng tiếc. Vì vậy tôi đã quyết định chọn và viết chuyên đề: *"Ứng dụng của phép nhân ma trận trong giải các bài toán Tin học*".

Trong chuyên đề tôi sẽ trình bày các kiến thức về ma trận, phép nhân ma trận và ứng dụng kĩ thuật nhân ma trận trong các bài toán Tin học. Hi vọng đây là một tài liệu thiết thực đối với các đồng nghiệp và các em học sinh.

PHẦN II: NỘI DUNG

I. MA TRẬN (Matrix), NHÂN MA TRẬN (Matrix multiplication)

1. Khái niệm ma trận

1.1. Khái niệm ma trận

Trong toán học, ma trận là một mảng chữ nhật gồm các số, ký hiệu, hoặc biểu thức, sắp xếp theo hàng và cột mà mỗi ma trận tuân theo những quy tắc định trước. Các ô trong ma trận được gọi là các phần tử. Ví dụ dưới đây là một ma trận có m hàng và n cột:

$$\begin{bmatrix} a_{11} & a_{12} & . & . & a_{1n} \\ a_{21} & a_{22} & . & . & a_{2n} \\ . & . & . & . & . \\ . & . & . & . & . \\ a_{m1} & a_{m2} & . & . & a_{mn} \end{bmatrix}$$

Khi định nghĩa một ma trận ta cần chỉ rõ số hàng m và số cột n của nó. Lúc này, mxn được gọi là cấp của ma trận.

Các phần tử trong ma trận được định danh bằng 2 địa chỉ hàng i và cột j tương ứng. Ví dụ phần tử hàng thứ i, cột thứ j sẽ được kí hiệu là: A_{ij}. Véc tơ có thể coi là trường hợp đặt biệt của ma trận với số hàng hoặc số cột là 1.

Ma trận vuông: Ma trận vuông là ma trận có số hàng bằng với số cột. Ví dụ một ma trận vuông cấp 3 (số hàng và số cột là 3) có dạng như sau:

$$\begin{bmatrix} 2 & 4 & 6 \\ 2 & 3 & 1 \\ 0 & 4 & 9 \end{bmatrix}$$

1.2. Khai báo ma trận trong ngôn ngữ lập trình C++

Sử dụng khai báo ma trận kiểu cấu trúc, giả sử khai báo cấu trúc ma trận có mảng dữ liệu **val** kích thước gồm 4 dòng, 4 cột; khởi tạo giá trị ban đầu cho các phần tử của ma trận đều bằng 0:

```
struct matrix  {
   long long  val[4][4];  //khai bao mang hai chieu chua du lieu cua ma tran
   matrix() //Ham khoi tao cho ma tran
      {
      memset(val,0,sizeof(val));    }
};
```

2. Phép nhân ma trận

2.1. Các khái niệm về phép nhân ma trận

Cho 2 ma trận: ma trận A kích thước M∗N và ma trận B kích thước N∗P (chú ý số cột của ma trận A phải bằng số hàng của ma trận B thì mới có thể thực hiện phép nhân).

Kết quả phép nhân ma trận A và B là ma trận C kích thước M∗P, với mỗi phần tử của ma trận C được tính theo công thức:

$$C_{ik} = \sum_{j=1}^{n} A_{ij} * B_{jk} \qquad \text{với i=1..m, k=1..p}$$

Hay viết $C_{ik}=[a_{i1}\ a_{i2}\ \dots\ a_{in}]\begin{bmatrix} b_{1k} \\ b_{2k} \\ . \\ . \\ b_{nk} \end{bmatrix}$ tức là phần tử ở dòng thứ i, cột thứ k của C là tích vô hướng của vectơ dòng thứ i của A với vectơ cột thứ k của B.

Ví dụ: Cho hai ma trận

$$A=\begin{bmatrix} -1 & 2 & 3 \\ 2 & 0 & -4 \end{bmatrix} \text{ và } B=\begin{bmatrix} 1 & 2 & 1 & -1 \\ 0 & 3 & 2 & 0 \\ -1 & 4 & 2 & 5 \end{bmatrix}$$

Phần tử C_{23} của ma trận tích A*B là tích vô hướng của vectơ dòng thứ hai của ma trận A và vectơ cột thứ 3 của ma trận B, ta có:

$$C_{23}= \begin{bmatrix} 2 & 0 & -4 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = -6$$

Ma trận tích A*B có dạng:

$$A*B= \begin{bmatrix} -1 & 2 & 3 \\ 2 & 0 & -4 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 & -1 \\ 0 & 3 & 2 & 0 \\ -1 & 4 & 2 & 5 \end{bmatrix} = \begin{bmatrix} -4 & 16 & 9 & 16 \\ 6 & -12 & -6 & -22 \end{bmatrix}$$

Chú ý: Với ma trận vuông A cấp p (gồm p dòng, p cột) ta có thể thực hiện phép lũy thừa với số mũ k nguyên dương bất kì, kí hiệu là: A^k; kết quả cũng là một ma trận vuông cấp p.

$$A^k = A*A*A*\dots*A \text{ (k lần ma trận A)}$$

2.2. Tính chất của phép nhân ma trận

- Phép nhân ma trận không có tính chất giao hoán: có thể A*B nhưng B không thể nhân được với A, nếu nhân được thì thường có A*B≠B*A.
- Tính chất kết hợp: (A*B)*C=A*(B*C)
- Tính chất phân phối: A*(B+D)=A*B+A*D, (B + C)D = BD + CD.

2.3. Cài đặt phép nhân ma trận trong ngôn ngữ lập trình C++

*** Phép nhân 2 ma trận:**

Cho hai ma trận: ma trận A kích thước m*n và ma trận B kích thước n*p. Phép nhân hai ma trận A*B được viết trong ngôn ngữ lập trình C++ như sau:

```
matrix a, b;
long long m,n,p;
matrix nhan(matrix a, matrix b)
{
   matrix c; //ma tran ket qua c=a*b
   for (int i = 1; i <= m; i++)
      for (int j = 1; j <= p; j++)
   {
      c.val[i][j] = 0; //thua vi da su dung ham memset
      for (int k = 1; k <= n; k++)
         c.val[i][j] = c.val[i][j] + a.val[i][k] * b.val[k][j];
   }
   return c;
}
```

Nhận xét: Độ phức tạp O(m*n*p).

Chú ý: Đối với phép nhân các ma trận vuông kích thước N∗N, có thuật toán nhân ma trận Strassen với độ phức tạp $O(N^{log7})$ theo tư tưởng chia nhỏ ma trận (tương tự cách nhân nhanh 2 số lớn). Tuy nhiên cài đặt rất phức tạp và trên thực tế với giá trị N thường gặp, cách này không chạy nhanh hơn nhân ma trận thông thường $O(N^3)$.

*** Tính lũy thừa của ma trận: A^k**

Cho ma trận vuông A kích thước n*n. Khi đó ta có phép tính ma trận A lũy thừa k (kí hiệu: A^k), với k là số nguyên dương bất kì. Cài đặt phép toán này ta có thể sử dụng 2 cách như sau:

Cách 1: Cài đặt bằng đệ quy (cách dễ cài đặt nhất)

Ta có A^k=
$$\begin{cases} A & if\ k = 1 \\ A^{k/2} * A^{k/2} & if\ k \bmod 2 = 0 \\ A^{k/2} * A^{k/2} * A & if\ k \bmod 2 = 1 \end{cases}$$

Code tính A^k:

```
matrix mu(matrix a, long long k)
{
   matrix x;
   if ( k == 1) return a;
   x = mu (a, k/2);
   x = nhan (x, x);
   if (k % 2) x = nhan (x, a);
   return x;
}
```

Cách 2:Cài đặt bằng vòng lặp

```
matrix mu(matrix a, long long k)
{
   matrix x;//x là ma trận đơn vị
   for (; k/=2; a=nhan(a,a))
        if (k&1) x=nhan(x,a); //nếu k lẻ
   return x;
}
```

Hoặc

```
matrix mu(matrix a, long long k)
{
   matrix x;//x là ma trận đơn vị
  while (k>0)
        { if (k&1) x=nhan(x,a); //nếu k lẻ
        k/=2;
        a=nhan(a,a);
```

```
        }
    return x;
}
```

Nhận xét: Thuật toán tính A^k có độ phức tạp $O(n^3 \log k)$, với n là kích thước của ma trận vuông A.

II. BÀI TẬP ỨNG DỤNG.

1. *Bài 1: Bài toán tính số Fibo thứ n.*

1.1. Đề bài

Dãy Fibonacci được định nghĩa như sau:

F(0) = 1

F(1) = 1

.....

F(i) = F(i-1) + F(i-2), i >= 2

Yêu cầu: Cho số nguyên dương n ($n \leq 10^{18}$), tính F(n) mod 10^9+7.

Input: fibo.inp

Một dòng duy nhất ghi số N.

Output: fibo.out

Ghi ra kết quả F(n) mod 10^9+7.

fibo.inp	fibo.out
4	5

❖ **Xác định bài toán:**

Input: số n nguyên dương, $n \leq 10^{18}$

Output: F(n) mod 10^9+7

1.2. Hướng dẫn giải thuật.

Đây là một bài toán quy hoạch động rất quen thuộc. Nếu làm theo cách bình thường chúng ta sử dụng vòng FOR với biến chạy từ 2 tới n để tính lần lượt các F(j) thì độ phức tạp là O(n). Nhưng nhận thấy dữ liệu đề bài cho $n \leq 10^{18}$, nên ta không thể sử dụng phương pháp đơn giản trên, mà phải sử dụng kĩ thuật khác. Một trong các kĩ thuật đó là: Nhân ma trận.

Ta có thể viết:

$F_i = 1.F_{i-1} + 1.F_{i-2}$

$F_{i-1}=1.F_{i-1}+0.F_{i-2}$

Ta đặt ma trận $A=\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}$

Từ hai biểu thức trên ta sẽ có: $\begin{bmatrix}F_i\\ F_{i-1}\end{bmatrix}=\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}*\begin{bmatrix}F_{i-1}\\ F_{i-2}\end{bmatrix}$ (lưu ý: ma trận bên trái thường cao hơn ma trận bên phải một bậc).

➡ $\begin{bmatrix}F_i\\ F_{i-1}\end{bmatrix}=A*\begin{bmatrix}F_{i-1}\\ F_{i-2}\end{bmatrix}=A^2*\begin{bmatrix}F_{i-2}\\ F_{i-3}\end{bmatrix}$ (vì $\begin{bmatrix}F_{i-1}\\ F_{i-2}\end{bmatrix}=A*\begin{bmatrix}F_{i-2}\\ F_{i-3}\end{bmatrix}$)

$=A^3*\begin{bmatrix}F_{i-3}\\ F_{i-4}\end{bmatrix}=.....=A^{i-1}*\begin{bmatrix}F_1\\ F_0\end{bmatrix}$

➡ $\begin{bmatrix}F_n\\ F_{n-1}\end{bmatrix}=A^{n-1}*\begin{bmatrix}F_1\\ F_0\end{bmatrix}=A^{n-1}*\begin{bmatrix}1\\ 1\end{bmatrix}$ bởi vì **F(0) = 1; F(1) = 1**

Đặt ma trận $res=A^{n-1}$

Kết quả cần tìm là: **F_n=res.val[1,1]*1+res.val[1,2]*1** (do thực hiện phép nhân ma trận $res*\begin{bmatrix}1\\ 1\end{bmatrix}$). Chú ý kết quả sau đó có mod 10^9+7.

1.3. Chương trình.

```
#include <bits/stdc++.h>
#define nmax 1005
#define MOD 1000000007
using namespace std;
struct matrix {int val [3][3];};
matrix a, c; int t; long long n;
matrix nhan(matrix a, matrix b)
{
   matrix c;
   for (int i = 1; i <= 2; i++)
      for (int j = 1; j <= 2; j++)
   {
      c.val[i][j] = 0;
      for (int k = 1; k <= 2; k++)
         c.val[i][j] = ((c.val[i][j] + (long long)a.val[i][k] * b.val[k][j] ) %
MOD)%MOD;
   }
   return c;
}
matrix mu (matrix a, long long k)
{
   matrix x;
```

```
    if ( k == 1) return a;
    x = mu (a, k/2);
    x = nhan (x, x);
    if (k % 2) x = nhan (x, a);
    return x;
}
int main()
{
    freopen("fibo.inp", "r", stdin);
    freopen("fibo.out", "w", stdout);
    cin >> n;
    a.val[1][1] = 1;
    a.val[1][2] = 1;
    a.val[2][1] = 1;
    a.val[2][2] = 0;
    c = mu(a, n-1); cout<<(c.val[1][1] + c.val[1][2])% MOD;
    return 0;
}
```

1.4. Độ phức tạp:

Chủ yếu thời gian của chương trình phụ thuộc vào thời gian tính c=mu(a, n-1), nên độ phức tạp là:

$$O(2^3 \log(n-1)) = O(\log(n-1))$$

Chú ý: trong khi nhân 2 ma trận, kích thước ma trận A chỉ có 2 dòng, 2 cột nên chỉ số chỉ chạy tới 2. Trong hàm nhân ma trận có câu lệnh:

*c.val[i][j] = ((c.val[i][j] + (long long)a.val[i][k] * b.val[k][j]) % MOD)%MOD;*

Do *a.val[i][k] * b.val[k][j]*kết quả có thể rất lớn nên ta phải ép kiểu nó thành *long long* không sợ tràn số, và tiến hành *%MOD* luôn tích này cho tổng cuối cùng giá trị nó nhỏ đi.

1.5. Test

https://drive.google.com/open?id=1L1EtM07Ux8_Fjr-MilRDFkrHauWThsto

2. Bài 2: Bài toán Lát Gạch TILE.

2.1. Đề bài

Cho một hình chữ nhật kích thước 2xN ($1<=N<10^9$). Hãy đếm số cách lát các viên gạch nhỏ kích thước 1x2 và 2x2 vào hình trên sao cho không có phần nào của các viên gạch nhỏ thừa ra ngoài, cũng không có vùng diện tích nào của hình chữ nhật không được lát.

Input: Tile.inp

Gồm nhiều test, dòng đầu ghi số lượng test T (T<=10). T dòng sau mỗi dòng ghi một số N.

Output: Tile.out

Ghi ra T dòng là số cách lát tương ứng lấy phần dư cho 10^9+7.

Example:

Tile.inp	Tile.out
3 2 8 12	3 171 2731

❖ **Xác định bài toán:**

Input: số n nguyên dương, $n \leq 10^{18}$

Output: gồm T dòng, mỗi dòng là số cách lát tương ứng lấy phần dư cho 10^9+7.

2.2. Hướng dẫn giải thuật.

Đây là một bài toán quy hoạch động.

Gọi F[i] là số cách lát hình chữ nhật kích thước 2*i.

Ta có F[0]=1; F[1]=1;

Công thức truy hồi: F[i]=F[i-1]+2*F[i-2]

Cách khác: Sử dụng toán học để đưa ra công thức truy hồi để xác định số hạng tổng quát $F[n]=\frac{2^{n+1}+(-1)^n}{3}$ Đến đây thì dùng kĩ thuật tính nhanh lũy thừa để tính.

Đây là một dạng tương tự bài toán Fibo ở trên. Nếu làm theo cách bình thường chúng ta sử dụng vòng FOR với biến chạy từ 2 tới n để tính lần lượt các F(i) thì độ phức tạp là O(n). Nhưng nhận thấy dữ liệu đề bài cho $n \leq 10^{18}$, nên ta phải sử dụng Nhân ma trận.

Ta có thể viết:

$F_i = 1.F_{i-1}+2.F_{i-2}$

$F_{i-1}=1.F_{i-1}+0.F_{i-2}$

Ta đặt ma trận $A=\begin{bmatrix}1 & 2\\ 1 & 0\end{bmatrix}$

Từ hai biểu thức trên ta sẽ có: $\begin{bmatrix}F_i\\ F_{i-1}\end{bmatrix}=\begin{bmatrix}1 & 2\\ 1 & 0\end{bmatrix}*\begin{bmatrix}F_{i-1}\\ F_{i-2}\end{bmatrix}$

➡ $\begin{bmatrix} F_i \\ F_{i-1} \end{bmatrix}$ = A*$\begin{bmatrix} F_{i-1} \\ F_{i-2} \end{bmatrix}$=$A^2$*$\begin{bmatrix} F_{i-2} \\ F_{i-3} \end{bmatrix}$ (vì $\begin{bmatrix} F_{i-1} \\ F_{i-2} \end{bmatrix}$=A*$\begin{bmatrix} F_{i-2} \\ F_{i-3} \end{bmatrix}$)

= A^3*$\begin{bmatrix} F_{i-3} \\ F_{i-4} \end{bmatrix}$ =.....= A^{i-1}*$\begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$

➡$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix}$= A^{n-1}*$\begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$= A^{n-1}*$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ bởi vì **F(0) = 1; F(1) = 1.**

Đặt ma trận c= A^{n-1}

Kết quả cần tìm là: **F_n=c.val[1,1]*1+c.val[1,2]*1** (do thực hiện phép nhân ma trận res*$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$). Chú ý kết quả sau đó có mod 10^9+7.

