

MỤC LỤC ATHENA XVI

Quy hoạch động theo lát cắt doc	4
Nguyên lý chung	4
Bài toán lát hình bằng quân đô mi nô	4
Bài toán trạng trí hoa văn	9
Quy hoạch động theo lát cắt gấp khúc	15
Nguyên lý xây dựng lát cắt	15
Dẫn xuất lát cắt	16
Ví dụ ứng dụng	21
Bài toán	21
GIẢI THUẬT AHO – CORASIK	22
Xây dựng rừng	22
Xây dựng ô tó mát	24
VZ29. TÍNH ĐỒNG NHẤT <i>Tên chương trình: UNIFORMITY.CPP</i>	27
VZ30. LƯỢT ĐI <i>Tên chương trình: PASS.CPP</i>	33
VZ31*. NHẠC NƯỚC <i>Tên chương trình: WMUSIC.CPP</i>	35
VZ32. SIÊU THỊ <i>Tên chương trình: MARKETS.CPP</i>	39
VZ33. IN PHUN 3D <i>Tên chương trình: P3D.CPP</i>	41
VZ34. HÌNH VUÔNG KHÁC NHAU <i>Tên chương trình: DIFFERENT.CPP</i>	43
VZ35. ĐƯỜNG XÓA ÓC <i>Tên chương trình: SPIRAL.CPP</i>	45
VZ36. KHÔNG CÓ ĐIỂM BÁT ĐỘNG <i>Tên chương trình: NOFIX.CPP</i>	48
VZ37. ƯỚC CHUNG LỚN NHẤT <i>Tên chương trình: GCD.CPP</i>	53
VZ38. DÃY CHỮA MAX <i>Tên chương trình: NUMMAX.CPP</i>	56
VZ39. BIÊN ĐỘI SỐ <i>Tên chương trình: TRANSFORM.CPP</i>	59
VZ41. HÁI VIỆT QUẤT <i>Tên chương trình: BLUEBERRIES.CPP</i>	69
VZ42. CHAT <i>Tên chương trình: CHAT.CPP</i>	72
VZ43. ĐỘI HÌNH THI ĐẤU <i>Tên chương trình: TEAM.CPP</i>	75
VZ44. VẬN CHUYỂN <i>Tên chương trình: TRANSPORT.CPP</i>	78
VZ45. CẤP ĐIỂM <i>Tên chương trình: POINTS.CPP</i>	81

VZ46. TIỀM NĂNG	<i>Tên chương trình: POTENTIAL.CPP</i>	88
VZ47. KHẢO CỐ	<i>Tên chương trình: ARCHAEOLOGY.CPP</i>	93
VZ48. CHỈ SỐ	<i>Tên chương trình: INDEX.CPP</i>	98
VZ49. CÁNH ĐỒNG THỬ NGHIỆM	<i>Tên chương trình: FIELD.CPP</i>	102
VZ50. THI BẢN NHANH	<i>Tên chương trình: COMPETITION.CPP</i>	107
WA01. CẤU HÌNH	<i>Tên chương trình: CONFIG.CPP</i>	109
WA02. ĐOẠN LỚN NHẤT	<i>Tên chương trình: MAXSEGM.CPP</i>	113
WA03. XÓA KÝ TỰ	<i>Tên chương trình: DEL_STR.CPP</i>	117
WA04. NỘI ĐỊA HÓA	<i>Tên chương trình: LOCAL.CPP</i>	121
WA05. VÔ ĐỊCH	<i>Tên chương trình: CHAMPION.CPP</i>	123
WA06. LỊCH BAY	<i>Tên chương trình: SCHEDULES.CPP</i>	125
WA07. TAM GIÁC	<i>Tên chương trình: TRIANGLE.CPP</i>	127
WA08. HÀM LƯỢNG PALINDROME	<i>Tên chương trình: MAXPAL.CPP</i>	129
WA09. MEX	<i>Tên chương trình: MEX.CPP</i>	132
WA10. XÂU CON	<i>Tên chương trình: SUBSTR.CPP</i>	135
WA11. PHƯƠNG TRÌNH	<i>Tên chương trình: EQUATION.CPP</i>	138
WA12. MUA CÔ CÁ	<i>Tên chương trình: COCA.CPP</i>	141
WA13. KÉO CẮT GIÁY	<i>Tên chương trình: SCISSOR.CPP</i>	143
WA14. HAI PHẦN	<i>Tên chương trình: TWOPARTS.CPP</i>	146
WA15. TẬP LỚN NHẤT	<i>Tên chương trình: MAX_SET.CPP</i>	149
WA16. MẶT XÍCH YẾU NHẤT	<i>Tên chương trình: NEXUS.CPP</i>	158
WA17. GIẢI THOÁT	<i>Tên chương trình: ESCAPE.CPP</i>	163
WA18. THÍCH ĐỎ NGỌT	<i>Tên chương trình: SWEET.CPP</i>	167
WA19. THUẾ ĐƯỜNG BỘ	<i>Tên chương trình: TAX.CPP</i>	171
WA20. CHIẾC MŨ THẦN BÍ	<i>Tên chương trình: HAT.CPP</i>	174
WA21. BIỂU THỨC NGOẶC	<i>Tên chương trình: BRACKETS.CPP</i>	177
WA22. DÃY BONG BÓNG	<i>Tên chương trình: BALLOONS.CPP</i>	180
WA23. CHƠI BÀI	<i>Tên chương trình: CARDS.CPP</i>	184
WA24. PHÂN TÍCH TẦN SỐ	<i>Tên chương trình: FREQUENCY.CPP</i>	186
WA25. BẢO HIỂM Y TẾ	<i>Tên chương trình: INSURANCE.CPP</i>	188

WA26. PHẦN HÈ	Tên chương trình: <i>PEDIGREE.CPP</i>	190
WA27. SINH BA	Tên chương trình: <i>TRINES.CPP</i>	194
WA28. THIẾU HỤT	Tên chương trình: <i>DEFICIENCY.CPP</i>	196
WA29. TRẠI HÈ TIN HỌC	Tên chương trình: <i>SUM_CAMP.CPP</i>	199

Quy hoạch động theo lát cắt doc

Quy hoạch động theo lát cắt doc (*Dynamic programming with profile*) là phương pháp tìm phương án ưu tú bằng quy hoạch động khi một trong số các kích thước của bài toán là đủ nhỏ.

Lát cắt doc (*Profile*) là một trong số các dòng (hoặc cột) thỏa mãn điều kiện bài toán. Lát cắt doc được dùng làm tham số điều khiển trong sơ đồ quy hoạch động.

Nguyên lý chung

Bài toán thường gặp là cho một bảng kích thước $n \times m$ và yêu cầu tìm số lượng cách bố trí hình cho trước lên bảng. Về nguyên tắc, có thể duyệt vét cạn mọi khả năng bố trí hình. Nhưng nếu áp dụng quy hoạch động theo lát cắt doc ta có thể giảm số khả năng duyệt để có độ phức tạp tuyến tính theo kích thước lát cắt.

Giả thiết ta đã xác định được quy tắc dàn hình và tính được số lượng cách dàn hình ở k lát cắt (k cột) đầu tiên. Gọi a là số lượng cách đặt hình cần xét ở mỗi ô và mỗi cách đặt hình có liên quan tới q cột

Để tính số cách đặt hình với sự tham gia của lớp cắt $k+1$ ta cần duyệt $a^{q \times n}$ khả năng và như vậy độ phức tạp của giải thuật sẽ là $a^{q \times n} \times m$. Nếu duyệt vét cạn độ phức tạp sẽ là $a^{n \times m}$.

Bài toán lát hình bằng quân đố mi nô

Bài toán

Cho lưới ô vuông kích thước $n \times m$. Hãy xác định số cách lát lưới bằng các viên gạch hình chữ nhật kích thước 1×2 , sao cho mọi ô của lưới đều được phủ bởi đúng một viên gạch. Khi lát có thể đặt viên gạch nằm ngang hoặc dọc. Hai cách lát gọi là khác nhau nếu tồn tại ít nhất một ô để trong một cách nó được phủ bởi viên gạch đặt ngang và trong cách lát kia – bởi viên gạch đặt dọc.

Hãy xác định số cách lát khác nhau có thể thực hiện.

Dữ liệu: Vào từ file văn bản TILING.INP gồm một dòng chứa 2 số nguyên n và m ($1 \leq n, m \leq 10$).

Kết quả: Đưa ra file văn bản TILING.OUT một số nguyên – số cách lát khác nhau.

Ví dụ:

TILING.INP

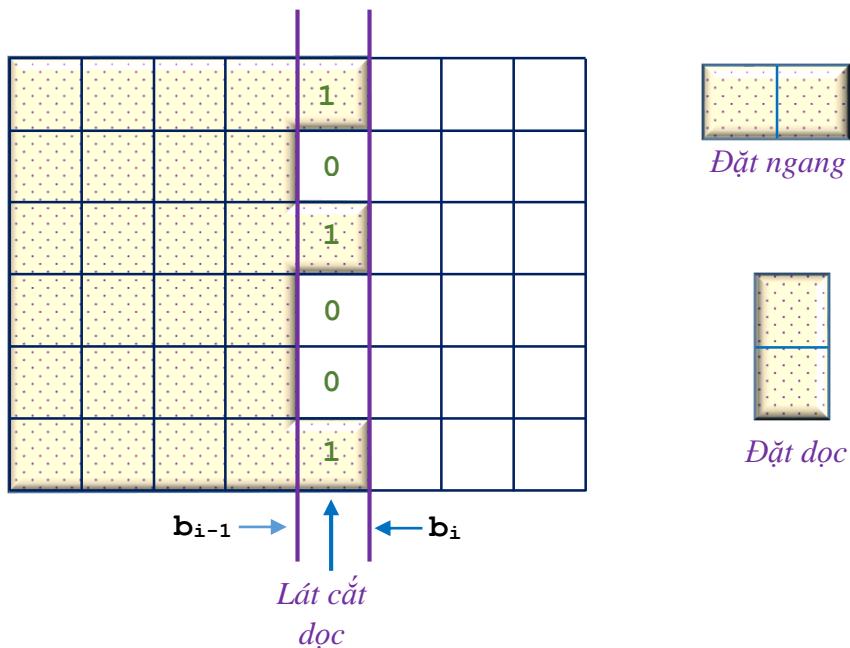
TILING.OUT



Giải thuật

Trong quá trình lát, mỗi ô của bảng có thể có một trong hai trạng thái: còn trống hay đã được lát. Để ghi nhận trạng thái các ô trong một cột chỉ cần dùng một biến p thuộc loại nguyên, bít thứ i của p bằng 0 nếu ô thứ i trong cột còn trống và bằng 1 nếu ô đó đã được lát. Biến p được gọi là *bản đồ bít* (hoặc còn gọi là *mặt nạ bít*) của cột.

Đường thẳng đứng (song song với trục O_y) đi qua các nút đỉnh ô của lưới được gọi là đường cơ sở.



Ký hiệu b_i là đường cơ sở mà cột i của lưới nhận làm đường biên phải. Như vậy cột i nằm giữa 2 đường cơ sở b_{i-1} và b_i .

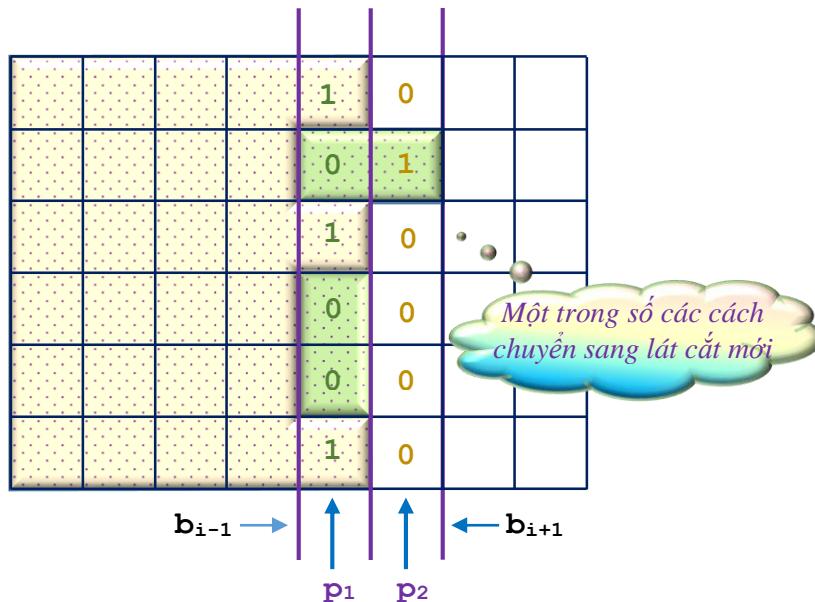
Xét trạng thái lát cắt dọc i với các điều kiện:

- ✚ Tất cả các ô nằm bên trái cột i đều đã được lát,
- ✚ Trên cột i không có các viên gạch đặt dọc,
- ✚ Các bên phải cột i chưa được lát.

Điều kiện đầu tiên đặt cơ sở cho việc xây dựng công thức lặp tính số cách lát. Các điều kiện 2 và 3 đảm bảo không đếm lặp cấu hình.

Trạng thái lát cắt được xác định bởi bản đồ bít. Ở hình trên, bản đồ bít của lát cắt là $100101_2 = 37_{10}$.

Từ mỗi bản đồ bít \mathbf{p}_1 của lát cắt đang xét có thể xác định tập các bản đồ \mathbf{p}_2 của lát cắt tiếp theo.



Ví dụ từ bản đồ $\mathbf{p}_1 = 37$ ta có thể chuyển sang bản đồ tiếp sau $\mathbf{p}_2 = 2$.

Ký hiệu $\mathbf{dp}_{i,j}$ là khả năng chuyển từ bản đồ i sang bản đồ j ở lát cắt tiếp theo, trong trường hợp này ta có $\mathbf{dp}_{37,2} = 1$.

Từ bản đồ \mathbf{p}_1 ta có thể chuyển sang bản đồ \mathbf{p}_2 nếu thỏa mãn các điều kiện:

- ⊕ Bít thứ u trong bản đồ \mathbf{p}_2 bằng 0 nếu bít thứ u tương ứng trong \mathbf{p}_1 bằng 1 (ở bên trái đã được lát và không thể đặt đô mi nô nằm ngang)
- ⊕ Bít thứ u trong bản đồ \mathbf{p}_2 có thể bằng 1 nếu bít thứ u tương ứng trong \mathbf{p}_1 bằng 0 (điều kiện có thể lát một viên *đô mi nô nằm ngang*),
- ⊕ Dãy các bít 0 liên tiếp còn lại trong \mathbf{p}_1 (ứng với bít 0 trong \mathbf{p}_2) phải tạo thành các nhóm có số lượng chẵn (điều kiện *lát kín* lát cắt dọc ứng với \mathbf{p}_1 bằng các *đô mi nô đặt dọc*).

Mỗi lát cắt chứa n ô, do đó bản đồ bít tương ứng với một trạng thái cụ thể của lát cắt là một số nằm trong phạm vi $[0 \dots 2^n)$. Như vậy dp phải là mảng 2 chiều kích thước $2^n \times 2^n$.

$dp_{i,j}$ bằng 1 nếu từ bản đồ i có thể chuyển sang bản đồ j ở lát cắt tiếp theo và bằng 0 trong trường hợp ngược lại.

Để xác định mỗi bít trong j ta cần biết khả năng lát kín các ô còn lại của p_1 ứng với bản đồ i . Việc này có thể thực hiện sơ đồ duyệt đệ quy.

```
void calc_profile(int p, int p2, int sz)
{
    if(sz==n) {dp[p][p2]=1; return;} // Đã duyệt hết các ô trong lát cắt
    if((p & (1<<sz))==0) // Ô ở vị trí sz trống
    {
        calc_profile(p,p2+(1<<sz),sz+1); //Đặt ngang.
        if(sz<n-1)
            if((p & (1<<(sz+1)))==0) calc_profile(p,p2,sz+2); // Còn trống thêm ô nữa => đặt dọc
    } else calc_profile(p,p2,sz+1); //Ô đang xét đã được lát => bỏ qua
    return;
}
```

Lát cắt trái nhất với đường cơ sở b_1 có bản đồ bít là 0 (không thể đặt ngang ô miêu để phần phải của lát cắt không bị lát).

Gọi $a_{u,i}$ là số cách lát $u-1$ cột đầu tiên và kết thúc bằng bản đồ i .

$a_{0,0} = 1$ (chỉ có một cách xuất phát: từ cột đầu tiên với bản đồ bít bằng 0).

Đặt $k = 2^n - 1$

$$a_{u,i} = \sum_{j=0}^k a_{u-1,j} \times dp_{j,i}$$

Số cách lát cần tìm sẽ là $a_{m,0}$ – lát kín lưới và kết thúc bằng bản đồ 0 tiếp theo.

Ta thấy, để tính các $a_{u,i}$ ta chỉ cần biết $a_{u-1,*}$ vì vậy trên thực tế chỉ cần lưu giá trị mảng a ứng với hai cột: cột cũ và cột mới, nhưng cần lưu ý *xóa giá trị biến chưa cột mới trước khi tính*.

Tổ chức dữ liệu:

- Mảng `int dp[1025][1025]={0}` – lưu bảng chuyển trạng thái,

- Mảng `int64_t a[2][1025]={0}` – phục vụ tính lặp theo sơ đồ quy hoạch động.

Độ phức tạp của giải thuật: O(2²ⁿm).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("domino.inp");
ofstream fo ("domino.out");
int n, m, k, u, v, dp[1025][1025]={0};
int64_t a[2][1025]={0};

void calc_profile(int p, int p2, int sz)
{
    if(sz==n) {dp[p][p2]=1; return;}
    if((p & (1<<sz))==0)
    {
        calc_profile(p,p2+(1<<sz),sz+1);
        if(sz<n-1)
            if((p & (1<<(sz+1)))==0) calc_profile(p,p2,sz+2);
        else calc_profile(p,p2,sz+1);
    }
    return;
}

int main()
{
    fi>>n>>m;
    if(n>m) swap(n,m);
    k = 1<<n;
    for(int i=0; i<k; ++i) calc_profile(i,0,0);

    a[0][0]=1; u=0; v=1;
    for(int q=1; q<=m; ++q)
    {
        for(int i=0; i<k; ++i)
            for(int j=0; j<k; ++j)a[v][i] += a[u][j]*dp[j][i];
        u^=1; v^=1;
        for(int ii=0; ii<k; ++ii)a[v][ii]=0;
    }

    fo<<a[u][0];
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```

Bài toán trang trí hoa văn

Bài toán

Cho lưới ô vuông kích thước $n \times m$ ô. Mỗi ô của lưới được tô một trong 2 màu vàng hoặc xanh. Hai cách tô gọi là khác nhau nếu tồn tại ít nhất một ô có màu khác nhau trong 2 cách tô.

Hãy xác định số cách tô khác nhau sao cho không tồn tại một hình vuông nào kích thước 2×2 cùng màu.

Dữ liệu: vào từ file văn bản PATTERN.INP gồm một dòng chứa 2 số nguyên n và m ($1 \leq n, m \leq 20, \min(n,m) \leq 10$).

Kết quả: Đưa ra file văn bản PATTERN.OUT một số nguyên – số cách lát khác nhau. Số cách tô có thể rất lớn, vì vậy chỉ cần đưa ra 9 chữ số cuối cùng trong giá trị tính được.

Ví dụ:

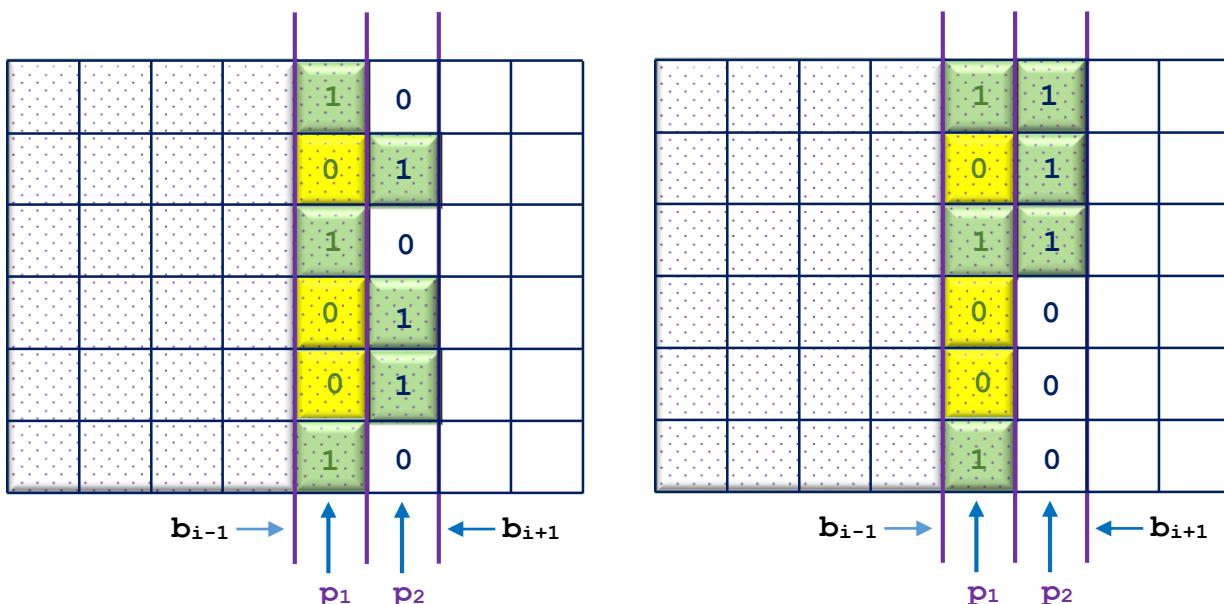
PATTER.INP	PATTER.OUT
6 6	11



Giải thuật 1

Xét lát cắt \mathbf{p}_1 xác định bởi đường cơ sở \mathbf{b}_i . Giả thiết phần lưới ô vuông bên trái đường cơ sở \mathbf{b}_{i-1} đã được tô đúng yêu cầu, phần bên phải \mathbf{b}_i chưa có ô nào được tô. Các ô màu vàng được gán giá trị 0, các ô màu xanh – giá trị 1.

Giả thiết $\mathbf{p}_1 = 100101_2 = 37_{10}$ là một cách tô đúng, khi đó lát cắt \mathbf{p}_2 xác định bởi đường cơ sở \mathbf{b}_{i+1} có thể tô với giá trị $011010_2 = 26_{10}$ hoặc $000111_2 = 7_{10}$. Khi đó ta có $\mathbf{dp}_{37, 26} = 1$ và $\mathbf{dp}_{37, 7} = 1$.



Bảng \mathbf{dp} có thể xây dựng bằng cách duyệt tất cả các giá trị \mathbf{p}_1 , $\mathbf{p}_1 = 0 \div 2^n - 1$, với mỗi \mathbf{p}_1 xác định và đánh dấu các \mathbf{p}_2 mà từ \mathbf{p}_1 có thể chuyển tới.

Gọi \mathbf{p}_{1i} và \mathbf{p}_{2i} là các bít thứ i của 2 số, $k = 2^n$. Từ \mathbf{p}_1 có thể chuyển sang \mathbf{p}_2 nếu số bít 1 trong các bít $\mathbf{p}_{1i}, \mathbf{p}_{1i+1}, \mathbf{p}_{2i}, \mathbf{p}_{2i+1}$ phải lớn hơn 0 và nhỏ hơn 4 với mọi $i = 0 \div k-2$. Hàm kiểm tra có thể có dạng:

```
bool check(int p1, int p2)
{
    int t;
    for(int i=0; i<n-1; ++i)
    {
        t = ((p1>>i)&1) + (((p1>>(i+1))&1)<<1) +
            (((p2>>i)&1)<<2) + (((p2>>(i+1))&1)<<3);
        if(t != 0 && t != 15) return false;
    }
    return true;
}
```

Trên cơ sở hàm **check** nêu trên ta có thể dễ dàng tính **dp**:

```
k = 1<<n;
for(int i=0; i<k; ++i)
    for(int j=0; j<k; ++j) dp[i][j]=check(i,j);
```

Cách tính số cách tô cũng tương tự như ở bài toán trước nhưng với 2 điểm khác biệt:

Giá trị xuất phát là $a_{0,i} = 1$, $i = 0 \div k-1$ – Có thể xuất phát từ cách tô bất kỳ của cột đầu,

Giá trị cần tìm là tổng các tối được cột cuối.

Độ phức tạp của giải thuật: O(2²ⁿm).

Chương trình 1

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("pattern.inp");
ofstream fo ("pattern.out");
const int m9=1e9;
int n, m, k, u, v, dp[1025][1025];
int64_t a[2][1025]={0};

bool check(int p1, int p2)
{
    int t;
    for(int i=0; i<n-1; ++i)
    {
        t = ((p1>>i)&1) + (((p1>>(i+1))&1)<<1) +
            (((p2>>i)&1)<<2) + (((p2>>(i+1))&1)<<3);
        return !(t==0 || t==15);
    }
    return 1;
}

int main()
{
    fi>>n>>m;
    if(n>m) swap(n,m);
    k = 1<<n;
    for(int i=0; i<k; ++i)
```

```

for(int j=0; j<k; ++j) dp[i][j]=check(i,j);

for(int i=0; i<k; ++i) a[0][i]=1;
u=0; v=1;
for(int q=1; q<m; ++q)
{
    for(int i=0; i<k; ++i)
        for(int j=0; j<k; ++j)
            a[v][i] = (a[v][i]+a[u][j]*dp[j][i])%m9;
    u^=1; v^=1;
    for(int ii=0; ii<k; ++ii)a[v][ii]=0;
}
int64_t res=0;
for(int i=0; i<k; ++i) res = (res+a[u][i])%m9;
fo<<res;
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```

Giải thuật 2

Không mất tính chất tổng quát, ta có thể coi $n \leq m$.

Với m đủ lớn ($m \leq 10^{18}$), sơ đồ tính toán trên không phù hợp vì độ phức tạp tỷ lệ với m . Có thể cải tiến sơ đồ tính toán để có giải thuật độ phức tạp $O(2^{3n} \log m)$.

Từ công thức lắp

$$a_{u,i} = \sum_{j=0}^k a_{u-1,j} \times dp_{j,i}$$

Nếu ký hiệu \mathbf{A}_u là dòng u của bảng \mathbf{A} , \mathbf{DP} là ma trận vuông với các phần tử $d_{p_i,j}$ ta có thể viết công thức trên bằng ngôn ngữ đại số tuyến tính: $\mathbf{A}_u = \mathbf{A}_{u-1} \times \mathbf{DP}$.

Như vậy:

$$\mathbf{A}_1 = \mathbf{A}_0 \times \mathbf{D}\mathbf{P},$$

$$A_2 = A_1 \times DP = A_0 \times DP^2,$$

$$A_3 = A_2 \times DP = A_0 \times DP^3,$$

• • • • • • • • • •

$$A_{m-1} = A_{m-2} \times DP = A_0 \times DP^{m-1}.$$

Như vậy, để tính \mathbf{A}_{m-1} ta chỉ cần thực hiện việc nâng nhanh DP lên lũy thừa $m-1$. Việc nhân ma trận vuông bậc k đòi hỏi chi phí $O(k^3)$ vì vậy sơ đồ tính toán dựa trên phương pháp nhân ma trận chỉ phù hợp với m đủ lớn.

Chương trình 2

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("pattern.inp");
ofstream fo ("pattern.out");
const int m9=1e9;
typedef vector<vector<int64_t>> vvi;
int n, m, k, u, v, p;
vvi dp, res, tmp;
vector<int64_t> a, b;
bool check(int p1, int p2)
{
    int t;
    for(int i=0; i<n-1; ++i)
    {
        t = ((p1>>i)&1) + (((p1>>(i+1))&1)<<1) +
            (((p2>>i)&1)<<2) + (((p2>>(i+1))&1)<<3);
        return !(t==0 || t==15);
    }
    return 1;
}

vvi mult(vvi &a, vvi &b)
{
    int64_t t;
    vvi tp(k, vector<int64_t> (k));
    for(int i=0; i<k; ++i)
        for(int j=0; j<k; ++j)
        {
            t=0;
            for(int u=0; u<k; ++u) t=(t+a[i][u]*b[u][j])%m9;
            tp[i][j]=t;
        }
    return tp;
}

vvi pow(vvi &a, int v)
{
    vvi tp(k, vector<int64_t> (k));
    vvi r(k, vector<int64_t> (k, 0));
    for(int i=0; i<v; ++i)
        r=mult(r, a);
    return r;
}
```

```

for (int i=0; i<k; ++i) r[i][i]=1;
tp=a;
while (v>0)
{
    if (v&1) r=mult(r, tp);
    v>>=1;
    tp=mult(tp, tp);
}
return r;
}

int main()
{
    fi>>n>>m;
    if (n>m) swap(n, m);
    k = 1<<n;
    dp.resize(k, vector<int64_t> (k));
    res.resize(k, vector<int64_t> (k));
    tmp.resize(k, vector<int64_t> (k));
    a.assign(k, 1);
    b.assign(k, 0);
    for (int i=0; i<k; ++i)
        for (int j=0; j<k; ++j) dp[i][j]=check(i, j);

    res = pow(dp, m-1);
    int64_t ans=0;
    for (int i=0; i<k; ++i)
        for (int j=0; j<k; ++j) ans=(ans+res[i][j])%m9;
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
}