2.3. Chương trình.

```
#include <bits/stdc++.h>
#define nmax 1005
#define MOD 1000000007
using namespace std;
struct matrix {long long val [3][3];};
matrix a, c;
int t;
long long n;
matrix nhan(matrix a, matrix b)
{
   matrix c;
   for (int i = 1; i <= 2; i++)
      for (int j = 1; j <= 2; j++)
   {
      c.val[i][j] = 0;
      for (int k = 1; k <= 2; k++)
         c.val[i][j] = ((c.val[i][j] + (long long)a.val[i][k] * b.val[k][j] ) %
MOD)%MOD;
   }
   return c;
}
matrix mu (matrix a, long long k)
{
   matrix x;
   if ( k == 1) return a;
   x = mu (a, k/2);
   x = nhan (x, x);
   if (k % 2) x = nhan (x, a);
   return x;
}
int main()
```

```
{
   freopen("tile.inp", "r", stdin);
   freopen ("tile.out", "w", stdout);
   cin >> t;
   for (int i = 1; i <= t; i++)  {
   cin >> n;
   a.val[1][1] = 1;
   a.val[1][2] = 2;
   a.val[2][1] = 1;
   a.val[2][2] = 0;
   if (n>1)
   {c = mu(a, n - 1);
   cout << (c.val[1][1] + c.val[1][2])% MOD;}
   if (n==1) or (n==0) cout<<1;
   cout << endl;
   }
   return 0;
}
```

2.4. Độ phức tạp:

Chủ yếu thời gian của chương trình phụ thuộc vào thời gian tính c=mu(a,n-1), nên độ phức tạp là: $O(2^3 \log(n-1))$

Chú ý: Bài toán tương tự trên SPOJ: Lát Gạch 4

Nguồn: http://vnoi.info/problems/show/LATGACH4/

2.5.Test

https://drive.google.com/file/d/13I35abqOtAVETi3pqP3b-w4Q-01sjyFB/view?usp=sharing

3. Bài 3: Bài toán Tổng FIBO.

3.1. Đề bài

Xét dãy số Fibonacci {***Fn***} theo định nghĩa:

$$F0=F1=1$$

$$F_n=F_{n-1}+F_{n-2}\ \forall n>1$$

Cho số ***n***, hãy tính tổng $S=F0+F1+F2+\cdots+Fn$ và đưa ra số dư của ***S*** chia cho (10^9+7).

Dữ liệu: vào từ file văn bản FIBOS.INP gồm một dòng duy nhất ghi số nguyên dương ***n*** ($n \le 10^{15}$).

Kết quả: ghi ra file văn bản FIBOS.OUT một số nguyên – số dư tìm được.

Ví dụ:

FIBOS.INP	FIBOS.OUT

3	**7**
5	**20**

❖ **Xác định bài toán:**

Input: số n nguyên dương, $n \leq 10^{15}$

Output: số dư của ***S*** chia cho (10^9+7).

3.2. Hướng dẫn giải thuật.

CÁCH 1: Từ công thức $S=F0+F1+F2+\cdots+Fn$

Suy ra: $S=F_{n+2}-1$ (có thể chứng minh bằng quy nạp) ? Công thức này dễ dàng đưa ra nhờ dự đoán.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
F	1	1	2	3	5	8	13	21	34	55	89	144	233	377
T	1	2	4	7	12	20	33	54	88	143	232	376	…	…

Bài toán quy về việc tính F[n+2]. Đây là một bài toán quy hoạch động, có dữ liệu lớn, tương tự bài toán Fibo và ta vẫn phải dùng tới Nhân ma trận.

Ta có thể viết:

$F_i = 1.F_{i-1}+1.F_{i-2}$

$F_{i-1}=1.F_{i-1}+0.F_{i-2}$

Ta đặt ma trận $A=\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}$

Từ hai biểu thức trên ta sẽ có: $\begin{bmatrix}F_i\\ F_{i-1}\end{bmatrix}=\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}*\begin{bmatrix}F_{i-1}\\ F_{i-2}\end{bmatrix}$

➡ $\begin{bmatrix}F_i\\ F_{i-1}\end{bmatrix} = A*\begin{bmatrix}F_{i-1}\\ F_{i-2}\end{bmatrix}=A^2*\begin{bmatrix}F_{i-2}\\ F_{i-3}\end{bmatrix}$ (vì $\begin{bmatrix}F_{i-1}\\ F_{i-2}\end{bmatrix}=A*\begin{bmatrix}F_{i-2}\\ F_{i-3}\end{bmatrix}$)

$= A^3*\begin{bmatrix}F_{i-3}\\ F_{i-4}\end{bmatrix} =\text{.....}= A^{i-1}*\begin{bmatrix}F_1\\ F_0\end{bmatrix}$

➡ $\begin{bmatrix}F_{n+2}\\ F_{n+1}\end{bmatrix}= A^{n+1}*\begin{bmatrix}F_1\\ F_0\end{bmatrix}= A^{n+1}*\begin{bmatrix}1\\ 1\end{bmatrix}$ bởi vì **F(0) = 1; F(1) = 1**

Đặt ma trận $res= A^{n+1}$

Kết quả cần tìm là: **F_n=res.val[1,1]*1+res.val[1,2]*1-1** (do thực hiện phép nhân ma trận res*$\begin{bmatrix}1\\1\end{bmatrix}$). Chú ý kết quả sau đó có mod 10^9+7.

❖ **Cài đặt chương trình theo cách 1:**

Tương tự phần cài đặt của bài toán Fibo ở trên. Chỉ khác ở chỗ tính ra kết quả cuối cùng:

```
matrix res;
c = mu(a, n+1);
cout<<(res.val[1][1] + res.val[1][2]-1)% MOD;
```

<u>CÁCH 2:</u>

Đặt T_n= $F0+F1+F2+\cdots+Fn$

T[0]=F[0]=1; T[1]=F[0]+F[1]=2; T[2]= F[0]+F[1]+F[2]=4

Ta thấy $T_{n-1}=T_{n-3}+F_{n-2}+F_{n-1}=T_{n-3}+F_n \Rightarrow F_n=T_{n-1}+T_{n-3}$

$$\Rightarrow T_n=T_{n-1}+F_n=2*T_{n-1}-T_{n-3}$$

Bài toán quay về tính số dư của T_n cho (10^9+7). Vì đây là bài toán quy hoạch động với kích thước n<10^{15}, nên ta sử dụng nhân ma trận như sau:

Từ T_n=2*T_{n-1} – T_{n-3} ta viết:

T_n=2*T_{n-1} +0*T_{n-2}– 1*T_{n-3}

T_{n-1}=1*T_{n-1} +0*T_{n-2}+ 0*T_{n-3}

T_{n-2}=0*T_{n-1} +1*T_{n-2}+0*T_{n-3}

Ta đặt ma trận A=$\begin{bmatrix}2 & 0 & -1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$

Từ các biểu thức trên ta sẽ có: $\begin{bmatrix}T_n\\T_{n-1}\\T_{n-2}\end{bmatrix}=\begin{bmatrix}2 & 0 & -1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}*\begin{bmatrix}T_{n-1}\\T_{n-2}\\T_{n-3}\end{bmatrix}$

➡ $\begin{bmatrix}T_n\\T_{n-1}\\T_{n-2}\end{bmatrix}$ = A*$\begin{bmatrix}T_{n-1}\\T_{n-2}\\T_{n-3}\end{bmatrix}$=$A^2$*$\begin{bmatrix}T_{n-2}\\T_{n-3}\\T_{n-4}\end{bmatrix}$ (vì $\begin{bmatrix}T_{n-1}\\T_{n-2}\\T_{n-3}\end{bmatrix}$ =A*$\begin{bmatrix}T_{n-2}\\T_{n-3}\\T_{n-4}\end{bmatrix}$)

$= A^3 * \begin{bmatrix} T_{n-3} \\ T_{n-4} \\ T_{n-5} \end{bmatrix} = \ldots\ldots\ldots = A^{n-2} * \begin{bmatrix} T_2 \\ T_1 \\ T_0 \end{bmatrix} = A^{n-2} * \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}$

Đặt ma trận res= A^{n-2}

Kết quả cần tìm là: **T_n=res.val[1,1]*4+res.val[1,2]*2+res.val[1,3]*1** (do thực hiện phép nhân ma trận res* $\begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}$). Chú ý kết quả sau đó có mod 10^9+7.

3.3. Chương trình.

(cài đặt theo cách số 2)

```
#include <bits/stdc++.h>
#define ll long long
#define MOD 1000000007
using namespace std;
struct matrix
{
   long long val [4][4];
   matrix()
   {
      memset(val, 0, sizeof(val));
   }
};
matrix a, c,d;
long long n;
matrix nhan(matrix a, matrix b)
{
   matrix c;
   for (int i = 1; i <= 3; i++)
      for (int j = 1; j <= 3; j++)
   {
      for (int k = 1; k <= 3; k++)
         c.val[i][j] = ((c.val[i][j] + (ll)a.val[i][k] * b.val[k][j] ) %
MOD+MOD)%MOD;
   }
   return c;
}
matrix mu (matrix a, long long k)
{
   matrix x;
   if ( k == 1) return a;
   x = mu (a, k/2);
   x = nhan (x, x);
```

```
    if (k % 2) x = nhan (x, a);
    return x;
}
int main()
{
    freopen("fibos.inp", "r", stdin);
    freopen("fibos.out", "w", stdout);
    cin >> n;
    a.val[1][1] = 2;
    a.val[1][2] = 0;
    a.val[1][3] = -1;
    a.val[2][1] = 1;
    a.val[2][2] = 0;
    a.val[2][3] = 0;
    a.val[3][1] = 0;
    a.val[3][2] = 1;
    a.val[3][3] = 0;
    if (n>2)
    {
    c = mu(a, n-2);
    d.val[1][1]=4;
    d.val[2][1]=2;
    d.val[3][1]=1;
    d=nhan(c,d);
    cout<<( ((c.val[1][1])*4)%MOD + (c.val[1][2]*2)%MOD + c.val[1][3])%
MOD << endl;
 }
    if (n==0) cout<<1;
    if (n==1) cout<<2;
    if (n==2) cout<<4;
    return 0;
}
```

3.4. Độ phức tạp.

Cách 1: Giống bài Fibo, Độ phức tạp với thời gian O(log n).

Cách 2: Chủ yếu thời gian của chương trình phụ thuộc vào thời gian tính c=mu(a,n-2), nên độ phức tạp là: $O(3^3 \log(n-2)) = O(\log (n-2))$.

3.5. Test.

https://drive.google.com/file/d/1FLPVi3OrOWBMQV5t9nG065320hndYY5U/view?usp=sharing

4. Bài 4: Bài toán Trò Chơi Lò Cò

(Nguồn: Duyên Hải 2015, http://vn.spoj.com/problems/DHLOCO/)

4.1. Đề bài

Carnaval Hạ Long 2015 với chủ đề "Hội tụ tinh hoa - Lan tỏa nụ cười", điểm mới của lễ hội là sự song hành giữa biểu diễn nghệ thuật "Nơi tinh hoa hội tụ" và diễu hành đường phố "Nụ cười Hạ Long" với sự góp mặt của hơn 2000 diễn viên quần chúng. Có rất nhiều chương trình vui chơi được tổ chức, một trong những trò chơi thu hút được nhiều du khách tham gia đó là trò chơi nhảy lò cò, cụ thể: người chơi cần vượt qua một đoạn đường dài *n* mét, mỗi bước, người chơi có ba cách nhảy với độ dài bước nhảy tương ứng là 1 mét, 2 mét, 3 mét. Một cách đi chuyển đúng là dãy các bước nhảy có tổng đúng bằng *n*.

Yêu cầu: Cho *n* và *M*, gọi *K* là số cách di chuyển đúng khác nhau để đi hết đoạn đường *n* mét, hãy tính phần dư của *K* chia *M*.

Dữ liệu: Vào từ file văn bản LOCO.INP: gồm một dòng chứa hai số nguyên dương *n*, *M* ($M \leq 2015$);

Kết quả: Đưa ra file văn bản LOCO.OUT một số nguyên là phần dư của *K* chia *M*.

Ví dụ:

LOCO.INP	**LOCO.OUT**
5 100	**13**

Ghi chú:

- Có 20% số test ứng với 20% số điểm có $n \leq 20$;
- Có 40% số test ứng với 40% số điểm có $n \leq 10^6$;
- Có 40% số test còn lại ứng với 40% số điểm có $n \leq 10^{15}$.

❖ **Xác định bài toán:**

Input: Các số nguyên dương *n* và *M* ($n \leq 10^{15}$, $M \leq 2015$)

Output: một số nguyên là phần dư của *K* chia *M*.

4.2. Hướng dẫn giải thuật.

Đây là một bài toán quy hoạch động.

Gọi F[i] là số cách nhảy để vượt qua đoạn đường dài i mét.

Ta có F[0]=1; F[1]=1;F[2]=2.

Công thức truy hồi: F[i]=F[i-1]+F[i-2]+F[i-3]

Kết quả cần tìm là: F[n] %M

Nếu làm theo cách bình thường chúng ta sử dụng vòng FOR với biến chạy từ 3 tới n để tính lần lượt các F(j) thì độ phức tạp là O(n). Với 2 sub đầu có thể dùng quy hoạch động thông thường, nhưng sub 3 với $n \leq 10^{15}$, nên ta phải sử dụng Nhân ma trận.

Ta có thể viết:

$F_i = 1.F_{i-1}+1.F_{i-2}+1.F_{i-3}$

$F_{i-1}=1.F_{i-1}+0.F_{i-2}+0.F_{i-3}$

$F_{i-2}=0.F_{i-1}+1.F_{i-2}+0.F_{i-3}$

Ta đặt ma trận $A=\begin{bmatrix}1 & 1 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$

Từ hai biểu thức trên ta sẽ có: $\begin{bmatrix}F_i\\ F_{i-1}\\ F_{i-2}\end{bmatrix}=\begin{bmatrix}1 & 1 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}*\begin{bmatrix}F_{i-1}\\ F_{i-2}\\ F_{i-3}\end{bmatrix}$

➡ $\begin{bmatrix}F_i\\ F_{i-1}\\ F_{i-2}\end{bmatrix}= A*\begin{bmatrix}F_{i-1}\\ F_{i-2}\\ F_{i-3}\end{bmatrix}=A^2*\begin{bmatrix}F_{i-2}\\ F_{i-3}\\ F_{i-4}\end{bmatrix}$ (vì $\begin{bmatrix}F_{i-1}\\ F_{i-2}\\ F_{i-3}\end{bmatrix}=A*\begin{bmatrix}F_{i-2}\\ F_{i-3}\\ F_{i-4}\end{bmatrix}$)

$= A^3*\begin{bmatrix}F_{i-3}\\ F_{i-4}\\ F_{i-5}\end{bmatrix}$ =.....= $A^{i-2}*\begin{bmatrix}F_2\\ F_1\\ F_0\end{bmatrix}$

➡ $\begin{bmatrix}F_n\\ F_{n-1}\\ F_{n-2}\end{bmatrix}= A^{n-2}*\begin{bmatrix}F_2\\ F_1\\ F_0\end{bmatrix}= A^{n-2}*\begin{bmatrix}2\\ 1\\ 1\end{bmatrix}$

Đặt ma trận $c= A^{n-2}$

Kết quả cần tìm là: **F_n=c.val[1,1]*2+c.val[1,2]*1** + **c.val[1,3]*1**. Chú ý kết quả sau đó có mod M.