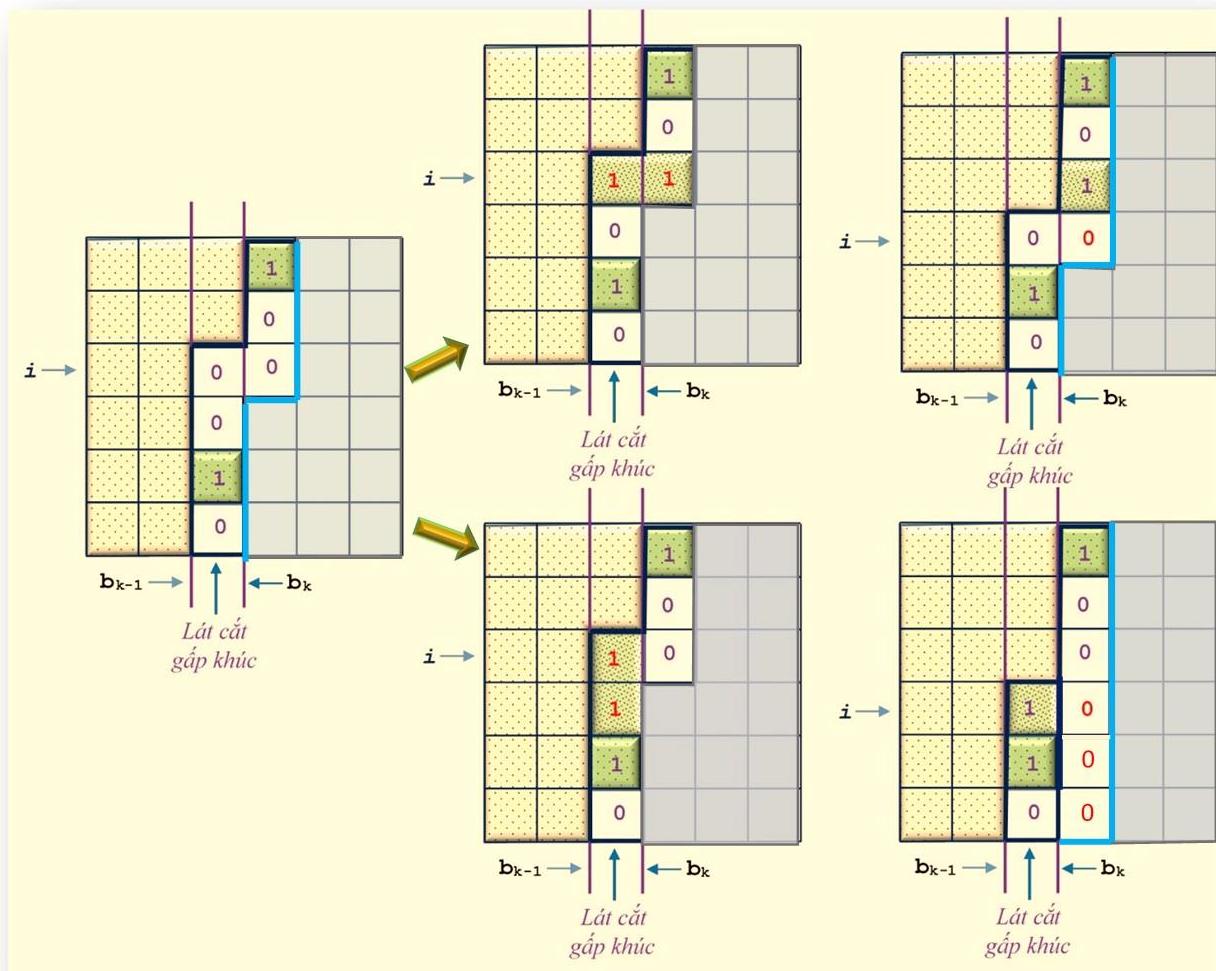
```

Quy hoạch động theo lát cắt gấp khúc

Lát cắt gấp khúc (*Broken Profile*) là lát cắt chứa một số cột và cùng với một số ô đầu tiên của cột tiếp theo. Đây là phương án tổng quát hóa lát cắt phẳng đã xét ở phần trên và là một cải tiến mang lại hiệu quả cao cho quy hoạch động theo lát cắt, chính vì vậy quy hoạch động theo lát cắt gấp khúc còn thường được gọi là “*Quy hoạch động nhanh theo lát cắt*”.

Nguyên lý xây dựng lát cắt

Tư tưởng chủ đạo của phương pháp này là xây dựng cấu trúc dữ liệu cho phép giảm đáng kể số phép chuyển từ một lát cắt sang lát cắt tiếp theo.



Để hiểu bản chất của lát cắt gấp khúc ta xét bài toán lát hình bằng quân đô mi nô.

Đường cơ sở có 2 phần: đi từ trên xuống khi qua dòng i thì chuyển sang đường dọc của cột trước.

Lát cắt là cặp (p, i) , trong đó p là thông tin về $n+1$ ô có điểm chung với đường cơ sở, i là dòng nơi đường cơ sở gấp khúc. Các ô được đánh số bắt đầu từ 0, từ trên xuống dưới. Ô góc có số thứ tự là $i+1$. Các hàng được đánh số từ 0 đến $n-1$.

Với hai lát cắt $pr1 = (p1, i1)$ và $pr2 = (p2, i2)$ ta có $d_{pr1, pr2} = 1$ khi và chỉ khi:

- ✚ Nếu $i1 < n-1$ thì $i2 = i1+1$, trong trường hợp ngược lại – $i2 = 0$.
- ✚ Có thể đặt quân đô mi nô (ngang hoặc dọc) phủ ô $i+1$ để sau đó $p2$ phản ánh đúng trạng thái các ô tương ứng.

Rõ ràng chỉ *có không quá 2 cách đặt* quân đô mi nô phủ ô $i+1$: đặt dọc hoặc đặt ngang và từ đó tạo ra lát cắt mới.

Lưu ý là nếu ô $i+1$ đã được phủ thì không cần đặt đô mi nô và (p, i) được đặt đồng nhất với $(p, i+1)$. Ở đây quy ước $i+1 = 0$ nếu $i = n-1$.

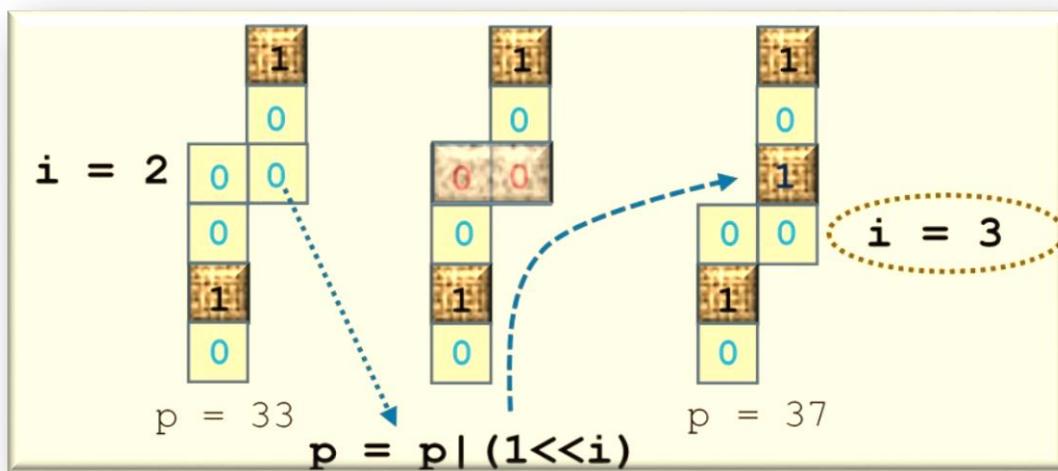
Dễ dàng thấy rằng số lát cắt tăng thêm $2 \times n$ lần (số thể hiện p có $n+1$ bit), nhưng số phép chuyển giảm từ 2^n xuống còn 2.

Dẫn xuất lát cắt

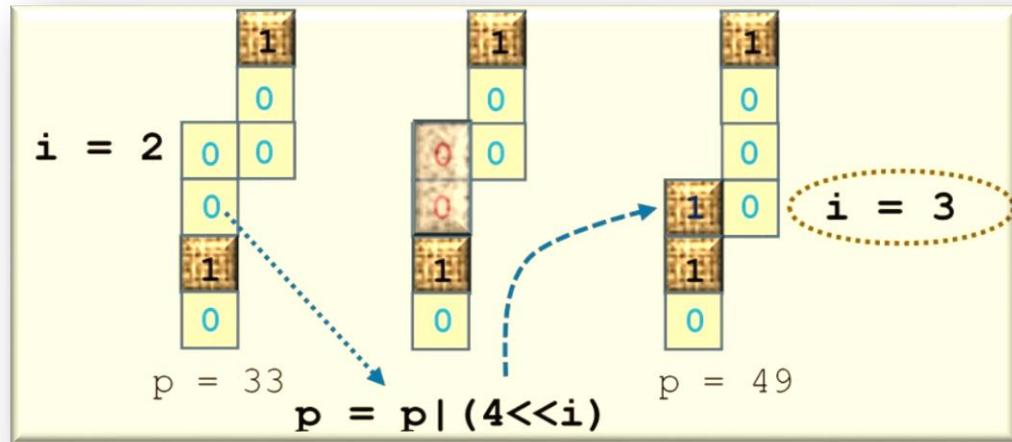
Giải thuật dẫn xuất các lát cắt tiếp theo từ lát cắt (p, i) cho trước:

Trường hợp $p_{i+1} = 0$ và $i < n-1$:

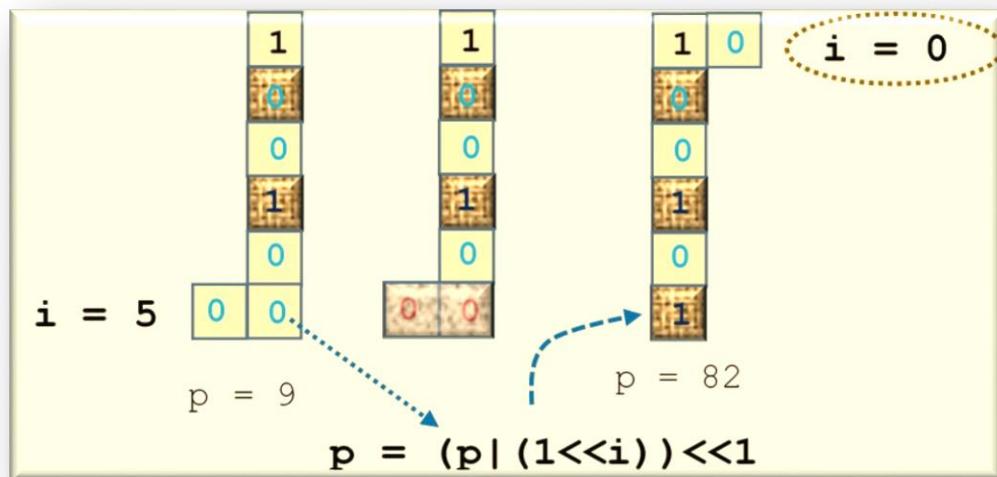
- Nếu $p_i = 0$ ($n = 6$):



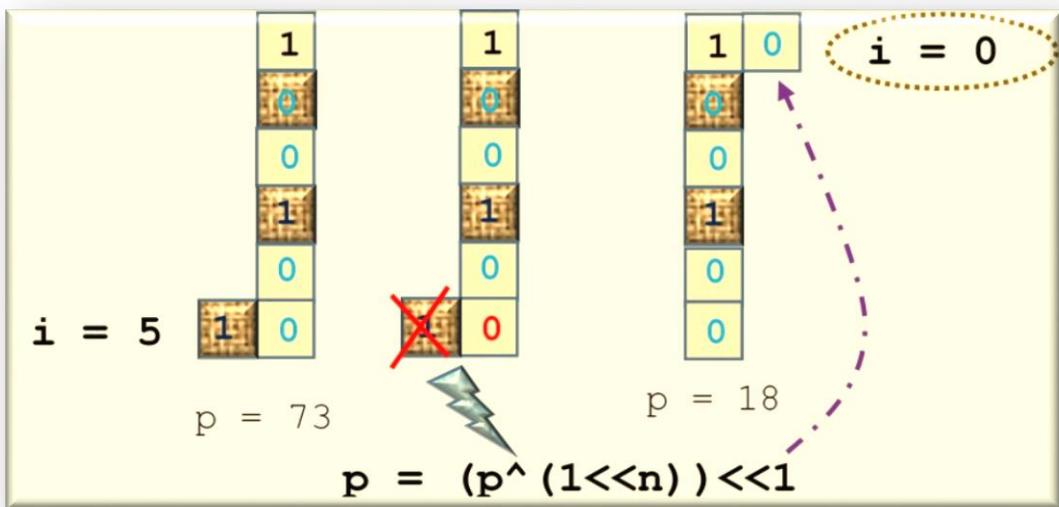
➤ Nếu $i < n-1$ và $p_{i+2} = 0$ ($n = 6$):



➤ Trường hợp $p_{i+1} = 0$ và $i = n-1$ ($n = 6$):



Trường hợp $p_{i+1} = 1$ và $i = n-1$:



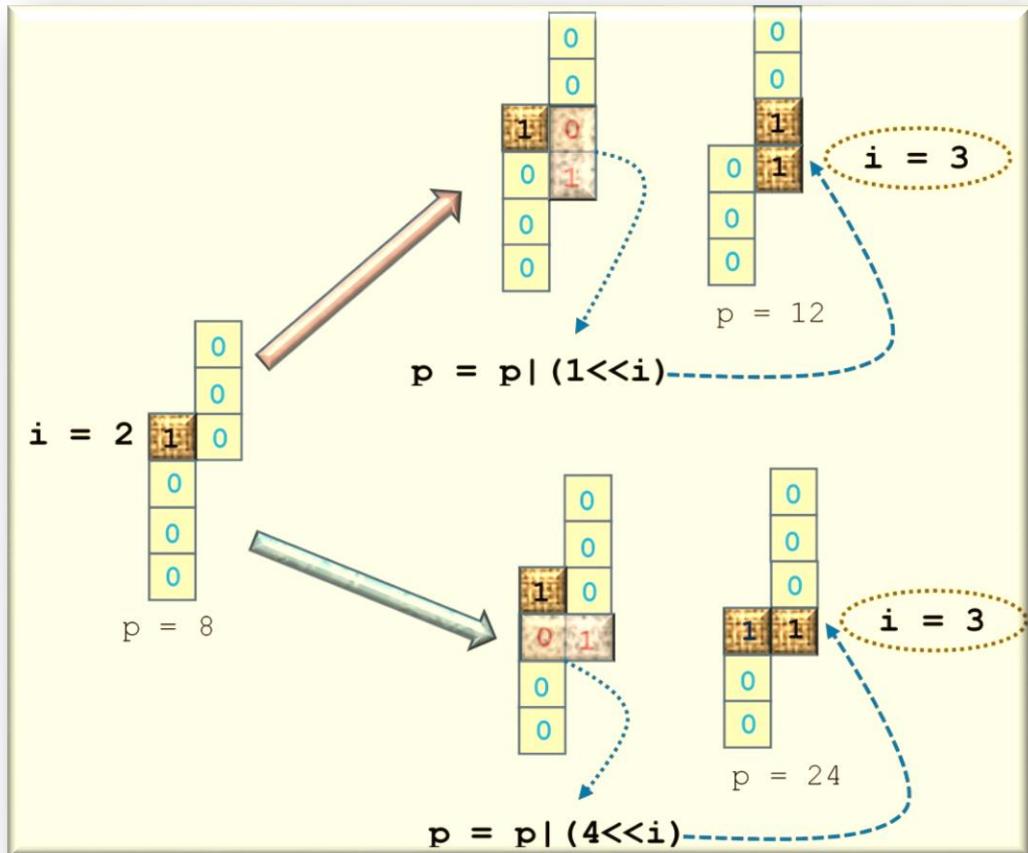
Trường hợp $p_{i+1} = 1$ và $i < n-1$:

Hàm dẫn xuất lát cắt:

```
void get_prof(int p, int i)
{
    if(!bit(p,i+1))
        if(i==n-1)
            fo<<((p|(1<<i))<<1)<<" 0"<<endl;
            else fo<<(p|(1<<i))<<" "<<i+1<<endl;

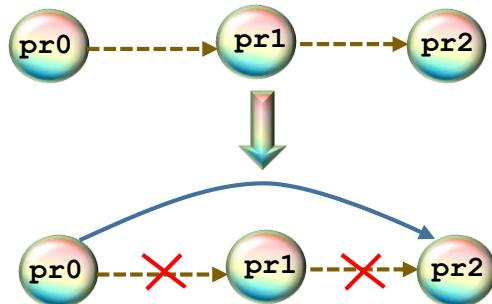
    else
        if(!bit(p,i))
            if(i==n-1) fo<<((p^(1<<n))<<1)<<" 0"<<endl;
            else fo<<(p^(1<<i))<<" "<<i+1<<endl;

    if(i<n-1 && (!bit(p,i+2)))
        fo<<(p+(4<<i))<<" "<<i+1<<endl;
}
```



Nhận xét:

- + Tồn tại không ít các lát cắt chỉ dẫn tới một lát cắt tiếp theo (ví dụ, các lát cắt có $p_{i+1} = 1$). Sử dụng quan hệ bắc cầu ta có thể thay thế cả đoạn đường đi chứa các trạng thái này bằng cách ghi nhận một phép chuyển từ trạng thái đầu của đoạn đường tới trạng thái cuối của đoạn. Ví dụ, từ **pr0** có thể tới **pr1**, từ **pr1** chỉ có một cách duy nhất tới **pr2**, tức là $d_{pr0, pr1} = 1$, $d_{pr1, pr2} = 1$, ta có thể ghi nhận việc chuyển trực tiếp từ **pr0** tới **pr2** và hủy các bước chuyển trung gian: $d_{pr0, pr2} = 1$, $d_{pr0, pr1} = 0$, $d_{pr1, pr2} = 0$.



Việc tối ưu hóa này cho phép giảm không ít hơn một nữa số lát cắt cần lưu trữ!

- + Có thể áp dụng sơ đồ loang theo chiều sâu (DFS) tích lũy kết quả cần tìm thay vì xây dựng trước bảng chuyển trạng thái.

Độ phức tạp của giải thuật: $O(2^n nm)$, hiệu quả hơn nhiều so với phương pháp dùng lát cắt phẳng.

Ví dụ ứng dụng

Bài toán

Xét giải thuật sử dụng lát cắt gáp khúc giải bài toán Lát đố mi nô đã xét ở trên.

```
#include <bits/stdc++.h> //Lat domino, Method:Dp_Broken_Profile
using namespace std;
ifstream fi ("dp_broken.inp");
ofstream fo ("dp_broken.out");
int n, m;
vector < vector<int64_t> > d;

void calc (int x , int y , int mask , int next_mask )
{
    if (x == m)
        return;
    if (y >= n)
        d[x+1][next_mask] += d[x][mask];
    else
    {
        int my_mask = 1 << y;
        if (mask & my_mask)
            calc (x, y+1, mask, next_mask);
        else
        {
            calc (x, y+1, mask, next_mask | my_mask);
            if ((y+1<n) && !(mask&my_mask) && !(mask&(my_mask<<1)))
                calc (x, y+2, mask, next_mask);
        }
    }
}

int main()
{
    fi >> n >> m;
    if ((n*m)&1) {fo<<'0'; return 0;}
    if (n>m) swap(n,m);
    d.resize (m+1, vector<int64_t> (1<<n));
    d[0][0] = 1;
    for (int x=0; x<m; ++x)
        for (int mask=0; mask<(1<<n); ++mask)
            calc (x, 0, mask, 0);

    fo << d[m][0];
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Nhận xét:

Với cơ chế tối ưu hóa đã nêu và phương pháp loang theo chiều sâu ta chỉ cần lưu 2^n trạng thái thay vì 2^{2n} như sơ đồ lý thuyết chuẩn!

GIẢI THUẬT AHO – CORASIK

Xét bảng chữ cái kích thước k và bộ các xâu với tổng độ dài là m . Với các xâu đã cho Giải thuật Aho – Corasik xây dựng cấu trúc dữ liệu rùng để lưu trữ và từ đó – xây dựng ô tô mát phục vụ cho nhiều loại bài toán trên tập các xâu đã cho, ví dụ tìm tất cả các lần xuất hiện các xâu của tập trong văn bản cho trước với độ phức tạp tuyến tính.

Giải thuật Aho – Corasik có độ phức tạp $O(m)$ và yêu cầu bộ nhớ $O(m \times k)$, do Alfred Vaino Aho và Margaret John Corasick đề xuất năm 1975.

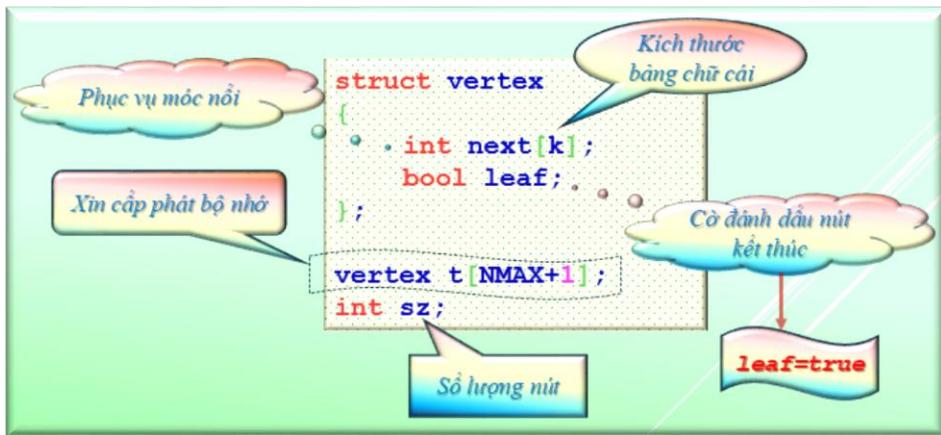
Xây dựng rùng

Rùng là một cây mà mỗi cạnh đi ra từ gốc được gắn với một ký tự (gọi là *nhân* của cạnh). Các cạnh đi ra từ mỗi đỉnh (không tính cạnh móc nối tới đỉnh cha, nếu có) được gắn với những ký tự khác nhau.

Xét đường đi bất kỳ từ gốc. Nếu ghi lại các ký tự gắn với cạnh theo trình tự gấp, ta được xâu ứng với đường đi. Ngược lại, với mỗi đỉnh bất kỳ của rùng cũng tương ứng với một xâu xác định đường đi từ gốc tới đỉnh đó.

Mỗi đỉnh có đường đi tương ứng với một xâu trong tập đã cho được đánh dấu là *lá* (mặc dù nó *có thể không phải là lá của cây*). Mỗi đỉnh được đánh dấu là lá tương ứng với một xâu trong tập và ngược lại, mỗi xâu trong tập tương ứng với một đỉnh có thuộc tính lá.

Cấu trúc mỗi đỉnh và cấu trúc dữ liệu lưu trữ rùng:



Như vậy rừng được lưu trữ dưới dạng mảng.

Ban đầu rừng chỉ chứa một nút gốc:

```

memset (t[0].next, 255, sizeof t[0].next);
sz = 1;

```

Rừng sẽ được mở rộng dần bằng cách lần lượt bổ sung các xâu từ tập đã cho.

Xét việc bổ sung xâu **s**. Xuất phát từ gốc, nếu đã có cạnh với ký tự được gán là **s₀** thì đi theo cạnh đó tới nút tiếp theo, trong trường hợp ngược lại – tạo cạnh mới với nhãn là **s₀** và đi tới đỉnh mới tạo. Từ đỉnh tới được xử lý tương tự với ký tự **s₁** và lặp lại quá trình trên cho đến hết xâu **s**. Sau khi đi hết hoặc tạo đường đi ứng với **s**, gán thuộc tính lá cho đỉnh cuối của đường đi.

Hàm bổ sung xâu mới:

```
void add_string (const string & s)
{
    int v = 0;
    for (size_t i=0; i<s.length(); ++i)
    {
        char c = s[i] - 'a';
        if (t[v].next[c] == -1)
        {
            memset (t[sz].next, 255,
                    sizeof t[sz].next);
            t[v].next[c] = sz++;
        }
        v = t[v].next[c];
    }
    t[v].leaf = true;
}
```

Phụ thuộc
bằng chữ cái

Xây dựng ô tô mát

Giả thiết đã xây dựng xong rùng cây quản lý các xâu đã cho. Mỗi đỉnh bất kỳ sẽ xác định xâu là tiền tố của một hoặc một số xâu đã cho, tức là mỗi đỉnh sẽ tương ứng với vị trí trong một hoặc một số xâu.

Việc di chuyển từ một đỉnh tới đỉnh khác sẽ đưa ta từ vị trí này đến vị trí khác của xâu trong bộ dữ liệu. Nếu coi vị trí đang đứng là một trạng thái trong tập các xâu, rùng cây đã xây dựng cho ta quy tắc di chuyển từ trạng thái này sang trạng thái khác, tức là ta có một *ô tô mát tiền định hữu hạn*. Ví dụ, nếu rùng cây chỉ chứa một xâu “**abc**” và ta đang ở trạng thái 2 (tương ứng với xâu “**ab**”), dưới tác động của ký tự ‘c’ ta sẽ chuyển sang trạng thái 3. Mỗi cạnh của rùng xác định một phép chuyển trạng thái theo nhãn của cạnh đó.

Tuy nhiên, khi chuyển trạng thái theo một xâu nào đó dựa vào rùng cây đã xây dựng và ta tới được trạng thái **p** ứng với tiền tố **t** của xâu đang xét. Tiếp theo cần di chuyển theo cạnh có nhãn **c**.

Sẽ có 2 trường hợp có thể xảy ra:

Tồn tại cạnh có nhãn **c** và ta chỉ việc di chuyển theo nó, tới đỉnh tương ứng với xâu **tc**,

Không tồn tại cạnh có nhãn **c** đi ra từ đỉnh đang đứng, khi đó ta phải *tìm hậu tố dài nhất* của **t** được lưu trong rừng mà từ đó có phép chuyển theo **c**.

Ví dụ, rừng được xây dựng với 2 xâu “**ab**” và “**bc**”. Xâu cần xử lý là “**abc**”. Khi đó dưới tác động của “**ab**” ta chuyển tới nút lá. Để đi tiếp ta cần lùi lại tới đỉnh trạng thái ‘**b**’ để có thể đi tiếp theo ký tự ‘**c**’.

Móc nối hậu tố của đỉnh **p** là *đỉnh nơi kết thúc của hậu tố dài nhất* của *xâu tương ứng với đỉnh p*. Trường hợp *đặc biệt duy nhất – gốc của rừng*. Để thuận tiện, ta cho *móc nối hậu tố của gốc rừng* chỉ tới *chính nó*.

Quy tắc chuyển trạng thái trong ô tô mát sẽ là như sau: Chừng nào từ đỉnh hiện tại của rừng không có cạnh với nhãn tiếp theo cần chuyển thì phải chuyển theo móc nối hậu tố.

Như vậy ta đã đưa bài toán xây dựng ô tô mát về bài toán tìm móc nối hậu tố của các đỉnh trong rừng cây. Nhưng điều đáng ngạc nhiên ở đây là móc nối hậu tố sẽ được xác định dựa trên bảng chuyển trạng thái của ô tô mát!

Nếu muốn biết móc nối hậu tố của một đỉnh **v** nào đó ta có thể chuyển tới đỉnh cha **p** của đỉnh hiện tại, trong đó **p** chưa phép chuyển tới **v** theo nhãn **c**, sau đó chuyển theo móc nối hậu tố của đỉnh này và từ đó – thực hiện phép chuyển theo nhãn **c**.

Tóm lại, bài toán tìm phép chuyển được đưa về bài toán tìm móc nối hậu tố, còn bài toán tìm móc nối hậu tố – đưa về bài toán tìm móc nối hậu tố và chuyển trạng thái, nhưng với những đỉnh gần gốc hơn. Ta có mối quan hệ đệ quy, nhưng hữu hạn và có thể giải quyết với chi phí thời gian tuyến tính.

Để xây dựng hàm tìm móc nối hậu tố, với mỗi đỉnh cần lưu trữ đỉnh **p** – cha của nó và nhãn **pch** chuyển từ đỉnh cha tới đỉnh đang xét. Ngoài ra, với mỗi đỉnh cần lưu trữ link – móc nối hậu tố của nó (bằng -1 nếu chưa được tính), mảng **go[k]** ghi nhận các phép chuyển theo từng ký tự (bằng -1 nếu chưa được tính).

Bảng -1 khi
chưa được tính

```
struct vertex
{
    int next[k]; //Bảng mốc nối tới đỉnh tiếp sau
    bool leaf; //Cờ đánh dấu đỉnh kết thúc
    int p; //Mốc nối tới đỉnh cha
    char pch; //Ký tự của cạnh dẫn tới đỉnh này
    int link; //Mốc nối hậu tố
    int go[k]; //Bảng mốc nối hậu tố
};

vertex t[NMAX+1];
int sz;

void init()
{
    t[0].p = t[0].link = -1;
    memset(t[0].next, 255, sizeof t[0].next);
    memset(t[0].go, 255, sizeof t[0].go);
    sz = 1;
}
```

```
int go (int v, char c); //Khai báo mẫu lời gọi
int get_link (int v)
{
    if (t[v].link == -1)
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go (get_link (t[v].p), t[v].pch);
    return t[v].link;
}

int go (int v, char c)
{
    if (t[v].go[c] == -1)
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v==0 ? 0 : go (get_link (v), c);
    return t[v].go[c];
}
```

Để sử dụng trước
khi khai báo hàm

Độ phức tạp:
Tuyến tính

VZ29. TÍNH ĐỒNG NHẤT

Tên chương trình: UNIFORMITY.CPP

Giải mã không bao giờ là một công việc đơn giản. Mỗi quốc gia đều có có thư viện riêng lưu trữ hồ sơ các mật mã: Cách mã hóa và giải mã, Ai đã sử dụng nó và khi nào, . . . Với một thông tin được mã hóa, việc đầu tiên người ta phải xác định một số đặc trưng để làm khóa tra cứu, tìm kiếm trong thư viện.

Bài tập cho các học viên hôm nay là cho thông tin được mã hóa thành dãy số nguyên a_1, a_2, \dots, a_n . Giả thiết a_1 gấp trong dãy số k_1 lần, a_2 – gấp k_2 lần, . . . Tính đồng nhất của dãy là số nguyên nhỏ nhất $c \geq 1$ và $c \neq k_i$ với mọi i . Yêu cầu xử lý q truy vấn, mỗi truy vấn thuộc một trong 2 dạng:

- 1 $lf\ rt$ – Tìm tính đồng nhất dãy đã cho trong đoạn từ vị trí lf đến vị trí rt (kể cả rt), $1 \leq lf \leq rt \leq n$,
- 2 $p\ x$ – thay a_p bằng x .

Với các truy vấn dạng 1 hãy đưa ra tính đồng nhất tìm được.

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

- Đòng đầu tiên chứa 2 số nguyên n và q ($1 \leq n, q \leq 10^5$),
- Đòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^9$, $i = 1 \div n$),
- Mỗi dòng trong q dòng sau chứa 3 số nguyên xác định một truy vấn.

Kết quả: Đưa ra file văn bản UNIFORMITY.OUT kết quả tìm được ứng với truy vấn loại 1, mỗi kết quả trên một dòng.

Ví dụ:

UNIFORMITY.INP	UNIFORMITY.OUT
<pre>10 5 1 2 3 1 1 2 2 2 9 9 1 1 1 1 2 8 2 7 1 2 8 5 1 2 8</pre>	<pre>2 3 4</pre>



Giải thuật: Thông kê tần số.

Mảng số ban đầu có **n** số, tối đa có **q** truy vấn thay đổi giá trị,

Trường hợp xấu nhất: mọi số đều khác nhau,

Như vậy có thể ánh xạ các số sang tập số nguyên từ 1 đến **n+q**, sao cho các số khác nhau tương ứng với số khác nhau sau ánh xạ, các số giống nhau - ánh xạ sang cùng một số.

Gọi **b_i** là ánh xạ của **a_i**, $0 \leq b_i \leq n+q-1$, $i = 0, 1, 2, \dots$

Để thực hiện ánh xạ cần có bản đồ dữ liệu **map<int, int> ids** và biến **cur_sz** lưu số lượng các số khác nhau đã gặp. Với số nguyên **x**, nếu nó đã xuất hiện trước đó thì nó được lưu giữ trong **ids** và ánh xạ của **x** là **ids[x]**. Trong trường hợp ngược lại, số lượng các số khác nhau đã gặp (**cur_sz**) tăng lên 1 và số này được nạp vào bản đồ ở vị trí **x** và sau đó – xác định ánh xạ của **x**.

Trên thực tế, ta không cần lưu giữ riêng **a_i** vì bản thân giá trị **a_i** sẽ không tham gia vào các xử lý tiếp theo.

Xử lý truy vấn: Để tìm **c** trong đoạn **[lf, rt]** cần duyệt mọi **b_i**, $i \in [lf, rt]$, với mỗi **b_i** cần xóa số lượng tần số xuất hiện cũ của **b_i** và ghi nhận số lần xuất hiện mới. Việc xác định kết quả đơn thuần là duyệt các tần số xuất hiện và ghi nhận tần số bằng không và sau đó khôi phục lại trạng thái bộ đếm tần số như trước khi tìm **c** trong khoảng đã cho.

Tổ chức dữ liệu:

- Mảng **int** **b** [**MAX_N**] – Ghi nhận ánh xạ dữ liệu vào sang khoảng $[0, 2 \times n]$,
- Mảng **int** **fr** [$2 * \text{MAX_N}$] – Ghi nhận tần số phục vụ tìm kiếm kết quả,
- Mảng **int** **cnt** [**MAX_N**] – Xác định sự xuất hiện của các tần số,
- Bản đồ **map<int, int> ids** - Phục vụ ánh xạ dữ liệu vào để xác định **b**.

Độ phức tạp của giải thuật: $O((n \times q) \log n)$.

Xử lý:

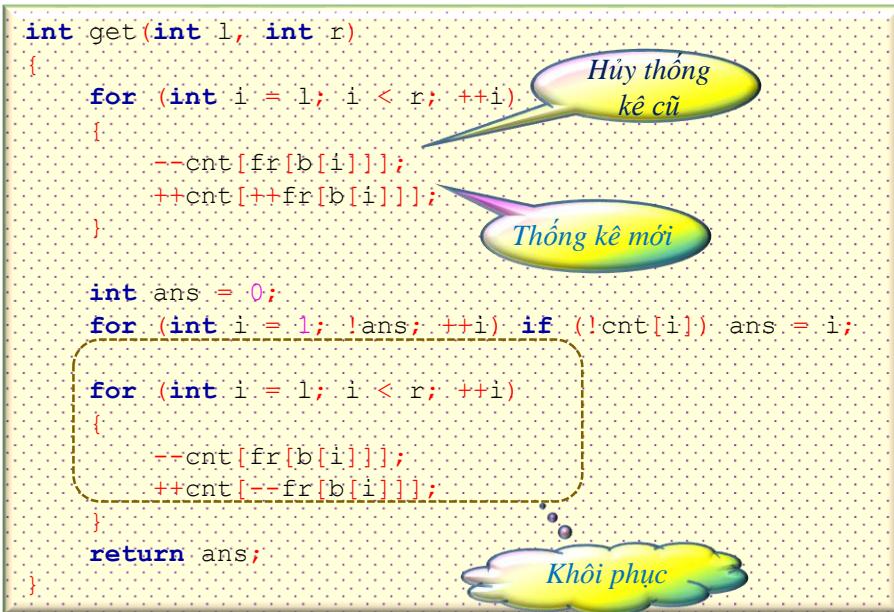
Ánh xạ dữ liệu về đoạn $[0, 2 \times n]$:



```
int get_id(int x)
{
    if (ids.find(x) == ids.end())
        ids[x] = cur_sz++;
    return ids[x];
}
```

Tìm c:

```
int get(int l, int r)
{
    for (int i = l; i < r; ++i)
    {
        --cnt[fr[b[i]]];
        ++cnt[++fr[b[i]]];
    }
    int ans = 0;
    for (int i = l; !ans; ++i) if (!cnt[i]) ans = i;
    for (int i = l; i < r; ++i)
    {
        --cnt[fr[b[i]]];
        ++cnt[--fr[b[i]]];
    }
    return ans;
}
```



Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi("uniformity.inp");
ofstream fo ("uniformity.out");

const int MAX_N = 100 * 1000 + 5;
int n, q;
int b[MAX_N];
int cnt[MAX_N];
int fr[2 * MAX_N];
int cur_sz = 0;
map<int, int> ids;

int get_id(int x)
{
    if (ids.find(x) == ids.end()) ids[x] = cur_sz++;
    return ids[x];
}

int get(int l, int r)
{
    for (int i = l; i < r; ++i)
    {
        --cnt[fr[b[i]]];
        ++cnt[++fr[b[i]]];
    }

    int ans = 0;
    for (int i = l; !ans; ++i) if (!cnt[i]) ans = i;

    for (int i = l; i < r; ++i)
    {
        --cnt[fr[b[i]]];
        ++cnt[--fr[b[i]]];
    }
    return ans;
}

int main()
{
    fi>>n>>q;
    for (int i = 0; i < n; ++i)
    {
        fi>>b[i];
        b[i] = get_id(b[i]);
    }
    int t, l, r;
    while (q--)
    {
        fi>>t>>l>>r;
```