4.3. Chương trình.

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
struct matrix
{
    long long val [4][4];
    matrix()
    {
        memset(val, 0, sizeof(val));
    } };
matrix a, c,d;
long long n; int MOD;
matrix nhan(matrix a, matrix b)
{
```

```
    matrix c;
    for (int i = 1; i <= 3; i++)
        for (int j = 1; j <= 3; j++)
    {
        for (int k = 1; k <= 3; k++)
            c.val[i][j] = ((c.val[i][j] + (ll)a.val[i][k] * b.val[k][j] ) % MOD+MOD)%MOD;}
    return c;}
matrix mu (matrix a, long long k)
{
    matrix x;
    if ( k == 1) return a;
    x = mu (a, k/2);
    x = nhan (x, x);
    if (k % 2) x = nhan (x, a);
    return x;}
int main()
{
    freopen("loco.inp", "r", stdin);
    freopen("loco.out", "w", stdout);
    cin >> n>>MOD;
    a.val[1][1] = 1;
    a.val[1][2] = 1;
    a.val[1][3] = 1;
    a.val[2][1] = 1;
    a.val[2][2] = 0;
    a.val[2][3] = 0;
    a.val[3][1] = 0;
    a.val[3][2] = 1;
    a.val[3][3] = 0;
    if (n>2)
    {c = mu(a, n-2);
    cout<<( ((c.val[1][1])*2)%MOD + (c.val[1][2]*1)%MOD + c.val[1][3])% MOD << endl;}
    if (n==0) cout<<1;
    if (n==1) cout<<1;
    if (n==2) cout<<2;
    return 0;
}
```

4.4. Độ phức tạp

Chủ yếu thời gian của chương trình phụ thuộc vào thời gian tính c=mu(a,n-2), nên độ phức tạp là:

$$O(3^3 \log(n-2)) = O(\log(n-2))$$

4.5. Test.

https://drive.google.com/file/d/1qAei3sKxm_UP_mLFRmXx-wzvTR9nWira/view?usp=sharing

5. Bài 5: Bài toán Đo nước.

5.1. Đề bài

Bờm đang nghiên cứu mực nước biển ở hành tinh Quạt Mo. Sau nhiều ngày theo dõi, Bờm nhận thấy rằng quy luật của mực nước biển là: mực nước biển của một ngày bất kì bằng trung bình cộng mực nước biển của ngày hôm trước và ngày hôm sau. Dựa vào ghi chép mực nước biển hai ngày đầu của Bờm, hãy tính toán mực nước biển ngày thứ N.

Dữ liệu vào: donuoc.inp:

- Dòng 1: chứa 2 số nguyên b, a là mực nước biển 2 ngày đầu ($-100 \le a, b \le 100$). Số a là mực nước ngày thứ nhất, số b là mực nước ngày thứ 2.
- Dòng 2: chứa số nguyên dương N ($3 \le N \le 10^{12}$).

Kết quả ra: donuoc.inp: mực nước biển ngày thứ N.

Ví dụ:

donuoc.inp	**donuoc.inp**
1 2 3	3
3 1 3	-1

Ghi chú: 50% số test có $n<=10^7$

50% số test còn lại ứng với $10^7<n<=10^{12}$

❖ **Xác định bài toán:**

Input: 2 số nguyên a,b ($100 \le a, b \le 100$). Số nguyên dương N ($3 \le N \le 10^9$).

Output: F[N] - mực nước biển ngày thứ N.

5.2. Hướng dẫn giải thuật.

Sub1: với $n<=10^7$

Gọi F[k] là mực nước biển ngày thứ k.

Ta có $F[k] = \frac{F[k-1] + F[k+1]}{2}$

tức F[k+1] = 2 * F[k] - F[k-1]

đặt i = k + 1 ta có F[i] = 2 * F[i-1] - F[i-2] với i>2

Bài toán trở về thành tìm số Fibonacci thứ n.

Độ phức tạp: O(N) và không có xử lý số lớn.

Sub2: $n<=10^{12}$

Dùng kỹ thuật nhân ma trận để giải, giống bài Fibonacci.

Độ phức tạp: $O(2^3 \log(N))$ và không có xử lý số lớn.

Ta có $F_i = 2.F_{i-1} - 1.F_{i-2}$

$F_{i-1} = 1.F_{i-1} + 0.F_{i-2}$

Ta đặt ma trận $A = \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix}$

Từ hai biểu thức trên ta sẽ có: $\begin{bmatrix} F_i \\ F_{i-1} \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} F_{i-1} \\ F_{i-2} \end{bmatrix}$

Biến đổi ta có:

$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = A^{n-2} * \begin{bmatrix} F_2 \\ F_1 \end{bmatrix} = A^{n-2} * \begin{bmatrix} b \\ a \end{bmatrix}$ bởi vì **F(1) = a; F(2) = b.**

Đặt ma trận $c = A^{n-2}$. Kết quả cần tìm là: **F_n=c.val[1,1]*b+c.val[1,2]*a**

5.3. Chương trình.

```
#include<bits/stdc++.h>
using namespace std;
long a, b;
long long n;
struct maxtrix
{
	long long val[3][3];
	maxtrix()
	{
		memset(val, 0, sizeof(val));
	}
};
maxtrix res, x;
maxtrix nhan(maxtrix A, maxtrix B)
{
	maxtrix C;
	for (int i = 1; i <= 2; i++)
		for (int j = 1; j <= 2; ++j)
			for (int k = 1; k <= 2; ++k)
				C.val[i][j] += A.val[i][k] * B.val[k][j];
	return C;
```

```
}
void nhap()
{
	scanf("%ld%ld%ld", &a, &b,&n);
}
void khoitao()
{
	res.val[1][1] = res.val[2][2] = 1;
	x.val[1][1] = 2;
	x.val[1][2] = -1;
	x.val[2][1] = 1; }
void lam()
{
	n -= 2;
	khoitao();
	for (; n > 0; x = nhan(x, x))
	{
		if (n % 2)res = nhan(res, x);
		n /= 2;}
	printf("%lld", res.val[1][1] * b + res.val[1][2] * a); }
int main()
{
	freopen("donuoc.inp", "r", stdin);
	freopen("donuoc.out", "w", stdout);
	nhap();
	lam();
	return 0;}
```

5.4. Độ phức tạp.

Sau khi sử dụng kĩ thuật nhân ma trận, bài toán sẽ có độ phức tạp rất nhỏ: $O(2^3 \log(N))$. Ngoài cách sử dụng nhân ma trận thì thấy rằng đây là cấp số cộng, nên kết quả đưa ra trong O(1)

5.5. Test.

https://drive.google.com/file/d/1Ud45kSs8Ohd60hsSkxuuKT8V9FJPd2bo/view?usp=sharing

6. Bài 6: Bài toán ONE4EVER.

(Nguồn: http://vnoi.info/problems/ONE4EVER/)

6.1. Đề bài.

Dù thông minh, đẹp trai, học giỏi nhưng vẫn không thoát khỏi kiếp FA vì chỉ có 8cm và lười tắm, Khánh 3508 lại buồn bã trở về vnoi code lại từ đầu. Trong một ngày chán như con gián, Khánh 3508 nhổ trộm bông hướng dương nhà hàng xóm và ngồi … đếm cánh hoa. Mỗi lần một cánh hoa rụng xuống là câu nói "Tắm", "Không tắm" lại vang lên. Đã ba năm trôi qua Khánh 3508 vẫn chỉ ngồi đếm lá và chưa tắm rồi đột

nhiên anh ta đứng lên và chạy về máy tính: "Đúng rồi số ngẫu nhiên!". Hóa ra Khánh 3508 đã nghĩ ra cách làm mới mà không phải nhổ trộm hoa + ngồi đếm số cánh hoa. Chúng ta biết rằng số ngẫu nhiên được sinh ra bởi bộ ba số a, b, m theo quy tắc:

+Số thứ nhất là x_1 =b mod m

+Số thứ k là (a*x_{k-1} +b) mod m với k>1

Khánh 3508 sẽ lấy số x_k để so với lịch xem nó có phải ngày đẹp hay không để quyết định tắm rửa. Tuy nhiên do thích chơi trội Khánh 3508 đã để a, b, m, k rất lớn khiến máy tính bị đơ. Bạn hãy giúp Khánh tính x_k thật nhanh để cậu ta có thể tắm.

Input:

- Dòng 1: Số nguyên T (1≤T≤10) là số lượng test
- T dòng tiếp theo, mỗi dòng gồm 4 số nguyên a, b, m, k (1≤a, b, m, k≤10^{15})

Output:

- Gồm T dòng, mỗi dòng in ra số x_k tương ứng.

Example:

Input	Output
3 1 1 1 1 2 5 100 6 1 8 777 6	0 15 48

6.2. Hướng dẫn giải thuật.

Gọi F[i] chính là giá trị của số x_i.

Có F[i]=a*F[i-1]+b , dễ nhận ra đây là bài toán nhân ma trận.

- Cách phân tích 1:

Nếu ta lấy ma trận 2x2 (gọi là ma trận (1)): $\begin{bmatrix} F[i-1] & 1 \\ F[i] & 1 \end{bmatrix}$

nhân với ma trận (gọi là ma trận (2)): $\begin{bmatrix} a & 0 \\ b & 1 \end{bmatrix}$

thì ta sẽ có ma trận kết quả là: $\begin{bmatrix} F[i] & 1 \\ F[i+1] & 1 \end{bmatrix}$

Như vậy muốn tính số thứ k thì ta đem ma trận (2) mũ k-2 rồi nhân với ma trận (1) với i=2. Ma trận (1) với i=2 có dạng là: $\begin{bmatrix} F[1] & 1 \\ F[2] & 1 \end{bmatrix}$, trong đó F[1]=b%m; F[2]=(a+1)*b%m;

Vậy đáp án sẽ là phần tử ở dòng 2 cột 1 của ma trận kết quả.

Việc còn lại là xử lý trường hợp k=1, và tính kết quả mod m (có thể dùng xử lý số lớn hoặc kĩ thuật nhân 2 số < mod để chống tràn số).

- Cách phân tích 2:

Từ công thức F[i]=a*F[i-1]+b ta có:

F[i]=a*F[i-1]+1*b

b =0*F[i-1]+1*b

nên suy ra: $\begin{bmatrix} F[i] \\ b \end{bmatrix} = \begin{bmatrix} a & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} F[i-1] \\ b \end{bmatrix}$

Đặt A=$\begin{bmatrix} a & 1 \\ 0 & 1 \end{bmatrix}$

Thì $\begin{bmatrix} F[k] \\ b \end{bmatrix} = A^{k-1} . \begin{bmatrix} F[1] \\ b \end{bmatrix} = A^{k-1} . \begin{bmatrix} b\%m \\ b \end{bmatrix}$

Đặt $x=A^{k-1}$

Kết quả cần tìm là: (x.val[1][1]*(b%m) + x.val[1][2]*b)% m;

Bài này phải dùng đến xử lý số lớn hoặc kĩ thuật nhân 2 số có giá trị nhỏ hơn mod để chống tràn số.

6.3. Chương trình.

❖ **Cài đặt chương trình theo cách 1:**

```
#include <bits/stdc++.h>
typedef long long ll;
typedef unsigned long long ull;
using namespace std;
struct matrix
{
ll M[2][2];
};
ll mod,k,a,b;
matrix F1,F2;
 ll fast_multi(ll a,ll b)
{
if (!b) return 0;
if (b==1) return a;
```

```
if (b&1LL) return (fast_multi(a,b>>1LL)*2+a)%mod;
else return (fast_multi(a,b>>1LL)*2)%mod;
}
matrix Mul(matrix A,matrix B)
{
matrix c;
ll res;
for (long i=0; i<2; ++i)
for (long j=0; j<2; ++j)
{
c.M[i][j]=0;
for (long k=0; k<2; ++k) c.M[i][j]=(c.M[i][j] +
fast_multi(A.M[i][k],B.M[k][j]))%mod;
}
return c;
}
matrix Power(matrix a,ll x)
{
if (x==1) return a;
matrix tmp=Power(a,x>>1LL);
tmp=Mul(tmp,tmp);
return (x&1LL)?Mul(tmp,a):tmp;
}
 int main()
{
long nTest;
scanf("%ld",&nTest);
while (nTest--)
{
scanf("%lld%lld%lld%lld",&a,&b,&mod,&k);
if (k==1) printf("%lld\n",b%mod);
else if (k==2) printf("%lld\n",fast_multi(a+1,b));
else
{
F1.M[0][0]=a; F1.M[0][1]=0;
F1.M[1][0]=b%mod; F1.M[1][1]=1;
F1=Power(F1,k-2);
F2.M[0][0]=b%mod; F2.M[0][1]=1;
F2.M[1][0]=fast_multi(a+1,b); F2.M[1][1]=1;
F2=Mul(F2,F1);
printf("%lld\n",F2.M[1][0]);
} } }
```

❖ **Cài đặt chương trình theo cách 2:**

```
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;
struct matrix {long long val [3][3];
```

```
matrix()
{
    memset(val, 0, sizeof(val));
}
};
matrix res, c;
int t;
long long a, b, m, k;
long long fast_multi(ll a,ll b)
{
   if (!b) return 0;
   if (b==1) return a;
   if (b&1LL) return (fast_multi(a,b>>1LL)*2+a)%m;  //b không chia hết cho 2
   else return (fast_multi(a,b>>1LL)*2)%m;
}
matrix nhan(matrix a, matrix b)
{
   matrix c;
   for (int i = 1; i <= 2; i++)
      for (int j = 1; j <= 2; j++)
   {
      c.val[i][j] = 0;
      for (int k = 1; k <= 2; k++)
          c.val[i][j] = (c.val[i][j] + fast_multi(a.val[i][k], b.val[k][j])+m)%m;
   }
   return c;
}
matrix power (matrix a, long long k)
{
   matrix c;
   c.val[1][1] = 1;
   c.val[1][2] = 0;
   c.val[2][1] = 0;
   c.val[2][2] = 1;
   for (; k; k /= 2, a = nhan (a, a))
   if (k & 1) c = nhan (c, a);
   return c;
}
matrix x;
int main()
{
   freopen("ONE4EVER.inp", "r", stdin);
   freopen("ONE4EVER.out", "w", stdout);
   cin >> t;
   for (int i = 1; i <= t; i++)
   {cin >>a>>b>>m>>k;
   res.val[1][1] = a;
   res.val[1][2] = 1;
```

```
res.val[2][1] = 0;
res.val[2][2] = 1;
if (k==1) cout<<b%m;
else if (k==2) cout<<fast_multi(a+1,b);
else
{
x = power(res, k - 1);
cout << (fast_multi(x.val[1][1],b%m) + fast_multi(x.val[1][2],b))% m;}
cout << endl;}
return 0;}
```

6.4. Độ phức tạp.

❖ Với cả 2 cách đều có độ phức tạp là:

$O(T *2^3*\log(k))=O(T*\log(k))$, T là số lượng test.

❖ **Chú ý:** Phép nhân hai số rồi lấy mod với các số có kích thước tới tận 10^{15} nên phải dùng xử lý số lớn hoặc kĩ thuật nhân 2 số có kích thước nhỏ hơn mod để chống tràn số. Tham khảo thuật toán tại: http://www.giaithuatlaptrinh.com/?p=1117 . Hãy kiểm tra lại cách quản lý tràn số của code này nhé!

6.5. Test

Có thể test trực tiếp trên SPOJ với mã bài ONE4EVER

Hoặc lấy bộ test tại:

https://drive.google.com/file/d/1KzMshj1kY66xivp081g3GWAXc1gduSQE/view?usp=sharing

7. Bài 7: *Bài toán Trò chơi bắt chước*

(Mã bài: ABA15E, nguồn https://www.codechef.com/ABCS2015/problems/ABA15E)

7.1. Đề bài.

Turing hiện đang làm việc để crack các máy bí ẩn.Nhưng ông thấy rằng có hai hàm toán học là f(n) và g(n) được sử dụng để mã hóa tin nhắn của người Đức. Ông muốn thử nghiệm khám phá của mình để bắt chước cách mã hóa của máy tính.

Các hàm được định nghĩa là:

g (n + 1) = 4 * g (n) + f (n + 1) + c
f (n + 2) = 3 * f (n + 1) + 2 * f (n)

Dữ liệu ban đầu:
f (0) = 1;
f (1) = 1;
g (-1) = 1;
g (0) = 1;
c = 2;

***Yêu cầu**:* Cho số nguyên n, cần phải tìm giá trị g(n) modulo 1000000007.

Đầu vào: Dòng đầu tiên ghi số lượng các trường hợp thử nghiệm T, T dòng tiếp theo mỗi dòng ghi một giá trị của n.

Đầu ra: Với mỗi test, xuất ra giá trị của g(n).

- **$1 \le T \le 1000$**
- **$1 \le n \le 10000000$**

Ví dụ:

ABA15E.INP	ABA15E.OUT
5 1 2 3 6 1000	7 35 159 12835 566998663

7.2. Hướng dẫn giải thuật

Ta có f(n + 2) = 3 * f (n + 1) + 2 * f (n) nên suy ra: f(n + 1) = 3 * f(n) + 2 * f (n-1)

Thay vào g(n + 1) = 4 * g(n) + f(n + 1) + c

Ta có: g(n + 1) = 4 * g(n) + 3 * f(n) + 2 * f (n-1)+ 1*c

f(n+1) = 0* g(n) + 3 * f(n) + 2 * f (n-1)+0*c

f(n) = 0* g(n) + 1 * f(n) + 0 * f (n-1)+0*c

c = 0* g(n) + 0 * f(n) + 0 * f (n-1)+1*c

$$\begin{bmatrix} g(n+1) \\ f(n+1) \\ f(n) \\ c \end{bmatrix} = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 0 & 3 & 2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} g(n) \\ f(n) \\ f(n-1) \\ c \end{bmatrix} \quad (1)$$

Ta đặt ma trận $A = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 0 & 3 & 2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Từ (1), biến đổi ta có:

$$\begin{bmatrix} g(n) \\ f(n) \\ f(n-1) \\ c \end{bmatrix} = A * \begin{bmatrix} g(n-1) \\ f(n-1) \\ f(n-2) \\ c \end{bmatrix} = A^{n-1} \begin{bmatrix} g(1) \\ f(1) \\ f(0) \\ c \end{bmatrix}$$ mà g(1)=4*g(0)+f(1)+c=4*1+1+2=7

Nên $$\begin{bmatrix} g(n) \\ f(n) \\ f(n-1) \\ c \end{bmatrix} = A^{n-1} \begin{bmatrix} 7 \\ 1 \\ 1 \\ 2 \end{bmatrix}$$

Đặt ma trận c= A^{n-1}.

Kết quả cần tìm là: **F_n=c.val[1,1]*7+c.val[1,2]*1+ c.val[1,3]*1+ c.val[1,4]*2**

7.3. Chương trình

```
#include<bits/stdc++.h>
#define MOD 1000000007
using namespace std;
typedef long long ll;
ll n, t;
struct matrix
{
    ll val[5][5];
    matrix()
    {
        memset(val, 0, sizeof(val));
    }
};
ll nhan2(ll a, ll b)
{
    if( b == 0 )
        return 0;
    if( b == 1 )
        return a;
    if( b & 1ll ) return (nhan2(a, b >> 1ll ) * 2 + a) % MOD;
    else return (nhan2(a, b >> 1ll) << 1 ) % MOD;
}
matrix nhan(matrix x, matrix y)
{
    matrix z;
    for(ll i = 1; i <= 4; i++)
        for(ll j = 1; j <= 4; j++)
        {
            z.val[i][j] = 0;
            for(ll t = 1; t <= 4; t++)
            {
                z.val[i][j] = z.val[i][j] + nhan2( x.val[i][t], y.val[t][j] );
                z.val[i][j] %= MOD;
```

```
            }
        }
    return z;
}
matrix mu(matrix x, ll y)
{
    if( y == 1 )
        return x;
    matrix z;
    z = mu(x, y >> 1);
    z = nhan(z, z);
    if( y % 2 )
        z = nhan(z, x);
    return z;
}
matrix x, y;
int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    freopen("aba15e.inp","r",stdin);
    freopen("aba15e.out","w",stdout);
    cin >> t;
    while( t-- )
    {
        cin >> n;
        if( n == 1 )
        {
            cout << 7 << "\n";
            continue;
        }
     x.val[1][1] = 4, x.val[1][2] = 3, x.val[1][3] = 2, x.val[1][4] = 1;
     x.val[2][1] = 0, x.val[2][2] = 3, x.val[2][3] = 2, x.val[2][4] = 0;
     x.val[3][1] = 0, x.val[3][2] = 1, x.val[3][3] = 0, x.val[3][4] = 0;
     x.val[4][1] = 0, x.val[4][2] = 0, x.val[4][3] = 0, x.val[4][4] = 1;
     x = mu(x, n - 1);
    cout<<(nhan2(x.val[1][1], 7)+x.val[1][2] + x.val[1][3]+nhan2(x.val[1][4], 2 ))
% MOD << "\n";
    }
return 0;
}
```

7.4. Độ phức tạp: **O(T*4^3*log(n)) = O(T*log(n))**

7.5. Test.

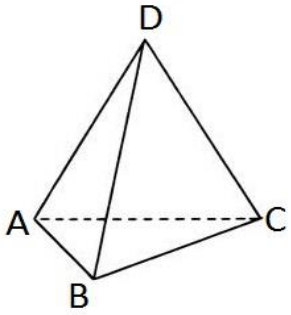
https://drive.google.com/file/d/1GcYZtWzv3So1yBomz-tMaOmIkr9MkDzB/view?usp=sharing

8. Bài 8: Bài toán tứ diện.

(Nguồn bài http://codeforces.com/problemset/problem/166/E)

8.1. Đề bài.

Cho một tứ diện, đánh dấu các đỉnh lần lượt là A, B, C, D.



Một con kiến đang đứng trên đỉnh *D* của tứ diện. Con kiến khá tích cực di chuyển và nó không chịu nhàn rỗi. Với mỗi bước đi, nó bước từ một đỉnh tới đỉnh khác dọc theo một số cạnh của tứ diện. Con kiến không bao giờ chịu đứng yên ở một chỗ.

Yêu cầu: đếm số cách mà con kiến có thể đi từ đỉnh *D* ban đầu rồi quay về chính nó trong đúng *n* bước. Nói cách khác, bạn sẽ được yêu cầu tìm ra số con đường tuần hoàn khác nhau có chiều dài *n* từ đỉnh *D* đến chính nó. Vì số có thể khá lớn nên bạn nên in theo modulo $(10^9 + 7)$.

Đầu vào:

Dòng đầu tiên chứa số nguyên duy nhất *n* $(1 \leq n \leq 10^7)$ - chiều dài của đường đi.

Đầu ra:

In số nguyên duy nhất là kết quả tìm được modulo $(10^9 + 7)$.

Ví dụ:

Tudien.inp	Tudien.out
2	3

Chú thích:

Có 3 cách đi có thể với bộ dữ liệu trên là là:

- *D - A - D*

- *D - B - D*
- *D - C - D*

8.2. Hướng dẫn giải thuật.

Bài này có thể giải theo cách khác với độ phức tạp O(n), tham khảo lời giải tại: http://codeforces.com/blog/entry/4173

Nhưng tôi xin trình bày cách làm khác tốt hơn với nhân ma trận với độ phức tạp O(logn):

Gọi s[i].a là số cách di chuyển đến đỉnh A tại thời điểm i. Tương tự cho s[i].b, s[i].c, s[i].d. Tại thời điểm thứ i ta có ma trận $\begin{bmatrix} s[i].a \\ s[i].b \\ s[i].c \\ s[i].d \end{bmatrix}$ và ta kí hiệu ma trận này là s[i], tương tự với ma trận s[i+1].

Nên suy ra:

s[i + 1].a = s[i].b + s[i].c + s[i].d

s[i + 1].b = s[i].a + s[i].c + s[i].d

s[i + 1].c = s[i].a + s[i].b + s[i].d

s[i + 1].d = s[i].a + s[i].b + s[i].c

=> $s[i+1] = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} * s[i] = M*s[i]$ (với $M=\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$)

=> $s[i + k] = M^k * s[i]$

Mà s[0].d = 1;

=> $s[n] = M^n * s[0] = M^n * \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

Kết quả = s[n].d

8.3. Chương trình.

```
#include <bits/stdc++.h>
#define maxn 5
#define MOD 1000000007
#define ll long long
```

```
using namespace std;
struct matrix
{
  ll val[maxn][maxn]  ;
   matrix()
    {
        memset(val,0,sizeof(val));
    }
};
ll nhannhanh(ll a,ll b)
{
    if(!b) return 0;
    if(b==1) return a;
    if (b&1ll) return (nhannhanh(a,b>>1ll)*2+a)%MOD;  //b không chia hết cho 2
    else return (nhannhanh(a,b>>1ll)*2)%MOD;
}
matrix nhan(matrix a,matrix b)
{
    matrix c;
    for(int i=1; i<=4; i++)
        for(int j=1; j<=4; j++)
        {
            c.val[i][j]=0;
            for(int k=1; k<=4; k++)
                c.val[i][j]=(c.val[i][j]+nhannhanh(a.val[i][k],b.val[k][j]))%MOD;
        }
    return c;
}
matrix mu(matrix a,ll n)
{
    matrix x;
    if(n==1) return a;
    x=mu(a,n/2);
    x=nhan(x,x);
    if(n%2) x=nhan(x,a);
    return x;
}
matrix A,res;
int n;
int main()
{
    freopen("tudien.inp","r", stdin);
     freopen("tudien.out","w",stdout);
    cin>>n;
    A.val[1][2]=A.val[1][3]=A.val[1][4]=A.val[2][1]=A.val[2][3]=A.val[2][4]=
    A.val[3][1]=A.val[3][2]=A.val[3][4]=A.val[4][1]=A.val[4][2]=A.val[4][3]=1;
    res=mu(A,n);
     cout<<res.val[4][4];
```

}

8.4. Độ phức tạp: O(log n)

8.5. Test.

https://drive.google.com/file/d/1lvH8ywc0g3apmQL8HP29JzD5oNKyPfYY/view?usp=sharing

9. Bài 9: Bài toán Giấc Mơ

9.1. Đề bài.

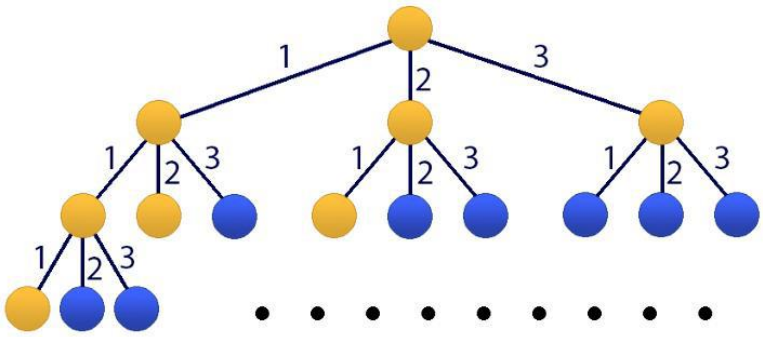
Sau một ngày mệt nhọc đón các đoàn về tham dự Trại hè tin học 2017, thầy Minh vô cùng mệt mỏi ngủ ngay khi học sinh về hết phòng. Trong giấc mơ, thầy Minh mơ đang vẽ một cây vô hạn, mỗi nút có đúng n nút con, khoảng cách từ nút cha tới các nút con của nó theo thứ tự từ trái sang phải là d1, d2, … , dn. Thầy Minh đang có một số k và rất muốn biết có bao nhiêu đỉnh trên cây mà khoảng cách từ đỉnh đó tới gốc không vượt quá k.

Dữ liệu: Vào từ file văn bản DREAM.INP.

- Dòng đầu chứa hai số nguyên dương n và k.
- Dòng thứ hai chứa n số nguyên dương d1, d2, … , dn (di ≤ 100)

Kết quả: Đưa ra file văn bản DREAM.OUT số lượng đỉnh mà khoảng cách từ đỉnh đó tới gốc không vượt quá k. Đưa ra theo số dư cho $10^9 + 7$.

Ví dụ:



DREAM.INP	DREAM.OUT
3 3 1 2 3	8

Ghi chú: 50% test có k ≤ 10000, n ≤ 100

50% test có 1000 < k ≤ 10^{18}, n≤ 100000

❖ **Xác định bài toán:**

Input: hai số nguyên dương k và n ($k \le 10^{18}$, $n \le 10^6$),

n số nguyên dương d1, d2, … , dn ($di \le 100$).

Output: (số lượng đỉnh mà khoảng cách từ đỉnh đó tới gốc không vượt quá k) mod ($10^9 + 7$).

9.2. Hướng dẫn giải thuật.

Gọi F_x là số cách sinh tổng x. Công thức tính F_x là:

$$F_x = \sum_{i=1}^{n} F_{x-di}$$

Kết quả cần tìm: $T_k=F_0+F_1+...+F_k$.

Sub 1: Sử dụng 2 vòng for:

```
For (int i=0;i<=k;i++)
{
    For (int j=0; j<=n;j++)
        F[i]=F[i]+F[i-d[j]];
        T[k]=T[k]+F[i];
}
```

Độ phức tạp O(n*k).

Sub 2: Sử dụng nhân ma trận

$$F_x = \sum_{i=1}^{n} F_{x-d_i} = \sum_{p=1}^{100} F_{x-p} \times c_p$$

Với c_p là số lần xuất hiện số p trong dãy d.

Biến đổi thành ma trận:

$$F_x = c_1 \times F_{x-1} + \cdots + c_{100} \times F_{x-100}$$

$$F_{x-1} = F_{x-1}$$

$$F_{x-2} = F_{x-2}$$

...

$$\begin{matrix} F_x \\ F_{x-1} \\ F_{x-2} \\ \\ \end{matrix} = \begin{matrix} c_1 & c_2 & \dots c_{100} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ \dots\dots & & \end{matrix} \times \begin{matrix} F_{x-1} \\ F_{x-2} \\ F_{x-3} \\ F_{x-100} \end{matrix}$$

Để tính F_x ta sử dụng bảng trên.

Tính $T_x=T_{x-1}+F_x=T_{x-1}+c_1\times F_{x-1}+...+c_{100}\times F_{x-100}$

Điền thêm giá trị T_x vào dòng đầu ma trận, ta có:

$T_x=1\times T_{x-1}+c_1\times F_{x-1}+ c_2\times F_{x-2}+.....+c_{100}\times F_{x-100}$

$F_x=0\times T_{x-1}+c_1\times F_{x-1}+ c_2\times F_{x-2}+.....+c_{100}\times F_{x-100}$

$F_{x-1}=0\times T_{x-1}+1\times F_{x-1}+ 0\times F_{x-2}+.....+0\times F_{x-100}$

$F_{x-2}=0\times T_{x-1}+0\times F_{x-1}+ 1\times F_{x-2}+.....+0\times F_{x-100}$

.....

$F_{x-99}=0\times T_{x-1}+0\times F_{x-1}+ .....1\times F_{x-99}+0\times F_{x-100}$

Gọi A= $\begin{bmatrix} T_x \\ F_x \\ F_{x-1} \\ F_{x-2} \\ . \\ . \\ . \\ F_{x-99} \end{bmatrix}$, ma trận B= $\begin{bmatrix} T_{x-1} \\ F_{x-1} \\ F_{x-2} \\ F_{x-3} \\ . \\ . \\ . \\ F_{x-100} \end{bmatrix}$

Ta có: A= D×B, với ma trận D là: D= $\begin{bmatrix} 1 & c1 & c2 & ... & c100 \\ 0 & c1 & c2 & ... & c100 \\ 0 & 1 & 0 & ... & 0 \\ 0 & 0 & 1 & ... & 0 \\ 0 & 0 & 0 & 1... & 0 \\ . & . & . & . & . \\ 0 & 0 & ... & 1 & 0 \end{bmatrix}$

Ma trận kết quả E, với:

$E=D^k\times \begin{bmatrix} T_0 \\ F_0 \\ F_{-1} \\ F_{-2} \\ . \\ . \\ . \\ F_{-99} \end{bmatrix}$

Trong đó $T_0=F_0=1$, các giá trị F_{-1}, F_{-2}, ..., F_{-99} đều bằng 0.

Gọi ma trận $res=D^k$

Kết quả cần tìm là: **(res.val[1][1]+res.val[1][2]) % (10^9+7).**

Độ phức tạp : O(100^3.logk)

9.3. Chương trình.

```
#include<bits/stdc++.h>
#define mod 1000000007
const long long T= 7ll*mod*mod;
using namespace std;
struct matran
{
    long long val[102][102];
    matran()
    {
        memset(val,0,sizeof(val));
    }
};
matran nhan(matran x,matran y,long m,long n,long p)
{
    matran z;
    long long num;
    for(long i=1;i<=m;++i)
        for(long j=1;j<=p;++j)
        {
            num=0;
            for(long k=1;k<=n;++k)
            {
                num+=x.val[i][k]*y.val[k][j];
                if(num>T)num-=T;
            }
            z.val[i][j]=num%mod;
        }
    return z;
}
matran a,res;
long long n,k;
long long c[120];
void nhap()
{
    memset(c,0,sizeof(c));
    scanf("%lld%lld",&n,&k);
    int d;
    for(long i=1;i<=n;++i)
    {
        scanf("%d",&d);
```