```
--l;  
    if (t == 1) fo<<get(l, r)<<'\\n';  
    else b[l] = get_id(r);  
}  
  
fo<<"\\nTime: "<<clock() / (double) 1000<<" sec";  
return 0;  
}
```



VZ30. LƯỢT ĐI

Tên chương trình: PASS.QPP

Steve là tài xế xe tải và có nhiệm vụ vận chuyển nguyên vật liệu từ nhà máy tới n xí nghiệp. Trên đường đi Steve phải qua một trạm thu phí BOT, vé mỗi lượt là k . Xí nghiệp thứ i cấp cho Steve vé điện tử qua trạm trị giá a_i , $i = 1 \div n$. Nhà máy cũng cung cấp một thẻ thanh toán qua tài khoản với giá trị p .

Hệ thống quét thẻ bao giờ cũng xử lý vé điện tử trước, sau đó mới quét tới thẻ thanh toán qua tài khoản và mỗi lần chỉ xử lý tối đa một vé điện tử. Nếu số tiền còn dư trên vé điện tử không đủ k hệ thống sẽ quét tiếp thẻ thanh toán và trừ tiếp số tiền còn thiếu để có một vé qua trạm. Sau mỗi lần qua trạm tiền trong vé điện tử (và có thẻ - cả ở thẻ thanh toán) giảm một các tương ứng với số tiền bị trừ.

Hãy xác định tối đa Steve có thể qua trạm thu phí bao nhiêu lần.

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

- ⊕ Dòng đầu tiên chứa 3 số nguyên n , p và k ($1 \leq n \leq 10^5$, $0 \leq p \leq 10^{18}$, $1 \leq k \leq 10^9$),
- ⊕ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($0 \leq a_i \leq 10^9$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản PASS.OUT một số nguyên – số lần tối đa có thể qua trạm.

Ví dụ:

PASS.INP
2 1000 2000
1299 1701

PASS.OUT
2



VZ30 Moo20190113 C A XVI

Giải thuật: Nguyên lý cực trị.

Tích lũy số lần qua trạm bằng vé điện tử và lưu lại số tiền còn dư không đủ qua trạm ở mỗi vé ($a[i] \% k$),

Sắp xếp a_i theo thứ tự *giảm dần*,

Dùng thẻ thanh toán để bù các vé điện tử theo trình tự đã sắp xếp,

Sau khi bù xong các vé điện tử: dùng số tiền còn lại trong thẻ thanh toán để mua vé.

Tổ chức dữ liệu:

Mảng `vector<int>` $a(n)$ lưu giá trị của vé điện tử.

Độ phức tạp của giải thuật: $O(n \log n)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("pass.inp");
ofstream fo ("pass.out");
int n,k;
int64_t p,ans=0;

int main()
{
    fi>>n>>p>>k;
    vector<int> a(n);
    for(int i=0; i<n; ++i)
    {
        fi>>a[i];
        ans+=a[i]/k;
        a[i]%=k;
    }

    sort(a.rbegin(), a.rend());
    for(int i=0; i<n; ++i)
    {
        if(a[i]==0) break;
        if(k-a[i]>p) break;
        ++ans; p-=(k-a[i]);
        if(p==0) break;
    }
    ans+=p/k;
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VZ31*, NHẠC NƯỚC

Tên chương trình: WMUSIC.CPP

Cuộc thi nhạc nước hàng năm hấp dẫn du khách không kém thi bắn pháo hoa. Mỗi màn trình diễn đòi hỏi sự phối hợp giữa làn điệu âm thanh với sự thay đổi huyền ảo của màu sắc trên nền các cột nước.

Kịch bản thay đổi màu được đưa vào hệ thống điều khiển các cột nước và đèn chiếu tia laser. Có n cột nước, đánh số từ 1 đến n . Mở màn, cột i có màu a_i . Cứ sau một đơn vị thời gian màu các cột nước thay đổi theo kịch bản $\mathbf{A} = (a_1, a_2, \dots, a_n)$ – cột nước a_i sẽ có màu hiện có ở cột i , tạo hiệu ứng như cột i chạy sang vị trí a_i . Các a_i, a_j khác nhau từng đôi một và nhận giá trị nguyên trong khoảng từ 1 đến n .



Màn biểu diễn kết thúc khi các cột nước có màu $\mathbf{B} = (b_1, b_2, \dots, b_n)$, trong đó các b_i, b_j khác nhau từng đôi một và nhận giá trị nguyên trong khoảng từ 1 đến n .

Hãy xác định với \mathbf{A} và \mathbf{B} cho trước màn biểu diễn có kết thúc được hay không và nếu kết thúc được thì sau bao nhiêu đơn vị thời gian.

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

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^6$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ,
- ✚ Dòng thứ 3 chứa n số nguyên b_1, b_2, \dots, b_n .
- ✚ Các số a_i, b_j – thỏa mãn điều kiện đã nêu.

Kết quả: Đưa ra file văn bản WMUSIC.OUT:

- ☛ Dòng đầu tiên chứa thông báo **Yes** nếu màn biểu diễn kết thúc được hoặc **No** trong trường hợp ngược lại,
- ☛ Nếu kết thúc được – dòng thứ 2 chứa một số nguyên – thời gian biểu diễn.

Ví dụ:

WMUSIC.INP
4
2 3 4 1
1 2 3 4

WMUSIC.OUT
Yes



VZ30 Moo20190113 C A XVI

Giải thuật: Bộ số chung nhỏ nhất .

Theo thời gian, mỗi màu sẽ chuyển sang màu khác. Tập các màu là hữu hạn vì vậy sau một thời gian hoạt động đủ dài với mỗi màu i tồn tại tập vị trí \mathbf{v}_i mà màu này có thể có mặt.

Nếu tồn tại i để $\mathbf{b}_i \notin \mathbf{v}_i$ – bài toán vô nghiệm.

Gọi t_i là thời gian nhỏ nhất để màu i tới vị trí \mathbf{b}_i .

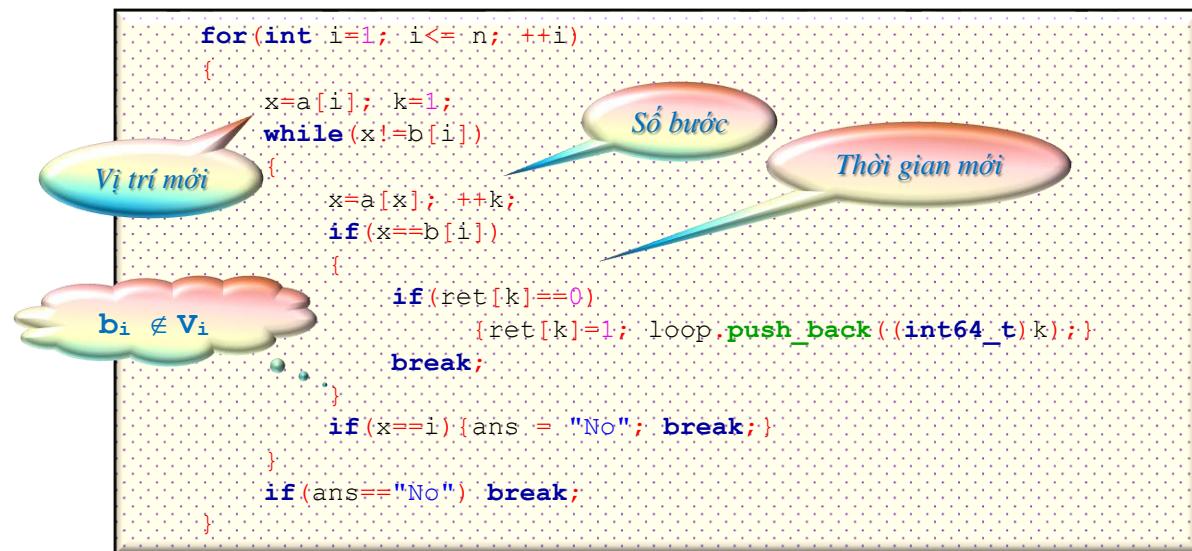
Thời gian nhỏ nhất đưa các màu về vị trí kết thúc là bội số chung nhỏ nhất của các t_i , $i = 1 \div n$.

Tổ chức dữ liệu:

- Các mảng `vector<int>` `a(n+1)`, `b(n+1)` – Lưu trữ dữ liệu input,
- Mảng `vector<int64_t>` `loop` – Ghi nhận thời gian nhỏ nhất để màu i tới vị trí \mathbf{b}_i ,
- Mảng `vector<int>` `ret(n+1, 0)` – Đánh dấu các chu trình đã được ghi nhận.

Xử lý:

Ghi nhận các chu trình thời gian:



Tính bội số chung nhỏ nhất:

```
if (ans=="Yes")
{
    int64_t z=1;
    for (int64_t i;loop) z=z*i/_gcd(z,i);
    fo<<z;
}
```

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("wmusic.inp");
ofstream fo ("wmusic.out");
int n,x,y,k;
string ans="Yes";

int main()
{
    fi>>n;
    vector<int> a(n+1), b(n+1), ret(n+1, 0);
    vector<int64_t> loop;
    for(int i=1; i<= n; ++i)
    {
        fi>>x; a[x]=i;
    }
    for(int i=1; i<=n; ++i) fi>>b[i];
    for(int i=1; i<= n; ++i)
    {
        x=a[i]; k=1;
        while(x!=b[i])
        {
            x=a[x]; ++k;
            if(x==b[i])
            {
                if(ret[k]==0) {ret[k]=1; loop.push_back((int64_t)k);}
                break;
            }
            if(x==i) {ans = "No"; break;}
        }
        if(ans=="No") break;
    }

    fo<<ans<<'\
';
    if(ans=="Yes")
    {
        int64_t z=1;
        for(int64_t i:loop) z=z*i/__gcd(z,i);
        fo<<z;
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VZ32. SIÊU THỊ

Tên chương trình: MARKETS.CPP

Con đường ven biển được quy hoạch thành khu du lịch, nghỉ dưỡng. Một bên đường là bãi biển, phía bên kia sẽ xây dựng ven theo đường các khách sạn, nhà ở và siêu thị. Khu vực xây dựng được chia thành n lô liên tiếp nhau và có cùng độ rộng, trong số đó có k lô dự kiến sẽ xây dựng siêu thị. Khách đến mua sắm thường có xu hướng tới siêu thị gần nơi ở nhất, vì vậy các siêu thị cần phân bố sao cho khoảng cách xa nhất từ mỗi nhà tới siêu thị gần nhất là nhỏ nhất.

Hãy xác định khoảng cách xa nhất từ mỗi nhà tới siêu thị gần nhất trong phương án quy hoạch tối ưu.

Dữ liệu: Vào từ file văn bản MARKETS.INP gồm một dòng chứa 2 số nguyên n và k ($2 \leq n \leq 10^{18}$, $1 \leq k < n$).

Kết quả: Đưa ra file văn bản MARKETS.OUT một số nguyên – khoảng cách tìm được.

Ví dụ:

MARKETS.INP
26 3

MARKETS.OUT
4



Giải thuật: Cơ sở lập trình.

Mỗi siêu thị thu hút khách từ nhóm các nhà ở phía trái và nhóm các nhà ở phía phải,



k siêu thị tạo thành $2 \times k$ nhóm nhà,

Số lượng nhà: $n - k$,

Gọi số lượng nhà trong nhóm là kích thước của nhóm,

Phương án tối ưu: chênh lệch kích thước giữa 2 nhóm bất kỳ là không quá 1.

Độ phức tạp của giải thuật: $O(1)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "Markets."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    uint64_t n, k, ans;
    fi >> n >> k;
    n-=k; k<<=1;
    ans = (n+k-1) / k;
    fo<<ans;
}
```



VZ33. IN PHUN 3D

Tên chương trình: P3D.CPP

Bài tập sử dụng máy in phun 3D là tạo ra các khối hộp chữ nhật từ thạch cao nhão. Hình hộp tạo ra còn khá mềm nên không thể lật hay dựng đứng hình.

Dựa vào kích thước chiều rộng, chiều dài của đáy và chiều cao của khối hộp sản phẩm sẽ được đánh giá là “**good**” hoặc “**bad**”.

Sản phẩm được đánh giá là “**good**” nếu tỷ lệ giữa cạnh bé của đáy với chiều cao phải ít nhất là 2 và tỷ lệ giữa cạnh lớn của đáy với cạnh bé của đáy không vượt quá 2.

Cho 2 kích thước đáy là **w**, **l** và chiều cao **h**.

Hãy đưa ra đánh giá đối với hình hộp chữ nhật đã cho.

Dữ liệu: Vào từ file văn bản P3D.INP gồm 3 số nguyên **w**, **l** và **h** ($10^3 \leq w, l, h \leq 10^4$), mỗi số trên một dòng.

Kết quả: Đưa ra file văn bản P3D.OUT thông báo **good** hoặc **bad**.

Ví dụ:

P3D.INP	P3D.OUT
26 3	4



VZ33 SptM20181210 A A XVI

Giải thuật; Cơ sở lập trình.

Chuẩn hóa dữ liệu: đưa về trường hợp $w \leq l$,

Thay việc tính trực tiếp tỷ lệ bằng phép so sánh tương đương chỉ sử dụng phép nhân.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "P3D."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    int w, l, h;
    fi >> w >> l >> h;
    if (w > l) swap(w, l);
    if (w >= h * 2 && l <= w * 2)
        fo << "good\n";
    else
        fo << "bad\n";
    return 0;
}
```



VZ34. HÌNH VUÔNG KHÁC NHAU

Tên chương trình: DIFFERENT.CPP

Để lát vỉa hè hay quảng trường người ta ép bột đá tạo thành các hình vuông đơn vị. Từ các hình vuông đơn vị này người ta ghép dán thành các hình vuông kích thước lớn hơn, tạo thành những viên gạch lát chịu lực tốt và vẫn để nước thẩm qua mặt lát.

Để quảng cáo cho mặt hàng mới này tại Hội chợ Công nghiệp hàng năm người ta làm các viên đá lát hình vuông kích thước khác nhau từ n viên đơn vị đang có sẵn để tạo thành sản phẩm mang ra trưng bày. Không nhất thiết phải sử dụng hết số viên đơn vị.

Hãy xác định nhiều nhất có bao nhiêu sản phẩm được mang ra trưng bày.

Dữ liệu: Vào từ file văn bản DIFFERENT.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 10^{18}$).

Kết quả: Đưa ra file văn bản DIFFERENT.OUT: một số nguyên – số lượng nhiều nhất các sản phẩm có thể mang ra trưng bày.

Ví dụ:

DIFFERENT.INP
15

DIFFERENT.OUT
3



Giải thuật: Cơ sở lập trình .

Để có nhiều hình vuông nhất cần bắt đầu từ những hình có kích thước nhỏ nhất,
Cần tìm **k** lớn nhất thỏa mãn điều kiện

$$\frac{k \times (k + 1) \times (2k + 1)}{6} \leq n$$

Sử dụng công thức trên để tìm **k** sẽ mất nhiều thời gian hơn việc tích lũy trực tiếp tổng bình phương các số tự nhiên đầu tiên và so sánh với **n**,

Để tránh khả năng tràn ô ở bước kiểm tra cuối cùng nên so sánh k^2 với phần còn lại của **n** sau mỗi lần xác định được hình vuông tiếp theo.

Độ phức tạp của giải thuật: $O(\sqrt[3]{n})$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "Different."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    int64_t n, k=0, s;
    fi >> n;
    s = n;
    while (s >=0)
    {
        k++;
        s -= k * k;
    }
    fo << k - 1 << "\n";
}

fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VZ35. ĐƯỜNG XOĂN ỐC

Tên chương trình: SPIRAL.CPP

Rô bốt di chuyển trên lưới ô vuông và vẽ một đường xoắn ốc. Ban đầu rô bót đứng ở ô tọa độ (0, 0) và hướng về phía tăng dần của tọa độ thứ nhất.

Rô bốt sẽ để lại vết sơn ở những ô mà nó đi qua, kể cả ô tọa độ (0, 0).

Quy tắc chuyển động của rô bốt là như sau: Rô bốt đi về phía trước d ô, quay sang trái và đi tiếp d ô nữa, quay sang trái và tăng d lên k lần, lặp lại các hành động đã nêu cho đến khi số ô được sơn là n .

Hãy đưa ra bức tranh dưới dạng hình chữ nhật có diện tích nhỏ nhất chứa tất cả các ô được sơn.

Dữ liệu: Vào từ file văn bản SPIRAL.INP: gồm một dòng chứa 3 số nguyên n , d và k ($1 \leq n \leq 1000$, $1 \leq d \leq 100$, $2 \leq k \leq 5$).

Kết quả: Đưa ra file văn bản SPIRAL.OUT:

- ✚ Dòng đầu tiên chứa 2 số nguyên p , q xác định số dòng và cột của hình chữ nhật kết quả,
- ✚ Mỗi dòng trong p dòng sau chứa xâu ký tự độ dài q mô tả một dòng của hình, ký tự '#' đánh dấu ô được sơn, ký tự '.' – ô trắng.

Ví dụ:

SPIRAL.INP	SPIRAL.OUT
13 2 2	5 5 ***** *...* *.*** *.... **...



VZ35 SptM20181210 C A XVI

Giải thuật: Mô phỏng ô tô mát.

Ngôn ngữ C/C++ không cho phép sử dụng chỉ số âm vì vậy cần tịnh tiến, đưa gốc tọa độ về điểm (**n**, **n**).

Số ô di chuyển cần xét của rô bốt không nhiều ($n \leq 1\ 000$), vì vậy chỉ cần theo vết đường di của rô bốt và đánh dấu các ô đã đi qua.

Gọi số ô di chuyển giữa 2 lần đổi hướng liên tiếp là bước đi.

Cần tạo các con lắc phục vụ:

- Thay đổi bước đi sau 2 lần đổi hướng,
- Hoán đổi việc thay đổi tọa độ từ trực Ox sang trực Oy và ngược lại,
- Đan xen việc tăng/giảm tọa độ tương ứng theo hướng đi.

Để xác định hình chữ nhật kết quả cần quản lý min và max tọa độ các ô được đánh dấu theo các trực tọa độ.

Tổ chức dữ liệu:

■ Các biến kiểu **int**:

- **int c = 1** – xác định tính chẵn lẻ của số lần đổi hướng,
- **int dx = 0, dy = 1** – số gia tọa độ để xác định ô tiếp theo cần đánh dấu,
- **int q = d** – số ô còn lại trong mỗi bước đi,
- **int minx = n, maxx = n, miny = n, maxy = n** – tham số xác định hình chữ nhật kết quả,

■ Mảng **vector<vector<char>> a (MAX, vector<char> (MAX, ' '))** – ghi nhận kết quả đánh dấu.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include<bits/stdc++.h>
#define NAME "spiral."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    int n, d, k;
    fi >> n >> d >> k;

    int MAX = 2 * n + 1;
    vector<vector<char>> a (MAX, vector<char>(MAX, ' '));
    int x = n, y = n;
    int dx = 0, dy = 1;
    a[x][y] = '*';
    int c = 1;
    int q = d;
    int minx = n, maxx = n;
    int miny = n, maxy = n;
    while (n > 0)
    {
        x += dx;
        y += dy;
        a[x][y] = '*';
        n--;
        q--;
        minx = min(x, minx);
        maxx = max(x, maxx);
        miny = min(y, miny);
        maxy = max(y, maxy);
        if (q == 0)
        {
            swap(dx, dy);
            dx = -dx;
            c^=1;
            if (c) d *= k;
            q = d;
        }
    }

    fo << (maxx - minx + 1) << ' ' << (maxy - miny + 1) << '\n';
    for (int i = minx; i <= maxx; i++)
    {
        for (int j = miny; j <= maxy; j++) fo << a[i][j];
        fo << '\n';
    }

    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VZ36. KHÔNG CÓ ĐIỂM BẤT ĐỘNG

Tên chương trình: NOFIX.CPP

Hoán vị các số từ 1 đến n là dãy số nguyên a_1, a_2, \dots, a_n , trong đó $1 \leq a_i \leq n$ và $a_i \neq a_j$ nếu $i \neq j$, với $1 \leq i, j \leq n$. Ví dụ, hoán vị các số nguyên từ 1 đến 3 là các dãy số (1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2) và (3, 2, 1).

Trong hoán vị a_1, a_2, \dots, a_n phần tử a_i được gọi là điểm bất động nếu $a_i = i$. Hoán vị (3, 1, 2) không có điểm bất động, hoán vị (2, 1, 3) có điểm bất động $a_3 = 3$. Với $n = 3$ có 2 hoán vị không có điểm bất động: (2, 1, 3) và (3, 1, 2).

Với n cho trước, xét các hoán vị các số từ 1 đến n không chứa điểm bất động và sắp xếp chúng theo thứ tự từ điển.

Hãy đưa ra t hoán vị đầu tiên trong dãy các hoán vị đã sắp xếp.

Dữ liệu: Vào từ file văn bản NOFIX.INP gồm một dòng chứa 2 số nguyên n và t ($2 \leq n \leq 1000$, $1 \leq t \leq 10^4$, $n \times t \leq 10^5$), dữ liệu đảm bảo có t hoán vị thỏa mãn điều kiện tìm kiếm.

Kết quả: Đưa ra file văn bản NOFIX.OUT theo thứ tự từ điển các hoán vị tìm được, mỗi hoán vị trên một dòng.

Ví dụ:

NOFIX.INP	NOFIX.OUT
4 2	2 1 4 3 2 3 1 4



VZ36 SptM20181210 D A XVI

Giải thuật: *Khởi tạo hoán vị có điều kiện.*

Ký hiệu a_1, a_2, \dots, a_n là một hoán vị của các số tự nhiên từ 1 đến n .

Trong thư viện của C++ có hàm `next_permutation(a.begin(), a.end())` cho hoán vị có số thứ tự từ điển tiếp theo của hoán vị đưa vào.

Với mỗi i cố định có $(n-1)!$ hoán vị trong đó i đứng ở vị trí i .

Nếu sử dụng hàm hệ thống để khởi tạo hoán vị ta phải mất $O(n)$ phép kiểm tra xem có tồn tại k bằng a_k hay không *với mỗi hoán vị nhận được*. Độ phức tạp của giải thuật sẽ xấp xỉ $O(n!)$.

Để giải bài toán có hiệu quả ta cần xây dựng hàm trực tiếp khởi tạo hoán vị theo thứ tự từ điển và lồng vào đó việc kiểm tra loại bỏ các hoán vị không phù hợp, *chuyển tiếp tục sang hoán vị có thứ tự từ điển tiếp theo*.

Hàm khởi tạo và dẫn xuất **t** hoán vị đầu tiên theo thứ tự từ điển:

```
#include <bits/stdc++.h>
#define NAME "Nofix."
using namespace std;
ifstream fi_(NAME"inp");
ofstream fo_(NAME"out");
vector<int> a;
vector<bool> used;
int n, t;

void gen(int p)
{
    if (p == n)
    {
        for (int x : a) fo_ << x << " ";
        fo_ << "\n";
        t--;
        return;
    }
    for (int i = 0; i < n; i++)
        if (!used[i])
        {
            used[i] = true;
            a[p] = i + 1;
            gen(p + 1);
            used[i] = false;
            if (t == 0) break;
        }
}

int main()
{
    n=4; t=10;
    a.resize(n);
    used.assign(n, false);
    gen(0);
    next_permutation(a.begin(), a.end());
    fo_ << "\n";
    for(int i:a) fo_ << i << " ";
    fo_ << "\nTime: " << clock()/(double)1000 << ".sec";
}
```

Cần bổ sung điều kiện loại bỏ điểm cố định

1	2	3	4
1	2	4	3
1	3	2	4
1	3	4	2
1	4	2	3
1	4	3	2
2	1	3	4
2	1	4	3
2	3	1	4
2	3	4	1
.....			
2	4	1	3
Time: 0.008 sec			

Hàm dẫn xuất t hoán vị đầu tiên không có điểm bất động:

```
void gen(int p)
{
    if (p == n)
    {
        for (int x : a) fo << x << " ";
        fo << "\n";
        t--;
        return;
    }
    for (int i = 0; i < n; i++)
        if (!used[i] && i != p)
        {
            used[i] = true;
            a[p] = i + 1;
            gen(p + 1);
            used[i] = false;
            if (t == 0) break;
        }
}
```

gen(0);

Lời gọi:

Độ phức tạp của giải thuật: $O(n \times t)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "Nofix."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
vector<int> a;
vector<bool> used;
int n, t;

void gen(int p)
{
    if (p == n)
    {
        for (int x : a) fo << x << " ";
        fo << "\n";
        t--;
        return;
    }
    for (int i = 0; i < n; i++)
        if (!used[i] && i != p)
        {
            used[i] = true;
            a[p] = i + 1;
            gen(p + 1);
            used[i] = false;
            if (t == 0) break;
        }
}
int main()
{
    fi >> n >> t;
    a.resize(n);
    used.assign(n, false);
    gen(0);
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VZ37. ƯỚC SỐ CHUNG LỚN NHẤT

Tên chương trình: GCD.CPP

Ước số chung lớn nhất của dãy số nguyên dương \mathbf{A} không rỗng là số nguyên dương d lớn nhất đồng thời là ước của mọi số trong dãy \mathbf{A} .

Cho mảng số nguyên dương a_1, a_2, \dots, a_n và số nguyên k .

Hãy tìm đoạn $a_i, a_{i+1}, \dots, a_{i+k-1}$ có ước số chung lớn nhất và đưa ra ước số chung đó.

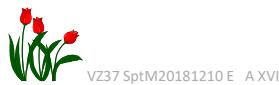
Dữ liệu: Vào từ file văn bản GCD.INP:

- ✚ Dòng đầu tiên chứa số nguyên n và k ($2 \leq n \leq 5 \times 10^5$, $2 \leq k \leq n$,
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^{18}$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản GCD.OUT một số nguyên – ước số chung lớn nhất tìm được.

Ví dụ:

GCD.INP	GCD.OUT
10 4 2 3 4 8 12 6 12 18 4 3	6



Giải thuật: Cây phân đoạn có cập nhật.

Phép lấy gcd có tính giao hoán và kết hợp:

- $gcd(x, y) = gcd(y, x)$
- $gcd(x, y, z) = gcd(x, gcd(y, z)) = gcd(gcd(x, y), z)$.

Để tiện xử lý các phần tử của mảng dữ liệu vào được đánh số bắt đầu từ 0.

Sử dụng cây phân đoạn t quản lý gcd . Ban đầu các nút lá của t chứa a_0, a_1, \dots, a_{k-1} . t_1 sẽ chứa gcd của k phần tử đầu tiên của dãy a .

Lần lượt cập nhật cây, thay a_0 bằng a_k , a_1 bằng a_{k+1}, \dots . Nói một cách khác, phần tử a_i sẽ được đưa vào thay thế cho phần tử nút lá thứ $i \% k$ của t và tính lại gcd chung cho các phần tử của nút lá, sau đó – cập nhật gcd kết quả.

Việc cập nhật cây được thực hiện với chi phí $O(lnk)$.

Số lần cập nhật cần thực hiện là $n-k$, như vậy *độ phức tạp của giải thuật* sẽ là $O(nlnk)$.

Tổ chức dữ liệu: Cần mảng t kích thước $4 \times k + 5$, mỗi phần tử thuộc loại nguyên 64 bits để tổ chức cây phân đoạn.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "Gcd."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,p,k,q;
vector<int64_t> a;
int64_t ans=0,x;

void upd(int i, int64_t val)
{
    int u=p+i;
    a[u]=val; u>>=1;
    while(u>0)
    {
        a[u]=__gcd(a[2*u],a[2*u+1]); u>>=1;
    }
}

int main()
{
    fi>>n>>k;
    for(int i=20; i>=0; --i)
    if((k>>i)&1){p=1<<i; break;}
    if((k&(-k))!=k)p<<=1;
    a.assign(4*k+5,0);
    for(int i=0; i<k; ++i)fi>>a[p+i];
    for(int i=p-1; i>0; --i)a[i]=__gcd(a[2*i],a[2*i+1]);
    ans=a[1];
    for(int i=k; i<n; ++i)
    {
        fi>>x;
        upd(i%k,x);
        if(ans<a[1]) ans=a[1];
    }

    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VZ38. DÃY CHÚA MAX

Tên chương trình: NUMMAX.CPP

Xét dãy số nguyên $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$. Dãy chứa các phần tử ở các vị trí liên tiếp của \mathbf{A} được gọi là dãy con. Hai dãy con được gọi là khác nhau nếu tồn tại ít nhất một vị trí mà phần tử của \mathbf{A} ở vị trí đó tham gia vào dãy con này và không tham gia vào dãy con kia.

Cho số nguyên b . Hãy xác định số lượng dãy con có giá trị lớn nhất của các phần tử trong dãy con bằng b .

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

- ⊕ Dòng đầu tiên chứa số nguyên n và b ($2 \leq n \leq 10^5$, $1 \leq b \leq 10^9$),
- ⊕ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^9$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản NUMMAX.OUT: một số nguyên – số lượng dãy con tìm được.

Ví dụ:

NUMMAX.INP	NUMMAX.OUT
4 5 1 3 5 2	6



VZ38 Ucr_14 B A XVI

Giải thuật: Kỹ thuật 2 con trỏ, nguyên lý bù – trừ.

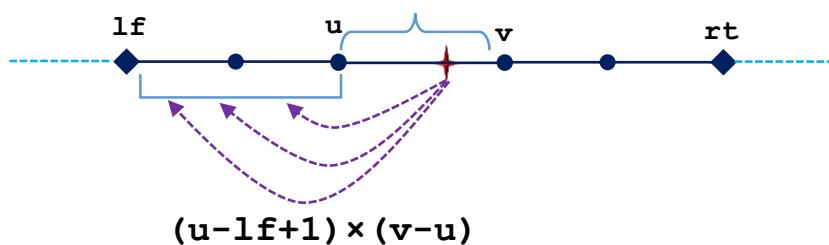
Tạo hàng rào: $a_0 = a_{n+1} = 10^9 + 1$.

Lần lượt từ trái qua phải tìm đoạn $[lf, rt]$ dài nhất các phần tử liên tiếp có giá trị không vượt quá b ,

Với mỗi đoạn $[lf, rt]$:

Duyệt a_i , $i = lf \div rt$: nạp các i có $a_i = b$ vào hàng đợi q .

Với u và v – hai phần tử liên tiếp trong hàng đợi, $u < v$, $v = rt$ nếu u là phần tử cuối cùng trong hàng đợi, số đoạn thỏa mãn điều kiện đầu bài và chứa phần tử a_u là $(u-lf+1) \times (v-u)$.



Mỗi đoạn với u mới không trùng với các đoạn đã tính ở u cũ vì có chứa phần tử a_u mới, trong một đoạn mở $[u, v)$ các đoạn được đếm khác nhau bởi điểm đầu hoặc điểm cuối hay cả hai.

Không có đoạn nào bị bỏ sót vì mỗi cực trị đều tham gia vào mọi đoạn chứa nó.

Tổ chức dữ liệu

- ─ Mảng `vector<int>` a – chứa dữ liệu input,
- ─ Hàng đợi `queue<int>` q – chứa tọa độ các cực trị trong một đoạn $[lf, rt]$.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("Nummax.inp");
ofstream fo ("Nummax.out");
int n, b, lf=0, rt=0, u, v;
int64_t ans=0;
vector<int> a;
queue<int> q;

int main()
{
    fi>>n>>b;
    a.resize(n+2);
    for(int i=1; i<=n; ++i) fi>>a[i];
    a[0]=a[n+1]=1e9+1;
    while(lf<=n)
    {
        while(a[lf]>b && lf<=n) ++lf;
        rt=lf;
        while(a[rt]<=b) ++rt;
        for(int i=lf; i<rt; ++i) if(a[i]==b) q.push(i);
        while(!q.empty())
        {
            u=q.front(); q.pop();
            if(q.empty()) v=rt; else v=q.front();
            ans+=(u-lf+1)*(v-u);
        }
        lf=rt+1;
    }
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VZ39. BIẾN ĐỔI SỐ

Tên chương trình: TRANSFORM.CPP

Cho hai phép biến đổi một số nguyên dương x :

- Tăng x lên 1,
- Tạo số mới không bắt đầu bằng chữ số 0 bằng cách đổi chỗ tùy ý các chữ số trong x .

Dễ dàng thấy rằng các phép biến đổi trên cho phép xuất phát từ $x = 1$ có thể nhận được số nguyên dương y bất kỳ.

Hãy xác định số phép biến đổi ít nhất để từ 1 nhận được số nguyên dương y cho trước.

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

- Dòng đầu tiên chứa số nguyên t – số lượng tests ($1 \leq t \leq 1000$),
- Mỗi dòng trong t dòng sau chứa một số nguyên y ($1 \leq y \leq 10^{18}$).

Kết quả: Đưa ra file văn bản TRANSFORM.OUT số phép biến đổi ít nhất cần thực hiện với mỗi số đã cho, mỗi kết quả đưa ra trên một dòng.

Ví dụ:

TRANSFORM.INP	TRANSFORM.OUT
3	5
5	17
26	44
843	



Giải thuật: Cáp số cộng, xử lý xâu.

$$z_n = \underbrace{100 \dots}_{n-1 \text{ số } 0} . \dots 0$$

Gọi n là số chữ số có nghĩa của y .

Để tạo được y trước hết cần tạo z_n :

Xuất phát từ $z_1 = 1$ tạo được $z_2 = 10$ với $9 = 10 - 1$ phép biến đổi.

Từ $z_2 = 10$ tạo được $z_3 = 100$ với $19 = 20 - 1$ phép biến đổi:

$$\begin{array}{ccccccc} 10 & \rightarrow & 19 & \rightarrow & 91 & \rightarrow & 99 \\ \text{Số phép biến đổi:} & & 9 & & 1 & & 8 \end{array} \rightarrow 100 \quad 1$$

Từ $z_3 = 100$ tạo được $z_4 = 1000$ với $29 = 30 - 1$ phép biến đổi:

$$\begin{array}{ccccccc} 100 & \rightarrow & 109 & \rightarrow & 190 & \rightarrow & 199 \\ \text{Số phép biến đổi:} & & 9 & & 1 & & 9 \end{array} \rightarrow \begin{array}{ccccccc} 991 & \rightarrow & 999 & \rightarrow & 1000 \\ 1 & & 8 & & 1 \end{array}$$

Bằng phương pháp quy nạp dễ dàng chứng minh được rằng từ z_{n-1} để dẫn xuất z_n cần $(n-1) \times 10 - 1$ phép biến đổi.

Như vậy, số phép biến đổi cần thiết để từ 1 dẫn xuất ra z_n là

$$\sum_{i=1}^{n-1} (10 \times i - 1)$$

Giả thiết $y = \overline{a_0 a_1 a_2 \dots a_{n-1}}$, trong đó $0 \leq a_i \leq 9$, $a_0 > 0$.

Từ z_n , nếu $a_0 \neq 1$ thì ta cần $a_0 + 1$ phép biến đổi để đưa chữ số trái nhất về a_0 .

$$100\dots00 \xrightarrow{a_0} 100\dots0a_0 \xrightarrow{1} a_000\dots01$$

Nếu $a_0 = 1$ – không cần thực hiện biến đổi.

Với mỗi $a_i \neq 0$ tiếp theo *chưa phải là chữ số cuối cùng*:

Trường hợp $a_0 \neq 1$:

- Để nhận được chữ số $a_i \neq 0$ tiếp theo cần thực hiện $(a_{i-1})+1=a_i$ phép biến đổi,
- Với các $a_i \neq 0$ còn lại: Cần thực hiện a_i+1 phép biến đổi.

Trường hợp $a_0 = 1$: Cần thực hiện a_i+1 phép biến đổi.

Với *chữ số cuối cùng*: xét tương tự, nhưng bớt 1 phép biến đổi vì không cần đổi vị trí.

Trường hợp đặc biệt: y chỉ chứa một chữ số khác không (ví dụ, $y = 600$), khi đó cần tính số phép biến đổi để nhận được $y-1$ và thêm 1 nữa để nhận được y .

Các số cần xử lý nên lưu ở dạng xâu để dễ dàng tính độ dài (số lượng chữ số) cũng như lấy ra từng chữ số.

Với mỗi số, độ phức tạp của giải thuật là $O(n)$, trong đó n là độ dài của số cần xử lý. Do $n \leq 18$, ta có thể coi độ phức tạp xử lý mỗi số là $O(1)$.

Như độ phức tạp độ phức tạp của toàn bài toán là $O(t)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("Transform.inp");
ofstream fo ("Transform.out");
int t, ans, n, m;
string y;

int solve(string s)
{
    int res = 0, nz = 0, d = 0;
    n=s.size();
    if(n==1) return (s[0]-49);
    for(int i=n-1; i>=0; --i)
        if(s[i]!='0') {m=i; break;}
    if(m==0)
    {
        ++res; s[0]=s[0]-1;
        for(int i=1; i<n; ++i) s[i]='9';
        if(s[0]=='0') {s.erase(0,1); --n;}
        m=n-1;
    }
    if(n==1) return (s[0]-48);
    res += n*(n-1)*5 - n + 1;
    for(int i=0; i<n; ++i)
    {
        res += s[i]-48;
        nz += s[i]!='0';
    }
    if(s[0]=='1') if(s[n-1]=='0') d=2; else d=3;
    if(s[0]!='1') if(s[n-1]=='0') d=1; else d=2;
    res += (nz-d);
    return res;
}

int main()
{
    fi>>t;
    for(int i=0; i<t; ++i)
    {
        fi>>y;
        ans = solve(y);
        fo<<ans<<'\n';
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Những tấm làm từ vật liệu dòn như thủy tinh hoặc gốm sứ khi bẻ sẽ cho vết gãy trơn và sắc. Tâm làm từ vật liệu chịu lực khi bị bẻ sẽ cho vết gãy sần sùi, nhấp nhô.

Độ bền của tấm vật liệu có thể đánh giá qua độ mấp mô của vết gãy.

Xét đường gãy của một tấm vật liệu composit. Đường gãy được chia thành n phần, độ cao trung bình ở phần thứ i của vết gãy là a_i , $i = 1 \div n$. Độ bền của tấm vật liệu được xác định như là số dãy con khác nhau các phần tử liên tiếp của dãy $\mathbf{A} = (a_1, a_2, \dots, a_n)$. Ví dụ, với $\mathbf{A} = (1, 2, 1, 2)$ ta có các dãy con khác nhau $(1), (2), (1, 2), (2, 1), (1, 2, 1), (2, 1, 2)$ và $(1, 2, 1, 2)$. Như vậy, độ bền của tấm vật liệu này được đánh giá là 7.

Cho 2 tấm vật liệu \mathbf{U} và \mathbf{V} có đường gãy được xác định tương ứng với các dãy số (u_1, u_2, \dots, u_n) và (v_1, v_2, \dots, v_m) . Khi ghép 2 tấm này thành một tấm, đường gãy của tấm ghép sẽ là hợp của 2 đường gãy ban đầu.

Hãy xác định ghép V vào bên phải hay bên trái của U sẽ cho tấm ghép có độ bền lớn hơn và đưa ra độ bền đó.

Dữ liệu: vào từ file văn bản STRENGTH.INP:

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên u_1, u_2, \dots, u_n ($1 \leq u_i \leq 10^3$, $i = 1 \div n$),
- ✚ Dòng thứ 3 chứa số nguyên m ($1 \leq m \leq 10^5$),
- ✚ Dòng thứ 4 chứa m số nguyên v_1, v_2, \dots, v_m ($1 \leq v_j \leq 10^3$, $j = 1 \div m$).

Kết quả: Đưa ra file văn bản STRENGTH.OUT một số nguyên – độ bền lớn nhất có thể đạt được.

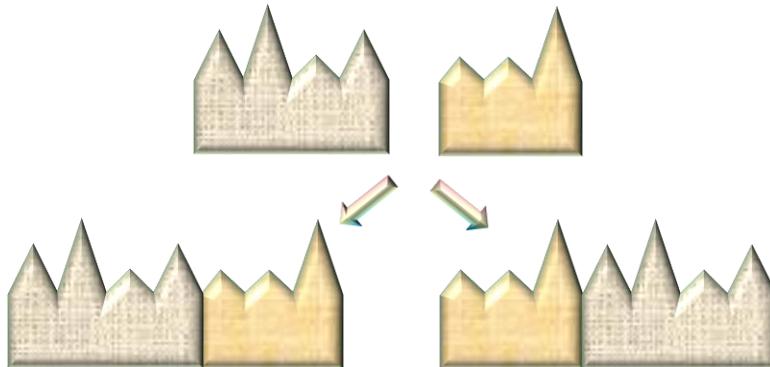
Ví dụ:

STRENGTH.INP	STRENGTH.OUT
<pre>4 1 2 1 2 4 1 2 3 1</pre>	<pre>29</pre>



Giải thuật: Tìm số lượng dãy con khác nhau .

Việc ghép các tám \mathbf{u} và \mathbf{v} có thể thực hiện theo 2 cách: ghép \mathbf{u} vào bên trái của \mathbf{v} hoặc ghép vào bên phải của \mathbf{v} .



Đường gãy của tám ghép tương ứng với dãy $n+m$ phần tử. Mỗi cách ghép sinh ra một đường gãy của mình.

Vấn đề còn lại là tìm số dãy con khác nhau ứng với mỗi đường gãy hình thành và chọn cách ghép cho nhiều dãy con khác nhau nhất.

Đặt $n = n+m$ và gọi dãy số nhận được là \mathbf{a} .

Tính số dãy con khác nhau của một dãy số nguyên không âm: Quá trình xử lý được thực hiện theo giải thuật tìm số lượng xâu con khác nhau của một xâu.

Xét các dãy con hậu tố của \mathbf{a} .

Trước hết tính độ dài của tiền tố dài nhất đối với mỗi cặp hậu tố liên tiếp nhau theo thứ tự từ điển, trên cơ sở đó – tính số dãy con mới bắt đầu từ các vị trí p_0, p_1, \dots . Vì các hậu tố được sắp xếp theo thứ tự từ điển nên hậu tố p_i đang xét sẽ cho tất cả các tiền tố mới của mình khác với các tiền tố do p_{i-1} cung cấp ngoại trừ những dãy con thuộc tiền tố chung dài nhất, tức là $1cp_{i-1}$ dãy con đầu tiên.

Độ dài của hậu tố \mathbf{p}_i đang xét là $n - \mathbf{p}_i$ nên số lượng dãy con khác nhau mới sẽ là $n - \mathbf{p}_i - 1$. Riêng hậu tố đầu tiên (\mathbf{p}_0) không sản sinh các dãy con trùng lặp và cho số lượng dãy con là $n - \mathbf{p}_0$. Như vậy kết quả cần tính sẽ là

$$\sum_{i=0}^n (n - p[i]) - \sum_{i=0}^{n-1} \text{lcp}[i]$$

Chi tiết về giải thuật tính **lcp** cho các dãy hậu tố: Xem ở tập IX.

Dộ phức tạp của giải thuật: O((n+m)log(n+m)).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "divert_1000."      //diff subsequentions u+v, v+u val - 1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int maxlen = 500010;
const int alph = 1000;
int n,l,k,m,len[100010],pl[maxlen],pr[maxlen];
vector<int> a,u,v;
vector<int>suff,ord,Lcp;
int64_t ans,r1,r2;

vector<int> buildLCP(vector<int> s,vector<int>& p)
{ int pos[maxlen];
  vector<int>lcp;
  lcp.reserve(n);
  for(int i=0;i<n;++i) pos[p[i]]=i;
  int k=0;
  for(int i=0;i<n;++i)
  {
    if(k>0) --k;
    if(pos[i]==n-1) lcp[n-1]=-1, k=0;
    else
    {
      int j=p[pos[i]+1];
      while(max(i+k,j+k)<n && (s[i+k]==s[j+k])) ++k;
      lcp[pos[i]]=k;
    }
  }
  return lcp;
}

vector<int> calc_suff(vector<int> s)
{vector<int>p,c,cnt,Lcp;
 p.reserve(maxlen);cnt.reserve(maxlen);
 c.reserve(maxlen);Lcp.reserve(maxlen);

 fill_n(cnt.begin(),maxlen,0);
 for (int i=0; i<n; ++i)++cnt[s[i]];
 for (int i=1; i<alph; ++i)cnt[i] += cnt[i-1];
 for (int i=0; i<n; ++i)p[--cnt[s[i]]] = i;
 c[p[0]] = 0;
 int classes = 1;
 for (int i=1; i<n; ++i)
```

```

{
    if (s[p[i]] != s[p[i-1]]) ++classes;
    c[p[i]] = classes-1;
}
vector<int> pn(maxlen), cn(maxlen);
for (int h=0; (1<<h)<n; ++h)
{
    for (int i=0; i<n; ++i)
    {
        pn[i] = p[i] - (1<<h);
        if (pn[i] < 0) pn[i] += n;
    }
    fill_n(cnt.begin(), maxlen, 0);
    for (int i=0; i<n; ++i) ++cnt[c[pn[i]]];
    for (int i=1; i<classes; ++i) cnt[i] += cnt[i-1];
    for (int i=n-1; i>=0; --i) p[--cnt[c[pn[i]]]] = pn[i];
    cn[p[0]] = 0;
    classes = 1;
    for (int i=1; i<n; ++i)
    {
        int mid1 = (p[i]+(1<<h))%n, mid2=(p[i-1] + (1<<h))% n;
        if (c[p[i]]!=c[p[i-1]]||c[mid1]!= c[mid2]) ++classes;
        cn[p[i]] = classes-1;
    }
    c=cn;
}
return p;
}

int main()
{
    fi>>l;
    u.resize(l);
    ord.reserve(maxlen); Lcp.reserve(maxlen);
    for(int &i:u) fi>>i;
    fi>>k;
    v.resize(k);
    for(int &i:v) fi>>i;
    n=k+l+1;
    a.resize(n);
    for(int i=0; i<l;++i) a[i]=u[i];
    for(int i=0; i<k;++i) a[i+l]=v[i];
    a[n-1]=0;
    suff=calc_suff(a);
    Lcp=buildLCP(a, suff);
    r1=n-suff[0];
    for(int i=1;i<n;++i) r1+=(n-suff[i]);
}

```

```
for(int i=0; i<n; ++i) r1-=Lcp[i];

for(int i=0; i<k; ++i) a[i]=v[i];
for(int i=0; i<l; ++i) a[i+k]=u[i];
suff=calc_suff(a);
Lcp=buildLCP(a, suff);
r2=n-suff[0];
for(int i=1; i<n; ++i) r2+=(n-suff[i]);
for(int i=0; i<n; ++i) r2-=Lcp[i];
ans=max(r1, r2);
fo<<ans-n-1;

fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
}
```



VZ41. HÁI VIỆT QUẤT

Tên chương trình: BLUEBERRIES.CPP

Việt quất phải thu hoạch trực tiếp bằng tay từng quả vì vậy giá thành khá cao. Viện Nghiên cứu Máy móc nông nghiệp đã chế tạo thành công máy thu hoạch tự động nhận dạng quả chín và hái.

Máy chạy ở chế độ tự động theo luồng, ở chế độ thăm dò và hái mỗi giờ xử lý được k mét. Những quả còn ương sẽ bị bỏ qua, ít nhất sau một giờ nữa mới chín và có thể hái được. Việc thăm dò khảo sát trong một chừng mực nào đó ảnh hưởng đến cây, vì vậy người ta cố tránh xử lý cây khi biết chắc các quả chín ở đó đã được hái hết.



Luồng có độ dài n mét. Ở phiên bản thử nghiệm hiện nay chỉ mới có một máy hái tự động. Thời gian đưa máy tới nơi bất kỳ là không đáng kể. Mỗi lần chuyển sang chế độ thu hoạch máy phải xử lý các đoạn có độ dài là bội của k .

Hãy xác định tối thiểu sau bao nhiêu giờ mỗi cây trên luồng được xử lý ít nhất một lần.

Dữ liệu: vào từ file văn bản BLUEBERRIES.INP gồm một dòng chứa 2 số nguyên n và k ($1 \leq k \leq n \leq 10^6$).

Kết quả: Đưa ra file văn bản BLUEBERRIES.OUT một số nguyên – số giờ tối thiểu xác định được.

Ví dụ:

BLUEBERRIES.INP
5 2

BLUEBERRIES.OUT
3



VZ41 Ucr II 20190424 A A XVI

Giải thuật: Kỹ năng phân tích giải thuật.

Phân biệt 3 trường hợp.

Trường hợp n chia hết cho k : số giờ là n/k .

$$n = 30, k = 5$$



Trường hợp n không chia hết cho k và $n > 2 \times k$:

$$n = 16, k = 6$$



Giờ thứ I:



Giờ thứ II:



Giờ thứ III:



Trường hợp n không chia hết cho k và $n < 2 \times k$:

$$n = 16, k = 10$$



Giờ thứ I:



Giờ thứ II:



Giờ thứ III:



Độ phức tạp của giải thuật: $O(1)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
int n, k, ans;

int main()
{
    ifstream cin ("blueberries.inp");
    ofstream cout ("blueberries.out");
    cin>>n>>k;
    ans=(n+k-1)/k;
    if(k!=n && k*2 > n) ++ans;
    cout<<ans;
}
```



VZ42. CHAT

Tên chương trình: CHAT.CPP

Steve có nhiệm vụ xây dựng hệ thống quản lý trò chuyện (*Chat*) trên mạng, phân tích chủ đề trò chuyện, ghi nhận từ khóa tìm kiếm, ...

Bước sang giai đoạn thử nghiệm Steve cần một số người chat với nhau để kiểm tra chất lượng hệ thống và sửa lỗi. Hệ thống động chạm đến vấn đề tế nhị là quyền bảo vệ thông tin riêng tư của người dùng vì vậy Công ty đề xuất Steve tìm thuê người ngoài thực hiện chat và nhóm người tham gia là ít nhất có thể.

Để test mỗi người trong nhóm phải soạn một văn bản có **a** từ và gửi tới tất cả những người còn lại trong nhóm. Người nhận phải trả lời cho người gửi bằng văn bản chứa **b** từ.

Ví dụ, nhóm có 3 người, **a** = 2 và **b** = 1. Người thứ nhất có thể gửi cho người thứ 2 và 3 câu “*Chào bạn*”, người thứ 2 gửi cho người thứ nhất và thứ 3 câu “*Gửi chúc*”, người thứ 3 gửi cho người thứ nhất và thứ 2 câu “*Ngủ chưa*”. Người thứ nhất trả lời cho hai người còn lại bằng các thông báo “2” và ”*Thank*”, câu trả lời của người thứ 2 cho người thứ nhất và người thứ 3 là “2” và “*Merci*”, còn câu trả lời của người thứ 3 là “*Hi*” và “*Gracia*”. Tổng cộng có 12 từ được trao đổi qua mạng.

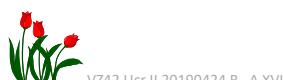
Hãy xác định số người tối thiêu cần có để số lượng từ trao đổi không ít hơn **n**.

Dữ liệu: Vào từ file văn bản CHAT.INP gồm một dòng chứa 3 số nguyên **n**, **a** và **b** ($1 \leq n \leq 10^{12}$, $0 \leq a, b \leq 10^6$, $0 < a+b$).

Kết quả: Đưa ra file văn bản CHAT.OUT một số nguyên – số người tối thiêu cần có trong nhóm.

Ví dụ:

CHAT.INP	CHAT.OUT
10 1 3	3



VZ42 Ucr II 20190424 B A XVI

Giải thuật: Tìm kiếm nhị phân .

Xét trường hợp có **k** người tham gia trong nhóm test hệ thống.

Tổng số từ *mỗi người phải soạn* để gửi thông báo đến người khác là **$k \times a$** ,

Tổng số từ *một người* phải soạn để trả lời cho những người còn lại là **$(k-1) \times b$** ,

Như vậy số từ được trao đổi trên mạng sẽ là

$$t = k \times a + k \times (k-1) \times b$$

Giá trị **t** tăng theo **k**, vì vậy có thể bằng phương pháp tìm kiếm nhị phân xác định **k** nhỏ nhất để **$t \geq n$** .

Trường hợp riêng: nếu **b = 0** $\rightarrow t = k \times a \rightarrow$ kết quả cần tìm là $\frac{n+a-1}{a}$.

Độ phức tạp của giải thuật: O(lnn).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
int64_t n, a, b, ans;

int main()
{
    ifstream cin ("chat.inp");
    ofstream cout ("chat.out");
    cin>>n>>a>>b;
    if(b==0) ans = (n+a-1)/a;
    else
    {
        int l=0, r=2000000;
        while(r-l>1)
        {
            int64_t mid=(r+l)/2;
            if(mid*a+mid*(mid-1)*b >= n) r=mid; else l=mid;
        }
        ans=r;
    }
    cout<<ans;
    cout<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VZ43. ĐỘI HÌNH THI ĐẤU

Tên chương trình: TEAM.CPP

Các môn thể thao phối hợp đòi hỏi vận động viên phải có sức mạnh và sự dẻo dai. Câu lạc bộ thể thao của nhà trường có n bạn tham gia, người thứ i có sức mạnh a_i và độ dẻo dai là b_i , $i = 1 \dots n$.

Đội thi đấu có k người. Trong đội sẽ có một đội trưởng, những người còn lại là thành viên. Tiềm năng của đội được đánh giá bằng tổng sức mạnh của đội trưởng với độ dẻo dai của các thành viên.

Để chuẩn bị đấu giao hữu với trường bạn, huấn luyện viên quyết định sẽ đưa ra đội hình có tiềm năng thấp nhất, chủ yếu là tạo điều kiện cho mọi người có dịp cọ xát với thực tế, đồng thời cũng thử nghiệm các chiến thuật thi đấu.

Ví dụ, với $n = 4$, sức mạnh và độ dẻo dai của mọi người tương ứng là $\mathbf{A} = (3, 7, 1, 6)$ và $\mathbf{B} = (6, 3, 8, 5)$. Nếu $k = 3$ thì để có đội hình tiềm năng thấp nhất cần chọn các người 2, 3, 4 và chỉ định người 3 làm đội trưởng. Khi đó tiềm năng của đội sẽ là $1+3+5 = 9$ – thấp nhất có thể!

Do không biết trước lần này cần phải cử bao nhiêu người đi nên huấn luyện viên phải lên phương án cho mọi khả năng với k từ 1 đến n .

Hãy đưa ra tiềm năng thấp nhất của đội được cử đi với k lần lượt nhận giá trị từ 1 đến n .

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

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 2 \times 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên a_i và b_i ($1 \leq a_i, b_i \leq 10^9$).

Kết quả: Đưa ra file văn bản TEAM.OUT n số nguyên – các tiềm năng thấp nhất tính được, mỗi số trên một dòng, số thứ i ứng với trường hợp $k = i$.

Ví dụ:

TEAM.INP	TEAM.OUT
4	1
3 6	4
7 3	9
1 8	
6 5	15



VZ43 Ucr II 20190424 C A XVI

Giải thuật: Nguyên lý cực trị.

Sắp xếp dữ liệu theo chiều tăng dần của cặp giá trị (b_i, a_i) , $i = 1 \dots n$.

Xét tiềm năng của đội k người. Gọi \mathbf{x} là sức mạnh của đội trưởng và \mathbf{y} – tổng độ dẻo dai của các người còn lại.

Dễ dàng thấy rằng chỉ có thể có 2 cách chọn:

- + Chọn k người đầu tiên trong dãy đã sắp xếp,
- + Chọn $k-1$ người đầu tiên trong dãy đã sắp xếp và một người trong số $n-k$ người (từ vị trí $k+1$ đến n) còn lại làm đội trưởng.

Tùy 2 cách chọn trên rút ra đội có tiềm năng nhỏ hơn.

Gọi ia là chỉ số nhỏ nhất thỏa mãn điều kiện $a_{ia} \leq a_i$ với $i = 1 \dots n$ trong dãy đã sắp xếp.

Nếu $ia > k$ thì đội cần chọn sẽ bao gồm $k-1$ người đầu tiên trong dãy đã sắp xếp làm thành viên và người thứ ia làm đội trưởng.

Nếu $ia \leq k$ – đội cần chọn sẽ gồm k người đầu tiên. Vấn đề còn lại – xác định đội trưởng.

Gọi s là tổng độ dẻo dai của k người đầu tiên.

$$s = \sum_{i=1}^k b_i$$

$$d = \max\{b_i - a_i, i = 1 \dots k\}$$

Tiềm năng cần tìm sẽ là $s-d$.

Các giá trị s và d được cập nhật với chi phí $O(1)$ khi chuyển từ k sang $k+1$.

Tổ chức dữ liệu

- ─ Mảng `vector<pll>` ba (n) – lưu các cặp giá trị nguyên 64 bits (b_i, a_i) ,
- ─ Mảng `vector<int64_t>` f (n) – lưu kết quả cần tìm.

Độ phức tạp của giải thuật: $O(n \log n)$.

Chương trình

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
typedef pair<int64_t,int64_t> pll;
int n,ia;
int64_t a,b,t,s=0;

int main()
{
    ifstream cin ("team.inp");
    ofstream cout ("team.out");
    cin>>n;
    vector<pll> ba(n);
    vector<int64_t> f(n);
    for(int i=0; i<n; ++i)
    {
        cin>>a>>b;
        ba[i]={b,a};
    }
    sort(ba.begin(),ba.end());
    a=1e9+1; b=0;
    for(int i=0; i<n; ++i) if(ba[i].ss<a) {a=ba[i].ss; ia=i;}
    for(int i=0; i<n; ++i)
    {
        t=ba[i].ff-ba[i].ss;
        if(b<t) b = t;
        if(i<ia) {f[i]=s+a; s+=ba[i].ff; continue;}
        s+=ba[i].ff;
        f[i]=s-b;
    }

    for(int i=0; i<n; ++i) cout<<f[i]<<'\n';

    cout<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VZ44. VẬN CHUYỂN

Tên chương trình: TRANSPORT.CPP

Công ty bán hàng qua mạng nhận được n đơn đặt hàng tới một tòa nhà cao tầng. Các hàng hóa được tập kết tại sảnh ở tầng 1, từ đó các rô bốt khác nhau sẽ mang hàng đến nơi đặt trong tòa nhà. Trong tòa nhà có một thang máy miễn phí vận chuyển hàng đi lên và một thang máy vận chuyển hàng dịch vụ, chi phí tỷ lệ với số tầng đi lên và tính riêng với từng người hoặc rô bốt sử dụng.

Trong thang máy miễn phí các nút bấm chọn tầng được đặt thành một hàng dọc, nút bấm tầng 1 ở độ cao h , nút bấm tầng 2 ở độ cao $h+1, \dots$, nút bấm tầng k ở độ cao $h+k-1$. Đơn hàng thứ i yêu cầu đưa hàng lên tầng f_i , cánh tay của rô bốt vận chuyển hàng này có tầm với cao r_i . Rô bốt có thể tự kích hoạt nút bấm tầng cần tới nếu độ cao nút chọn không vượt quá tầm với của mình. Ngoài ra rô bốt cũng có thể phát tín hiệu xin trợ giúp. Nếu trong thang máy đang có rô bốt có thể trợ giúp thì sẽ kích hoạt dùm nút cần thiết. Bộ điều khiển của thang máy chỉ lưu một tín hiệu xác định tầng cần dừng. Sau khi thang máy dừng ở một tầng nào đó để các rô bốt mang hàng ra thì mới có thể đưa ra yêu cầu lên tầng mới và việc chọn tầng chỉ có thể khi cửa thang máy đã đóng. Nếu một rô bốt đi quá tầng cần đến thì sẽ không thể mang hàng xuống vì không có phương tiện di chuyển xuống. Nếu rô bốt ra sớm ở tầng thấp hơn nơi cần tới thì có thể dùng thang máy dịch vụ để tới đích.

Các rô bốt đều biết cách ra hợp lý để tổng chi phí phải trả cho phần dịch vụ là nhỏ nhất.

Hãy xác định tổng nhỏ nhất số lượng tầng cần phải dùng thang máy dịch vụ.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và h ($1 \leq n \leq 10^5$, $1 \leq h \leq 10^9$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên r_i và f_i ($1 \leq r_i, f_i \leq 10^9$).

Kết quả: Đưa ra file văn bản TRANSPORT.OUT một số nguyên – tổng nhỏ nhất số lượng tầng cần phải dùng thang máy dịch vụ.

Ví dụ:

TRANSPORT.INP	TRANSPORT.OUT
3 10 30 17 1 4 3 8	0



VZ44_1020181124.D A XVI

Giải thuật: Tìm cực trị.

Gọi \mathbf{p} – tầng cao nhất rô bốt thứ i có thể tới.

Ta có: $\mathbf{h+p-1} = \mathbf{r}_i \rightarrow \mathbf{p} = \mathbf{r}_i - \mathbf{h+1}$.

Ban đầu các rô bốt được tập kết ở tầng 1, vì vậy $\mathbf{p} = \max(1, \mathbf{r}_i - \mathbf{h+1})$.

Trên thực tế, rô bốt thứ i sẽ ra ở tầng \mathbf{f}_i nếu $\mathbf{p} > \mathbf{f}_i$.

Như vậy tầng mà rô bốt i sẽ ra nếu đi một mình là $\min(\mathbf{f}_i, \mathbf{p})$.

Khi đi cùng nhau, rô bốt ra muộn nhất (theo khả năng hoặc nhu cầu của mình) sẽ hỗ trợ các rô bốt ra sớm ra đúng tầng của rô bốt đó.

Gọi f_{\max} là tầng cao nhất các rô bốt di chuyển theo thang máy miễn phí.

Tổng số tầng dùng thang máy dịch vụ sẽ là

$$\sum_{f_i > f_{\max}} (f_i - f_{\max})$$

Lưu ý: Tổng tìm được có thể rất lớn.

Tổ chức dữ liệu:

Các mảng **vector<int>r (n)**, **f (n)** – lưu dữ liệu vào.

Độ phức tạp của giải thuật: $O(n)$.

Ràng buộc:

Nhóm I: $n \leq 1\ 000$, Mọi rô bốt đủ cao để chọn tầng của mình, tổng điểm 20;

Nhóm II: $n \leq 10\ 000$, $1 \leq r_i, f_i \leq 10\ 000$, tổng điểm 20;

Nhóm III: Cho mọi khả năng của dữ liệu, tổng điểm 60.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("Transport.inp");
ofstream fo ("Transport.out");
int n,h,fm=1;
int64_t ans=0;

int main()
{
    fi>>n>>h;
    vector<int>r(n),f(n);
    for(int i=0; i<n; ++i) fi>>r[i]>>f[i];
    for(int i=0; i<n; ++i)
        fm = max(fm,min(f[i],max(1,r[i]-h+1)));
    for(int i=0;i<n;++i) if(f[i]>fm)
        ans+=(int64_t)(f[i]-fm);
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



VZ45. CẤP ĐIỂM

Tên chương trình: POINTS.CPP

Với nhiều học sinh hình học thường mang lại nỗi khiếp sợ vô hình. Để chứng minh rằng cái đáng sợ là cấu trúc dữ liệu và giải thuật chứ không phải hình học thầy giáo ra một bài có nội dung hình học: Cho n điểm trên trực hoành, điểm thứ i có tọa độ $(x_i, 0)$ và n điểm trên trực tung, điểm thứ i có tọa độ $(0, y_i)$, $i = 1 \dots n$. Tất cả các điểm đều có tọa độ nguyên và không có điểm nào trùng với gốc tọa độ. Khi nối một điểm trên trực hoành với một điểm trên trực tung ta có một đoạn thẳng.

Hãy xác định có bao nhiêu cách nối mỗi điểm trên trực hoành với một điểm trên trực tung sao cho không có hai đoạn thẳng nào cắt nhau và đưa ra theo mô đun 998244353.

Dữ liệu vào từ file văn bản POINTS.INP:

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên x_1, x_2, \dots, x_n ($-10^9 \leq x_1 < x_2 < \dots < x_n \leq 10^9$),
- ✚ Dòng thứ 3 chứa n số nguyên y_1, y_2, \dots, y_n ($-10^9 \leq y_1 < y_2 < \dots < y_n \leq 10^9$).

Kết quả: Đưa ra file văn bản POINTS.OUT số nguyên tính được.

Ví dụ:

POINTS.INP	POINTS.OUT
<pre>2 -1 1 -1 2</pre>	<pre>2</pre>



Giải thuật: Số tổ hợp, tính nhanh lũy thừa và nghịch đảo mô đun .

Gọi số lượng điểm trên trực hoành có tọa độ dương là $\mathbf{x+}$, số điểm có tọa độ âm là $\mathbf{x-}$, số lượng điểm trên trực tung có tọa độ dương là $\mathbf{y+}$, số điểm có tọa độ âm là $\mathbf{y-}$.

Giả thiết ta có $\mathbf{P++}$ là số lượng cặp điểm được chọn, trong đó cả 2 tọa độ đều dương.

Khi đó có thể tính được số đoạn thẳng $\mathbf{P+-}$ nối điểm hoành độ dương với điểm tung độ âm: $\mathbf{P+-} = \mathbf{x+} - \mathbf{P++}$ (vì $\mathbf{P++} + \mathbf{P+-} = \mathbf{x+}$).

Tương tự như vậy, có thể tính $\mathbf{P-+}$ – số đoạn thẳng nối điểm hoành độ âm với điểm tung độ dương: $\mathbf{P-+} = \mathbf{y++} - \mathbf{P++}$ và $\mathbf{P--}$ – số đoạn thẳng nối điểm hoành độ âm với điểm tung độ âm: $\mathbf{P--} = \mathbf{x-} - \mathbf{P-+}$.

Biết các giá trị $\mathbf{P??}$, trong đó ? là ký tự + hoặc -, ta có thể xác định số cách chọn điểm cho mỗi nhóm.

Ký hiệu \mathbf{C}_u^v là tổ hợp chập v từ u phần tử, ta có số cách chọn các điểm hoành độ dương để có $\mathbf{P++}$ là $\mathbf{C}_{\mathbf{x+}}^{P++}$ và số cách chọn các điểm tung độ dương để có $\mathbf{P++}$ là $\mathbf{C}_{\mathbf{y+}}^{P++}$.

Gọi $\mathbf{A+}$ là tập các điểm đã chọn các điểm trong $\mathbf{x+}$ và $\mathbf{B+}$ là tập các điểm trong $\mathbf{y+}$ tham gia tạo $\mathbf{P++}$, cách nối thỏa mãn điều kiện bài toán là nối điểm có *giá trị lớn nhất trong $\mathbf{A+}$* với *điểm giá trị lớn nhất trong $\mathbf{B+}$* , nối điểm có *giá trị lớn thứ hai trong $\mathbf{A+}$* với *điểm giá trị lớn thứ hai trong $\mathbf{B+}$* , ... Các đoạn thẳng nối như vậy sẽ *không cắt nhau* và nằm gọn trong *phản tư I* của mặt phẳng tọa độ.

Các điểm tham gia tạo $\mathbf{P+-}$ cũng được nối theo cách tương tự: nối điểm có *giá trị lớn nhất trong $\mathbf{A+}$* với *điểm giá trị tuyệt đối lớn nhất trong $\mathbf{B-}$* , nối điểm có *giá trị lớn thứ hai trong $\mathbf{A+}$* với *điểm giá trị tuyệt đối lớn thứ hai trong $\mathbf{B-}$* , ... Các đoạn thẳng nối như vậy sẽ *không cắt nhau* và nằm gọn trong *phản tư IV* của mặt phẳng tọa độ.

Các đoạn thẳng tạo $\mathbf{P-+}$ và $\mathbf{P--}$ cũng được tạo theo cách nối điểm tương tự đã nêu, chúng nằm gọn trong phản tư II và phản tư III, do đó không có đoạn thẳng nào giao nhau.

Một khi đã chọn các điểm thuộc $\mathbf{Y}+$ tham gia tạo $\mathbf{P}++$, số điểm thuộc $\mathbf{Y}+$ tham gia tạo $\mathbf{P}-+$ trở nên tiền định và $\mathbf{P}-+$ chỉ còn phụ thuộc các chọn điểm trong $\mathbf{X}-$. Để có $\mathbf{P}-+$, số cách chọn điểm có hoành độ âm là C_{X-}^{P-+} .

Số cách chọn điểm có tung độ âm để có $\mathbf{P}+-$ chỉ còn phụ thuộc vào cách chọn điểm trong $\mathbf{Y}-$ và sẽ là C_{Y-}^{P+-} .

Với mỗi cách chọn điểm để có $\mathbf{P}++$, nếu các giá trị $\mathbf{P}+-$, $\mathbf{P}-+$ và $\mathbf{P}--$ không âm thì tổng số cách nối điểm sẽ tăng thêm một lượng là $C_{X+}^{P++} \times C_{Y+}^{P++} \times C_{X-}^{P-+} \times C_{Y-}^{P+-}$.

Tính $C_u^v \bmod q$, trong đó q – số nguyên tố:

$$C_u^v = \mathbf{x} \rightarrow \frac{u!}{v! \times (u-v)!} = \mathbf{x} \rightarrow u! = dv \times \mathbf{x}, \text{ trong đó } dv = v! \times (u-v)!$$

Từ đây có:

$$\begin{aligned} u! \times dv^{q-2} &= dv^{q-1} \times \mathbf{x} \\ (u! \times dv^{q-2}) \bmod q &= (dv^{q-1} \times \mathbf{x}) \bmod q \\ &= (dv^{q-1} \bmod q) \times (\mathbf{x} \bmod q) \\ &= 1 \times (\mathbf{x} \bmod q) \\ &= \mathbf{x} \bmod q \end{aligned}$$

Như vậy để tính nhanh theo mô đun q số tổ hợp C_u^v ta cần chuẩn bị bảng giá trị các giai thừa theo mô đun q và sơ đồ nâng nhanh lũy thừa một số (tính theo mô đun q).

Tổ chức dữ liệu:

- ▀ Mảng `int64_t f[maxn]` – Lưu giá trị các giai thừa theo mô đun q ,
- ▀ Các biến `pos`, `diff`, `next` quản lý số điểm đang xét tương ứng thuộc $\mathbf{X}+$, $\mathbf{Y}-$ và $\mathbf{X}-$.

Xử lý:

Sơ đồ đệ quy tính nhanh $a^b \text{ mod } \text{MOD}$:

```
int64_t power(int64_t a, int64_t b)
{
    if (b == 0) return 1;
    if (b % 2 == 0)
    {
        int64_t x = power(a, b / 2);
        return (x * x) % MOD;
    } else return (power(a, b - 1) * a) % MOD;
}
```

Tính số tò hợp C_n^k theo mô đun **MOD**:

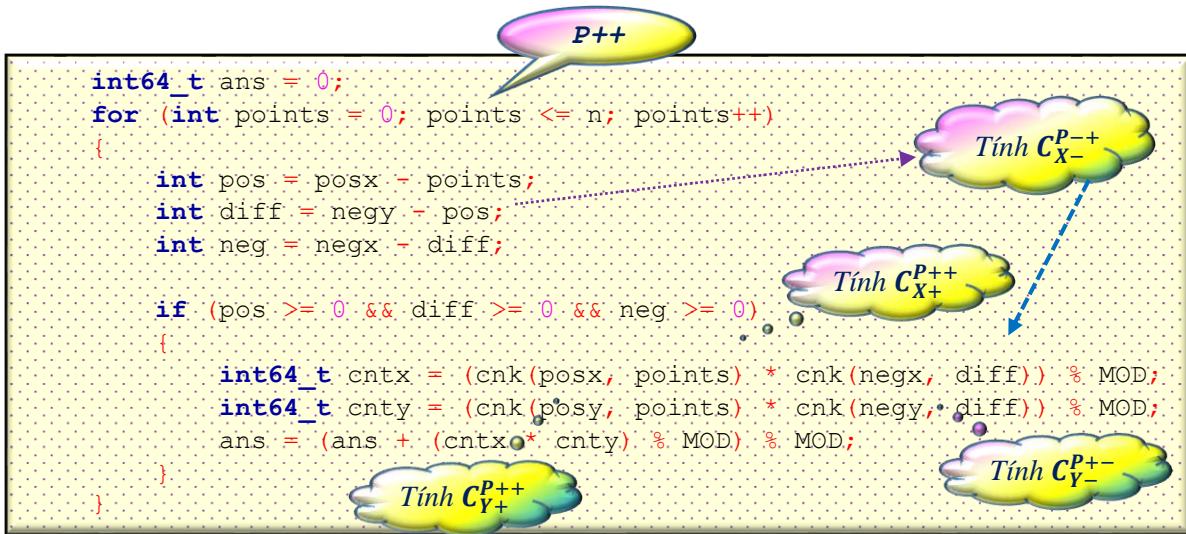
```
int64_t inv(int64_t a, int64_t mod)
{
    return power(a, mod - 2);
}

int64_t divide(int64_t a, int64_t b)
{
    return (a * inv(b, MOD)) % MOD;
}

int64_t cnk(int n, int k)
{
    if (0 <= k && k <= n)
    {
        int64_t b = (f[k] * f[n - k]) % MOD;
        return divide(f[n], b);
    }
    return 0;
}
```

*Tính nghịch đảo
theo mô đun*

Xác định tham số các tập điểm và chỉnh lý kết quả:



Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
const int MOD = 998244353;
const int maxn = 1e5 + 10;

int64_t f[maxn];

int64_t power(int64_t a, int64_t b)
{
    if (b == 0) return 1;
    if (b % 2 == 0)
    {
        int64_t x = power(a, b / 2);
        return (x * x) % MOD;
    } else return (power(a, b - 1) * a) % MOD;
}

int64_t inv(int64_t a, int64_t mod)
{
    return power(a, mod - 2);
}

int64_t divide(int64_t a, int64_t b)
{
    return (a * inv(b, MOD)) % MOD;
}

int64_t cnk(int n, int k)
{
    if (0 <= k && k <= n)
    {
        int64_t b = (f[k] * f[n - k]) % MOD;
        return divide(f[n], b);
    }
    return 0;
}

int main()
{
    ifstream cin ("points.inp");
    ofstream cout ("points.out");
    f[0] = 1;
    for (int i = 1; i < maxn; i++) f[i] = (f[i - 1] * i) % MOD;
    int n;
    cin >> n;
    int posx = 0, negx = 0, posy = 0, negy = 0;
    for (int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
```

```

        if (x > 0) posx++; else negx++;
    }
    for (int i = 0; i < n; i++)
    {
        int y;
        cin >> y;
        if (y > 0) posy++; else negy++;
    }

int64_t ans = 0;
for (int points = 0; points <= n; points++)
{
    int pos = posx - points;
    int diff = negy - pos;
    int neg = negx - diff;

    if (pos >= 0 && diff >= 0 && neg >= 0)
    {
        int64_t cntx = (cnk(posx, points) * cnk(negx, diff)) % MOD;
        int64_t cnty = (cnk(posy, points) * cnk(negy, diff)) % MOD;
        ans = (ans + (cntx * cnty) % MOD) % MOD;
    }
}

cout << ans << '\n';
cout<<"\nTime: "<<clock()/(double)1000<<" sec";
}

```



VZ46. TIỀM NĂNG

Tên chương trình: POTENTIAL.CPP

Xét số nguyên dương x . Tiềm năng của x được xác định là tích các chữ số của nó. Ví dụ, $x = 123$, tiềm năng của x là $1 \times 2 \times 3 = 6$, còn với $x = 509$, tiềm năng là $5 \times 0 \times 9 = 0$.