```
            c[d]++;
        }
    }
    void khoitao()
    {
        for(long i=1;i<=101;++i)res.val[i][i]=1;
        a.val[1][1]=1;
        a.val[1][2]=a.val[2][2]=c[1];
        for(long i=3;i<=101;++i)
        {
            a.val[i][i-1]=1;
            a.val[1][i]=a.val[2][i]=c[i-1];
        }
    }
    void lam()
    {
        khoitao();
        long long b=k;
        for(;b>0;a=nhan(a,a,101,101,101))
        {
            if(b%2)res=nhan(a,res,101,101,101);
            b=b/2;
        }
        long long kq=(res.val[1][1]+res.val[1][2])%mod;
        kq=(kq+mod)%mod;
        printf("%lld",kq);
    }
    int main()
    {
        freopen("DREAM.INP","r",stdin);
        freopen("DREAM.OUT","w",stdout);
        nhap();
        lam();
        return 0;
    }
```

9.4. Độ phức tạp.

Thuật toán có độ phức tạp là $O(n+101^3\log(k))=O(n+10^6\log(k))$.

9.5. Test.

https://drive.google.com/file/d/1-JFB-UYhADdPeBUu2Vl45nOOyvjK5FVZ/view?usp=sharing

10. Bài 10: Bài toán FIB3

10.1. Đề bài.

Dãy Fibonacci là dãy vô hạn các số tự nhiên bắt đầu bằng hai phần tử 0 và 1, các phần tử sau đó được thiết lập theo quy tắc *mỗi phần tử luôn bằng tổng hai phần tử*

trước nó. Dãy số Fibonacci rất đặc biệt này được Leonardo Fibonacci (hay còn có tên tên khác là Leonarda da Pisa) là một nhà toán học người Ý công bố vào năm 1202 trong cuốn sách Liber Abacci - Sách về toán đố qua 2 bài toán: Bài toán con thỏ và bài toán số các "cụ tổ" của một ong đực.

Henry E Dudeney (1857 - 1930) (là một nhà văn và nhà toán học người Anh) nghiên cứu ở bò sữa, cũng đạt kết quả tương tự.

Thế kỉ XIX, nhà toán học Edouard Lucas (người Pháp) xuất bản một bộ sách bốn tập với chủ đề toán học giải trí, ông đã dùng tên Fibonacci để gọi dãy số kết quả của bài toán từ cuốn Liber Abaci – bài toán đã sinh ra dãy Fibonacci.

Dãy số này hầu như biến hóa vô tận. Chính đều đó làm cho bao nhà toán học (chuyên nghiệp lẫn nghiệp dư) và ngay cả chúng ta say mê nghiên cứu, khám phá về nó.

Xét dãy số *fib3* là một biến thể của dãy số Fibonacci, với ba số nguyên không âm a, b, c ta xây dựng dãy số theo quy tắc sau:

$$fib3(n) = \begin{cases} n & if \quad n \le 3 \\ a.fib3(n-1) + b.fib3(n-2) + c.fib3(n-3) & if \quad n\%3 = 1 \\ b.fib3(n-1) + c.fib3(n-2) + a.fib3(n-3) & if \quad n\%3 = 2 \\ c.fib3(n-1) + a.fib3(n-2) + b.fib3(n-3) & if \quad n\%3 = 0 \end{cases}$$

Yêu cầu: Cho 5 số nguyên không âm a, b, c, k, n. Hãy tính số fib3(n)%k.

Input:

- Một dòng chứa 5 số nguyên không âm (a, b, c, k, n) (a, b, c, k≤10^9)

Output:

- Chứa một số là kết quả tìm được tương ứng với dữ liệu vào.

fib3.inp	fib3.out
1 1 1 100 4	6

Subtask 1: n≤10^6 **;**

Subtask 2: n≤10^9, a=b=c=1;

Subtask 3: n≤10^9.

10.2. Hướng dẫn thuật toán

Đây là một bài toán rất hay và khó.

Với subtask 1: n<=10^6

Ta có thể dùng vòng For để tính lần lượt các Fib3[i]. Độ phức tạp O(n).

Với subtask 2: n<=10^9, a=b=c=1.

Tất cả các Fib3(i), với i>3, đều có một công thức tính chung, không phân biệt n chia cho 3 dư bao nhiêu. Với điều kiện này ta có thể sử dụng phương pháp nhân ma trận đơn giản như sau:

Có Fib3[i]=i nếu i<=3 và

Fib3[i]=1.Fib3[i-1]+ 1.Fib3[i-2]+ 1.Fib3[i-3]

Fib3[i-1]=1.Fib3[i-1]+ 0.Fib3[i-2]+ 0.Fib3[i-3]

Fib3[i-2]=0.Fib3[i-1]+ 1.Fib3[i-2]+ 0.Fib3[i-3]

Nên suy ra: $\begin{bmatrix} fib3_i \\ fib3_{i-1} \\ fib3_{i-2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} . \begin{bmatrix} fib3_{i-1} \\ fib3_{i-2} \\ fib3_{i-3} \end{bmatrix}$, đặt A= $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

Tương tự như các bài trước, ta sẽ có $\begin{bmatrix} fib3_n \\ fib3_{n-1} \\ fib3_{n-2} \end{bmatrix}$ =A^{n-3}. $\begin{bmatrix} fib3_3 \\ fib3_2 \\ fib3_1 \end{bmatrix}$ = A^{n-3}. $\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$

Đặt res=A^{n-3}.

Kết quả cần tìm là **fib3(n)=res.val[1,1]*3 + res.val[1,2]*2 + res.val[1,3]*1**

Độ phức tạp: O(3^3log(n-3)) = O(log(n-3))

Với subtask 3: n<=10^9, với sub này ta phải sử dụng tới 3 ma trận khác nhau:

+ Với n%3=1: Ta có Fib3[i]=a.Fib3[i-1]+ b.Fib3[i-2]+ c.Fib3[i-3]

Làm tương tự sub 2, ta có $\begin{bmatrix} fib3_i \\ fib3_{i-1} \\ fib3_{i-2} \end{bmatrix} = \begin{bmatrix} a & b & c \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} . \begin{bmatrix} fib3_{i-1} \\ fib3_{i-2} \\ fib3_{i-3} \end{bmatrix}$ (1)

Đặt ma trận mx_1= $\begin{bmatrix} a & b & c \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

+ Với n%3=2, tương tự ta có $\begin{bmatrix} fib3_i \\ fib3_{i-1} \\ fib3_{i-2} \end{bmatrix} = \begin{bmatrix} b & c & a \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} . \begin{bmatrix} fib3_{i-1} \\ fib3_{i-2} \\ fib3_{i-3} \end{bmatrix}$ (2)

Đặt $mx_2=\begin{bmatrix} b & c & a \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

+ Với n%3=0, tương tự ta có $\begin{bmatrix} fib3_i \\ fib3_{i-1} \\ fib3_{i-2} \end{bmatrix} = \begin{bmatrix} c & a & b \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} . \begin{bmatrix} fib3_{i-1} \\ fib3_{i-2} \\ fib3_{i-3} \end{bmatrix}$ (3)

Đặt $mx_0=\begin{bmatrix} c & a & b \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

Bây giờ phải làm thế nào để tính các F[i], mà với 3 giá trị F[i] liên tiếp nhau thì lại tính qua 3 ma trận khác nhau?

Ta phân tích như sau:

Giả sử n là số chia hết cho 3.

Giả sử đặt $A_n=\begin{bmatrix} fib3_n \\ fib3_{n-1} \\ fib3_{n-2} \end{bmatrix}$, $A_{n-1}=\begin{bmatrix} fib3_{n-1} \\ fib3_{n-2} \\ fib3_{n-3} \end{bmatrix}$,....., $A_3=\begin{bmatrix} fib3_3 \\ fib3_2 \\ fib3_1 \end{bmatrix}$; Khi đó:

$A_n=mx_0.A_{n-1}$ (theo (3)); n-1 bây giờ là số chia cho 3 dư 2

$= mx_0.mx_2.A_{n-2}$ (theo (2)); n-2 là số chia 3 dư 1

$= mx_0.mx_2.mx_1. A_{n-3}$ (theo (1)); n-3 là số chia 3 dư 0

$= mx_0.mx_2.mx_1.mx_0. A_{n-4}$; n-4 chia 3 dư 2

$= mx_0.mx_2.mx_1.mx_0. mx_2. A_{n-5}$

$= mx_0.mx_2.mx_1.mx_0. mx_2.mx_1. A_{n-6}$

.....

$= mx_0.mx_2.mx_1.mx_0. mx_2.mx_1..... mx_0. mx_2.mx_1.A_3$

$= mx_0.mx_2.mx_1.mx_0. mx_2.mx_1..... mx_0. mx_2.mx_1.\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$

Đặt ma trận $mx= mx_0. mx_2.mx_1$, số lần lặp tích $mx_0. mx_2.mx_1$ là b=[n/3]-1.

Đặt $res=mx^b$.

Nếu n%3=1 thì res=nhan(mx_1,res); *(tức là $res=mx_1*res$)*

Nếu n%3=2 thì res=nhan(nhan(mx_2,mx_1),res);*(tức là res=mx_2*mx_1*res)*

Kết quả cần tìm là: ***kq=3*res.val[1][1]+2*res.val[1][2]+res.val[1][3];***

10.3. Chương trình

```
#include <bits/stdc++.h>
using namespace std;
struct matran
{
   long long val[4][4];
   matran()
   {
      memset(val,0,sizeof(val));
      val[2][1]=val[3][2]=1;
   }
};
long long a,b,c,k,n;
matran res,mx,mx1,mx2,mx0;
matran nhan(matran A,matran B)
{
   matran C;
   memset(C.val,0,sizeof(C.val));
   for(long i=1;i<=3;++i)
      for(long j=1;j<=3;++j)
      {
         for(long z=1;z<=3;++z)
            C.val[i][j]+=A.val[i][z]*B.val[z][j];
         C.val[i][j]=C.val[i][j]%k;
         while(C.val[i][j]<0)C.val[i][j]+=k;
      }
   return C;
}
void khoitao()
{
   mx1.val[1][1]=mx2.val[1][3]=mx0.val[1][2]=a;
   mx1.val[1][2]=mx2.val[1][1]=mx0.val[1][3]=b;
   mx1.val[1][3]=mx2.val[1][2]=mx0.val[1][1]=c;
   memset(res.val,0,sizeof(res.val));
   res.val[1][1]=res.val[2][2]=res.val[3][3]=1;
   mx=nhan(mx0,mx2);
   mx=nhan(mx,mx1);
}
void lam()
{
   if(n<=3)
   {
      printf("%ld ",n%k);
      return;
```

```
    }
    b=n/3-1;
    for(;b>0;mx=nhan(mx,mx))
    {
        if(b&1)res=nhan(res,mx);
        b=b>>1;
    }
    if(n%3==1)
    {
        res=nhan(mx1,res);
    }
    if(n%3==2)
    {
        res=nhan(nhan(mx2,mx1),res);
    }
    long long kq=3*res.val[1][1]+2*res.val[1][2]+res.val[1][3];
    kq=kq%k;
    while(kq<0)
        kq+=k;
    cout<<kq<<endl;
}
int main()
{
    freopen("fib3.inp","r",stdin);
    freopen("fib3.out","w",stdout);
    while (cin>>a>>b>>c>>k>>n)
  {
    khoitao();
    lam();
  }
    return 0;
}
```

10.4. Độ phức tạp

Chủ yếu là thời gian tính ma trận res mũ [n/3]-1 và thời gian nhân 3 ma trận kích thước 3. Vậy độ phức tạp là: $O(3^3.\log(n/3)+3^3*3^3) \approx O(\log(n/3))$.

Nhận xét: *Điều thú vị trong bài toán này là phải sử dụng tới 3 ma trận, và kết hợp chúng với nhau một cách khéo léo. Nếu chúng ta thay đổi thứ tự nhân giữa các ma trận, giả sử với phân tích như trên thứ tự phép nhân là mx_0. $mx_2.mx_1$ nhưng nếu ta vô tình thực hiện $(mx_2.mx_1).mx_o$ thì sẽ dẫn tới kết quả sai. Do đó khi làm việc với ma trận cần phải chú ý thực hiện thứ tự phép nhân cho chính xác.*

10.5. Test.

https://drive.google.com/file/d/1oSMcV4CDWdI-YkUtfqc_4a1RN03Xrt7N/view?usp=sharing

11. Bài 11. Bài toán Dãy Fibonacci – FIBSEQ.

(Nguồn: Đề thi học sinh giỏi quốc gia năm 2017)

11.1. Đề bài.

Năm 1202, Leonardo Fibonacci, nhà toán học người Ý, tình cờ phát hiện ra tỉ lệ vàng 0.618 được tiệm cận bằng thương của hai số liên tiếp trong một loại dãy số vô hạn được một số nhà toán học ẤN ĐỘ xét đến từ năm 1150. Sau đó dãy số này được đặt tên là dãy số Fibonacci $\{F_i: i=1, 2, ...\}$, trong đó $F_1=F_2=1$ và mỗi số tiếp theo trong dãy được tính bằng tổng của hai số ngay trước nó. Đây là 10 số đầu tiên của dãy số Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34, 35. Người ta đã phám phá ra mối liên hệ chặt chẽ của số Fibonacci và tỉ lệ vàng với sự phát triển trong tự nhiên (cánh hoa, cành cây, vân gỗ), trong vũ trụ (hình xoáy trôn ốc dải ngân hà, khoảng cách giữa các hành tinh), hay sự cân đối của cơ thể con người. Đặc biệt số Fibonacci được ứng dụng mạnh mẽ trong kiến trúc (Kim tự tháp Ai Cập, tháp Eiffel), trong mỹ thuật (các bức tranh của Leonardo da Vinci), trong âm nhạc (các bản giao hưởng của Mozart) và trong nhiều lĩnh vực khoa học kĩ thuật.

Trong toán học, dãy Fibonacci là một đối tượng tổ hợp quan trọng có nhiều tính chất đẹp, có nhiều phương pháp hiệu quả liệt kê và tính các số Fibonacci như phương pháp lặp hay phương pháp nhân ma trận.

Sau khi được học về dãy số Fibonacci, Sơn rất muốn phát hiện thêm những tính chất của dãy số này. Vì thế Sơn đặt ra bài toán sau đây: Hỏi rằng có thể tìm được một tập con các số trong n số Fibonacci liên tiếp bắt đầu từ số thứ i, sao cho tổng của chúng chia hết cho một số nguyên dương k (k<=n) cho trước hay không? Nhắc lại, một tập con q số của một dãy n số là một cách chọn ra q số bất kỳ trong n số của dãy đó, mỗi số được chọn không quá một lần.

Yêu cầu: Hãy giúp Sơn giải quyết bài toán đặt ra.

Dữ liệu: Vào từ file văn bản FIBSEQ.INP bao gồm:

- Dòng thứ nhất ghi số nguyên dương T (T<=10) là số lượng bộ dữ liệu.
- Mỗi dòng trong T dòng tiếp theo chứa ba số nguyên dương n, i và k là thông tin của một bộ dữ liệu.

Các số trên cùng dòng được ghi cách nhau bởi dấu cách.

Kết quả: Ghi ra file văn bản FIBSEQ.OUT bao gồm T dòng tương ứng với kết quả của T bộ dữ liệu đầu vào, mỗi dòng có cấu trúc như sau: Đầu tiên ghi số nguyên q là số lượng các số trong tập con tìm được, tiếp đến ghi q số nguyên là các số thứ tự trong dãy Fibonacci của q số trong tập con tìm được. Nếu không tìm được tập con thỏa mãn điều kiện đặt ra thì ghi ra một số 0.

Nếu có nhiều cách chọn thì chỉ cần đưa ra một cách chọn bất kỳ.

Ràng buộc:

- Có 20% số lượng test thỏa mãn điều kiện: $n \leq 20$, $i \leq 10^6$
- Có 20% số lượng test thỏa mãn điều kiện: $n \leq 10^3$, $i \leq 10^6$
- Có 20% số lượng test thỏa mãn điều kiện: $n \leq 10^6$, $i \leq 10^6$
- Có 10% số lượng test thỏa mãn điều kiện: $n \leq 20$, $i \leq 10^{15}$
- Có 10% số lượng test thỏa mãn điều kiện: $n \leq 10^3$, $i \leq 10^{15}$
- Có 20% số lượng test còn lại thỏa mãn điều kiện: $n \leq 10^6$, $i \leq 10^{15}$

Ví dụ:

FIBSEQ.INP	FIBSEQ.OUT
1 10 3 9	2 5 7

Giải thích: Trong ví dụ trên một tập con thỏa mãn điều kiện đặt ra là tập gồm 2 số $F_5=5$, $F_7=13$ với tổng bằng 18.

11.2. Hướng dẫn giải thuật.

Ta cần phải tính n số Fibonacci bắt đầu từ số thứ i (viết tắt là F_i). Để thuận tiện khi lập trình, ta sẽ lưu n số Fibonacci, bắt đầu từ số Fibonacci thứ i vào mảng **val**:

val[1]=F_i, val[2]=F_{i+1}, ..., val[n]=F_{i+n-1}. Chú ý các số Fibonacci đã được mod cho k.

+ Với $i \leq 10^6$, ta chỉ cần thực hiện vòng lặp để tính các số Fibonacci chia lấy dư cho k.

+ Với $i \leq 10^{15}$, phải sử dụng kĩ thuật nhân ma trận để tính các số Fibonacci chia lấy dư cho k.

- Sub1: $n \leq 20$, $i \leq 10^6$

Dùng vòng lặp để tính các số Fibonacci chia lấy dư cho k: O(i)

Duyệt các dãy con liên tiếp các số Fibonacci tính từ số thứ i, với mỗi dãy con đó tính tổng các số trong dãy, nếu gặp được dãy có tổng chia hết cho k thì dừng. Dùng 3 vòng lặp lồng nhau: 2 biến i, j (i, j:1→n) để duyệt qua vị trí đầu và cuối của dãy con; biến z trong vòng lặp thứ 3 để tính tổng các số Fibo từ vị trí thứ i đến vị trí thứ j: $O(n^3)$

Độ phức tạp: $O(T*(i+n^3))$ với T là số lượng test.

- Sub2: $n \leq 10^3$, $i \leq 10^6$

Dùng vòng lặp để tính các số Fibonacci chia lấy dư cho k: O(i)

Dùng mảng S tính tổng dồn của các số Fibonacci. Dùng 2 vòng lặp lồng nhau: 2 biến i, j (i, j:1→n) để duyệt qua vị trí đầu và cuối của dãy con; tính tổng các số trong mỗi dãy

con dựa vào mảng tính tổng dồn, sau đó kiểm tra điều kiện. Nếu gặp một dãy con nào đó thỏa mãn thì dừng vòng lặp ngay.

Độ phức tạp: $O(T*(i+n^2))$ với T là số lượng test.

- Sub3: $n \leq 10^6$, $i \leq 10^6$

Dùng vòng lặp để tính các số Fibonacci chia lấy dư cho k: O(i)

Dùng mảng S tính tổng dồn của các số Fibonacci. Nhưng ở đây ta phải sử dụng một thuật toán tối ưu hơn thuật toán ở sub2. Đó là dựa vào *định lí Dirichle.*

Mảng S sẽ có n + 1 phần tử từ 0 đến n, mà có dữ kiện $k \leq n$ nên theo nguyên lí Dirichle thì trong mảng S chắc chắn có 2 số S_p, S_q có cùng số dư cho k (hay tổng $S_{p+1}+...S_q$ chia hết cho k). Vậy các phần tử ở vị trí từ p+1 tới q trong mảng val sẽ chia hết cho k. Nên sẽ đưa ra được vị trí của các phần tử đó trong dãy Fibonacci ban đầu.

Sử dụng kĩ thuật đánh dấu các giá trị tổng dồn đã có vào mảng index, khi tạo ra một giá trị tổng dồn mới, kiểm tra trong mảng đánh dấu đã có chưa, nếu có rồi thì in ra kết quả và kết thúc.