Cho 2 số nguyên dương lf và rt ($lf \leq rt$). Hãy tìm số x có tiềm năng lớn nhất và thỏa mãn điều kiện $lf \leq x \leq rt$. Nếu có nhiều số cùng tiềm năng lớn nhất thì đưa ra số lớn nhất trong các số đó.

Dữ liệu: vào từ file văn bản POTENTIAL.INP:

- ✚ Dòng thứ nhất chứa số nguyên lf ($1 \leq lf \leq 10^{10^5}$)
- ✚ Dòng thứ 2 chứa số nguyên rt ($lf \leq rt \leq 10^{10^5}$).

Kết quả: Đưa ra file văn bản POTENTIAL.OUT số nguyên tìm được.

Ví dụ:

POTENTIAL.INP	POTENTIAL.OUT
1	1
30	29



Giải thuật: Phân tích và nhận dạng tình huống, Kỹ thuật bảng phương án.

Các số **lf** và **rt** cần lưu trữ dưới dạng xâu.

Xét 2 trường hợp:

- ✚ **lf** và **rt** cùng độ dài,
- ✚ **lf** có độ dài ngắn hơn **rt**.

a – Trường hợp **lf** và **rt** cùng độ dài:

Nếu **lf** = **rt** → kết quả là **lf**.

Nếu tiền tố chung của **lf** và **rt** chứa 0 → kết quả là **rt**.

Nếu ngay sau phần tiền tố chung phần tiếp theo của **rt** có dạng **111...10...** thì không có cách giảm **rt** để kết quả không chứa chữ số 0, do đó kết quả cần tìm sẽ là **rt**.

Trường hợp còn lại:

b – Trường hợp **lf** ngắn hơn **rt**:

Chương trình

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

const int MAGIC = 5;

int pw(int x, int y)
{
    int ans = 1;
    for (int i = 0; i < y; i++) ans *= x;
    return ans;
}

int main()
{
    ifstream cin ("potential.inp");
    ofstream cout ("potential.out");
    string l, r;
    cin >> l >> r;

    if (l == r)
    {
        cout << l << '\n';
        return 0;
    }

    if (l.size() == r.size())
    {
        string t;
        int it = 0;
        int n = l.size();
        bool zero=false;
        while (it < n && l[it] == r[it])
        {
            if (r[it]=='0') {zero=true; break;}
            it++;
        }
        if (!zero) while (it < n && r[it] == '1') it++;
        if (zero || r[it] == '0')
        {
            cout << r << '\n';
            return 0;
        }
        bool needChange = false;
        for (int i = it + MAGIC; i < n; i++)
            if (r[i] != '9') needChange = true;

        vector<int> w;
        for (int i = it; i < it+MAGIC && i<n; i++) w.push_back(r[i]-'0');
        int mx = -1, where = -1;
        int k = w.size();
    }
}
```

```

int d = 1;
if (needChange)
{
    where = it;
    mx = w[0] - 1;
    for (int i = 1; i < k; i++) mx *= w[i];
}
for (int i = 0; i <= k; i++)
{
    if (i == k)
    {
        if (needChange) break;
        int cnt = d;
        if (cnt > mx)
        {
            where = INT_MAX;
            mx = cnt;
        }
        break;
    }
    int cnt = d * (w[i] - 1);
    cnt *= pw(9, k - i - 1);
    if (mx < cnt)
    {
        mx = cnt;
        where = i + it;
    }
    d *= w[i];
}
for (int i = 0; i < n; i++)
{
    if (i == where) r[i]--;
    if (i > where) r[i] = '9';
}
cout << r << '\n';
}
else
{
    int n = r.size();
    if (r[0] == '1')
    {
        for (int i = 1; i < (int)r.size(); i++) cout << 9;
        cout << '\n';
        return 0;
    }
    vector<int> w;
    for (int i=0;i<MAGIC && i<(int)r.size(); i++) w.push_back(r[i]-'0');
    bool needChange = false;
    for (int i = MAGIC; i < n; i++)
        if (r[i] != '9') needChange = true;
    int mx = -1, where = -1;
    int k = w.size();
}

```

```

int d = 1;
if (needChange)
{
    where = 0;
    mx = w[0] - 1;
    for (int i = 1; i < k; i++) mx *= w[i];
}
for (int i = 0; i <= k; i++)
{
    if (i == k)
    {
        if (needChange) break;
        int cnt = d;
        if (cnt > mx)
        {
            where = INT_MAX;
            mx = cnt;
        }
        break;
    }
    int cnt = d * (w[i] - 1);
    cnt *= pw(9, k - i - 1);
    if (mx < cnt)
    {
        mx = cnt;
        where = i;
    }
    d *= w[i];
}
for (int i = 0; i < n; i++)
{
    if (i == where) r[i]--;
    if (i > where) r[i] = '9';
}
cout << r << '\n';
}
}

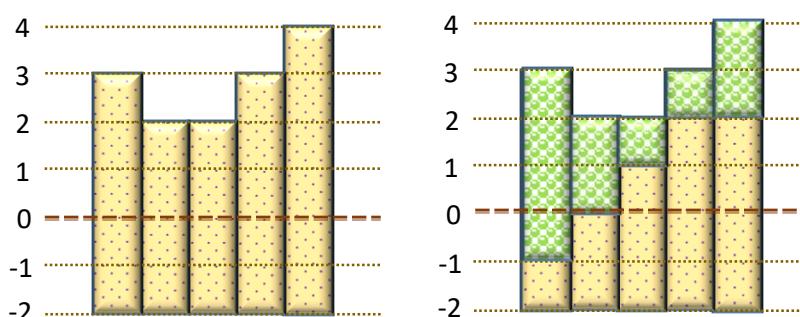
```



VZ47. KHẢO CỐ

Tên chương trình: ARCHAEOLO.CPP

Một khu vực có nhiều di tích cổ được phát hiện. Thiết diện phẳng của khu vực được chia thành n phần bằng nhau, mỗi phần là một cột. Khi tiến hành đào sâu để tìm kiếm, độ cao các cột cạnh nhau phải đảm bảo chênh lệch không quá 1 để tránh sạt lở. Mùa mưa tới, việc khai quật phải tạm dừng. Độ cao cột i lúc đó là h_i , $i = 1 \dots n$.



Ở mùa khô tiếp theo các nhà khảo cổ quay lại nơi cũ tiếp tục công việc khai quật. Cứ mỗi ngày họ có thể dọn đất đá, hạ độ cao ở một cột tùy chọn xuống 1 đơn vị. Đợt khai quật này dự kiến kéo dài t ngày. Mục tiêu của đợt này là đào xuống được sâu nhất có thể.

Hãy xác định độ sâu nhất có thể đạt được.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và t ($1 \leq n \leq 10^5$, $1 \leq t \leq 10^{18}$),
- ✚ Dòng thứ 2 chứa n số nguyên h_1, h_2, \dots, h_n ($1 \leq h_i \leq 10^9$, $i = 1 \dots n$).

Kết quả: Đưa ra file văn bản ARCHAEOLO.OUT một số nguyên – độ sâu đạt được.

Ví dụ:

ARCHAEOLO.INP	ARCHAEOLO.OUT
5 10 3 2 2 3 4	-1



Giải thuật: Tìm kiếm nhị phân.

Nếu đào để đưa cột \mathbf{x} về độ cao \mathbf{d} thì cần đảm bảo độ cao 2 cột $\mathbf{x}-1$ và $\mathbf{x}+1$ không vượt quá $\mathbf{d}+1$, độ cao 2 cột $\mathbf{x}-2$ và $\mathbf{x}+2$ không vượt quá $\mathbf{d}+2$, ...

Gọi \mathbf{i} là cột *gần nhất bên trái* \mathbf{x} thỏa mãn điều kiện $\mathbf{h}_i \leq \mathbf{d} + (\mathbf{x} - i)$.

Với mọi $\mathbf{j} \leq \mathbf{i}$ ta có $\mathbf{h}_j \leq \mathbf{d} + (\mathbf{x} - j)$,

Với các \mathbf{j} thỏa mãn $\mathbf{i} < \mathbf{j}$ có $\mathbf{h}_j > \mathbf{d} + (\mathbf{x} - j)$.

Như vậy cần tồn thời gian điều chỉnh độ cao các cột từ $\mathbf{i}+1$ đến $\mathbf{x}-1$.

Để có chi phí thời gian ít nhất cần đưa các cột từ $\mathbf{i}+1$ đến \mathbf{x} về trạng thái bậc thang thấp dần, bắt đầu từ $\mathbf{h}_{\mathbf{i}-1}$ đến \mathbf{d} .

Thời gian đưa *phản trái* của \mathbf{x} về trạng thái bậc thang sẽ là:

$$\mathbf{t_left} = \sum_{j=i+1}^x \mathbf{h}_j - \sum_{j=i+1}^x (\mathbf{d} + \mathbf{x} - j)$$

Tương tự như vậy, ta có thể tìm cột \mathbf{k} gần nhất bên phải \mathbf{x} thỏa mãn điều kiện

$$\mathbf{h}_k \leq \mathbf{d} + (k - x)$$

và đưa *phản phải* của \mathbf{x} về trạng thái bậc thang với chi phí

$$\mathbf{t_right} = \sum_{j=x+1}^k \mathbf{h}_j - \sum_{j=x+1}^k (\mathbf{d} + j - x)$$

Như vậy tổng chi phí thời gian đưa cột \mathbf{x} về độ cao \mathbf{d} sẽ là

$$\sum_{j=i+1}^x \mathbf{h}_j - \sum_{j=i+1}^x (\mathbf{d} + x - j) + \sum_{j=x+1}^k \mathbf{h}_j - \sum_{j=x+1}^k (\mathbf{d} + j - x)$$

Với mỗi giá trị \mathbf{x} và \mathbf{d} , bằng phương pháp *tìm kiếm nhị phân* ta có thể xác định được \mathbf{i} và \mathbf{k} .

Chi phí thời gian để đưa cột \mathbf{x} về độ cao \mathbf{d} sẽ tăng dần khi \mathbf{d} giảm, vì vậy bằng phương pháp tìm kiếm nhị phân có thể xác định được \mathbf{d} theo \mathbf{t} cho trước.

Gọi \mathbf{d}_x là độ cao nhỏ nhất có thể đạt được ở cột \mathbf{x} sau khoảng thời gian \mathbf{t} , kết quả cần tìm sẽ là $\min_{i=0 \dots n-1} d_i$.

Tổ chức dữ liệu:

- Mảng `vector<int>` `h(n)` – Lưu trữ dữ liệu độ cao ban đầu,
- Mảng `vector<ll>` `pref(n+1)` – Tích lũy tổng tiền tố độ cao các cột.

Xử lý:

Để tính tổng độ cao các cột liên tục từ `i` đến `k` cần xây dựng tổng tiền tố các độ cao, giá trị các tổng tiền tố có thể vượt quá 10^9 ,

Việc tích lũy tổng thời gian đưa cột `x` về độ cao `d` có thể cho giá trị vượt quá khả năng lưu trữ của kiểu dữ liệu `int64_t`, dĩ nhiên đó là kết quả bị loại (vì $t \leq 10^{18}$). Để nhận dạng trường hợp bị loại cần tính trước gần đúng tổng thời gian với kiểu dữ liệu `long double` và gán kết quả trả về là ∞ .



```
ll all_tm(ll l, ll r)
{
    [double tmp = (double) (r - l) * (l + r - 1) / 2;
     if (tmp < -INFL * 4) return -INFL * 2;

    ll ret = (r - l) * (l + r - 1) / 2;
    return ret;
}
```

Độ phức tạp của giải thuật: $O(n \log^2(n))$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long double ld;
ll const INFL = (ll)1e18 + 1e6;

ll all_tm(ll l, ll r)
{
    double tmp = (double) (r - l) * (l + r - 1) / 2;
    if (tmp < -INFL * 4) return -INFL * 2;
    ll ret = (r - l) * (l + r - 1) / 2;
    return ret;
}

int main()
{
    ifstream cin ("archaeol.inp");
    ofstream cout ("archaeol.out");
    int n, a;
    ll t;
    cin >> n >> t;
    vector<int> h(n);
    vector<ll> pref(n+1);
    pref[0]=0;

    for (int i = 0; i < n; ++i)
    {
        cin >> a;
        h[i]=a;
        pref[i+1]=pref[i]+a;
    }

    ll ans = *min_element(h.begin(), h.end());

    for (int i = 0; i < n; ++i)
    {
        ll lh = -t, rh = h[i];
        while (rh - lh > 1)
        {
            ll midh = (lh + rh) / 2;
            int llb = -1, rlb = i;
            while (rlb - llb > 1)
            {
                int midl = (llb + rlb) / 2;
                if (h[midl] <= midh + (i - midl)) llb = midl;
```

```

        else rlb = midl;
    }

int lrb = i, rrb = n;
while (rrb - lrb > 1)
{
    int midr = (lrb + rrb) / 2;
    if (h[midr] <= midh + (midr - i)) rrb = midr;
    else lrb = midr;
}

ll sum = pref[lrb + 1] - pref[rlb];
sum -= all_tm(midh, midh + (i - rlb) + 1) +
       all_tm(midh + 1, midh + (lrb - i) + 1);
if (sum > t) lh = midh;
else rh = midh;
}
ans = min(ans, rh);
}
cout << ans << "\n";
cout << "\nTime: "<<clock() / (double)1000<<" sec";
}

```



VZ48. CHỈ SỐ

Tên chương trình: INDEX.CPP

Một dây chuyền sản xuất hiện đại mới được nhập về. Dây chuyền được lắp ráp từ n linh kiện, linh kiện thứ i có mã sản phẩm là a_i – một số nguyên dương không vượt quá 10^9 , $i = 1 \div n$.

Để tiện bảo dưỡng và đặt mua linh kiện dự trữ thay thế người ta lập một danh mục các linh kiện khác nhau, đánh số liên tục từ 0 trở đi. Số ghi trong danh mục được gọi là chỉ số thiết bị. Các linh kiện cùng mã sản phẩm phải tương ứng với cùng một chỉ số thiết bị. Hai linh kiện khác mã sản phẩm phải có chỉ số thiết bị khác nhau.

Hãy xác định số lượng số sản phẩm khác nhau và chỉ ra một cách đánh chỉ số.

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

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^9$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản INDEX.OUT:

- ▣ Dòng đầu tiên chứa số nguyên m – số loại linh kiện có mã khác nhau,
- ▣ Dòng thứ 2 chứa n số nguyên b_1, b_2, \dots, b_n , trong đó b_i là chỉ số thiết bị của sản phẩm thứ i , $i = 1 \div n$.

Ví dụ:

INDEX.INP	INDEX.OUT
11	6
1 2 3 4 5 1 2 1 2 7 5	0 1 2 3 4 0 1 0 1 5 4



Giải thuật: Ánh xạ sang dãy số nguyên liên tiếp.

Có nhiều cách ánh xạ dãy số nguyên a_1, a_2, \dots, a_n sang tập \mathcal{K} các số nguyên liên tiếp nhau từ 0 đến k , thỏa mãn các điều kiện:

- Nếu $a_i = a_j$ thì a_i và a_j cùng được đặt tương ứng với một số nguyên thuộc \mathcal{K} ;
- Nếu $a_i \neq a_j$ thì a_i và a_j tương ứng với những số khác nhau trong \mathcal{K} .

Giải thuật I: Chỉ sử dụng các cấu trúc dữ liệu tuyến tính.

Áp dụng khi *không có các truy vấn thay đổi giá trị của dãy* cần ánh xạ.

Thông tin vào được lưu trữ dưới dạng các cặp (a_i, i) và được sắp xếp theo thứ tự tăng dần. Các phần tử có a_i giống nhau sẽ đứng liên tiếp tạo thành một nhóm.

Như vậy a_i sẽ được đặt tương đương với số thứ tự của nhóm và được ghi vào mảng b , ở vị trí i .

Chương trình I

```
#include <bits/stdc++.h>
#define NAME "index."
#define ff first
#define ss second
using namespace std;
typedef pair<int,int> pii;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
pair<int,bool> r;
int n,x,k=-1;

int main()
{
    fi>>n;
    vector<pii> a(n);
    vector<int> b(n);
    for(int i=0; i<n; ++i) {fi>>x; a[i]={x,i};}
        //for(auto i:a) fo<<i.ff<<' ' ; fo<<'\n';
    sort(a.begin(),a.end());
    x=-1;

    for(int i=0; i<n; ++i)
        if(a[i].ff!=x)
        {
            ++k; b[a[i].ss]=k; x=a[i].ff;
        }
        else b[a[i].ss]=k;

    fo<<k+1<<'\n';
    for(auto i:b) fo<<i<<' ';
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```

Giải thuật II: Trường hợp có các truy vấn thay đổi giá trị dãy số ban đầu.

Các truy vấn thay đổi giá trị a_i có thể tồn tại các số trong \mathcal{K} không tương ứng với giá trị a_i nào hiện tại. Việc khắc phục hiện tượng này không quá phức tạp (với chi phí $O(n)$), nhưng thường không cần thiết trong nhiều bài toán thực tế.

Việc phân nhóm được tiến hành bằng cấu trúc dữ liệu **map**. Cơ chế sử dụng **map** cho phép ánh xạ đổi tượng bất kỳ (số nguyên, xâu, vector, ...) sang tập \mathcal{K} .

Chương trình II

```
#include <bits/stdc++.h>
#define NAME "index."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,x,sz=0;
map<int,int> index;
int main()
{
    fi>>n;
    vector<int> b (n);
    for(int i=0; i<n; ++i)
    {
        fi>>x;
        if (index.find(x) == index.end()) index[x] = sz++;
        b[i] = index[x];
    }
    fo<<sz<<'\
';
    for(int i=0; i<n; ++i) fo<<b[i]<<' ';
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



VZ49. CÁNH ĐỒNG THỬ NGHIỆM

Tên chương trình: FIELD.CPP

Cánh đồng thử nghiệm trồng rau sạch được chia thành các ô vuông và có kích thước $n \times n$ ô. Các hàng và cột được đánh số bắt đầu từ 1 trở đi. Có n rô bốt được dùng để kiểm tra và điều chỉnh các tham số kỹ thuật ở mỗi ô, duy trì độ ẩm và các vi lượng ở mức phù hợp. Mỗi rô bốt có một ô cơ sở để bổ sung năng lượng, nước và các hoạt chất cần thiết cho hoạt động của mình. Từ ô cơ sở rô bốt sẽ di chuyển theo các ô cùng hàng hoặc cùng cột với ô cơ sở. Tại mỗi thời điểm, trên mỗi ô có không quá một rô bốt.

Để kiểm soát toàn cánh đồng cần có n rô bốt. Đáng tiếc, hiện tại người ta không có đủ n rô bốt cùng loại nên phải dùng những loại khác nhau, rô bốt thứ i thuộc loại a_i , $i = 1 \div n$. Rô bốt khác loại có kiến trúc ở ô cơ sở khác nhau. Nếu một rô bốt đi qua ô cơ sở của loại khác nó sẽ làm hỏng ô cơ sở và hỏng cả chính nó.

Mỗi loại rô bốt có tín hiệu điều khiển riêng. Khi phát tín hiệu điều khiển cho một loại nào đó, các rô bốt cùng loại sẽ đồng thời hoạt động, mỗi rô bốt sẽ chọn ngẫu nhiên hướng đi cho mình.

Để tiện bảo dưỡng người ta bố trí cơ sở cho các rô bốt cùng loại ở nhóm các ô tạo thành miền liên thông kề cạnh.

Hãy đưa ra một phương án chọn ô cơ sở cho các rô bốt.

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

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_i ($1 \leq a_i \leq n$).

Kết quả: Đưa ra file văn bản FIELD.OUT n cặp số nguyên (x_i, y_i) , mỗi cặp số trên một dòng, dòng thứ i xác định tọa độ ô cơ sở của rô bốt thứ i ($i = 1 \div n$).

Ví dụ:

FIELD.INP	FIELD.OUT
3	2 2
2 1 2	1 1 2 3



VZ49 Io201811124 C A XVI

Giải thuật I: Kỹ thuật phân nhóm. Phương pháp sử dụng bộ nhớ tĩnh.

Các rô bót cùng loại tạo thành một nhóm.

Gọi k là số lượng nhóm và m_i là số lượng rô bốt ở nhóm i , $i = 1 \div k$.

Cơ sở của các rô bốt cùng loại sẽ được đặt ở các ô liên tiếp cùng hàng, bắt đầu từ cột trái nhất còn trống.

Cơ sở cho rô bót ở nhóm tiếp theo – ở hàng tiếp sau hàng dành cho nhóm trước.

I	I				
		II	II	II	II

Cơ sở của các rô bốt nhóm I bắt đầu từ ô $(1, 1)$.

Tuy vậy, *trong trường hợp ở bài toán này* ta có thể làm đơn giản hơn: Nếu nhóm i có các *rô bót loại a_j* thì cơ sở của chúng được bố trí vào *dòng a_j* .

Để phân nhóm, dữ liệu cần lưu trữ dưới dạng mảng các cặp (a_i, i).

Vì $1 \leq a_i \leq n$ nên không phải thực hiện ánh xạ a_i về giá trị trong khoảng $[1..n]$.

Sắp xếp $\{(a_i, i)\}$ theo thứ tự tăng dần, các rô bót cùng loại sẽ tương ứng với nhóm phần tử liên tiếp nhau trong mảng. Việc xác định ô đặt cơ sở sẽ *không đòi hỏi kiểm tra nhóm!*

Tổ chức dữ liệu:

- ¶ Mảng `pair<int, int>` `a` [`(int) 1e5+10`] – Lưu thông tin dữ liệu vào,
¶ Các mảng `int x` [`(int) 1e5+10`], `y` [`(int) 1e5+10`] – Lưu toa đô cơ sở cần tìm.

Độ phức tạp của giải thuật: $O(n \log n)$.

Chương trình I

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

using namespace std;

int x[(int)1e5 + 10], y[(int)1e5 + 10];
pair<int, int> a[(int)1e5 + 10];

int main()
{
    ifstream cin ("field.inp");
    ofstream cout ("field.out");
    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cin >> a[i].first;
        a[i].second = i;
    }
    sort(a, a + n);
    for (int i = 0; i < n; i++)
    {
        y[a[i].second] = i + 1;
        x[a[i].second] = a[i].first;
    }
    for (int i = 0; i < n; i++)
        cout << x[i] << " " << y[i] << "\n";
    cout<<"\nTimes: "<<clock() / (double)1000<<" sec";
}
```

Giải thuật II: Kỹ thuật phân nhóm. Phương pháp sử dụng bộ nhớ động.

Sử dụng mảng động hai chiều cnt để lưu trữ các nhóm:

- Rô bốt thuộc cùng một loại – lưu trữ trong cùng một dòng,
- Các rô bốt khác loại: ở những dòng khác nhau.

Như vậy có thể vòng tránh việc sắp xếp dữ liệu!

Tổ chức dữ liệu:

- Mảng `vector <vector<int>> cnt(n)` – Lưu dữ liệu vào,
- Mảng `vector <pii> ans(n)` – Lưu tọa độ các ô cơ sở cần tìm.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
#define ss second
#define ff first
typedef int64_t ll;
typedef pair<int, int> pii;

int main()
{
    int n;
    ifstream cin ("field.inp");
    ofstream cout ("field.out");
    cin >> n;
    vector<vector<int>> cnt(n);
    for (int i = 0; i < n; ++i)
    {
        int x;
        cin >> x;
        cnt[x - 1].push_back(i);
    }
    vector<pii> ans(n);
    for (int i = 0, j = 0; i < n && j < n; i++)
        for (auto x : cnt[i])
        {
            ans[x] = {i + 1, j + 1};
            j++;
        }
    for (int i = 0; i < n; ++i)
        cout << ans[i].ff << ' ' << ans[i].ss << "\n";

    cout << "\nTimes: " << clock() / (double)1000 << " sec";
}
```



VZ50. THI BẮN NHANH

Tên chương trình: COMPETITION.CPP

Cảnh sát phải hành động nhanh và chuẩn xác khi trấn áp tội phạm nguy hiểm. Một trong những bài tập là trong khoảng thời gian ngắn nhất phải bắn hạ **n** mục tiêu. Súng được trang bị có băng đạn chứa được **m** viên. Người có thao tác chuẩn có thể nạp đầy băng trong **a** giây, còn nạp một viên vào băng mất **b** giây và cứ mỗi giây bắn được một phát.

Đĩa không thể bắn từ băng đạn rỗng và cũng không thể nạp vào băng quá m viên đạn.

Ban đầu băng đạn lắp ở súng chưa có viên nào.

Hãy xác định thời gian tối thiểu hoàn thành bài tập.

Dữ liệu: vào từ file văn bản COMPETITION.INP gồm một dòng chứa 4 số nguyên **n**, **m**, **a** và **b** ($1 \leq n, m, a, b \leq 10^4$).

Kết quả: Đưa ra file văn bản COMPETITION.OUT một số nguyên – thời gian tối thiểu tính được.

Ví dụ:

COMPETITION.INP
3 2 1 1

COMPETITION.OUT
5



Io20181014 A V16

Giải thuật: Cơ sở lập trình .

Nếu $a > m \times b$ tức là nạp từng viên đạn một để đầy băng nhanh hơn thao tác nạp đồng thời cả nhóm đạn vào băng thì chỉ sử dụng thao tác nạp từng viên một.

Số lần chắc chắn cần nạp đầy băng đạn là n/m .

Số viên đạn lẻ còn lại là $n \% m$ có thể được nạp lần lượt từng viên hoặc nạp cả băng tùy thuộc vào tham số thời gian.

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int main()
{
    ifstream cin ("competition.inp");
    ofstream cout ("competition.out");
    int64_t n, m, a, b, r;
    r = n / m;
    cout << r * min(a, b * m) + min(a, (n % m) * b) + n;
}
```



WA01. CÁU HÌNH

Tên chương trình: CONFIG.CPP

Sân đua rô bốt là có hình chữ nhật được chia thành n hàng và m cột. Có n rô bốt cho mỗi lượt đua. Rô bốt thứ i xuất phát ở vị trí ô $(i, 1)$, $i = 1 \dots n$ và đích của mọi rô bốt là ô (n, m) .

Từ ô (i, j) rô bốt có thể chuyển sang ô $(i+1, j)$ hoặc ô $(i, j+1)$ nếu ô thứ hai vẫn nằm trong sân đua. Các rô bốt là đùi bé vì vậy ở mỗi ô có thể có đồng thời nhiều rô bốt và chúng không cản trở lẫn nhau trong chuyển động.

Giữa 2 ô có thể có hoặc không có gờ cản. Nếu không có gờ cản thời gian chuyển động sang ô mới là 0, còn nếu có gờ cản – thời gian là 1. Các gờ cản được bố trí đảm bảo công bằng: nếu mọi rô bốt biết chọn đường tối ưu thì sẽ có cùng thời gian tối thiểu để về đích. Bản đồ bố trí vật cản được ghi vào bộ nhớ rô bốt tại vị trí xuất phát.

Alice có nhiệm vụ kiểm tra tính đúng đắn của việc bố trí các gờ cản. Khi về nhà cô muốn rà soát lại một lần nữa cho chắc chắn, nhưng sau một ngày làm việc mệt mỏi được cô chỉ còn nhớ được trạng thái k cặp ô, biết giữ chúng có hay không có gờ cản. Với những thông tin ít ỏi này Alice tự hỏi liệu có bao nhiêu bản đồ bố trí gờ cản thỏa mãn điều kiện công bằng.

Hãy tính số bản đồ mà Alice xác định được và đưa ra theo mô đun 998244353.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n \leq 5$, $1 \leq m \leq 15$),
- ✚ Dòng thứ 2 chứa số nguyên k ($0 \leq k \leq 2nm-n-m$),
- ✚ Mỗi dòng trong k dòng sau chứa 5 số nguyên $r1, c1, r2, c2$ và w ($1 \leq r1, r2 \leq n$, $1 \leq c1, c2 \leq m$, w bằng 0 hoặc 1, trong đó giữa cặp ô $(r1, c1)$ và $(r2, c2)$ không có gờ cản nếu $w = 0$ và có gờ cản trong trường hợp $w = 1$). Đảm bảo hai ô này có cạnh chung và không có dòng nào bị đưa lặp.

Kết quả: Đưa ra file văn bản CONFIG.OUT một số nguyên – số lượng bản đồ tính được theo mô đun 998244353.

Ví dụ:

CONFIG.INP	CONFIG.OUT
2 2 1 1 1 1 2 0	5



WA01_Io20181028_F_A XVI

Giải thuật: Quy hoạch động theo lát cắt.

Để tiện xử lý theo sơ đồ quy hoạch động cần đưa bài toán ban đầu về bài toán tương đương: *Tìm số cách đặt gờ cản để từ ô (1, 1) có thể tới ô bất kỳ ở cột phải nhất với cùng thời gian như nhau. Quy tắc di chuyển vẫn là chỉ được sang ô phải hoặc ô dưới kè cạnh.*

Chương trình

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

const int INF = 1000000000;
const int MOD = 998244353;

vector<int> go(vector<int> a, int i, int j, int w1, int w2)
{
    if (i == 0 && j == 0) return a;
    int v = INF;
    if (i > 0) v = min(v, a[j] + w2);
    if (j > 0) v = min(v, a[j - 1] + w1);
    a[j] = v;

    return a;
}

bool ok(vector<int> d)
{
    for (auto v : d)
        if (v != d[0]) return false;
    return true;
}

int main()
{
    ifstream cin ("config.inp");
    ofstream cout ("config.out");
    int n, m;
    cin >> n >> m;
    int k;
    cin >> k;
    vector<vector<int>> up(n, vector<int>(m, -1));
    vector<vector<int>> lft(n, vector<int>(m, -1));
    for (int i = 0; i < k; i++)
    {
        int r1, c1, r2, c2, w;
        cin >> r1 >> c1 >> r2 >> c2 >> w;
        r1 = n - r1;
        r2 = n - r2;
        c1 = m - c1;
        c2 = m - c2;
        if (r1 == r2) lft[r1][max(c1, c2)] = w;
        else up[max(r1, r2)][c1] = w;
    }

    vector<vector<map<vector<int>, long long>>>
        dp(m + 1, vector<map<vector<int>, long long>>(n + 1));
    vector<int> cur(n);
    dp[0][0][cur] = 1;
```

```

for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
    {
        int ii = i;
        int jj = j + 1;
        if (jj == n) {ii++; jj = 0;}
        for (auto& p : dp[i][j])
        {
            p.second %= MOD;
            int w1min = 0;
            int w1max = j == 0 ? 0 : 1;
            int w2min = 0;
            int w2max = i == 0 ? 0 : 1;
            if (up[j][i] != -1) w1min = w1max = up[j][i];
            if (lft[j][i] != -1) w2min = w2max = lft[j][i];

            for (int w1 = w1min; w1 <= w1max; w1++)
                for (int w2 = w2min; w2 <= w2max; w2++)
                {
                    auto a1 = go(p.first, i, j, w1, w2);
                    dp[ii][jj][a1] += p.second;
                }
        }
    }

long long ans = 0;
for (auto& p : dp[m][0])
    if (ok(p.first)) ans += p.second % MOD;

cout << ans % MOD << endl;
cout << "\nTime: " << clock() / (double)1000 << "sec ";
return 0;
}

```



WA02. ĐOẠN LỚN NHẤT

Tên chương trình: MAXSEGM.CPP

Trong giờ học về việc tạo số ngẫu nhiên nhiều bạn trong lớp đã nhanh chóng làm xong bài tập tạo dãy số nguyên a_1, a_2, \dots, a_n và bắt đầu nói chuyện.

Để giữ trật tự, thầy giáo yêu cầu những ai đã tạo xong dãy số thực hiện tiếp yêu cầu tìm đoạn con có phần tử đầu giống phần tử cuối và có tổng các phần tử trong đoạn là lớn nhất. Nói một cách khác, tìm lf và rt thỏa mãn các điều kiện:

- ▀ $1 \leq lf \leq rt \leq n$,
- ▀ $a_{lf} + a_{lf+1} + \dots + a_{rt}$ là lớn nhất.

Những người làm được yêu cầu bỗng sung, đưa ra được tổng lớn và các chỉ số lf , rt tương ứng sẽ được giải phóng bài tập về nhà cả tuần.

Cả lớp trở nên rất hào hứng vì nghe có vẻ đơn giản, nhưng sau đó mới nhận thấy đây là một quả hồ đào khó nhăn!

Với dãy số đã có, hãy đưa ra các kết quả của yêu cầu bỗng sung.

Dữ liệu: vào từ file văn bản MAXSEGM.INP:

- ▀ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^6$),
- ▀ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($|a_i| \leq 10^9$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản MAXSEGM.OUT, dòng đầu tiên chứa tổng các phần tử của đoạn tìm được, dòng thứ 2 chứa 2 số nguyên lf và rt tương ứng.

Ví dụ:

MAXSEGM.INP	MAXSEGM.OUT
14	12
2 3 1 -6 3 2 3 -15 3 -1 3 4 3 1	9 13



Mzo20180110 B V16

Giải thuật: Tổng tiền tố.

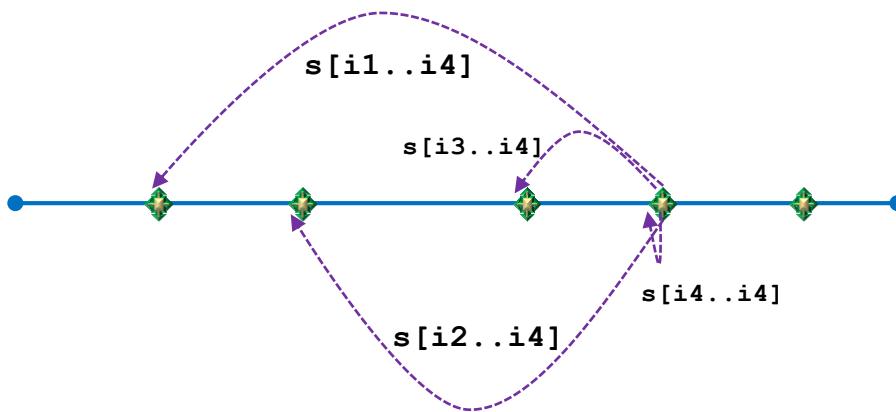
Giả thiết có $a_{i1} = a_{i2} = a_{i3} = \dots = a_{ik}$, $i1 < i2 < \dots < ik$,

Ký hiệu:

- $s[u..v] = \sum_{i=u}^v a_i$ – tổng phần tử ở các vị trí liên tiếp từ u đến v ,
- f_i – tổng tiền tố i phần tử đầu tiên của dãy số đã cho.

Khi đó có $s[u..v] = f_v - f_{u-1}$.

Với mỗi chỉ số i trong tập các chỉ số có giá trị a_i bằng nhau ta phải xác định $j \leq i$ thuộc tập, thỏa mãn $s[j..i]$ là lớn nhất. So sánh các giá trị tính được với mọi i thuộc tập ta sẽ có giá trị tổng lớn nhất các phần tử thuộc đoạn có các phần tử đầu và cuối bằng nhau.



Ví dụ với $i4$ ta phải so sánh $s[i1..i4]$, $s[i2..i4]$, $s[i3..i4]$ và $s[i4..i4]$, xác định max trong số các giá trị trên. Các giá trị này tương ứng bằng $f_{i4} - f_{i1-1}$, $f_{i4} - f_{i2-1}$, $f_{i4} - f_{i3-1}$ và $f_{i4} - f_{i4-1}$.

Dễ dàng thấy được giá trị cần tìm sẽ là

$$f_{i4} - \min(f_{i1-1}, f_{i2-1}, f_{i3-1}, f_{i4-1})$$

Sử dụng *giá trị a_i làm khóa* ta chỉ cần lưu *tổng tiền tố nhỏ nhất* từ đầu dãy đến vị trí $i-1$ và *vị trí đạt min* với *tập các phần tử cùng giá trị*.

Cấu trúc dữ liệu kiểu `map<int, pair<int64_t, int>>` hỗ trợ hiệu quả cho mục đích nêu trên.

Mỗi giá trị a_i khác nhau tương ứng với một phần tử của `map`.

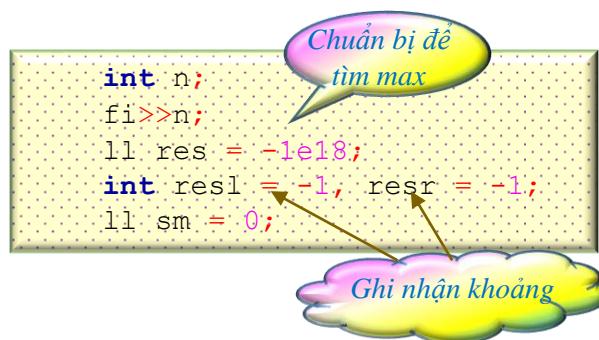
Để có thể dẫn xuất kết quả cần lần lượt so sánh các giá trị lớn nhất đạt được, lưu max và cắp chỉ số tương ứng nơi đạt max.

Bản thân các giá trị của dãy ban đầu không cần lưu lại vì đã được ghi nhận qua khóa của bản đồ dữ liệu quản lý max.

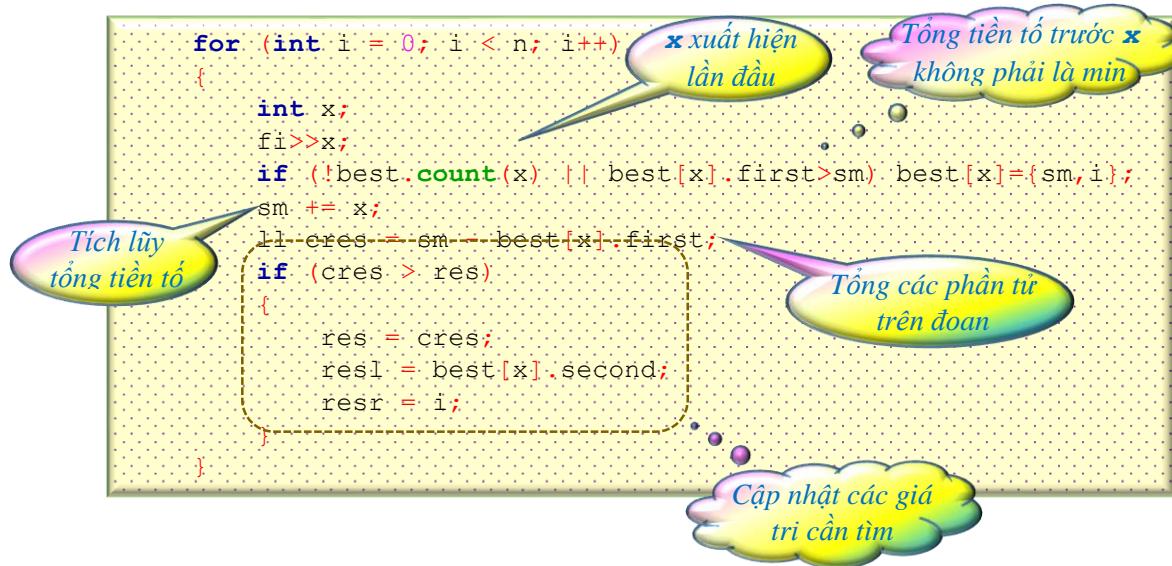
Tổ chức dữ liệu: Bản đồ `map<int, pair<ll, int>> best` và 3 biến đơn lưu kết quả cần tìm.

Xử lý:

Chuẩn bị:



Duyệt mảng:



Độ phức tạp của giải thuật: $O(n \log n)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("maxsegm.inp");
ofstream fo ("maxsegm.out");
typedef long long ll;
typedef long double ld;
map<int, pair<ll, int> > best;

int main()
{
    int n;
    fi>>n;
    ll res = -1e18;
    int resl = -1, resr = -1;
    ll sm = 0;
    for (int i = 0; i < n; i++)
    {
        int x;
        fi>>x;
        if (!best.count(x) || best[x].first > sm) best[x] = {sm, i};
        sm += x;
        ll cres = sm - best[x].first;
        if (cres > res)
        {
            res = cres;
            resl = best[x].second;
            resr = i;
        }
    }
    fo<<res<<'\
'<<resl + 1<<' ' <<resr + 1;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA03. XÓA KÝ TỰ

Tên chương trình: DEL_STR.CPP

Một nhóm bạn trên mạng xã hội thống nhất mỗi người trong nhóm phải có biệt danh (nickname) là một xâu con nhận được bằng cách xóa một số ký tự (có thể là 0) từ xâu **s** cho trước chung cho cả nhóm. Biệt danh nhận được sẽ là xâu các ký tự còn lại, giữ nguyên trình tự ban đầu trong s. Trường hợp ngoại lệ là người quản lý nhóm có biệt danh **t** xác định trước, độc lập với xâu **s**. Để mọi người chọn biệt danh không trùng với trưởng nhóm, bộ phận quản lý quyết định xóa bớt một số ít nhất các ký tự trong **s** để từ xâu **s** mới không ai có thể có biệt danh giống như của người trưởng nhóm. Các xâu chỉ chứa các ký tự la tinh thường và hoa, phân biệt chữ hoa và chữ thường.

Hãy xác định xâu **s** mới từ **s** và **t** ban đầu.

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

- ✚ Dòng đầu tiên chứa xâu **s** độ dài không quá 10 000,
- ✚ Dòng thứ 2 chứa xâu **t** độ dài không quá 1000.

Cả 2 xâu đã cho chỉ chứa các ký tự la tinh thường và hoa.

Kết quả: Đưa ra file văn bản DEL_STR.OUT một xâu **s** mới tùy chọn thỏa mãn các điều kiện đã nêu.

Ví dụ:

DEL_STR.INP	DEL_STR.OUT
abacaba	aacaa
aba	



WA03 Mzo20180110 A XVI

Giải thuật: Quy hoạch động .

Xử lý tương tự bài toán tìm xâu con chung dài nhất của 2 xâu.

Ghi nhận 3 kiểu chuyển trạng thái trong sơ đồ quy hoạch động:

- ☒ 2 – chuyển khi gặp cặp ký tự giống nhau,
- ☒ 1 – chuyển khi gặp cặp ký tự khác nhau,
- ☒ 0 – giá trị trong sơ đồ quy hoạch động không phản ánh bước chuyển trực tiếp.

Đưa ra kết quả:

- Xuất phát từ giá trị lớn nhất ở cột cuối cùng,
- Chỉ làm việc với các ký tự có kiểu chuyển là 1 hoặc 2,
- Cách biến đổi chỉ số: tương tự trường hợp bài toán tìm xâu con chung dài nhất của 2 xâu,
- Nên ghi nhận các ký tự theo trình tự duyệt ngược, sau đó đảo xâu.

Tổ chức dữ liệu:

- ☒ Mảng `int dp[10100][1010]` – Phục vụ quy hoạch động,
- ☒ Mảng `int go[10100][1010]` – Đánh dấu kiểu chuyển.

Độ phức tạp của giải thuật: $O(n \times m)$.

Chương trình

```
#include <bits/stdc++.h>           //final version
using namespace std;
string s, t;
int go[10100][1010];
int dp[10100][1010];
int n, m;

int main()
{
    ifstream cin ("del_str.inp");
    ofstream cout ("del_str.out");
    cin >> s >> t;
    n = s.size(); m = t.size();

    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
    {
        if (s[i] == t[j])
        {
            if (dp[i + 1][j + 1] < dp[i][j] + 1)
                dp[i + 1][j + 1] = dp[i][j] + 1, go[i + 1][j + 1] = 2;
        }
        else
        {
            dp[i + 1][j] = dp[i][j] + 1;
            go[i + 1][j] = 1;
        }
        if (dp[i + 1][j] < dp[i][j])
        {
            dp[i + 1][j] = dp[i][j];
            go[i + 1][j] = 0;
        }
    }

    int ans = -1;
    int cur = -1;
    for (int i = 0; i < m; ++i)
        if (dp[n][i] > ans)
        {
            ans = dp[n][i];
            cur = i;
        }

    int now = n;
    string res;
    while (now > 0)
    {
        if (go[now][cur])
        {
            res += s[now - 1];
            now--;
            cur--;
        }
    }
}
```

```
        if (go[now][cur] == 2) --cur;
    }
--now;
}

reverse(res.begin(), res.end());
cout << res << '\n';
cout<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



WA04. NỘI ĐỊA HÓA

Tên chương trình: LOCAL.CPP

Ô tô của một Công ty liên doanh được lắp ráp từ n loại linh kiện khác nhau, loại thứ i có mã vạch b_i , được sử dụng với số lượng k_i và có đơn giá a_i , $i = 1 \dots n$. Như vậy ô tô có $\sum_{i=1}^n k_i$ linh kiện và có giá trị là $\sum_{i=1}^n k_i \times a_i$.

Mã vạch là số nguyên 13 chữ số (có thể có các chữ số 0 ở đầu), trong đó 3 chữ số ở hàng cao nhất (3 chữ số đầu tiên tính từ trái) xác định quốc gia sản xuất linh kiện và được gọi là Mã quốc gia.



Quốc gia nơi Công ty hợp tác sản xuất có Mã quốc gia là c . Linh kiện nội địa là linh kiện có mã quốc gia bằng c .

Tỷ lệ nội địa hóa được xác định bằng 2 số:

- p – tỷ lệ phần trăm của số linh kiện nội địa trên tổng số linh kiện của một ô tô,
- q – tỷ lệ phần trăm của giá trị các linh kiện nội địa trên giá trị của ô tô.

Hãy xác định tỷ lệ nội địa hóa.

Dữ liệu: vào từ file văn bản LOCAL.INP:

- ✚ Dòng đầu tiên chứa 2 số nguyên n và c ($1 \leq n \leq 10^6$, $1 \leq c < 1000$),
- ✚ Dòng thứ i trong n dòng sau chứa 3 số nguyên b_i , k_i và a_i ($1 \leq k_i \leq 10^9$, $1 \leq a_i \leq 10^9$).

Kết quả: Đưa ra file văn bản LOCAL.OUT 2 số thực p và q với độ chính xác 2 chữ số sau dấu chấm thập phân, mỗi số trên một dòng.

Ví dụ:

LOCAL.INP	LOCAL.OUT
5 893 6904546278524 6 215 8937300567361 10 21 7603456217657 20 32 8937612342130 15 6 8858100503452 40 20	27.47 9.90



Giải thuật; Cơ sở lập trình.

Lưu ý kiểu nhập dữ liệu và cách xuất kết quả.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("local.inp");
ofstream fo ("local.out");

const int64_t m10 = 1e10;
int n,c,k,a,bc;
int64_t b;

int main()
{
    fi>>n>>c;
    int64_t vp=0, vq=0, sv=0, sl=0;
    for(int i=0; i<n; ++i)
    {
        fi>>b>>k>>a;
        sv+=k*a;
        sl+=k;
        bc=b/m10;
        if(bc==c) vp+=k, vq+=k*a;
    }
    fo<<fixed<<setprecision(2)<<(double)vp/sl*100<<' \n '<<(double)vq/sv*100;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA05. VÔ ĐỊCH

Tên chương trình: CHAMPION.CPP

Có 26 võ sỹ quyền anh hạng nặng, mỗi người có một biệt danh là một chữ cái la tinh in hoa và không có ai có biệt danh giống nhau. Người vô địch được giữ đai vô địch. Cuối năm vừa qua đai vô địch đang được người **P1** giữ.

Hàng năm có một số cuộc thách đấu giữa 2 võ sỹ. Ai thắng được người giữ đang đai vô địch sẽ được giữ đai vô địch.

Từ đầu năm đến nay đã có **n** cuộc thi đấu.

Hãy xác định hiện nay ai đang giữ đai vô địch và bao nhiêu người khác nhau đã có dịp giữ đai vô địch trong năm nay.

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

- ✚ Dòng đầu tiên chứa ký tự la tinh in hoa xác định biệt danh của người **P1**,
- ✚ Dòng thứ 2 chứa số nguyên **n** ($1 \leq n \leq 100$),
- ✚ Mỗi dòng trong **n** dòng sau chứa 2 ký tự la tinh in hoa **a** và **b** xác định biệt danh các võ sỹ của một cặp thách đấu và người **a** thắng cuộc, các ký tự ghi cách nhau một dấu cách.

Các cuộc đấu diễn ra theo trình tự nêu trong file.

Kết quả: Đưa ra file văn bản CHAMPION.OUT, dòng đầu tiên xác định biệt danh người hiện giữ đai vô địch, dòng thứ hai chứa số nguyên xác định số người khác nhau đã có dịp giữ đai vô địch trong năm nay.

Ví dụ:

CHAMPION.INP	CHAMPION.OUT
A	C
3	3
B A	
C B	
D A	



WA05 Cr04_20190119 A A XVI

Giải thuật: o sở lập trình .

Ghi nhận người đang giữ đai vô địch,

Đánh dấu những người đã giữ đai,

Khi chuyển người giữ đai: kiểm tra có phải là người chưa giữ đai lần nào hay không và cập nhật số người đã giữ đai nếu cần.

Tổ chức dữ liệu: Mảng `int flg[26]={0}` – Đánh dấu người đã giữ đai vô địch.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("champion.inp");
//ifstream fi ("10");
ofstream fo ("champion.out");
int n, res = 1, flg[26]={0};
char a,b,c;

int main()
{
    fi>>c;   flg[c-65]=1;
    fi>>n;
    for(int i = 0; i<n; ++i)
    {
        fi>>a>>b;
        if(b==c)
        {
            c=a;
            if(flg[c-65]==0) flg[c-65]=1, ++res;
        }
    }
    fo<<c<<'\\n'<<res;
}
```



WA06. LỊCH BAY

Tên chương trình: SCHEDULES.CPP

Alice phụ trách điều độ các chuyến bay. Sân bay chỉ có một đường băng, nhưng mỗi ngày phải đảm bảo n chuyến bay đi. Mùa hè, chuyến bay thứ i cất cánh vào thời điểm t_i , $i = 1 \div n$ và $t_i > t_{i-1}$ với mọi $i \geq 2$.

Mùa đông, tuyết rơi nhiều. Sau mỗi lần cất cánh tuyết trên đường băng bị nén lại làm đường băng trở nên trơn trượt, ánh hướng dẫn sự an toàn của việc cất cánh, vì vậy sau mỗi lượt cất cánh người ta lại phải làm sạch đường băng. Việc làm sạch đường băng mất k đơn vị thời gian. Như vậy chuyến bay thứ i phải cất cánh sau chuyến bay trước không ít hơn k đơn vị thời gian và không sớm hơn thời điểm t_i . Chuyến bay đầu tiên vẫn cất cánh vào thời điểm t_1 vì trước đó chưa cần làm sạch đường băng.

Alice phải lên lịch thời điểm các chuyến bay trong chế độ mùa đông để công bố cho hành khách.

Hãy đưa ra thời điểm cất cánh mới của tất cả các chuyến bay.

Dữ liệu: vào từ file văn bản SCHEDULES.INP:

- ✚ Dòng đầu tiên chứa 2 số nguyên n và k ($1 \leq n \leq 10^6$, $1 \leq k \leq 10^9$),
- ✚ Dòng thứ 2 chứa n số nguyên t_1, t_2, \dots, t_n ($1 \leq t_1 < t_2 < t_3 < \dots < t_n \leq 10^{18}$).

Kết quả: Đưa ra file văn bản SCHEDULES.OUT trên một dòng thời điểm cất cánh mới của các chuyến bay theo trình tự tăng dần của thời gian.

Ví dụ:

SCHEDULES.INP
6 3
1 2 7 10 12 15

SCHEDULES.OUT
1 4 7 10 13 16



WA06 Moz_20180110 E A XVI

Giải thuật: Cơ sở lập trình .

Gọi **f** là thời gian cất cánh của chuyến bay đang xét.

Ban đầu **f = t₁**.

Giả thiết đã xác định thời gian cất cánh của **i-1** chuyến bay đầu tiên.

Thời gian cất cánh của chuyến bay thứ **i** sẽ là **f = max(f+k, t_i)**, **i = 2 ÷ n**.

Lưu ý kiểu dữ liệu của các biến.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("Sheldules.inp");
ofstream fo ("Sheldules.out");
int n, k;
int64_t f, t;

int main()
{
    fi>>n>>k>>t;
    f=t;
    fo<<t<<' ';
    for(int i=1; i<n; ++i)
    {
        fi>>t;
        if(t-f>=k) f=t; else f+=k;
        fo<<f<<' ';
    }
}
```



WA07. TAM GIÁC

Tên chương trình: TRIANGLE.CPP

Alice có 3 thanh nhựa độ dài tương ứng là a , b và c . Alice định lắp một hình tam giác diện tích khác 0 có cạnh là các thanh nói trên, mỗi thanh là một cạnh.

Nếu không thể làm điều đó từ các thanh ban đầu thì Alice có thể hơ nóng và kéo dài một hoặc vài thanh để lắp. Cứ mỗi phút Alice kéo dài thanh được chọn thêm 1cm.

Hãy xác định thời gian tối thiểu cần thiết cho việc kéo dài thanh để lắp được tam giác.

Dữ liệu: Vào từ file văn bản TRIANGLE.INP gồm một dòng chứa 3 số nguyên a , b , c ($1 \leq a, b, c \leq 10^9$).

Kết quả: Đưa ra file văn bản TRIANGLE.OUT một số nguyên – thời gian tối thiểu (tính theo số phút) cần thiết cho việc kéo dài thanh để lắp được tam giác.

Ví dụ:

TRIANGLE.INP
100 10 10

TRIANGLE.OUT
81



WA07 Mos KB 20180110 BC V16

Giải thuật: Cơ sở lập trình, Kỹ năng phân tích giải thuật.

Chuẩn hóa dữ liệu: Đưa về trạng thái $\mathbf{a} \geq \mathbf{b} \geq \mathbf{c}$.

Điều kiện để lắp được tam giác: $\mathbf{b+c > a}$.

Nếu $\mathbf{b+c > a} \rightarrow$ Thời gian kéo dài thanh bằng 0.

Trong trường hợp ngược lại $\mathbf{ans = a - (b+c) + 1}$.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("triangle.inp");
ofstream fo ("triangle.out");
int a,b,c,ans=0;

int main()
{
    fi>>a>>b>>c;
    if(a<b) swap(a,b);
    if(a<c) swap(a,c);
    if(a>=b+c) ans = a-b-c+1;
    fo<<ans;
}
```

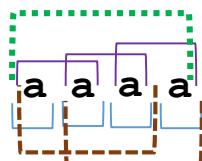


WA08. HÀM LƯỢNG PALINDROME

Tên chương trình: MAXPAL.CPP

Xét xâu s chỉ chứa các ký tự la tinh thường. Hai xâu con các ký tự liên tiếp nhau của s được coi là khác nhau nếu chúng khác nhau về độ dài hoặc có cùng độ dài nhưng khác nhau ở vị trí ban đầu. Toàn bộ xâu s cũng được coi như một xâu con.

Ví dụ, với $s = "aaaa"$ ta có 10 xâu con khác nhau:



Số lượng xâu con khác nhau của s là palindrome được gọi là hàm lượng palindrome của s . Ở ví dụ trên, hàm lượng palindrome của s là 10.

Với xâu s cho trước, bằng cách đổi chỗ các ký tự trong xâu ta có thể thay đổi hàm lượng palindrome của nó. Hãy tìm cách đổi chỗ các ký tự để có xâu với hàm lượng palindrome lớn nhất, đưa ra hàm lượng palindrome nhận được.

Dữ liệu: Vào từ file văn bản MAXPAL.INP gồm một dòng chứa xâu s độ dài không quá 10^6 và chỉ bao gồm các ký tự la tinh thường.

Kết quả: Đưa ra file văn bản MAXPAL.OUT một số nguyên – hàm lượng palindrome lớn nhất có thể nhận được.

Ví dụ:

MAXPAL.INP	MAXPAL.OUT
gagadbcgghhhchbdf	29



WA08 Mos KB 20180110 BB V16

Giải thuật: Cơ sở lập trình, Kỹ năng phân tích giải thuật.

Có nhiều cách đổi chỗ ký tự để nhận được xâu có hàm lượng palindrome lớn nhất.

Ví dụ, với $s = \text{"abaab"}$ các xâu “**ababa**” và “**aaabb**” đều cùng có hàm lượng palindrome lớn nhất và bằng 9.

Không khó để chứng minh được rằng xâu nhận được từ s bằng cách đổi chỗ các ký tự để các ký tự giống nhau đứng cạnh nhau sẽ có hàm lượng palindrome lớn nhất.

Hàm lượng palindrome của xâu đã biến đổi bằng tổng hàm lượng palindrome từ các xâu con chỉ chứa một loại ký tự.

Nếu xâu con chỉ chứa một loại ký tự có độ dài k thì hàm lượng palindrome của nó sẽ là $\frac{k \times (k+1)}{2}$.

Không cần biến đổi trực tiếp xâu s ta cũng có thể tính được độ dài của các xâu con này.

Số lượng xâu con không quá 26 tương ứng với các ký tự từ **a** đến **z** và độ dài mỗi xâu bằng tổng số ký tự tương ứng xuất hiện trong xâu s .

Việc tính tần số xuất hiện các ký tự có thể thực hiện bằng cách duyệt xâu một lượt từ đầu đến cuối.

Tổ chức dữ liệu: Mảng `int z[26]={0}` – Lưu tần số xuất hiện các ký tự.

Độ phức tạp của giải thuật: O(n), trong đó n – độ dài xâu s .

Lưu ý: Kiểu dữ liệu của biến kết quả.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("maxpal.inp");
ofstream fo ("maxpal.out");
int ls, k, z[26]={0};
int64_t ans;
string s;

int main()
{
    fi>>s; ls=s.size();
    for(int i=0; i<ls; ++i) ++z[s[i]-97];
    for(int i=0; i<26; ++i) ans+=(int64_t)z[i]*(z[i]+1)/2;
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Trong lý thuyết trò chơi hàm **mex** đóng vai trò quan trọng. Hàm **mex** được định nghĩa như sau: Cho tập số nguyên dương **A**. **mex(A)** là số nguyên dương nhỏ nhất không có trong tập **A**. Ví dụ, với **A = {2, 1, 3, 5, 100}**, **mex(A) = 4**, với **A = {2, 3, 4, 5}**, **mex(A) = 1**.

Alice rất thích thú với vai trò và ứng dụng của hàm **mex**. Sẵn có trong tay dãy số nguyên dương **A = (a₁, a₂, ..., a_n)**, trong đó các số khác nhau từng đôi một, Alice quyết định thực hiện **k** lần phép bổ sung **mex** vào dãy, mỗi lần đưa thêm vào **A** số **mex** tìm được và làm tăng số phần tử của dãy lên 1.

Hãy xác định giá trị của phần tử cuối cùng được bổ sung vào dãy.

Dữ liệu: Vào từ file văn bản MEX.INP:

- ✚ Dòng đầu tiên chứa 2 số nguyên **n** và **k** ($1 \leq n \leq 10^5$, $1 \leq k \leq 10^9$),
- ✚ Dòng thứ 2 chứa **n** số nguyên khác nhau **a₁, a₂, ..., a_n** ($1 \leq a_i \leq 10^5$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản MEX.OUT một số nguyên – giá trị số cuối cùng được bổ sung vào dãy.

Ví dụ:

MEX.INP	MEX.OUT
7 10 1 3 20 2 7 45 5	15



Giải thuật: Kỹ thuật tổ chức dữ liệu .

Tạo bảng **f** đánh dấu các giá trị **a_i** xuất hiện, **i = 1 ÷ n**,

Ghi nhận **mxa** = $\max\{a_i\}$,

Lưu các giá trị nhỏ hơn **mxa** chưa được đánh dấu vào vector **b** theo thứ tự tăng dần của giá trị được lưu,

Gọi **p** là số lượng các số được lưu trong **b**.

$$\text{ans} = \begin{cases} b_{k-1} & \text{nếu } k \leq p, \\ mxa + k - p & \text{trong trường hợp ngược lại.} \end{cases}$$

Tổ chức dữ liệu:

- Mảng **int** **flg** [100001]={0} – Đánh dấu sự xuất hiện các phần tử của dãy **A**,
- Mảng **vector<int>** **b** – Ghi nhận các phần tử nhỏ hơn **mxa** và không có trong dãy **A**.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("mex.inp");
ofstream fo ("mex.out");

int main()
{
    int n, k, a, mx=0, p=0, ans, flg[100001]={0};
    vector<int> b;
    fi>>n>>k;
    for(int i=0; i<n; ++i)
    {
        fi>>a;
        if(a>mx) mx=a;
        flg[a]=1;
    }
    for(int i=1; i<=mx; ++i)
        if(flg[i]==0)
        {
            b.push_back(i); ++p;
        }
    if(k<=p) ans = b[k-1]; else ans = mx+k-p;
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA10. XÂU CON

Tên chương trình: SUBSTR.CPP

Cho hai xâu **s** và **t** chỉ chứa các ký tự la tinh thường. Xâu con của **s** là dãy các ký tự liên tiếp trong **s**. Hai xâu con gọi là khác nhau nếu chúng khác nhau ở vị trí bắt đầu hoặc vị trí kết thúc.

Xâu con các ký tự liên tiếp nhau bắt đầu từ vị trí đầu tiên của xâu được gọi là tiền tố của xâu đó.

Nói xâu **t** phủ lên xâu con của **s** nếu tồn tại một cách đổi chỗ các ký tự của **t** để nó có một tiền tố bằng xâu con đang xét.

Hãy xác định số lượng xâu con khác nhau của **s** được xâu **t** phủ.

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

- ✚ Dòng thứ nhất chứa xâu **s**,
- ✚ Dòng thứ 2 chứa xâu **t**.

Cả 2 xâu chỉ chứa các ký tự la tinh thường và mỗi xâu có độ dài không quá 10^6 .

Kết quả: Đưa ra file văn bản SUBSTR.OUT một số nguyên – số lượng xâu con tìm được.

Ví dụ:

SUBSTR.INP	SUBSTR.OUT
abacaba	
abc	15



WA10.lj20191005 - A XVI

Giải thuật: Kỹ thuật 2 con trỏ.

Nếu một xâu con của **s** được **t** phủ thì các xâu con của xâu con này cũng được phủ bởi **s**.

Ký hiệu **s[i..j]** là xâu con các ký tự liên tiếp nhau của **s** bắt đầu từ vị trí **i** cho đến vị trí **j** (kể cả **j**).

Xét các xâu con bắt đầu từ vị trí **i** của **s** (**i = 0 ÷ s.size() - 1**).

Gọi **r_i** là vị trí phải nhất trong **s** ứng với **i** để **s[i..r_i]** được phủ bởi **t**.

Dễ dàng thấy rằng **r_i ≤ r_{i+1}**.

Để **s[i..r_i]** được phủ bởi **t** thì tần số xuất hiện các ký tự khác nhau trong đó phải nhỏ hơn hoặc bằng tần số xuất hiện các ký tự tương ứng trong **t**.

Số xâu con thỏa mãn điều kiện bài toán trong **s[i..r_i]** sẽ là **r_i-i+1**.

Gọi **f_j** – tần số xuất hiện ký tự **j** trong **t**.

$$j = 0 \rightarrow a$$

$$j = 1 \rightarrow b$$

⋮ ⋮ ⋮ ⋮ ⋮

$$j = 25 \rightarrow z$$

Tìm **r₀**: duyệt các ký tự **s₀, s₁, s₂, ...**, giảm các **f_j** tương ứng chừng nào **f_j** còn lớn hơn 0. Khi gặp **k** có giá trị **f** tương ứng với **s_k** bằng 0 ta có **r₀ = k-1**.

Tìm **r₁**: Tăng giá trị **f** tương ứng với **s₀** lên 1 và duyệt tiếp tương tự từ vị trí **k** trong **s**.

Xử lý tương tự để tìm **r₂, r₃, ...**

Tổ chức dữ liệu:

Mảng **vector<int> f(26, 0)** – Lưu tần số xuất hiện các ký tự trong **t**.