```
memset(index, -1, sizeof index);//mảng đánh dấu
for (int i=0;i<n+1;i=i+1){
//mảng tổng dồn là mảng sum
//nếu chưa có giá trị sum[i] thì đánh dấu chỉ số, ngược lại thì in ra kết
//quả rồi kết thúc
 if (index[sum[i]] < 0) index[sum[i]] = i;
else { int t = index[sum[i]];
        //in ra kết quả
         cout << i - t;
         FOR(j=t+1;j<=i; j++) cout << " " << s + j - 1;
         cout << endl;
         return; } }
```

Độ phức tạp: O(T*(i+n)) với T là số lượng test.

- Sub4: $n \leq 20$, $i \leq 10^{15}$

Làm tương tự sub1 nhưng phải sử dụng kĩ thuật nhân ma trận để tính các số Fibonacci chia lấy dư cho k.

Độ phức tạp: $O(T*(\log(i)+n^3))$ với T là số lượng test.

- Sub5: $n \leq 10^3$, $i \leq 10^{15}$

Làm tương tự sub2 nhưng phải sử dụng kĩ thuật nhân ma trận để tính các số Fibonacci chia lấy dư cho k.

Độ phức tạp: O(T*(log(i)+n^2)) với T là số lượng test.

- Sub6: $n \leq 10^6$, $i \leq 10^{15}$

Làm tương tự sub3 nhưng phải sử dụng kĩ thuật nhân ma trận để tính các số Fibonacci chia lấy dư cho k.

Độ phức tạp: O(T*(log(i)+n)) với T là số lượng test.

11.3. Chương trình.

```
#include<bits/stdc++.h>
#define FOR(i,a,b) for (int i=(a),_b=(b);i<=_b;i=i+1)
#define FORD(i,b,a) for (int i=(b),_a=(a);i>=_a;i=i-1)
#define REP(i,n) for (int i=0,_n=(n);i<_n;i=i+1)
using namespace std;
int MOD;
struct Matrix {  //Cấu trúc ma trận
   int m, n, d[3][3];
   Matrix(int m = 0, int n = 0) {
      this->m = m;
      this->n = n;
      memset(d, 0, sizeof d);
   }
   Matrix operator * (const Matrix &a) const { //PHEP NHAN HAI MA TRAN
      int x = m;
      int y = n;
      int z = a.n;
      Matrix res(x, z);
      REP(i, x) REP(j, y) REP(k, z)
        res.d[i][k] = (res.d[i][k] + 1LL * d[i][j] * a.d[j][k]) % MOD;
      return res;
   }
   Matrix operator ^ (long long k) const { //TINH MA TRAN LUY THUA K
      Matrix res(n, n);
      Matrix mul = *this;
      REP(i, n) REP(j, n) res.d[i][j] = i == j ? 1 : 0;
      while (k > 0) {
         if (k & 1) res = res * mul;
         mul = mul * mul;
         k >>= 1;
      }
      return res;
   }
};
int getFib(long long k, int mod) { // TIM SO FIBO THU K, CHIA LAY DU CHO mod
   MOD = mod;
```

```
    Matrix res(1, 2);
    REP(i, 2) res.d[0][i] = i;
    Matrix mul(2, 2);
    mul.d[1][0] = mul.d[0][1] = mul.d[1][1] = 1;
    mul.d[0][0] = 0;
    res = res * (mul ^ k);
    return res.d[0][0];
}
#define MAX   1000100
int val[MAX], sum[MAX], index[MAX];
//Tim day cac so Fibo lien tiep, Trong n Fibo bat dau tu so thu s ma chia het cho mod
void solve(long long s, int n, int mod) {
//tinh va gan cho val[1]=F[i] , val[2]=F[i+1]
REP(i, 2) val[i + 1] = getFib(s + i, mod);
FOR(i, 3, n) { //tinh tiep mang val
        val[i] = val[i - 2] + val[i - 1];
        if (val[i] >= mod) val[i] -= mod;
    }
FOR(i, 1, n) { //tinh mang tong don: sum
        sum[i] = sum[i - 1] + val[i];
        if (sum[i] >= mod) sum[i] -= mod;
    }
    memset(index, -1, sizeof index);  //mang danh dau
    //neu tim thay tong trong mang tong don thi ket thuc, nguoc lai thi danh dau lai
    REP(i, n + 1) {
        if (index[sum[i]] < 0) index[sum[i]] = i;
        else {
            int t = index[sum[i]];
            cout << i - t;
            FOR(j, t + 1, i) cout << " " << s + j - 1;
            cout << endl;
            return;
        }
    }
    assert(false);
}
int main(void) {
    freopen("fibseq.inp", "r", stdin);
    freopen("fibseq.out", "w", stdout);
    int t;
    cin >> t;
    REP(sotest, t) {
        int n, mod;
        long long s; cin >> n >> s >> mod;
        solve(s, n, mod);
    }
    return 0;
}
```

11.4. Độ phức tạp.

Bài toán được cài đặt với độ phức tạp: O(T*(log(i)+n)) với T là số lượng test.

11.5. Test.

https://drive.google.com/file/d/1msUN5jJjvkcIYlAGuIjiy6ZhwcDyjqo3/view?usp=sharing

12. Bài 12: Bài toán SEQ - Recursive Sequence.

(Nguồn https://www.spoj.com/problems/SEQ/)

12.1. Đề bài.

Cho dãy (ai) các số tự nhiên được định nghĩa như sau:

$a_i = b_i$ (với $i <= k$)

$a_i = c_1a_{i-1} + c_2a_{i-2} + ... + c_ka_{i-k}$ (với $i > k$)

với b_i và c_i là các số tự nhiên cho trước ($1<=j<=k$). Hãy tính a_n modulo 10^9, *với* n cho trước.

Input:

Dòng đầu tiên chứa số c là số lượng test (1<=c<=1000)

Mỗi test chứa 4 dòng:

Dòng 1: chứa số k, là số phần tử của dãy b và c ($1 <= k <= 10$)
Dòng 2: chứa các số $b_1,...,b_k$ với $0 <= b_j <= 10^9$, mỗi số cách nhau một dấu cách.
Dòng 3:chứa các số $c_1,...,c_k$ với $0 <= c_j <= 10^9$, mỗi số cách nhau một dấu cách.

Dòng 4: chứa số n ($1 <= n <= 10^9$)

Output: Gồm c dòng, mỗi dòng là kết quả của một test, ghi giá trị: a_n modulo 10^9

Ví dụ:

SEQ.INP	SEQ.OUT

3 3 5 8 2 32 54 6 2 3 1 2 3 4 5 6 6 3 24 354 6 56 57 465 98765432	8 714 257599514

12.2. Hướng dẫn giải thuật.

+ Với i<=k thì $a_i = b_i$, nên kết quả là b_i % 10^9.

+ Ngược lại ta có: $a_i = c_1a_{i-1} + c_2a_{i-2} + ... + c_ka_{i-k}$

Ta phân tích như sau

$a_{i-k+1} = 0.a_{i-1} + 0.a_{i-2} + 0.a_{i-3} + ... + 1.a_{i-k+1} + 0.a_{i-k}$

.....

.....

$a_{i-1} = 1.a_{i-1} + 0.a_{i-2} + 0.a_{i-3} + ... + 0.a_{i-k+1} + 0.a_{i-k}$

$a_i = c_1.a_{i-1} + c_2.a_{i-2} + c_3.a_{i-3} + ... + c_{k-1}.a_{i-k+1} + c_k.a_{i-k}$

Nên ta có:

$$\begin{bmatrix} a_{i-k+1} \\ ..... \\ ..... \\ a_{i-1} \\ a_i \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & ... & 1 & 0 \\ . & . & . & ... & . & . \\ . & . & . & ... & . & . \\ 1 & 0 & 0 & ... & 0 & 0 \\ c_1 & c_2 & c_3 & ... & c_{k-1} & c_k \end{bmatrix} * \begin{bmatrix} a_{i-k} \\ ..... \\ ..... \\ a_{i-2} \\ a_{i-1} \end{bmatrix} = T * \begin{bmatrix} a_{i-k} \\ ..... \\ ..... \\ a_{i-2} \\ a_{i-1} \end{bmatrix}$$

Với T= $\begin{bmatrix} 0 & 0 & 0 & ... & 1 & 0 \\ . & . & . & ... & . & . \\ . & . & . & ... & . & . \\ 1 & 0 & 0 & ... & 0 & 0 \\ c_1 & c_2 & c_3 & ... & c_{k-1} & c_k \end{bmatrix}$

Ngoài ra $\begin{bmatrix} a_1 \\ a_2 \\ .... \\ a_{k-1} \\ a_k \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ .... \\ b_{k-1} \\ b_k \end{bmatrix}$

Nên $\begin{bmatrix} a_{n-k+1} \\ ..... \\ ..... \\ a_{n-1} \\ a_n \end{bmatrix} = T^{n-k} * \begin{bmatrix} b_1 \\ b_2 \\ .... \\ b_{k-1} \\ b_k \end{bmatrix}$

Đặt res=T^{n-k} * I, với I= $\begin{bmatrix} b_1 \\ b_2 \\ .... \\ b_{k-1} \\ b_k \end{bmatrix}$

Vậy a_n= res.val[k,1].

12.3. Chương trình.

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 15;
const int MODULO = 1000000000;
struct matrix //KHAI BAO CAU TRUC MA TRAN
{
    int rowSize = MAX - 1;
    int columnSize = MAX - 1;
```

```
    int data[MAX][MAX];
    matrix () {
        for (int i = 1; i <= rowSize; i ++) {
            for (int j = 1; j <= columnSize; j ++)
                {
                data[i][j] = 0;
                }   }
            }
    matrix operator * (const matrix other) const{ //PHEP NHAN MA TRAN
        assert(columnSize == other.rowSize);
        matrix result = matrix();
        for (int i = 1; i <= rowSize; i ++) {
            for (int j = 1; j <= other.columnSize; j ++) {
                for (int k = 1; k <= columnSize; k ++) {
                    result.data[i][j] = (result.data[i][j] + 1LL * data[i][k] * other.data[k][j]
% MODULO) % MODULO;
                }
            }
        }
        result.rowSize = rowSize;
        result.columnSize = other.columnSize;
        return result;
    }
    matrix power(int e) {  //TINH LUY THUA e CUA MA TRAN
        if (e == 1) {
            return *this;
        }
        if (e & 1) {
            return (*this) * power(e - 1);
        }
        matrix half = power(e >> 1);
        return half * half;
    }
};
int T, N, K;
int b[MAX], c[MAX];
int main() {
    cin >> T;
    for (int cases = 0; cases < T; cases ++) {
        cin >> K;
        for (int i = 1; i <= K; i ++) {
            cin >> b[i];
        }
        for (int i = 1; i <= K; i ++) {
            cin >> c[i];
        }
        cin >> N;
        if (N <= K) {
```

```
            cout << b[N] << endl;
            continue;
        }
        matrix t, I;
        t.rowSize = K;
        t.columnSize = K;
        for (int i = 2; i <= K; i ++) {
            t.data[i - 1][i] = 1;
        }
        for (int i = 1; i <= K; i ++) {
            t.data[K][i] = c[K - i + 1];
        }
        I.rowSize = K;
        I.columnSize = 1;
        for (int i = 1; i <= K; i ++) {
            I.data[i][1] = b[i];
        }
        matrix result = t.power(N - K) * I;
        cout << result.data[K][1] << endl;
    }
    return 0;
}
```

12.4. Độ phức tạp.

$O(c * K^3 * log(N))$, *với c là số test.*

12.5. Test.

https://drive.google.com/file/d/1aMQBQJ8Fc-1XLBa7odaCa41OSzkishQi/view?usp=sharing

13. Bài 13: Bài toán NKBITI.

(Nguồn https://infoarena.ro/problema/nkbiti)

13.1. Đề bài.

Có bao nhiêu dãy *N* bit với tối đa *K* bit *0* liên tiếp?

Input: nkbiti.inp.

Chứa một dòng với hai số tự nhiên *N* và *K* cách nhau bằng dấu cách.

Output: nkbiti.out.

Kết quả tìm được modulo *666777*.

Hạn chế:

- $1 \le N \le 1\ 000\ 000\ 000$
- $1 \le K \le 40$
- 40% test có $N \le 100\ 000$

- 60% test có $N \le 6\ 000\ 000$

Ví dụ:

nkbiti.inp	nkbiti.out
4 2	13

13.2. Hướng dẫn giải thuật.

(Tham khảo thuật toán tại http://codeforces.com/topic/21274/en83)

Gọi $D_{n,k}$ là số lượng dãy nhị phân độ dài n với k bít 0 ở cuối. Với mỗi một vị trí trong dãy ta có 2 cách chọn là bít 0 hoặc bít 1. Từ trạng thái (n,k) ta có thể di chuyển tới trạng thái (n+1,k+1) và (n+1,0). Nên $D_{n,0}=\sum_{i=0}^{k} D_{n-1,i}$ và $D_{n,k} = D_{n-1,k-1}$

Nên ta sẽ có:

$D_{n,0}=\sum_{i=0}^{k} D_{n-1,i}$

$D_{n,1} = D_{n-1,0}$

$D_{n,2} = D_{n-1,1}$

.....

$D_{n,k} = D_{n-1,k-1}$

Hay là:

$D_{n,0} = D_{n-1,0}+D_{n-1,1}+ D_{n-1,2} +.....+D_{n-1,k-1}+D_{n-1,k}$

$D_{n,1} = 1.D_{n-1,0}+0.D_{n-1,1}+ 0.D_{n-1,2}+.....+0.D_{n-1,k-1}+0.D_{n-1,k}$

$D_{n,2} = 0.D_{n-1,0}+1.D_{n-1,1}+ 0.D_{n-1,2} +.....+0.D_{n-1,k-1}+0.D_{n-1,k}$

$D_{n,3} = 0.D_{n-1,0}+0.D_{n-1,1}+ 1.D_{n-1,2}+..... +0.D_{n-1,k-1}+0.D_{n-1,k}$

.....

$D_{n,k} = 0.D_{n-1,0}+0.D_{n-1,1}+ 0.D_{n-1,2}+.....+1.D_{n-1,k-1}+0.D_{n-1,k}$

Suy ra:

$$\begin{bmatrix} D_{n,0} \\ D_{n,1} \\ D_{n,2} \\ D_{n,3} \\ ..... \\ D_{n,k} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & ..... & 1 & 1 \\ 1 & 0 & 0 & ..... & 0 & 0 \\ 0 & 1 & 0 & ..... & 0 & 0 \\ 0 & 0 & 1 & ..... & 0 & 0 \\ . & . & . & ..... & . & . \\ 0 & 0 & 0 & ..... & 1 & 0 \end{bmatrix} * \begin{bmatrix} D_{n-1,0} \\ D_{n-1,1} \\ D_{n-1,2} \\ D_{n-1,3} \\ ..... \\ D_{n-1,k} \end{bmatrix}$$, đặt $T=\begin{bmatrix} 1 & 1 & 1 & ..... & 1 & 1 \\ 1 & 0 & 0 & ..... & 0 & 0 \\ 0 & 1 & 0 & ..... & 0 & 0 \\ 0 & 0 & 1 & ..... & 0 & 0 \\ . & . & . & ..... & . & . \\ 0 & 0 & 0 & ..... & 1 & 0 \end{bmatrix}$, T là ma trận vuông kích thước k+1.

Biến đổi ta sẽ có:

$$\begin{bmatrix} D_{n,0} \\ D_{n,1} \\ D_{n,2} \\ D_{n,3} \\ ..... \\ D_{n,k} \end{bmatrix} = T^{n-k} * \begin{bmatrix} D_{k,0} \\ D_{k,1} \\ D_{k,2} \\ D_{k,3} \\ ..... \\ D_{k,k} \end{bmatrix}$$

Tính được $D_{k,k}=1$; $D_{k,k-1}=2$; $D_{k,k-2}=2^2$;$D_{k,k-3}=2^3$; .....

Và từ đó tính được kết quả của bài toán.

13.3. Chương trình.

```
#include <iostream>
#include <fstream>
#define MOD 666777
using namespace std;
ifstream f("nkbiti.inp");
ofstream g("nkbiti.out");
long long  a[43][43],mask[43][43],sol[43][43];
int n,k;
void Nhan_matran(long long  a[43][43],long long  b[43][43])
{
   long long c[43][43]={};
   for(int i=1;i<=k;i++)
      for(int j=1;j<=k;j++)
      {
         for(int q=1;q<=k;q++)
            c[i][j]=(c[i][j]+ (a[i][q]*b[q][j] ) %MOD ) %MOD;
      }
   for(int i=1;i<=k;i++)
      for(int j=1;j<=k;j++)
         a[i][j]=c[i][j];
}
void power( long long a[43][43],int n)// Tinh ma tran luy thua n
{
   long long aux[43][43]={};
   for(int i=0;i<=k;i++)
      aux[i][i]=1;
   while(n)
   {
      if(n%2)
      Nhan_matran(aux,a);
      Nhan_matran(a,a);
      n/=2;
   }
  for(int i=1;i<=k;i++)
    for(int j=1;j<=k;j++)
```

```
        a[i][j]=aux[i][j];
}
int main()
{
   f>> n >> k;
   long long  pw=1;
   k++;
   for(int i=1;i<=k;i++)
   {
      a[i][1]=pw;
      pw=(pw*2)%MOD;
   }
   for(int i=1;i<=k;i++)
   {
      mask[k][i]=1;
      mask[i][i+1]=1;
   }
   power(mask,n-k+1);
   Nhan_matran(mask,a);
   int sum=0;
   sum=mask[k][1];
   g << sum % MOD;
   return 0;
}
```

13.4. Độ phức tạp.

$O(k^3 \log(n))$

13.5. Test.

https://drive.google.com/file/d/1ZktnSNBbcizg5Xbul5Zk5OZfOE0t4ULJ/view?usp=sharing

Chú ý: Bài tương tự nhưng kích thước k, n nhỏ; có thể tham khảo đề tại: http://vn.spoj.com/problems/SPBINARY/. Thuật giải cho bài này tham khảo trong KCBOOK3, trang 8.

14. Bài 14: Bài toán 852B - *Neural Network country*

(Nguồn: https://codeforces.com/contest/852/problem/B)

14.1. Đề bài.

Do sự phổ biến gần đây của Deep learning các quốc gia mới đang bắt đầu trông giống như mạng thần kinh. Đó là, các nước đang được xây dựng với nhiều lớp, mỗi lớp có thể có nhiều thành phố. Mỗi nước cũng có một điểm đầu và một điểm cuối.

Mỗi nước có chính xác L lớp, mỗi lớp có N thành phố. Quan sát hai lớp kề nhau L1 và L2. Mỗi thành phố thuộc lớp L1 được nối với một thành phố thuộc lớp L2 với một chi phí di chuyển là c_{ij} với mọi i, j,$\in$ {1, 2, ..., N} và mỗi cặp lớp lân cận có cùng chi phí

ở giữa các thành phố của chúng với nhau như bất kỳ cặp nào khác. Chi phí di chuyển của mọi thành phố thuộc lớp L1 đến một thành phố thuộc lớp L2 là giống nhau, nghĩa là c_{ij} là như nhau với mọi i ∈ {1, 2, ..., N} và j cố định.

Yêu cầu: Tìm số lượng đường đi để một người bắt đầu đi tại điểm đầu và kết thúc tại điểm cuối sao cho chi phí đi lại của người đó có thể chia hết cho số M cho trước.

Input:

Dòng đầu chứa số $N\,(1 \le N \le 10^6)$, $L\,(2 \le L \le 10^5)$ và $M\,(2 \le M \le 100)$, là số lượng thành phố trong từng lớp, số lượng lớp và số M là chi phí của đường đi cần phải chia hết.

Dòng thứ hai, ba, bốn chứa N số nguyên (mỗi số biểu thị chi phí di chuyển *cost* mà $0 \le cost \le M$) lần lượt là chi phí tới lớp đầu tiên, chi phí giữa hai lớp kề nhau như mô tả ở trên và chi phí từ lớp cuối cùng tới điểm kết thúc.

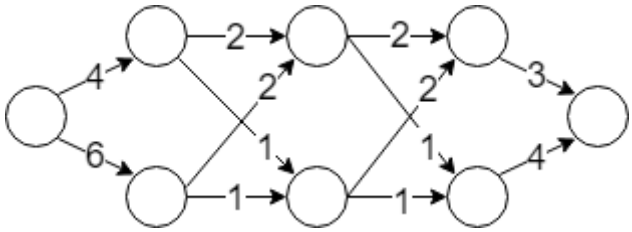
Output:

Chứa một số nguyên là số lượng đường đi mà tổng chi phí chia hết cho M, modulo $10^9 + 7$.

Ví dụ:

852B.INP	852B.OUT
2 3 13 4 6 2 1 3 4	2

Ghi chú: Hình vẽ mô tả test ở trên:



Đây là một đất nước có 3 lớp, mỗi lớp có 2 thành phố. Chỉ có hai đường đi $6 \to 2 \to 2 \to 3$, và $6 \to 2 \to 1 \to 4$ có tổng chi phí chia hết cho 13. Chú ý các cạnh đầu vào cho các thành phố trong một lớp có chi phí như nhau, và giống nhau cho tất cả các lớp.

14.2. Hướng dẫn giải thuật.

(Tham khảo thuật toán tại ***http://bubblecup.org/Content/Media/Booklet2017.pdf)***

Gọi $Dp[i][j]$ là số lượng đường đi đến lớp thứ i mà tổng chi phí đường đi bằng j%M. Ma trận Dp có kích thước L*M. Ta có:

$$\text{Dp[i][j]}=\sum_{k=0}^{N-1} Dp[i-1][(j-w[k])\ \%\ M]$$

Với w là mảng trọng số giữa các lớp. Từ mảng chi phí giữa điểm đầu và lớp đầu tiên, giả sử là mảng a, ta tính toán $Dp[0][a[k]]$ += 1, với mọi k chạy từ 0 đến N-1. Tính kết quả cuối cùng sẽ khó khăn hơn một chút. Để tính kết quả cuối cùng, chúng ta không chỉ cần tính $Dp[L][0..N-1]$, mà còn phải biết từng con đường đó kết thúc tại nút nào. Chúng ta có thể làm bằng cách tính mảng Dp tới lớp thứ L-1, và tính bước cuối một cách thủ công bằng cách gộp chi phí di chuyển của hai tập cạnh cuối cùng, đơn giản là ta sẽ cộng $w[i]$ với $b[i]$, ở đây b là mảng chi phí di chuyển giữa lớp cuối cùng với điểm kết thúc, với mọi i từ 0 đến N-1, và đưa ra kết quả sau khi modulo M:

$$\text{Res} = \sum_{i=0}^{N-1} Dp[L-1][(-w[i]-b[i])\ \%\ M]$$

Cách làm trên với độ phức tạp O(L*N*M), và bộ nhớ tốn O(L*M), quá giới hạn cho phép.

Chúng ta sẽ giảm độ phức tạp bằng cách: Lưu số lượng mỗi w[i[%M vào mảng num[i]. Bây giờ ma trận Dp có thể tính toán trong $O(L*M^2)$:

$$\text{Dp[i][j]}=\sum_{k=0}^{M-1} num[k] * Dp[i-1][(j-k)\ \%\ M]$$

Nhìn vào công thức ta thấy có thể sử dụng phép nhân giữa một ma trận và một vecto. Nhưng chúng ta sẽ chuyển đổi ma trận kích thước M*M thành một vecto thể hiện trạng thái hiện tại (số lượng đường đi có tổng chi phí i=0..M-1, theo modulo M đến lớp hiện tại), thao tác đưa ra vecto trạng thái tiếp theo tương ứng với lớp tiếp theo, kết quả của bài toán được tính trong thời gian $O(M^3 * \log L)$ sử dụng cách tính lũy thừa ma trận. Lớp cuối cùng vẫn phải được xử lí riêng giống như mô tả từ trước.

Cụ thể hơn, có thể xem code dưới đây:

14.3. Chương trình.

```
#include<bits/stdc++.h>
#define md 1000000007
using namespace std;
int N, L, M, P, i, j, S[1000000];
struct depe {
	long long A[100];
} awa, ten, akh, ans, kos;

depe kal(depe X,depe Y) //Phep Nhan
{
	depe ret=kos;
	for (i=0;i<M;i++) {
		for (j=0;j<M;j++) {
			ret.A[(i+j)%M]=(ret.A[(i+j)%M]+X.A[i]*Y.A[j])%md;
```

```
                        }
                    }
        return ret;
}
depe pan(int X) {  //ma tran luy thua X
        if (X==1) {
                return ten;
        }
        depe ret=pan(X/2);
        ret=kal(ret,ret);
        if (X%2) {
                ret=kal(ret,ten);
        }
        return ret;
}
int main ()
{
        ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
        cin>>N>>L>>M;
        for (i=0;i<N;i++) {
                cin>>P;
//mang dem so luong canh noi tu diem dau den lop thứ nhất co trong so theo %M
                awa.A[P%M]++;           }

        for (i=0;i<N;i++) {
                cin>>S[i];  //chi phi di chuyen giua 2 lop
//mang dem so luong canh noi giua 2 lop co trong so theo %M
                ten.A[S[i]%M]++;
        }
        for (i=0;i<N;i++) {
                cin>>P;
//gop 2 tap canh cuoi cung, mang dem so luong canh co trong so theo %M
                akh.A[(P+S[i])%M]++;
        }
        if (L==2) {
                ans=kal(awa,akh);
        }
        else {
                ans=kal(kal(awa,pan(L-2)),akh);
        }
        cout<<ans.A[0]<<'\n'; //A[0]nghia la chia het cho M
}
```

14.4. Độ phức tạp: $O(M^3 \log L)$

14.5. Test:

https://drive.google.com/file/d/1Lpdg74GtTsVFaAScyfXi2RE84-ELvqWg/view?usp=sharing

15. Bài 15: 821E - Okabe and El Psy Kongroo

(Nguồn: https://codeforces.com/contest/821/problem/E)

15.1. Đề bài.

Okabe thích đi bộ nhưng biết rằng gián điệp có thể ở bất cứ đâu; đó chính là lý do tại sao anh ấy muốn biết có bao nhiêu cách khác nhau mà anh ta có thể đi trong thành phố của mình một cách an toàn. Thành phố của Okabe có thể được biểu diễn bởi các điểm (x, y) sao cho x và y không âm. Okabe bắt đầu đi tại điểm gốc (điểm (0, 0)) và cần đi đến điểm (k, 0). Nếu Okabe đang ở điểm (x, y), trong một bước anh ta có thể đi đến (x + 1, y + 1), (x + 1, y), hoặc (x + 1, y - 1).

Ngoài ra, có n đoạn đường ngang, đoạn thứ i từ $x=a_i$ đến $x=b_i$ và ở tung độ $y=c_i$. Luôn đảm bảo rằng $a_1=0$, $a_n \le k \le b_n$ và $a_i=b_{i-1}$ với $2 \le i \le n$. Đoạn đường thứ i buộc Okabe phải đi với giá trị y trong phạm vi $0 \le y \le c_i$, còn giá trị x phải thoả mãn $a_i \le x \le b_i$, nếu không anh ta có thể bị theo dõi. Điều này cũng có nghĩa là anh ta bắt buộc phải ở dưới hai đoạn đường khi một đoạn đường kết thúc và một đoạn khác bắt đầu.

Okabe muốn biết có bao nhiêu cách đi an toàn từ điểm đầu (0,0) tới điểm (k,0), kết quả modulo $10^9 + 7$.

Input:

Dòng đầu chứa hai số nguyên n và k ($1 \le n \le 100$, $1 \le k \le 10^{18}$) là số đoạn đường ngang và tọa độ x của điểm kết thúc.

N dòng tiếp theo, mỗi dòng chứa 3 số nguyên a_i, b_i và c_i ($0 \le a_i < b_i \le 10^{18}$, $0 \le c_i \le 15$) tương ứng là hoành độ đầu mút trái, hoành độ đầu mút phải, tung độ của một đoạn đường ngang.

Dữ liệu luôn đảm bảo rằng $a_1 = 0$, $a_n \le k \le b_n$, và $a_i = b_{i-1}$ với $2 \le i \le n$.

Output:

Ghi số lượng cách đi thỏa mãn điều kiện, kết quả modulo $10^9 + 7$.

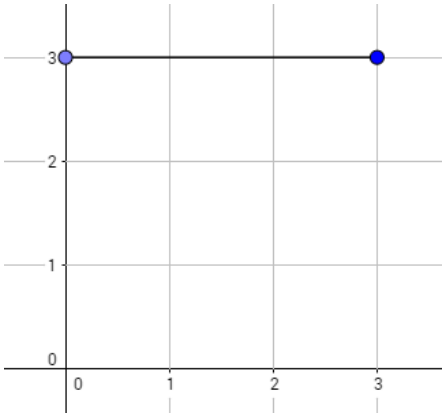
Ví dụ 1:

Okabe.inp	Okabe.out
1 3 0 3 3	4

Ví dụ 2:

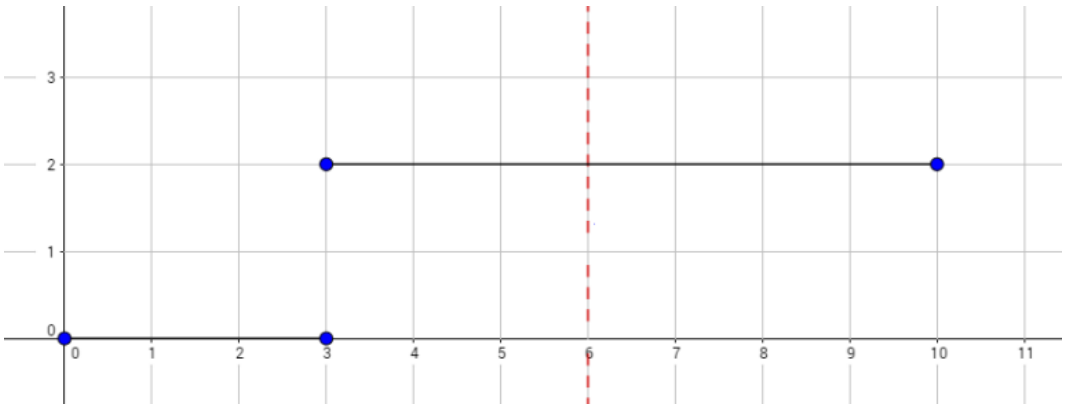
Okabe.inp	Okabe.out
2 6 0 3 0 3 10 2	4

Chú thích:



Đồ thị trên tương ứng với ví dụ 1. Các cách đi có thể là:

- $(0,0) \to (1,0) \to (2,0) \to (3,0)$
- $(0,0) \to (1,1) \to (2,0) \to (3,0)$
- $(0,0) \to (1,0) \to (2,1) \to (3,0)$
- $(0,0) \to (1,1) \to (2,1) \to (3,0)$



Đồ thị trên tương ứng với ví dụ 2. Ở đây chỉ có thể nghiên cứu cách đi từ $(3,0)$. Sau đó, các cách đi có thể là:

- $(3,0) \to (4,0) \to (5,0) \to (6,0)$
- $(3,0) \to (4,0) \to (5,1) \to (6,0)$
- $(3,0) \to (4,1) \to (5,0) \to (6,0)$
- $(3,0) \to (4,1) \to (5,1) \to (6,0)$

15.2. Hướng dẫn thuật toán.

Thuật toán đơn giản dễ nhận ra là quy hoạch động. Gọi f(x,y) là số lượng cách đi tới điểm (x, y). Thì f(x, y)= $f(x-1, y-1) + f(x-1, y) + f(x-1, y+1)$, cẩn thận với các đoạn đường ngang.

Để tăng tốc độ của thuật toán ta sẽ sử dụng kĩ thuật nhân ma trận. Vì x rất lớn trong khi y lại nhỏ nên ta sẽ biểu diễn trạng thái hiện tại theo y ($0 \le y \le 15$).

Mỗi một đoạn đường ngang với hoành độ đầu mút trái, hoành độ đầu mút phải, tung độ của đoạn đường ngang tương ứng là l, r, c ta sẽ xây dựng một ma trận như sau:

f(x,0)=f(x-1,0) + f(x-1,1)

f(x,1)=f(x-1,0)+f(x-1,1)+f(x-1,2)

f(x,2)=f(x-1,1)+f(x-1,2)+f(x-1,3)

.....

f(x,c)=f(x-1,c-1)+f(x-1,c) (vì không được đi lên trên đoạn đường ngang nên không có f(x-1,c+1), với $0 \le c \le 15$)

nên suy ra:

$$\begin{bmatrix} f(x,0) \\ f(x,1) \\ f(x,2) \\ ..... \\ f(x,c) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & ... & 0 & 0 \\ 1 & 1 & 1 & .... & 0 & 0 \\ 0 & 1 & 1 & .... & 0 & 0 \\ . & . & . & . & . & . \\ . & . & . & . & 1 & 1 \end{bmatrix} * \begin{bmatrix} f(x-1,0) \\ f(x-1,1) \\ f(x-1,2) \\ ..... \\ f(x-1,c) \end{bmatrix}$$

Đặt A= $\begin{bmatrix} 1 & 1 & 0 & ... & 0 & 0 \\ 1 & 1 & 1 & .... & 0 & 0 \\ 0 & 1 & 1 & .... & 0 & 0 \\ . & . & . & . & . & . \\ . & . & . & . & 1 & 1 \end{bmatrix}$ ma trận gồm c+1 dòng, c+1 cột; dòng thứ i (0<i<15) sẽ có các cột i-1, i, i+1 có giá trị 1; chỉ số dòng và cột của ma trận tính từ 0.

Ta cần tính số cách đi từ điểm đầu x=r đến điểm cuối x=l nên phải tính $\mathbf{A^{r-l}}$

Khởi tạo ma trận kết quả ban đầu **ans** là ma trận vuông kích thước 16*16 vì $0 \le c_i \le 15$, trong đó các phần tử trên đường chéo chính bằng 1.

Khi duyệt qua từng đoạn đường ngang, ta tính: ans=ans* $\mathbf{A^{r-l}}$**.** Ma trận là kết quả của đoạn đường ngang trước được dùng để nhân với ma trận $\mathbf{A^{r-l}}$ của đoạn đường ngang sau. Vì trạng thái kết thúc của người đi trên đoạn đường ngang trước lại là trạng thái bắt đầu đi trên đoạn đường ngang sau. Cứ làm như vậy khi duyệt hết n đoạn đường ngang.

Sau khi duyệt qua hết n đoạn đường ngang, kết quả cuối cùng là **res.val[0, 0].**

15.3. Chương trình.

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 16, mod = 1e9 + 7;
long long n, k;
struct mtx{  //khai bao cau truc ma tran
	long long a[maxn][maxn];
	mtx() {
		for(int i = 0; i < maxn; i++)
		for(int j = 0; j < maxn; j++)
			a[i][j] = 0;
	}
	mtx operator* (mtx b) { //phep nhan ma tran
		mtx res = mtx();
		for(int i = 0; i < maxn; i++)
		for(int j = 0; j < maxn; j++)
		for(int k = 0; k < maxn; k++)
			res.a[i][j] = (res.a[i][j] + ((a[i][k] * b.a[k][j]) % mod)) % mod;
	return res;
	}
};
mtx bp(mtx a, long long k) { //Tinh ma tran a luy thua k
	mtx res = mtx();
	for(int i = 0; i < maxn; i++)
		res.a[i][i] = 1;
	if(k == 0)
		return res;
	res = (res * bp(a, k / 2));
	res = res * res;
	if(k & 1) res = res * a;
	return res;
}
mtx ans;
int main() {
   freopen("Okabe.inp","r",stdin);
   freopen("Okabe.out","w",stdout);

	for(int i = 0; i < maxn; i++)
		ans.a[i][i] = 1; //khoi tao ma tran ket qua
	cin >> n >> k;
	for(int i = 0; i < n; i++) { //duyet qua tung doan duong ngang
	    long long l, r, c;
	   cin >> l >> r >> c;
	   l = min(l, k);
	   r = min(r, k);
	  mtx cur = mtx(); //cur: ma tran ung voi doan duong ngang thu i
```

```
            for(long long j = 0; j <= c; j++) //từ tung độ j có thể đi tới các tung độ  j-1, j, j+1
                for(long long f = max(0LL, j - 1); f <= min(j + 1, c); f++)
                        cur.a[j][f] = 1;
            cur = bp(cur, r - l);
            ans = ans * cur;//lan luot nhan cac ma tran thanh phan ung voi tung doan duong ngang
        }
        cout << ans.a[0][0];
        return 0;
    }
```

15.4. Độ phức tạp.

O(n*h^3*log(k) với n là số đoạn đường ngang, h=16, k là tọa độ điểm đích.

15.5. Test.

https://drive.google.com/file/d/13u53lZATzcZ3DFKosyYlGaxkabL44vYI/view?usp=sharing

16. Hướng dẫn thuật toán của một số bài toán tham khảo khác

16.1. Bài toán INKPRINT - Mực in

Đề bài: https://vn.spoj.com/problems/INKPRINT/

Thuật toán:

Thuật toán trâu, được 30 điểm:

Gọi f[i] là số lượng các số có thể viết ra khi dùng i đơn vị mực in

->f[i]=$\sum$ f[i − m[j]] với 1≤i≤s, 1≤j≤n

Kết quả bài toán : f[s]

Thuật toán 100 điểm: Sử dụng nhân ma trận 26×26 và xử lí số lớn.

Theo thuật toán ở trên ta có: f[i]=$\sum$ f[i − m[j]] với 1≤i≤s, 1≤j≤n

Vì 1≤m[j]≤n và n≤26 nên thay vì biểu diễn f[i] như trên thì ta biểu diễn f[i] thành:

f[i]=$\sum$ k ∗ f[i − j] với 1≤j≤26, k=0 hoặc 1.

k=0 nếu không có bất cứ giá trị m[t] (1≤t≤n) nào bằng với j.

k=1 trong trường hợp ngược lại.

Đầu tiên ta sẽ phân tích để tính ra công thức tổng của f[i] theo cách như trên, giả sử phân tích được:

f[i]=f[i-1]+f[i-2]+.....+f[i-25]+f[i-26]

thì ta sẽ tìm ma trận như sau:

f[i] =1*f[i-1]+1*f[i-2]+.....+1*f[i-25]+1*f[i-26]

f[i-1]= 1*f[i-1]+0*f[i-2]+.....+0*f[i-25]+0*f[i-26]

f[i-2]= 0*f[i-1]+1*f[i-2]+.....+0*f[i-25]+0*f[i-26]

.....

f[i-25]= 0*f[i-1]+0*f[i-2]+.....+1*f[i-25]+0*f[i-26]

Suy ra:

$$\begin{bmatrix} f[i] \\ f[i-1] \\ .f[i-2] \\ ..... \\ f[i-25] \end{bmatrix} = \begin{bmatrix} 1 & 1 & .... & 1 & 1 \\ 1 & 0 & .... & 0 & 0 \\ 0 & 1 & .... & 0 & 0 \\ . & . & . & . & . \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} f[i-1] \\ f[i-2] \\ .f[i-3] \\ ..... \\ f[i-26] \end{bmatrix} = T^i * \begin{bmatrix} f[1] \\ f[2] \\ .f[3] \\ ..... \\ f[26] \end{bmatrix}$$

$$\text{Với } T = \begin{bmatrix} 1 & 1 & .... & 1 & 1 \\ 1 & 0 & .... & 0 & 0 \\ 0 & 1 & .... & 0 & 0 \\ . & . & . & . & . \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Các giá trị từ f[1] tới f[26] ta sẽ tính theo thuật toán trâu, tính kết quả bài toán f[s] theo cách nhân ma trận, có xử lí số lớn.

16.2. Bài toán QTLOVE2 - Hoa hướng dương

Đề bài: https://vn.spoj.com/problems/QTLOVE2/

Thuật toán: Sử dụng quy hoạch động kết hợp nhân ma trận:

Gọi f[i][1] là số cách tô i cánh, trong đó màu của cánh hoa thứ i khác với màu cánh hoa thứ nhất và màu cánh hoa trước nó, và f[i][2] là số cách tô i cánh hoa, trong đó màu của cánh hoa thứ i khác màu cánh hoa trước nó và giống màu cánh hoa thứ nhất.

->Cơ sở quy hoạch động: f[i][1]=0, f[i][2]=m.

Công thức quy hoạch động:

f[i][1]=(m-2)*f[i-1][1]+(m-1)*f[i-1][2] (1)

f[i][2]=f[i-1][1]. (2)

Kết quả: f[n][1].

Do N<=10^{18} nên phải sử dụng nhân ma trận để giải quyết bài toán trong thời gian O(log(n)).

Từ (1) và (2) ta có: f[i][1]=(m-2)*f[i-1][1]+(m-1)*f[i-2][1]

Ta xây dựng ma trận như sau:

f[i][1]=(m-2)*f[i-1][1]+(m-1)*f[i-2][1]

f[i-1][1]=1*f[i-1][1]+0*f[i-2][1]

$\Rightarrow \begin{bmatrix} f[i][1] \\ f[i-1][1] \end{bmatrix} = \begin{bmatrix} m-2 & m-1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} f[i-1][1] \\ f[i-2][1] \end{bmatrix}$, đặt $T=\begin{bmatrix} m-2 & m-1 \\ 1 & 0 \end{bmatrix}$

Nên $\begin{bmatrix} f[n][1] \\ f[n-1][1] \end{bmatrix} = T^{n-2} * \begin{bmatrix} f[2][1] \\ f[1][1] \end{bmatrix} = T^{n-2} * \begin{bmatrix} m(m-1) \\ 0 \end{bmatrix}$ (vì f[1][1]=0, f[2][1]=m*(m-1))

Từ đó ta tính được f[n][1].

16.3. Bài toán VOSTRIBO - Tribonacci

Đề bài: https://vn.spoj.com/problems/VOSTRIBO/

Thuật toán: Sử dụng quy hoạch động kết hợp nhân ma trận:

Từ F(n-1) = T(1) + T(2) + T(3) + ... + T(n-1) + T(n-1).

Và T(n) = T(n-1) + T(n-2) + T(n-3)

Nên ta có: F[n]=F[n-1]+T[n-1]+T[n-2]+T[n-3]

Ta xây dựng ma trận như sau:

F[n] =1*F[n-1]+1*T[n-1]+1*T[n-2]+1*T[n-3]

T[n] =0*F[n-1]+1*T[n-1]+1*T[n-2]+1*T[n-3]

T[n-1]=0*F[n-1]+1*T[n-1]+0*T[n-2]+0*T[n-3]

T[n-2]=0*F[n-1]+0*T[n-1]+1*T[n-2]+0*T[n-3]

Suy ra:

$$\begin{bmatrix} F[n] \\ T[n] \\ T[n-1] \\ T[n-2] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} F[n-1] \\ T[n-1] \\ T[n-2] \\ T[n-3] \end{bmatrix}, \text{ đặt } T=\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Nên $\begin{bmatrix} F[n] \\ T[n] \\ T[n-1] \\ T[n-2] \end{bmatrix} = T^{n-3} * \begin{bmatrix} F[3] \\ T[3] \\ T[2] \\ T[1] \end{bmatrix}$, mà T[1]=1, T[2]=2, T[3]=3, F[3]=6

Do đó $\begin{bmatrix} F[n] \\ T[n] \\ T[n-1] \\ T[n-2] \end{bmatrix} = T^{n-3} * \begin{bmatrix} 6 \\ 3 \\ 2 \\ 1 \end{bmatrix}$

Cuối cùng ta sẽ tính được F[n].

16.4. Bài toán FBRICK – Xếp Hình

Đề bài: https://vn.spoj.com/problems/FBRICK/

Thuật toán: Sử dụng quy hoạch động kết hợp nhân ma trận:

Gọi T[k] là tổng thể tích của k hình lăng trụ từ hình 1 đến hình thứ k.

Ta có T[k]=$\sum_{i=1}^{k} A[i]^2$

Vậy ta cần tìm T[n].

Ta có $T[n]=T[n-1]+A[n]^2=T[n-1]+(2A[2]*A[n-1]-A[n-2])^2$

$=T[n-1]+4*A[2]^2*A[n-1]^2+A[n-2]^2-4A[2]*A[n-1]*A[n-2]$ (1)

$A[n]^2=(2A[2]*A[n-1]-A[n-2])^2=4*A[2]^2*A[n-1]^2+A[n-2]^2-4A[2]*A[n-1]*A[n-2]$

$= 0*T[n-1] + 4*A[2]^2*A[n-1]^2+A[n-2]^2-4A[2]*A[n-1]*A[n-2]$ (2)

$A[n-1]^2=0*T[n-1] + 1*A[n-1]^2 +0*A[n-2]^2 + 0*A[n-1]*A[n-2]$ (3)

$A[n]*A[n-1]= (2A[2]*A[n-1]-A[n-2])*A[n-1]$

$=2A[2]*A[n-1]^2 –A[n-1]*A[n-2]$

$=0*T[n-1] + 2A[2]*A[n-1]^2 +0*A[n-2]^2 -1*A[n-1]*A[n-2]$ (4)

Từ (1), (2), (3) và (4) ta có:

$$\begin{bmatrix} T[n] \\ A[n]^2 \\ A[n-1]^2 \\ A[n]*A[n-1] \end{bmatrix} = \begin{bmatrix} 1 & 4A[2]^2 & 1 & -4A[2] \\ 0 & 4A[2]^2 & 1 & -4A[2] \\ 0 & 1 & 0 & 0 \\ 0 & 2A[2] & 0 & -1 \end{bmatrix} * \begin{bmatrix} T[n-1] \\ A[n-1]^2 \\ A[n-2]^2 \\ A[n-1]*A[n-2] \end{bmatrix}$$

Đặt $T=\begin{bmatrix} 1 & 4A[2]^2 & 1 & -4A[2] \\ 0 & 4A[2]^2 & 1 & -4A[2] \\ 0 & 1 & 0 & 0 \\ 0 & 2A[2] & 0 & -1 \end{bmatrix}$

Ta sẽ có $\begin{bmatrix} T[n] \\ A[n]^2 \\ A[n-1]^2 \\ A[n]*A[n-1] \end{bmatrix} = T^{n-2} * \begin{bmatrix} T[2] \\ A[2]^2 \\ A[1]^2 \\ A[2]*A[1] \end{bmatrix} = T^{n-2} * \begin{bmatrix} A[2]+1 \\ A[2]^2 \\ 1 \\ A[2] \end{bmatrix}$ (vì A[1]=1, T[2]=A[2]+A[1]=A[2]+1)

Cuối cùng ta sẽ tính được T[n].

16.5. Bài toán FLIB - Flibonakki

Đề bài: https://www.spoj.com/problems/FLIB/

Thuật toán: Sử dụng quy hoạch động kết hợp nhân ma trận:

(Tham khảo thuật toán tại https://mukeshiiitm.wordpress.com/2011/06/25/spoj-7487-flibonakki/)

Ta cần biểu diễn mối liên quan giữa phần tử thứ n+1 và n.

Từ g(n) = g(n-1) + f(4n-1) , với n > 0

Ta có g(n+1) = g(n)+f(4n+3)

Mà f(4n+3)=f(4n+2)+f(4n+1)=f(4n+1)+f(4n)+f(4n+1)=2f(4n+1)+f(4n).

Nên g(n+1)=g(n)+ 2f(4n+1)+f(4n).

Ta xây dựng ma trận như sau:

g(n+1) =g(n)+ 2f(4n+1)+f(4n)

f(4n+5)=0g(n) + 5f(4n+1)+3f(4n) (biến đổi theo biểu thức của dãy fibonacci)

f(4n+4)= 0g(n) + 3f(4n+1)+2f(4n) (biến đổi theo biểu thức của dãy fibonacci)

nên ta có:

$$\begin{bmatrix} g(n+1) \\ f(4n+5) \\ f(4n+4) \end{bmatrix} = \begin{bmatrix} 0 & 2 & 1 \\ 0 & 5 & 3 \\ 0 & 3 & 2 \end{bmatrix} * \begin{bmatrix} g(n) \\ f(4n+1) \\ f(4n) \end{bmatrix}$$

Hay $\begin{bmatrix} g(n+1) \\ f(4*(n+1)+1) \\ f(4*(n+1)) \end{bmatrix} = \begin{bmatrix} 0 & 2 & 1 \\ 0 & 5 & 3 \\ 0 & 3 & 2 \end{bmatrix} * \begin{bmatrix} g(n) \\ f(4n+1) \\ f(4n) \end{bmatrix}$

Nên $\begin{bmatrix} g(n) \\ f(4n+1) \\ f(4n) \end{bmatrix} = \begin{bmatrix} 0 & 2 & 1 \\ 0 & 5 & 3 \\ 0 & 3 & 2 \end{bmatrix} * \begin{bmatrix} g(n-1) \\ f(4*(n-1)+1) \\ f(4*(n-1)) \end{bmatrix}$

Đặt $T = \begin{bmatrix} 0 & 2 & 1 \\ 0 & 5 & 3 \\ 0 & 3 & 2 \end{bmatrix}$

Ta sẽ có $\begin{bmatrix} g(n) \\ f(4n+1) \\ f(4n) \end{bmatrix} = T^n * \begin{bmatrix} g(0) \\ f(1) \\ f(0) \end{bmatrix} = T^n * \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

Cuối cùng ta tính được g(n) là kết quả bài toán.

16.6. Một vài bài toán khác

https://codeforces.com/contest/497/problem/E

https://codeforces.com/contest/107/status/D

https://codeforces.com/problemset/problem/989/E

https://codeforces.com/problemset/problem/954/F

https://codeforces.com/problemset/problem/696/D

http://vn.spoj.com/problems/C11DK2/

http://vn.spoj.com/problems/PA06ANT/

http://vn.spoj.com/problems/C11CAL/

http://codeforces.com/topic/21274/en83

https://codeforces.com/problemset/problem/691/E

.....

PHẦN III: KẾT LUẬN

Trong chuyên đề, tôi đã trình bày về khái niệm, các phép toán trên ma trận và ứng dụng các phép toán đó để giải một số bài toán trong Tin học. Giúp giảm độ phức tạp của bài toán từ O(N) xuống còn O(logN).

Sau khi áp dụng kĩ thuật *Nhân ma trận* vào một số bài toán Tin học, đặc biệt là các bài toán quy hoạch động, tôi thấy nó mang lại hiệu quả rất rõ rệt. Nó thay đổi cách tiếp cận, phương pháp làm các dạng bài quy hoạch động có kích thước lớn, bây giờ chúng ta đã có một phương pháp đơn giản, dễ dàng hơn để thực hiện. Hầu hết giáo viên và học sinh đều đánh giá đây là một phương pháp hay, cần phổ biến rộng, củng cố, luyện tập để nâng cao được chất lượng giảng dạy, học tập, thi cử của học sinh trong các kì thi học sinh giỏi hiện nay.

Tôi nhận thấy cần phải phổ biến rộng kĩ thuật này tới các giáo viên và học sinh trong các nhà trường, đặc biệt là các học sinh giỏi Tin, học sinh chuyên Tin. Nhằm nâng cao được chất lượng giảng dạy, học tập, thi học sinh giỏi môn Tin học.

Do thời gian còn 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 được hoàn thiện hơn.

TÀI LIỆU THAM KHẢO

1. Tài liệu giáo khoa chuyên Tin tập 1, 2, 3.
2. Website: http://vn.spoj.com/
3. Website: http://vnoi.info/wiki/algo/trick/matrix-multiplication
4. Website: https://vi.wikipedia.org/wiki/Ma_trận_(toán_học)
5. Website: https://www.codechef.com
6. Website: http://codeforces.com

MỤC LỤC