Độ phức tạp của giải thuật: O(n), trong đó **n = max(s.size(), t.size())**.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("substr.inp");
ofstream fo ("substr.out");

int main()
{
    string s, t;
    fi >> s >> t;
    vector<int> f(26, 0);
    for (char c : t) f[c - 97]++;
    int q = 0, n = s.size();
    int64_t ans = 0;
    for (int i = 0; i < n; ++i)
    {
        while (q < n && f[s[q] - 97] > 0)
        {
            --f[s[q] - 97];
            ++q;
        }
        ans += q - i;
        ++f[s[i] - 97];
    }
    fo << ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA11. PHƯƠNG TRÌNH

Tên chương trình: EQUATION.CPP

Việc giải phương trình đại số Boolean với sự tham gia của các phép tính lô gic và xử lý bít khác khá xa với việc giải các phương trình đại số thông thường.

Để chứng minh cho điều đó, thầy giáo yêu cầu cả lớp về nhà xác định số lượng nghiệm không âm của từng phương trình trong số t phương trình, mỗi phương trình có dạng:

$$a - (a^x) - x = 0$$

trong đó:

- **x** là ẩn số,
- $0 \leq a < 2^{30}-1$,
- \wedge là phép tính XOR.

Dữ liệu: Vào từ file văn bản EQUATION.INP

- Dòng đầu tiên chứa số nguyên **t** ($1 \leq t \leq 1\,000$),
- Mỗi dòng trong **t** dòng sau chứa số nguyên **a** ($0 \leq a < 2^{30}$).

Kết quả: Đưa ra file văn bản EQUATION.OUT **t** số nguyên, mỗi số trên một dòng, số thứ **i** xác định số lượng nghiệm không âm của phương trình thứ **i**, $i = 1 \div t$.

Ví dụ:

EQUATION.INP	EQUATION.OUT
3	1
0	2
2	1073741824
1073741823	



WA10 Mos KB 20180110 BC A XVI

Giải thuật: Xử lý bit.

$$\mathbf{a} - (\mathbf{a}^{\wedge} \mathbf{x}) - \mathbf{x} = 0$$



$$\mathbf{a} = (\mathbf{a}^{\wedge} \mathbf{x}) + \mathbf{x}$$

Dễ dàng thấy được $\mathbf{x} = 0$ là một nghiệm của phương trình.

Hai toán hạng của vế phải đều không âm.

Gọi \mathbf{a}_i và \mathbf{x}_i là bít ở vị trí i của \mathbf{a} và \mathbf{x} , \mathbf{r}_i – bít thứ i của tổng nhận được ở vế phải phương trình.

$\mathbf{a}_i = \mathbf{r}_i$ khi và chỉ khi:

- $\mathbf{a}_i = \mathbf{x}_i = 0$,
- $\mathbf{a}_i = 1 \rightarrow \mathbf{x}_i = 1$ hoặc $\mathbf{x}_i = 0$.

Gọi k là số lượng bít 1 trong \mathbf{a} , số cách chọn \mathbf{x} sẽ là

$$C_k^0 + C_k^1 + C_k^2 + \dots + C_k^k = 2^k$$

Độ phức tạp của giải thuật: $O(t)$.

Chương trình I

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("equation.inp");
ofstream fo ("equation.out");

int main()
{
    int t,n,ans,r;
    fi>>t;
    for(int i=0; i<t; ++i)
    {
        fi>>n; r=0;
        for(int i=30; i>=0; --i) r+=(n>>i)&1;
        ans=1<<r;
        fo<<ans<<'\n';
    }
}
```

Chương trình II

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("equation.inp");
ofstream fo ("equation.out");

int main()
{
    int t,n,ans,r;
    fi>>t;
    for(int i=0; i<t; ++i)
    {
        fi>>n;
        ans=1<<__builtin_popcount(n);
        fo<<ans<<'\n';
    }
}
```



WA12. MUA CÔ CA

Tên chương trình: COCA.CPP

Giữa giờ nghỉ chuyên tiết học Alice và Tôm tới ô tô mát bán nước giải khát để mua Cô ca. Không may trong máy không còn một lon Cô ca nào. Hai bạn quyết định chạy ra phố mua và dĩ nhiên chỉ cần một người đi là đủ. Trời nắng gắt và ai cũng ngại đi. Hai bạn quyết định chơi một trò chơi nhỏ và ai thua sẽ phải đi.

Trong tay Alice đang có một băng giấy gồm các ô vuông, mỗi ô được tô một trong 2 màu Đỏ (**R**) hoặc Xanh (**B**). Độ rộng băng giấy bằng độ rộng ô vuông. Hai người lần lượt cắt băng thành các đoạn độ dài (*số ô trên đoạn*) lớn hơn 0

Quy tắc chơi là hai người lần lượt đi, ai đến lượt mình đi chọn một đoạn có ô ở hai đầu khác màu nhau và cắt đoạn đó ở vị trí tùy chọn để nhận được 2 đoạn mỗi đoạn có độ dài lớn hơn 0.

Ai đến lượt đi nhưng không thể chọn được đoạn để cắt là thua và phải đi mua Cô ca.

Alice đi trước.

Cho trạng thái băng giấy. Hãy xác định Alice sẽ thắng hay thua và đưa ra thông báo tương ứng là Win hoặc Lose với giả thiết cả hai đều biết cách đi tối ưu.

Dữ liệu: vào từ file văn bản COCA.INP gồm một dòng chứa xâu s mô tả trạng thái ban đầu của băng giấy, s chỉ chứa các ký tự trong tập {**R**, **B**}, độ dài không vượt quá 10^5 .

Kết quả: Đưa ra file văn bản COCA.OUT thông báo tương ứng tìm được.

Ví dụ:

COCA.INP	COCA.OUT
RBRB	Win



WA12 Io20191005 I V16

Giải thuật; Cơ sở lập trình.

Trường hợp ký tự hai đầu giống nhau: Alice không có sự lựa chọn và thua.

Trường hợp ký tự 2 đầu khác nhau: Sẽ tồn tại ít nhất một i để $s_{i-1} = s_0$ và $s_i = s_{n-1}$, trong đó n – độ dài của xâu s .

Thực hiện lát cắt ở giữa s_{i-1} và s_i sẽ có 2 đoạn, mỗi đoạn đều có ký tự 2 đầu giống nhau và Alice thắng.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("coca.inp");
ofstream fo ("coca.out");
string s;
int n;
int main()
{
    fi>>s;
    n=s.size()-1;
    if(s[0]==s[n]) fo<<"Lose"; else fo<<"Win";
}
```

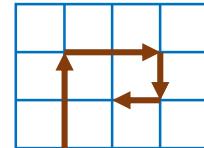


WA13. KÉO CẮT GIẤY

Tên chương trình: SCISSOR.CPP

Để trang trí phòng phục vụ tổ chức sinh nhật cho một người bạn Alice lấy một tờ giấy màu thủ công kẻ ô vuông kích thước $n \times m$ (n hàng và m cột), cắt thành hình lò xo xoắn theo hướng phải sang trái và có độ rộng của đường bằng 1:

- Bắt đầu từ biên phải cột 0 cắt lên trên cho đến khi cách lề trên một ô,
- Cắt sang phải theo đường biên dưới cho đến khi cách lề phải một ô,
- Cắt xuống dưới, rồi sang trái, sau đó lên trên, . . . để có băng giấy độ rộng 1 ô,
- Quá trình cắt sẽ dừng khi không cách cắt tiếp mà không làm đứt băng giấy.



Hãy tính tổng độ dài đường cắt theo đơn vị ô.

Dữ liệu: Vào từ file SCISSOR.INP gồm một dòng chứa 2 số nguyên n và m ($2 \leq n, m \leq 10^9$).

Kết quả: Đưa ra file văn bản SCISSOR.OUT một số nguyên – độ dài đường cắt.

Ví dụ:

SCISSOR.INP	SCISSOR.OUT
3 4	6



WA13 Io20191005 G V16

Giải thuật: Phân tích mô hình toán học.

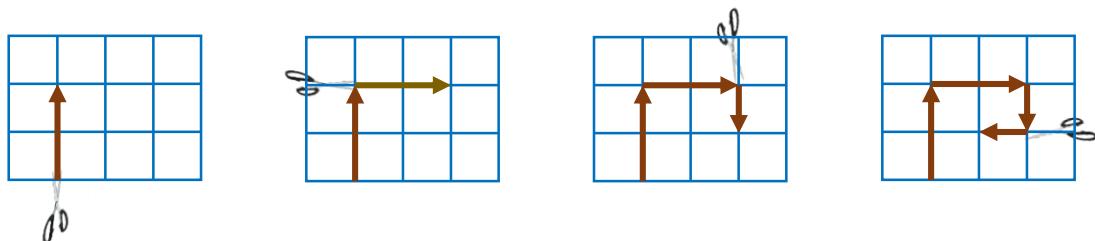
Đầu tiên xét trường hợp $n < m$.

Nhát cắt đầu tiên trước khi đổi hướng cắt có độ dài $n-1$.

Độ rộng của giấy phần bên phải của kéo là $m-1$,

Xoay kéo và ta có độ dài đường cắt tiếp theo là $m-2$,

Sau mỗi lần xoay kéo độ dài đường cắt sẽ giảm 1 so với độ dài đường cắt cùng hướng trước đó.



Việc cắt sẽ được thực hiện cho đến khi phần độ rộng bên phải kéo (sau khi cắt) có độ rộng 1.

Lần cắt	Độ dài lát cắt dọc	Độ dài lát cắt ngang
1	$n-1$	$m-1-1$
2	$n-2$	$m-1-2$
3	$n-3$	$m-1-3$
...
i	$n-i$	$m-1-i$
...
$n-1$	1	$m-1-(n-1)$

Như vậy số lần cắt theo chiều rộng và theo chiều dài là $n-1$.

Tổng độ dài các đường cắt sẽ là

$$n-1+m-2+n-2+m-3+\dots+1+(m-1-(n-1)) = \sum_{i=1}^{n-1} i + \sum_{m-2}^{m-n} i = (n-1) \times (m-1)$$

Kết quả vai trò của n và m là đối xứng, không phụ thuộc vào việc n lớn nhỏ hay bằng m .

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ifstream fi ("scissor.inp");
ofstream fo ("scissor.out");

int main()
{
    int64_t n,m,ans;
    fi>>n>>m;
    ans = (n-1) * (m-1);
    fo<<ans;
}
```



WA14. HAI PHẦN

Tên chương trình: TWOPARTS.CPP

Tổng Công ty trúng thầu xây dựng tuyến đường cao tốc bắc – nam có hai công ty con: Công ty Bắc và Công ty Nam. Con đường được chia thành n đoạn, đánh số từ 1 bắt từ bắc, đoạn thứ i đòi hỏi vốn chi phí thi công là a_i , $i = 1 \dots n$.

Công ty Bắc phụ trách thi công xây dựng phần đầu tuyến đường, Công ty Nam – phần cuối. Ở giai đoạn I, công ty Bắc thi công các đoạn đường từ 1 đến $1f$ (kể cả $1f$), công ty Nam – thi công các đoạn từ rt đến n .

Để đảm bảo việc bỏ vốn không chênh lệch quá nhiều giữa 2 công ty người ta quyết định chọn $1 \leq 1f < rt \leq n$, sao cho chênh lệch tổng vốn đầu tư phải ra giữa 2 công ty là nhỏ nhất khi xây dựng các đoạn từ 1 đến $1f$ và thi công xây dựng các đoạn từ rt tới n .

Hãy xác định giá trị chênh lệch nhỏ nhất có thể đạt được và các $1f$, rt tương ứng.

Dữ liệu: Vào từ file TWOPARTS.INP:

- ✚ Dòng đầu tiên chứa số nguyên n ($2 \leq n \leq 10^6$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^9$, $i = 1 \dots n$).

Kết quả: Đưa ra file văn bản TWOPARTS.OUT trên một dòng 3 số nguyên – giá trị chênh lệch nhỏ nhất có thể đạt được và các $1f$, rt tương ứng.

Ví dụ:

TWOPARTS.INP
4
1 2 3 4

TWOPARTS.OUT
1 2 4



Giải thuật: *Tổng tiền tố, Phương pháp 2 con trỏ.*

Chi phí đầu tư của Công ty Bắc là tổng tiền tố của dãy số chi phí,

Chi phí đầu tư của Công ty Nam là tổng hậu tố của dãy số chi phí,

Các tổng này đều đơn điệu tăng, do đó, xuất phát từ $\mathbf{lf} = 1$ và $\mathbf{rt} = n$, tăng dần \mathbf{lf} và giảm dần \mathbf{rt} , tìm chênh lệch nhỏ nhất giữa tổng tiền tố và tổng hậu tố tương ứng.

Nếu tổng tiền tố bé hơn tổng hậu tố thì tăng tổng tiền tố, tương ứng với việc tăng \mathbf{lf} .

Nếu tổng hậu tố bé hơn tổng tiền tố thì tăng tổng hậu tố, tương ứng với việc giảm \mathbf{rt} .

Tổ chức dữ liệu:

Mảng `vector<int64_t> a(n + 1)` – Lưu tổng tiền tố.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("twoparts.inp");
ofstream fo ("twoparts.out");

int main()
{
    int n;
    fi >> n;
    vector<int64_t> a(n + 1); a[0]=0;
    for (int i = 0; i < n; i++)
    {
        fi >> a[i + 1];
        a[i + 1] += a[i];
    }

    int l = 1, r = n - 1, resl = 0, resr = 0;
    int64_t diff = numeric_limits<int64_t>::max();
    while (l <= r)
    {
        int64_t lsum = a[l], rsum = a[n] - a[r];
        if (rsum > lsum)
        {
            if (diff > rsum - lsum)
            {
                diff = rsum - lsum;
                resl = l;
                resr = r;
            }
            l++;
        } else if (rsum < lsum)
        {
            if (diff > lsum - rsum)
            {
                diff = lsum - rsum;
                resl = l;
                resr = r;
            }
            r--;
        } else
        {
            diff=0; resl=l; resr=r; break;
        }
    }

    fo << diff << ' ' << resl << ' ' << resr + 1 << '\n';
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA15. TẬP LỚN NHẤT

Tên chương trình: MAX_SET.CPP

Cho dãy số nguyên dương $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$. Hãy tìm tập nhiều phần tử nhất có cùng ít nhất một ước số chung lớn hơn 1 và đưa ra số phần tử trong tập tìm được.

Ví dụ, với $\mathbf{A} = (6, 15, 10, 42)$, tập $\{6, 10, 42\}$ chứa các số cùng chia hết cho 2 và là tập nhiều phần tử nhất có cùng ít nhất một ước số chung lớn hơn 1. Số lượng các phần tử trong tập là 3.

Dữ liệu: Vào từ file MAX_SET.INP:

- ⊕ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 1000$),
- ⊕ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($2 \leq a_i \leq 10^{18}$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản MAX_SET.OUT một số nguyên – số lượng phần tử của tập tìm được.

Ví dụ:

MAX_SET.INP
4
6 15 10 42

MAX_SET.OUT
3



WA15 Io20191005 D A XVI

Giải thuật I: Phương pháp Leman phân tích ra thừa số nguyên tố.

Giải thuật Leman phân tích số nguyên ra thừa số

Xét số nguyên n lẻ và lớn hơn 8.

B1. Với $a = 2, 3, 4, \dots, \lfloor n^{1/3} \rfloor$ kiểm tra điều kiện n chia hết cho a , nếu tìm được các thừa số thì ghi nhận và giảm n một cách tương ứng.

Ký hiệu $m = \lfloor n^{1/3} \rfloor + 1$, nếu $m^2 \leq n$ – việc phân tích ra thừa số chưa kết thúc, chuyển tới bước B2.

B2. Có 2 khả năng:

- ✚ n – số nguyên tố,
- ✚ n – hợp số, khi đó $n = p^{2/3} \cdot q$ và q – nguyên tố, $d = 0, \dots, \left\lfloor \frac{n^{1/6}}{4\sqrt{k}} \right\rfloor + 1$ với $p < q <$

Xét với mọi $k = 1, 2, \dots, \lfloor n^{1/3} \rfloor$ và mọi kiểm tra số

$$(\lfloor \sqrt{4kn} \rfloor + d)^2 - 4kn$$

có phải là một số chính phương hay không. Nếu đó là một số chính phương thì với

$A = \lfloor \sqrt{4kn} \rfloor + d$ và $B = \sqrt{A^2 - 4kn}$ kiểm tra điều kiện:

$$\begin{aligned} A^2 &\equiv B^2 \pmod{n} \\ \text{hoặc } (A-B)(A+B) &\equiv 0 \pmod{n} \end{aligned}$$

Nếu điều kiện trên thỏa mãn thì $d^* = \gcd(A - B, n)$ tính và kiểm tra điều kiện $1 < d^* < n$.

Nếu tìm thấy d^* thỏa mãn điều kiện đã nêu $n = d^* \cdot (n/d^*)$ thì ta có , trong trường hợp ngược lại – có n là số nguyên tố.

Giải thuật Leman có thể được sử dụng để kiểm tra n có phải là số nguyên tố hay không.

Bước 2 của giải thuật Leman là sự phát triển của giải thuật Ferma và tìm ước của n dựa trên đẳng thức $x^2 - y^2 = 4 \times k \times n$. Giải thuật đặt nền móng trên cơ sở của định lý sau:

Định lý: Nếu n là một hợp số và $n = p \times q$, trong đó p, q – lẻ, là các số nguyên tố cùng nhau và thỏa mãn điều kiện $n^{1/3} < p < q < n^{2/3}$, khi đó tồn tại các số nguyên $x, y, k \geq 1$ thỏa mãn các điều kiện:

$$\text{Lý thuyết số} \quad \mathbf{x}^2 - \mathbf{y}^2 = 4kn \text{ với } k < n^{1/3},$$

$$0 \leq x - \lfloor \sqrt{4kn} \rfloor < \frac{n^{1/6}}{4\sqrt{k}} + 1$$

Bổ đề:

Nếu các điều kiện của định lý được thực hiện thì tồn tại các số nguyên \mathbf{r} và \mathbf{s} thỏa mãn các điều kiện $\mathbf{r} \times \mathbf{s} < n^{1/3}$ và $|\mathbf{p} \times \mathbf{r} - \mathbf{q} \times \mathbf{s}| < n^{1/3}$.

Chứng minh bổ đề:

Nếu $\mathbf{p} = \mathbf{q}$, tức là $n = \mathbf{p}^2$, thì ta có $\mathbf{r} = \mathbf{s} = 1$.

Xét trường hợp $\mathbf{p} < \mathbf{q}$. Phân tích q/p ra phân số liên tục. Gọi p_j/q_j là phân số thích hợp thứ j .

Ta có $\mathbf{p}_0 = \lfloor \mathbf{q}/\mathbf{p} \rfloor$, $q_0 = 1$ và $0 < \mathbf{p}_0 \times \mathbf{q}_0 < n^{1/3}$ vì $\mathbf{q}/\mathbf{p} = \mathbf{n}^{2/3}/\mathbf{n}^{1/3} = \mathbf{n}^{1/3}$.

Chọn m nhỏ nhất thỏa mãn điều kiện

$$\begin{cases} p_m q_m < n^{1/3} \\ p_{m+1} q_{m+1} > n^{1/3} \end{cases}$$

Số m tồn tại bởi vì ở phân số thích hợp cuối cùng mẫu số $\mathbf{q}_m = \mathbf{p} > n^{1/3}$. Ta sẽ chứng minh rằng $\mathbf{r} = \mathbf{p}_m$ và $\mathbf{s} = \mathbf{q}_m$ thỏa mãn điều khẳng định của bổ đề.

Rõ ràng là $\mathbf{r} \times \mathbf{s} < n^{1/3}$.

Từ tính chất của các phân số thích hợp có:

$$\left| \frac{r}{s} - \frac{q}{p} \right| \leq \left| \frac{r}{s} - \frac{p_{m+1}}{q_{m+1}} \right| = \frac{1}{sq_{m+1}}$$

Xét trường hợp

$$\frac{p_{m+1}}{q_{m+1}} \leq \frac{q}{p}$$

Ta có

$$|pr - qs| = ps \left| \frac{r}{s} - \frac{q}{p} \right| \leqslant \frac{ps}{sq_{m+1}} = \frac{p}{q_{m+1}} = \sqrt{\frac{p}{q_{m+1}}} \sqrt{\frac{p}{q_{m+1}}} \leqslant \sqrt{\frac{p}{q_{m+1}}} \sqrt{\frac{q}{p_{m+1}}} = \sqrt{\frac{n}{p_{m+1}q_{m+1}}} < \frac{n^{1/2}}{n^{1/6}} = n^{1/3}$$

Và đó là điều cần chứng minh.

Trường hợp

$$\frac{p_{m+1}}{q_{m+1}} > \frac{q}{p}$$

Ta có

$$\begin{aligned} \frac{p_{m+1}}{q_{m+1}} &> \frac{q}{p} > \frac{p_m}{q_m} \\ \frac{q_m}{p_m} &> \frac{p}{q} > \frac{q_{m+1}}{p_{m+1}} \end{aligned}$$

Từ đó suy ra

Theo tính chất của phân số liên tục, tồn tại bất đẳng thức:

Từ đó suy ra

$$\begin{aligned} \frac{1}{rq} &\leqslant \left| \frac{s}{r} - \frac{p}{q} \right| \leqslant \left| \frac{s}{r} - \frac{q_{m+1}}{p_{m+1}} \right| = \frac{1}{rp_{m+1}} \\ 1 &\leqslant |sq - pr| = rq \left| \frac{s}{r} - \frac{p}{q} \right| \leqslant \frac{rq}{rp_{m+1}} = \frac{q}{p_{m+1}} = \sqrt{\frac{q}{p_{m+1}}} \sqrt{\frac{q}{p_{m+1}}} \leqslant \sqrt{\frac{q}{p_{m+1}}} \sqrt{\frac{p}{q_{m+1}}} = \sqrt{\frac{n}{p_{m+1}q_{m+1}}} < \frac{n^{1/2}}{n^{1/6}} = n^{1/3} \end{aligned}$$

Đó là điều cần chứng minh.

Chứng minh định lý:

Gọi \mathbf{p} và \mathbf{q} là 2 ước lẻ của \mathbf{n} .

Đặt $\mathbf{x} = \mathbf{p} \times \mathbf{r} + \mathbf{q} \times \mathbf{s}$ và $\mathbf{y} = \mathbf{p} \times \mathbf{r} - \mathbf{q} \times \mathbf{s}$, trong đó \mathbf{r} và \mathbf{s} – các số thỏa mãn bù đê.

Khi đó:

$$x^2 - y^2 = (pr + qs)^2 - (pr - qs)^2 = 4rspq = 4kn$$

trong đó $\mathbf{k} = \mathbf{r} \times \mathbf{s}$.

Theo bù đê, \mathbf{k} thỏa mãn điều kiện $\mathbf{k} < \mathbf{n}^{1/3}$. Kết luận thứ nhất của định lý được chứng minh.

Chứng minh kết luận thứ 2:

Đặt $z = x - \lfloor \sqrt{4kn} \rfloor = pr + qs - \lfloor \sqrt{4kn} \rfloor$.

Từ $x^2 = 4kn + y^2$ suy ra $x \geqslant \sqrt{4kn}$ và $z \geqslant 0$.

Xuất phát từ đánh giá cận trên của y ta có:

$$n^{2/3} > y^2 = x^2 - 4kn = (pr+qs+\sqrt{4kn})(pr+qs-\sqrt{4kn}) \geq 2\sqrt{4kn}(pr+qs-\sqrt{4kn}) \geq 2\sqrt{4kn}(z-1)$$

Từ đó suy ra

$$z < \frac{n^{2/3}}{2\sqrt{4kn}} + 1 = \frac{n^{1/6}}{4\sqrt{k}} + 1$$

Định lý được chứng minh.

Danh giá độ phức tạp của giải thuật:

Ký hiệu $\lceil y \rceil$ là số nguyên nhỏ nhất lớn hơn hoặc bằng y .

Ở giai đoạn một cần thực $\lceil n^{1/3} \rceil$ hiện phép chia để tìm các ước nhỏ của n .

Độ phức tạp của bước 2 được xác định bởi số phép tính cần thiết để kiểm tra xem số có $(\lfloor \sqrt{4kn} \rfloor + d)^2 - 4kn$ phải là một chính phương hay không.

Ta nhận thấy với $k < \frac{n^{1/6}}{4}$ chính phương chỉ có thể đạt được một giá trị $d = 1$.

Như vậy độ phức tạp của giai đoạn 2 bị chặn trên bởi đại lượng

$$\frac{n^{1/6}}{4} \sum_{k=1}^{\lfloor n^{1/6} \rfloor} \frac{1}{\sqrt{k}} + 2(\lceil n^{1/3} \rceil - \lfloor n^{1/6} \rfloor) < 3\lceil n^{1/3} \rceil$$

Tổng kết chung, giải thuật có độ phức tạp $O(n^{1/3})$.

Các bước giải bài toán

Với mỗi số a_i :

- ❖ Phân tích ra thừa số nguyên tố, kết quả phân tích ở vector **divisors**,
- ❖ Lọc các thừa số nguyên tố khác nhau,
- ❖ Tích lũy tất cả số xuất hiện của các thừa số nguyên tố vào bản đồ tất cả số của toàn mảng số đã cho.

Tìm và đưa ra tất cả số xuất hiện lớn nhất.

Tổ chức dữ liệu:

- ❑ Mảng **vector<int64_t>** a ($m+1$) – Lưu trữ dữ liệu vào,
- ❑ Mảng **vector<uint64_t>** divisors – Lưu kết quả phân tích ra thừa số nguyên tố của một số,
- ❑ Bản đồ **map<int64_t, int>** mp – Lưu tất cả số xuất hiện của các thừa số nguyên tố.

Độ phức tạp của giải thuật: Gọi k là giá trị trung bình của các số trong dãy đã cho, độ phức tạp giải thuật sẽ là $O(n \times k)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "maxset."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

vector<uint64_t> find_prime_divisors_leman(uint64_t n)
{
    vector<uint64_t> divisors;
    uint64_t sqrt3_n_up=ceil(pow((long double)n, (long double)1.0/3))+2;

    // finding divisors <= n ^ (1 / 3)
    uint64_t m = 2;
    while (m * m <= n && m < sqrt3_n_up) {
        while (n % m == 0) {
            divisors.push_back(m);
            n /= m;
        }
        ++m;
    }
    if (m * m > n)
    {
        if (n != 1)
            divisors.push_back(n);
        return divisors;
    }

    // either n is prime or n = p*q, n ^ (1/3) < p <= q < n ^ (2/3)
    long double sqrt6_n = pow((long double)n, (long double)1.0 / 6);
    long double sqrt_n = sqrt((long double)n);
    for(int k = 1; k < sqrt3_n_up; ++k) {
        long double sqrt_k = sqrt((long double)k);
        uint64_t sqrt_4nk = ceil(2 * sqrt_k * sqrt_n) + 2;

        // diff = ([sqrt(4nk)]^2 - 4nk)
        uint64_t diff = sqrt_4nk*sqrt_4nk-4 * k*n;
        while (diff >= 2 * sqrt_4nk - 1) {
            diff = diff - 2 * sqrt_4nk + 1;
            --sqrt_4nk;
        }
        int max_d = ceil(sqrt6_n / (4 * sqrt_k)) + 3;
        for (int d = 0; d < max_d; ++d) {
            uint64_t a = sqrt_4nk + d;
            uint64_t b = round(sqrt((long double)diff));
            if (a * b == n)
                divisors.push_back(a);
        }
    }
}
```

```

        if (b * b == diff) {
            // diff is a perfect square, a ^ 2 = b ^ 2 mod n
            uint64_t d1 = __gcd(a + b, n);
            uint64_t d2 = __gcd(a - b, n);
            if (1 < d1 && d1 < n || 1 < d2 && d2 < n) {
                uint64_t p = (1 < d1 && d1 < n) ? d1 : d2;
                divisors.push_back(p);
                divisors.push_back(n / p);
                return divisors;
            }
        }
        diff += 2 * a + 1;
    }
}

// n is prime
divisors.push_back(n);
return divisors;
}

int main()
{
    uint64_t n,m,k,ans=0;
    map<int64_t,int> mp;
    vector<uint64_t> divisors;
    fi>>m;
    vector<int64_t> a(m+1); a[0]=0;
    for(int i=1; i<=m; ++i) fi>>a[i];
    sort(a.begin(),a.end());
    for(int i=1;i<=m;++i)
    {
        n=a[i];
        if(a[i]!=a[i-1])
        {
            divisors = find_prime_divisors_leman(n);
            k=unique(divisors.begin(),divisors.end())-divisors.begin();
        }

        for(int i=0; i<k; ++i) ++mp[divisors[i]];
    }

    for(auto e:mp) if(ans<e.second) ans=e.second;
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```

Giải thuật II: Phân nhóm theo ước số chung .

Với mỗi số a_i khác với các số đã gặp tìm các ước chung nguyên tố với các số trước đó trong dãy,

Tích lũy các ước tìm được vào mảng groups.

Tính tần số xuất hiện mỗi ước vào các số trong dãy ban đầu.

Đưa ra tần số lớn nhất tìm được.

Tổ chức dữ liệu:

- Mảng **vector** `<ll>` a – Lưu mảng ban đầu,
- Mảng **vector** `<ll>` groups – Lưu các ước số nguyên tố,
- Mảng **vector** `<int>` cnt (k) – Lưu tần số tham gia của ước vào các số trong dãy ban đầu.

Dộ phức tạp của giải thuật: $\approx O(n^2 \log n)$

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
typedef int64_t ll;
ifstream fi ("maxset.inp");
ofstream fo ("maxset.out");
vector<ll> a;
vector<ll> groups;

void solve(ll x)
{
    ll pw = 1;
    for (ll &y : groups)
    {
        ll val = y;
        ll t = __gcd(y, x);
        if (1 < t)
        {
            ll go = x / t;
            y = t;
            solve(go);
            solve(val / t);
            return;
        }
    }
    if (x != 1) groups.push_back(x);
}

int main()
{
    int n, k;
    fi >> n;
    a.resize(n);
    for(int i=0; i<n; ++i) fi>>a[i];
    sort(a.begin(), a.end()); groups.push_back(a[0]);

    for(int i=1; i<n; ++i) if(a[i] != a[i-1]) solve(a[i]);

    k=groups.size();
    vector<int> cnt(k);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < k; j++)
            if (a[i] % groups[j] == 0) cnt[j]++;
}

fo << *max_element(cnt.begin(), cnt.end()) << '\n';
fo << "\nTime: " << clock() / (double)1000 << " sec";
}
```



WA16. MẶT XÍCH YẾU NHẤT

Tên chương trình: NEXUS.CPP

Đoàn thanh niên tổ chức một cuộc chạy kêu gọi mọi người hưởng ứng phong trào “Vì một cuộc sống không có rác thải nhựa”. Có n người tham gia cuộc chạy, mỗi người mặc một áo phông có in số ở lưng, áo người thứ i có số là a_i , $i = 1 \div n$.

Sau cuộc chạy mọi người tập trung ở quảng trường thành phố, tham gia trò chơi tập thể “Khâu yếu nhất”. Các bạn đứng thành một vòng tròn, cạnh người thứ 2 là người thứ nhất và thứ 3, cạnh người thứ 3 là người thứ 2 và thứ tư, . . . , cạnh người thứ n là người thứ $n-1$ và người thứ nhất. Trò chơi bao gồm nhiều lượt đi. Ở mỗi lượt, những ai có số áo nhỏ hơn số áo hai người cạnh mình bước ra khỏi hàng, những người còn lại đứng dồn khít thành vòng tròn nhỏ hơn. Trò chơi kết thúc khi trong vòng tròn chỉ còn có 2 người hoặc khi không có ai phải bước ra ngoài.

Với mỗi người hãy xác định lượt đi mà họ phải bước ra ngoài. Những người còn lại trong vòng tròn khi trò chơi kết thúc có số của lượt đi ra là 0. Các lượt đi đánh số từ 1.

Dữ liệu: Vào từ file NEXUS.INP:

- ✚ Dòng đầu tiên chứa số nguyên n ($2 \leq n \leq 2 \times 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^9$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản NEXUS.OUT trên một dòng n số nguyên, số thứ i xác định lượt đi ra của người thứ i , $i = 1 \div n$.

Ví dụ:

NEXUS.INP	NEXUS.OUT
5	0 1 2 1 0
5 1 3 1 5	



WA16 Mos KB 20180110 BH A XVI

Giải thuật: *Ứng dụng stack*.

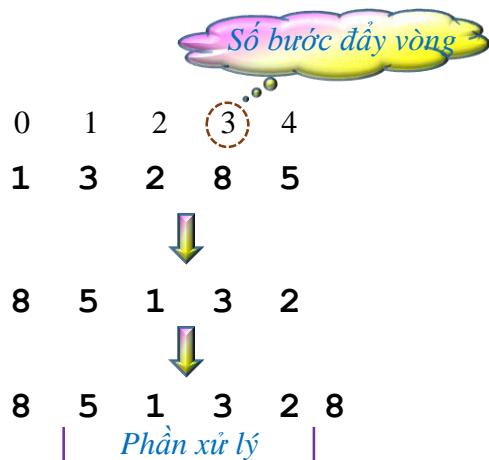
Nhận xét: Người có số lớn nhất sẽ không bị loại ra khỏi hàng,

Chuẩn hóa dữ liệu: Đánh số lại các người trong hàng, bắt đầu từ người có số lớn nhất để đưa về trường hợp người thứ nhất không bao giờ bị loại,

Xử lý dữ liệu vòng tròn: Bổ sung vào cuối hàng số của người thứ nhất.

Ta có hàng rào: Người thứ nhất và người cuối cùng trong dãy đã tạo ra không bị loại, bài toán xử lý vòng tròn trở thành bài toán xử lý tuyến tính.

Ví dụ: Dãy số ban đầu:



Lần lượt xét các phần tử trong phần cần xử lý.

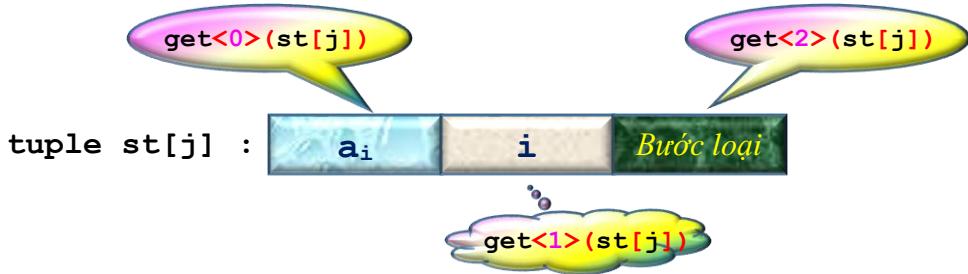
Mỗi phần tử a_i chỉ liên quan tới phần tử đứng sau nó trong dãy và *phần tử còn lại cuối cùng* ở bên trái.

Như vậy cần có stack lưu giữ các phần tử còn lại.

Việc ứng dụng trực tiếp stack sẽ làm tăng độ phức tạp giải thuật vì việc lấy một phần tử ra khỏi stack rồi nạp trở lại chính nó có thể xảy ra nhiều lần.

Sử dụng vector để mô phỏng stack cho phép ta dễ dàng *làm việc với hai phần tử đầu* trong stack.

Mỗi phần tử của stack có dạng nhóm 3:



Phân tử mới được đưa vào stack với *Bước loại* bằng 0.

Bước loại của phân tử sẽ tăng khi phân tử sau nó bị loại.

Nếu một phân tử bị loại, cần ghi nhận bước loại *theo vị trí ban đầu* của phân tử đó.

Tổ chức dữ liệu:

- ▀ Mảng `vector<int>` `a(n)` – Lưu các giá trị trong vòng tròn,
- ▀ Mảng `vector<int>` `st(n+1)` – Mô phỏng stack,
- ▀ Mảng `vector<int>` `rounds(n, 0)` – Lưu kết quả cần dẫn xuất.

Xử lý: Chuẩn bị:

```

fi>>n;
vector<int> a(n), rounds(n, 0);
for(int i=0; i<n; ++i)
{
    fi>>tp;
    a[i]=tp;
    if(mxa<tp) mxa=tp, imxa=i;
}
rotate(a.begin(), a.begin() + imxa, a.end());
a.push_back(mxa);

```

Số bước đầy vòng

Dụng hàng rào

Thực hiện đầy vòng

Xác định bước loại:

```
vector<tuple> st(n+1);
st[0]=make_tuple(0,0,0); st[1]=make_tuple(mxa,imxa,0);
tp=1; k=0;
for (int i=1; i<n; ++i)
{
    if (a[i]<a[i-1] && a[i]<a[i+1]) k=0;
    st[++tp] = make_tuple(a[i],i,0);
}
while (get<0>(st[tp-1])>get<0>(st[tp]) &&
       a[i+1]>get<0>(st[tp])
       && !(i==n-1 && tp==2))
{
    rounds[(get<1>(st[tp])+imxa)%n]=++k; --tp;
    get<2>(st[tp])=k;
}
}
```

Nạp phần tử mới

vào stack

Chuẩn bị xử lý

Điều kiện loại

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("nexus.inp");
ofstream fo ("nexus.out");
typedef tuple<int,int,int> tii;
int n,mxa=-1,imxa,v,id,rd,tp,k;

int main()
{
    fi>>n;
    vector<int> a(n), rounds(n,0);
    for(int i=0; i<n; ++i)
    {
        fi>>tp;
        a[i]=tp;
        if(mxa<tp) mxa=tp, imxa=i;
    }
    rotate(a.begin(),a.begin()+imxa,a.end());
    a.push_back(mxa);

    vector<tii>st(n+1);
    st[0]=make_tuple(0,0,0); st[1]=make_tuple(mxa,imxa,0); tp=1;
    k=0;
    for(int i=1; i<n; ++i)
    {
        if(a[i]<a[i-1] && a[i]<a[i+1]) k=0;
        st[++tp] = make_tuple(a[i],i,0);
        while(get<0>(st[tp-1])>get<0>(st[tp]) && a[i+1]>get<0>(st[tp])
            && !(i==n-1 & tp==2))
        {
            rounds[(get<1>(st[tp])+imxa)%n]=++k; --tp;
            get<2>(st[tp])=k;
        }
    }

    for(int i:rounds) fo<<i<<' ';
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA17. GIẢI THOÁT

Tên chương trình: `ESCAPE.CPP`

Một thiết bị thăm dò điều khiển từ xa được thả xuống khảo sát mặt đáy của một bể hóa chất hình chữ nhật kích thước $n \times m$ ô vuông. Thiết bị có thể được điều khiển để đi từ một ô sang ô kề cạnh. Trục chuyển động theo chiều ngang (sang ô cùng hàng) được kết nối với thiết bị làm lạnh, cứ mỗi lần di chuyển sang ô kề cạnh cùng hàng nhiệt độ bên trong thiết bị giảm đi 1. Trục chuyển động theo chiều dọc (sang ô cùng cột) được gắn với thiết bị làm nóng, cứ mỗi lần di chuyển sang ô kề cạnh cùng cột nhiệt độ bên trong thiết bị tăng thêm 1.

Sau khi hoàn thành nhiệm vụ khảo sát thiết bị đang ở ô được đánh dấu là '**s**' và có nhiệt độ bên trong là 0. Thật không may ống hút đưa thiết bị lên trên bị kẹt và chỉ có thể thu hồi thiết bị khảo sát nếu nó ở ô được đánh dấu '**f**'. Ngoài ra, khi đưa lên thiết bị cần có nhiệt độ gần 0 nhất có thể.

Tình trạng đáy của bể hóa chất được xác định bởi bản đồ **B** kích thước $n \times m$. $B_{i,j}$ được đánh dấu '.' nếu là ô trống và thiết bị thăm dò có thể đi qua. Nếu ô (i, j) có vật cản, không thể đi vào thì $B_{i,j}$ được đánh dấu là '#'. Tồn tại một ô được đánh dấu '**s**' và một ô khác – đánh dấu '**f**'.

Hãy xác định chênh lệch nhiệt độ tối thiểu (so với 0) thiết bị thăm dò có thể đạt được khi thoát ra khỏi bể.

Dữ liệu: Vào từ file `ESCAPE.INP`:

- ⊕ Dòng đầu tiên chứa 2 số nguyên **n** và **m** ($1 \leq n, m \leq 1000$),
- ⊕ Dòng thứ **i** trong **n** dòng sau chứa xâu độ dài **m** chứa các ký tự đã nêu, mô tả dòng thứ **i** của bản đồ **B**.

Kết quả: Đưa ra file văn bản `ESCAPE.OUT` một số nguyên – chênh lệch nhiệt độ tối thiểu có thể đạt được. Nếu không thể cứu được thiết bị thì đưa ra số -1.

Ví dụ:

ESCAPE.INP	ESCAPE.OUT
4 3 ..f ..# s## ...	0



WA17 Io20191005 H A XVI

Giải thuật: Duyệt đồ thị (DFS hoặc BFS).

Bằng kỹ thuật loang (DFS hoặc BFS) kiểm tra từ ô được đánh dấu **s** có thể tới ô được đánh dấu **f** hay không, nếu không tới được – đưa ra kết quả **-1**.

Đánh dấu trường hợp đi ngang và đi dọc.

Nếu tồn tại cách chuyển động theo cả hai hướng: có thể điều chỉnh nhiệt độ ở mức tùy ý bằng cách di chuyển nhiều lần theo một hướng. Khi đó nhiệt độ ra sẽ là 0 hoặc 1 tùy theo sự chênh lệch hàng và cột giữa hai ô s và f.

Nếu không tồn tại cách chuyển động của một hướng nào đó, nhiệt độ ra sẽ bằng độ dài đường đi.

Tổ chức dữ liệu:

- Mảng `vector<vector<int>> field(n, vector<int>(m))` – Lưu bản đồ của bể hóa chất,
- Mảng `vector<vector<bool>> used(n, vector<bool>(m))` – Đánh dấu kết quả thăm,
- Mảng `bool used_dir[2] = {false, false}` – Đánh dấu hướng chuyển động,
- Hàng đợi `queue<pii> qu` – phục vụ loang theo chiều rộng.

Độ phức tạp của giải thuật: $O(n \times m)$

Chương trình

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
typedef pair<int, int> pii;
ifstream fi ("escape.inp");
ofstream fo ("escape.out");
typedef long long ll;
int dx[4] = {0, 1, 0, -1};
int dy[4] = {1, 0, -1, 0};

int main()
{
    int n, m;
    fi >> n >> m;
    vector<vector<int>> field(n, vector<int>(m));
    int sx = -1, sy = -1, fx = -1, fy = -1;
    for (int i = 0; i < n; ++i)
    {
        string s;
        fi >> s;
        for (int j = 0; j < m; ++j)
        {
            if (s[j] == '#') field[i][j] = 1;
            if (s[j] == 's') sx = i, sy = j;
            if (s[j] == 'f') fx = i, fy = j;
        }
    }
    vector<vector<bool>> used(n, vector<bool>(m));
    bool used_dir[2] = {false, false};
    queue<pii> qu;
    qu.push({sx, sy});
    used[sx][sy] = true;
    while (!qu.empty())
    {
        int x=qu.front().ff;
        int y=qu.front().ss;
        qu.pop();
        for (int d = 0; d < 4; ++d)
        {
            int nx = x + dx[d];
            int ny = y + dy[d];
            if (0 <= nx && nx < n && 0 <= ny &&
                ny < m && field[nx][ny] == 0 && !used[nx][ny])
            {
                used_dir[d % 2] = true;
                used[nx][ny] = true;
                qu.push({nx, ny});
            }
        }
    }
}
```

```
    }
if (!used[fx][fy]) {fo << "-1\n"; return 0;}
if (used_dir[0] && used_dir[1])
    fo << (abs(fx - sx) + abs(fy - sy)) % 2 << '\n';
else fo<< abs(fx - sx) + abs(fy - sy) << '\n';
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA18. THÍCH ĐỒ NGỌT

Tên chương trình: SWEET.CPP

Trong đêm đốt lửa trại các bạn nắm tay nhau múa tập thể vòng quanh đồng lửa. Khi điệu múa kết thúc Alice chạy đi và lát sau quay về với một hộp kẹo lớn, đựng k chiếc kẹo.

Trong vòng tròn còn lại n người, đánh số từ 1 đến n theo chiều kim đồng hồ, bắt đầu từ Tôm – người bạn thân của Alice. Alice đưa hộp kẹo cho bạn số $1b$. Hộp kẹo được chuyền tay nhau theo chiều kim đồng hồ. Mỗi người khi nhận hộp kẹo (kể cả bạn số $1b$) lấy 1 hoặc 2 chiếc kẹo. Những người thích đồ ngọt lấy 2 chiếc, không thích – lấy một. Một người có thể được bốc nhiều lần vì hộp kẹo được chuyền đi vòng tròn.

Sau khi bốc hết kẹo người bốc viên kẹo cuối cùng trả hộp lại Alice và nếu Alice không nhầm thì đó là bạn số re . Sở dĩ Alice quan tâm đến bạn cuối cùng vì có thể trong hộp khi nhận được chỉ còn 1 chiếc kẹo và dù bạn ấy thích đồ ngọt thì cũng chỉ có thể bốc một chiếc. Nghĩ rộng hơn, Alice tự hỏi không biết trong số các bạn nhận kẹo vừa rồi tối đa có bao nhiêu bạn thích đồ ngọt.

Hãy xác định số lượng nhiều nhất các bạn thích đồ ngọt trong hàng.

Dữ liệu: Vào từ file SWEET.INP gồm một dòng chứa 4 số nguyên n , $1b$, re và k ($1 \leq n \leq 1000$, $1 \leq 1b, re \leq n$, $1 \leq k \leq 10^9$).

Kết quả: Đưa ra file văn bản SWEET.OUT một số nguyên – số lượng nhiều nhất các bạn thích đồ ngọt. Nếu Alice nhầm lẫn – đưa ra số -1.

Ví dụ:

SWEET.INP	SWEET.OUT
5 3 4 10	3



WA18 Mos KB 20180110 BI A XVI

Giải thuật: Vết cạn, Phân tích tình huống lô gic.

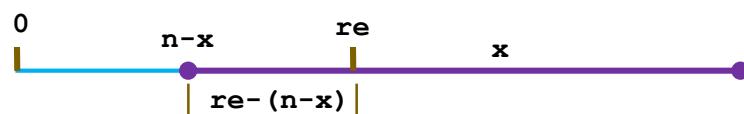
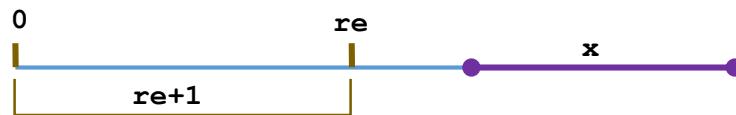
Đánh số lại bắt đầu từ 0, đưa **1b** về 0 và xác định giá trị **re** tương ứng,

Giả thiết có **x** bạn thích đồ ngọt, sau mỗi vòng chuyền tay nhau, số kẹo trong hộp giảm một lượng bằng $a = 2 \times x + n - x$.

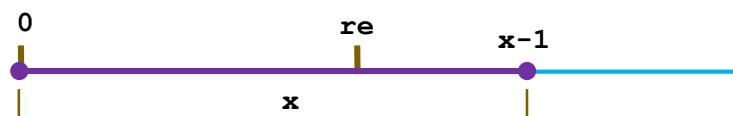
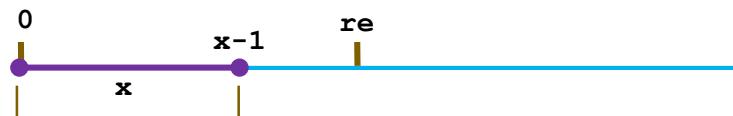
Số lượng kẹo còn trong hộp khi bắt đầu vòng cuối cùng là $b = k \% a$.

b sẽ nhỏ nhất khi tất cả những người thích đồ ngọt đứng ở cuối hàng và lớn nhất khi họ đứng ở đầu hàng.

Gọi **mn** – số lượng kẹo nhỏ nhất có thể trong hộp ở vòng cuối và **mx** – số lượng lớn nhất có thể.



$$mn = \begin{cases} re+1 & \text{nếu } re < n-x, \\ 2 \times re - n + x + 1 & \text{trong trường hợp ngược lại.} \end{cases}$$



$$mx = \begin{cases} re+x+1 & \text{nếu } re > x-1, \\ 2 \times re + 2 & \text{trong trường hợp ngược lại.} \end{cases}$$

Giá trị b có thể bằng 0 khi $re = n-1$ hoặc phải thỏa mãn điều kiện

$$mn \leq b \leq mx$$

Lần lượt kiểm tra các điều kiện trên với $x = n, n-1, n-2, \dots, 1, 0$, đưa ra giá trị x gặp đầu tiên thỏa mãn các điều kiện nêu trên.

Nếu không tìm được x – đưa ra **-1**.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("sweet.inp");
ofstream fo ("sweet.out");
int n,l,r,k,ans,mn,mx,b;

bool check(int x)
{
    int b;
    b=2*x+n-x;
    b=k%b;
    if(r<n-x) mn=r+1; else mn=2*r-n+x+1;
    if(x-1<r) mx=r+x+1; else mx=2*r+2;
    return ((b==0 && r==n-1) || (b>=mn && b<= mx));
}

int main()
{
    fi>>n>>l>>r>>k;
    ans=-1;
    r=(r-l+n)%n; l=0;
    for(int i=n; i>=0; --i)
        if(check(i)){ans=i; break;}
    fo<<ans<<'\n';
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Để có kinh phí bảo trì cầu đường bộ Giao thông Vận tải ra quy định người mua ô tô phải trả phí bảo trì phụ thuộc vào công suất của động cơ tính theo mã lực và thu khi đến đăng ký lái biển số.

Thuế được tính dựa theo bảng n cặp giá trị (p_i, t_i) , trong đó $p_1 = 0 < p_2 < p_3 < \dots < p_n$, $0 \leq t_1 \leq t_2 \leq \dots \leq t_n$, p_i là công suất của động cơ tính theo mã lực, t_i là hệ số thuế, $i = 1 \div n$.

Hệ số thuế người mua ô tô phải đóng là t_i nếu công suất động cơ lớn hơn p_i và nhỏ hơn hoặc bằng p_{i+1} . Nếu công suất động cơ lớn hơn hoặc bằng p_n thì hệ số sẽ là t_n . Thuế phải đóng bằng tích của hệ số thuế với công suất động cơ.

Hôm nay có m xe đến đăng ký lái biển số, xe thứ j có công suất động cơ là q_j , $j = 1 \div m$.

Hãy xác định thuế phải đóng của mỗi xe.

Dữ liệu: Vào từ file TAX.INP:

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên p_i và t_i ($0 \leq p_i, t_i \leq 10^9$ và thỏa mãn điều kiện đã nêu),
- ✚ Dòng tiếp theo chứa số nguyên m ($1 \leq m \leq 10^5$),
- ✚ Dòng thứ j trong m dòng sau chứa một số nguyên q_j ($1 \leq q_j \leq 10^9$).

Kết quả: Đưa ra file văn bản TAX.OUT m số nguyên, mỗi số trên một dòng xác định thuế phải đóng với mỗi xe.

Ví dụ:

TAX.INP	TAX.OUT
5	3745
0 24	5005
100 35	7600
150 50	8500
200 75	5250
250 150	
5	
107	
143	
152	
170	
150	



Giải thuật: Cơ sở lập trình.

Mô hình toán học:

Cho biết:

- n cặp số nguyên (p_i, t_i) sắp xếp theo thứ tự tăng dần $i = 1 \div n$,
- $p_0 = 0$,
- m số nguyên dương q_1, q_2, \dots, q_m .

Yêu cầu: với mỗi số q_j , $j = 1 \div m$:

- ❖ Nếu $q_j \leq p_n \rightarrow$ Tìm k thỏa mãn điều kiện $p_k < q_j \leq p_{k+1}$,
- ❖ Nếu $q_j > p_n \rightarrow$ Đặt $k = n$,
- ❖ Tính và đưa ra $q_j \times t_k$.

Giải thuật:

Lưu p_i và t_i với mọi i vào 2 mảng riêng biệt: mảng p và mảng t , chỉ số bắt đầu từ 0,

Tạo hàng rào: $p_n = 10^9 + 1$, $t_n = t_{n-1}$,

Với mỗi giá trị q_j :

- ❖ Dùng hàm **upper** tìm k – vị trí phần tử nhỏ nhất trong p có $p_k \geq q_j$,
- ❖ Nếu $p_k = q_j$ – giảm k đi một,
- ❖ Đưa ra $q_j \times t_{k-1}$.

Lưu ý:

- ➡ Kiểu dữ liệu của kết quả đưa ra,
- ➡ Không cần lưu mảng chứa các giá trị q_j .

Tổ chức dữ liệu:

- ❑ Mảng `vector<int> p(n+1)` – Lưu các p_i ,
- ❑ Mảng `vector<int> t(n+1)` – Lưu các t_i .

Độ phức tạp của giải thuật: $O(m \log n)$

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("tax.inp");
ofstream fo ("tax.out");
int n,m,q,k;
int64_t ans;

int main()
{
    fi>>n;
    vector<int> p(n+1),t(n+1);
    for(int i=0; i<n; ++i) fi>>p[i]>>t[i];
    p[n]=1e9+1; t[n]=t[n-1];
    fi>>m;
    for(int i=0; i<m; ++i)
    {
        fi>>q;
        k=upper_bound(p.begin(),p.end(),q)-p.begin()-1;
        if(q==p[k]) --k;
        ans=(int64_t)t[k]*q;
        fo<<ans<<'\n';
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA20. CHIẾC MŨ THẦN Bí

Tên chương trình: HAT.CPP

Sau khi làm bài thi các đội tập trung chơi trò chơi *Chiếc mũ thần bí*. Người ta viết mỗi tên đồ vật vào một mảnh giấy và bỏ vào mũ. Các từ được viết khác nhau từng đôi một, mỗi từ có độ dài không quá 10 và khác rỗng.

Mỗi lượt chơi một đội được chọn cử 2 người lên sân khấu. Người thứ nhất bốc từ mũ ra một mảnh giấy và cố gắng mô tả để người thứ 2 hiểu và nói đúng tên đồ vật viết trong giấy. Nếu người thứ 2 nói đúng đội đó sẽ được 1 điểm và tờ giấy đã rút ra sẽ bị hủy. Nếu người thứ 2 không nói đúng tên đồ vật, tờ giấy được bỏ trở lại vào mũ và đội đó không được điểm.

Sau mỗi lượt máy tính lại chọn ngẫu nhiên một đội nào đó (có thể trùng với đội vừa rời sân khấu) lên chơi. Có tất cả n đội tham gia, đánh số từ 1 đến n và trò chơi kéo dài m lượt.

Alice ngồi ghi biên bản. Nhưng do bị cuốn hút vào khói sôi nổi của hội trường Alice đã quên ghi điểm của đội chơi sau mỗi lượt, vì vậy biên bản có dạng m dòng, mỗi dòng chứa số nguyên xác định đội chơi và tên đồ vật ghi trên giấy được bốc ra.

Hãy giúp Alice tính điểm của mỗi đội biết rằng những tên đồ vật còn lại trong mũ (nếu có) đều là tên chưa được đội nào bốc ra.

Dữ liệu: Vào từ file HAT.INP

- ⊕ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n, m \leq 10^5$),
- ⊕ Dòng thứ i trong m dòng sau chứa số nguyên t_i và xâu s mà đội t_i bốc được.

Kết quả: Đưa ra file văn bản HAT.OUT trên một dòng n số nguyên – điểm số của mỗi đội.

Ví dụ:

HAT.INP	HAT.OUT
2 3 1 hat 1 shirt 2 hat	1 1



Giải thuật: Lọc dữ liệu.

Mô hình toán học:

Cho m cặp dữ liệu (t_i, w_i) ,

$1 \leq t_i \leq n$, w_i khác rỗng, chỉ chứa các ký tự la tinh thường độ dài không quá 10.

Yêu cầu: Đếm tần số của các t_i ứng với lần xuất hiện cuối cùng của w_i trong dữ liệu đã cho.

Giải thuật:

Lưu t_i và w_i ở 2 mảng riêng, $i = 1 \div m$,

Duyệt với i từ m lùi về 1:

- Nạp w_i vào tập s lưu xâu ký tự,
- Nếu nạp được \rightarrow tăng tần số của biến kết quả res_i .

Đưa ra các giá trị của res .

Tổ chức dữ liệu:

- Mảng `vector<int> team(m)` – Lưu số của đội lên sân khấu,
- Mảng `vector<string> w(m)` – Lưu xâu tên các đồ vật bốc được,
- Mảng `vector<int> res(n+1, 0)` – Lưu kết quả các đội,
- Tập `set<string> s` – Lưu tên các đồ vật được bốc.

Độ phức tạp của giải thuật: $O(m \log m)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("hat.inp");
ofstream fo ("hat.out");
int n,m;
set<string> s;

int main()
{
    fi>>n>>m;
    vector<int> team(m);
    vector<string> w(m);
    vector<int> res(n+1,0);
    for(int i=0; i<m; ++i) fi>>team[i]>>w[i];
    for(int i=m-1; i>=0; --i)
        if(s.insert(w[i]).second) ++res[team[i]];
    for(int i=1; i<=n; ++i) fo<<res[i]<<' ';
    fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
}
```



WA21. BIỂU THỨC NGOẶC

Tên chương trình: BRACKETS.CPP

Cho xâu **s** chỉ chứa các ký tự từ tập { ‘(‘, ’)’ , ‘[‘, ’]’ , ‘{‘, ’}’ } và các chữ cái la tinh thường.

Hãy xác định có cách bổ sung vào xâu đã cho các số và dấu phép tính, thay thế các ký tự chữ cái la tinh bằng chữ số hoặc số và dấu phép tính hay dấu phép tính và số hoặc dấu phép tính để nhận được một biểu thức số học đúng hay không. Dấu phép tính được sử dụng là +, -, *, hoặc /.

Dữ liệu: Vào từ file BRACKETS.INP gồm một dòng chứa xâu **s** khác rỗng độ dài không quá 10^6 chỉ chứa các ký tự trong tập đã nêu.

Kết quả: Đưa ra file văn bản BRACKETS.OUT thông báo **Yes** nếu có cách thay thế và bổ sung hoặc **No** trong trường hợp ngược lại.

Ví dụ:

BRACKETS.INP	BRACKETS.OUT
([abc] { }de)	Yes



WA21_A XVI

Giải thuật: Ứng dụng stack.

Việc thay ký tự chữ cái la tinh bằng chữ số hay chữ số và dấu phép tính hoặc dấu phép tính tương đương với việc xóa ký tự đó khỏi xâu.

Sau khi xóa các ký tự chữ cái latin ta được xâu **s'** chỉ chứa các ngoặc.

Nếu tìm thấy trong **s'** nếu tồn tại xâu con 2 ký tự liên tiếp nhau là “()”, “[]” hoặc “{}” thì có thể xóa xâu con đó mà không ảnh hưởng đến tính đúng đắn của **s'**.

Nếu kết quả xóa cho xâu **s'** rỗng ta có biểu thức ngoặc đúng và câu trả lời sẽ là **Yes**, trong trường hợp ngược lại – kết quả **No**.

Việc tìm xâu con để xóa được thực hiện bằng stack. Nếu ký tự tiếp theo trong **s'** là dấu ngoặc đóng và ký tự đỉnh stack là dấu ngoặc mở cùng loại – xóa ký tự ở đỉnh stack và xét ký tự tiếp theo của **s'**. Nếu gặp dấu ngoặc mở - nạp dấu ngoặc đó vào stack.

Tổ chức dữ liệu:

- Xâu **string s** – Lưu dữ liệu vào,
- Ngăn xếp **stack<char> st** – Lưu các ngoặc mở chưa bị xóa.

*Độ phức tạp của giải thuật: O(n), trong đó **n** – độ dài xâu **s**.*

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("brackets.inp");
ofstream fo ("brackets.out");
int n;
stack<char> st;
string s,ans;

int main()
{
    fi>>s; n=s.size();
    ans="Yes";
    for(int i=0; i<n; ++i)
    {
        if(s[i]>=97) continue;
        switch (s[i])
        {
            case '!': st.push(s[i]); break;
            case '[': st.push(s[i]); break;
            case '{': st.push(s[i]); break;
            case ')':if(st.top() != '(') ans="No"; else st.pop();break;
            case ']':if(st.top() != '[') ans="No"; else st.pop();break;
            case '}':if(st.top() != '{') ans="No"; else st.pop();break;
        }
        if(ans=="No") break;
    }
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA22.DÃY BÓNG BÓNG

Tên chương trình: BALLOONS.CPP

Có n bong bóng nổi thành một hàng dài trên mặt nước. Người ta phun hóa chất vào mỗi bong bóng, làm cho bong bóng thứ i có màu a_i , $i = 1 \dots n$. Các hóa chất phủ lên bong bóng có tính năng hấp thụ nhiệt và dãy bong bóng liên tiếp nhau cùng màu càng dài thì độ hấp thụ nhiệt càng lớn do hiệu ứng cộng hưởng. Vì vậy, nếu có 3 hay nhiều hơn bong bóng cùng màu đứng cạnh nhau thì chúng sẽ bị nổ ngay lập tức và các bong bóng bên cạnh sẽ dòn lại. Việc dòn lại sẽ kết thúc khi trong dãy không còn có 3 hay nhiều hơn quả bóng cùng màu đứng cạnh nhau.

Hãy xác định số bong bóng bị nổ.

Dữ liệu: Vào từ file BALLOONS.INP:

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^6$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^9$, $i = 1 \dots n$).

Kết quả: Đưa ra file văn bản BALLOONS.OUT một số nguyên – số lượng bong bóng bị nổ.

Ví dụ:

BALLOONS.INP
10
3 3 2 1 1 1 2 2 3 3

BALLOONS.OUT
10



Giải thuật: *Ứng dụng stack và Tích lũy tần số.*

Việc phát hiện dãy bong bóng bị nổ được dựa vào bong bóng đang xét,

Dãy bị nổ là dãy bong bóng cuối cùng được lưu trong quá trình xét.

Vì vậy, cấu trúc dữ liệu thích hợp sẽ là stack.

Khi nhập dữ liệu:

Mỗi dãy liên tiếp cùng màu sẽ được ghi nhận tần số xuất hiện của màu,

Gặp màu khác: Tần số được xác lập lại bằng 1.

Việc sáp nhập 2 nhóm cùng màu: Cập nhật lại tần số.

Khi gặp bóng khác màu: Kiểm tra tần số nhóm cuối cùng và tích lũy kết quả nếu tần số lớn hơn 2.

Cần lưu ý kiểm tra tần số nhóm cuối cùng còn lại.

Tổ chức dữ liệu:

- Mảng `vector<int>` ball (`n+1`) – Lưu màu của cụm bong bóng liên tiếp cùng màu,
- Mảng `vector<int>` cnt (`n+1`) – Lưu số lượng bong bóng cùng màu trong dãy.

Độ phức tạp của giải thuật: O(n).

Chương trình I Mô phỏng stack.

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("balloons.inp");
ofstream fo ("balloons.out");
int n, b, k=0, ans=0;

int main()
{
    fi>>n;
    vector<int> ball(n+1), cnt(n+1);
    ball[0]=0; cnt[0]=1;
    for(int i=0; i<n; ++i)
    {
        fi>>b;
        if(b==ball[k]) ++cnt[k];
        else
        {
            if(cnt[k]>=3) {ans+=cnt[k]; --k;}
            if(b==ball[k]) ++cnt[k]; else {cnt[++k]=1; ball[k]=b;}
        }
    }
    if(cnt[k]>2) ans+=cnt[k];
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```

Chương trình II Sử dụng stack tương minh.

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
typedef pair<int,int> pii;
ifstream fi ("balloons.inp");
ofstream fo ("balloons.out");
int n, b, ans=0;
stack<pii>st;

int main()
{
    fi>>n;
    st.push({0,0});
    for(int i=0; i<n; ++i)
    {
        fi>>b;
        if(b==st.top().ff) ++st.top().ss;
        else
        {
            if(st.top().ss>2) {ans+=st.top().ss; st.pop();}
            if(b==st.top().ff) ++st.top().ss;
            else st.push({b,1});
        }
    }
    if(st.top().ss>2) ans+=st.top().ss;
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA23. CHƠI BÀI

Tên chương trình: CARDS.CPP

Có $2n$ quân bài, trên mỗi quân bài in một số trong phạm vi từ 0 đến $2n-1$, không có 2 quân bài nào có số giống nhau. Các quân bài được tráo cẩn thận và chia thành 2 bộ, mỗi bộ n quân. Hai người chơi, mỗi người lấy một bộ.

Mỗi lượt đi hai người mở quân bài trên cùng ở bộ bài của mình. Ai có số 0 hoặc có số lớn hơn số trên quân bài đối phương sẽ thắng ở lượt đi này. Người thắng thu về cả 2 quân bài đã mở, nhét quân bài của mình vào cuối bộ bài, sau đó nhét tiếp quân bài đối phương, như vậy quân bài này sẽ nằm cuối cùng trong bộ bài người thắng ở lượt đi.

Sau một số lượt đi người nào không còn bài thì thua. Nếu sau 10^6 lượt đi mà chưa có ai thua thì được coi là hòa.

Hãy xác định người thắng hoặc thông báo hòa.

Dữ liệu: Vào từ file CARDS.INP:

- ✚ Dòng đầu tiên chứa số nguyên n , $1 \leq n \leq 10\,000$),
- ✚ Dòng thứ 2 chứa n số nguyên xác định các quân bài của người thứ nhất,
- ✚ Dòng thứ 3 chứa n số nguyên xác định các quân bài của người thứ hai.

Các quân bài được nêu theo thứ tự từ trên xuống dưới ở mỗi bộ bài.

Kết quả: Đưa ra file văn bản CARDS.OUT thông báo **First** nếu người thứ nhất thắng, thông báo **Second** nếu người thứ 2 thắng hoặc **Draw** nếu kết quả hòa. Trong trường hợp có người thắng thì đưa ra số lượt đi trên cùng dòng thông báo người thắng.

Ví dụ:

CARDS.INP
5
1 3 5 7 9
2 4 6 8 0

CARDS.OUT
Second 5



Giải thuật: Ứng dụng hàng đợi.

Các quân bài được lấy ra ở đầu cỗ bài,

Nếu thêm bài: bổ sung vào cuối.

Như vậy, cần tổ chức hàng đợi để lưu trữ.

Trò chơi kết thúc khi có một hàng đợi rỗng hay số nước đi đã thực hiện là 10^6 .

Tổ chức dữ liệu:

Hai hàng đợi để lưu trữ cỗ bài của mỗi người chơi.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("cards.inp");
ofstream fo ("cards.out");
int n,ca,cb,cnt=0;
queue<int> a, b;
string ans;
bool stop=false;

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i) {fi>>ca; if(ca==0) ca= 2*n+1; a.push(ca);}
    for(int i=0; i<n; ++i) {fi>>cb; if(cb==0) cb= 2*n+1; b.push(cb);}
    while(!stop)
    {
        ca=a.front(); a.pop();
        cb=b.front(); b.pop();
        if(ca>cb) {a.push(ca); a.push(cb);}
        else {b.push(cb); b.push(ca);}
        ++cnt;
        stop =a.empty() || b.empty() || cnt>=1000000;
    }
    if(b.empty()) ans="First"; else ans="Second";
    if(cnt>=1000000) fo<<"Draw"; else fo<<ans<<' '<<cnt;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA24. PHÂN TÍCH TẦN SỐ

Tên chương trình: FREQUENCY.CPP

Trong xử văn bản, việc tính tần số xuất hiện các từ là rất quan trọng và được sử dụng trong nhiều bài toán thực tế.

Cho một văn bản gồm nhiều dòng, mỗi dòng chứa một số từ, các từ trong một dòng cách nhau ít nhất một dấu cách, mỗi từ chỉ chứa các ký tự la tinh thường và có độ dài không quá 25 ký tự.

Hãy đưa ra các từ khác nhau trong văn bản theo trình tự giảm dần của tần số xuất hiện từ đó. Các từ có cùng tần số - đưa ra theo thứ tự từ điển.

Dữ liệu: Vào từ file FREQUENCY.INP gồm nhiều dòng chứa văn bản cần khảo sát. Số lượng từ khác nhau trong văn bản là không quá 10^5 .

Kết quả: Đưa ra file văn bản FREQUENCY.OUT danh sách các từ khác nhau theo yêu cầu đã nêu, mỗi từ trên một dòng.

Ví dụ:

FREQUENCY.INP	FREQUENCY.OUT
hi	damme
hi	is
what is your name	name
my name is bond	van
james bond	bond
my name is damme	claude
van damme	hi
claude van damme	my
jean claude van damme	james



WA24 A XVI

Giải thuật: Kỹ thuật tổ chức dữ liệu.

Việc tính tần số xuất hiện các từ khác nhau có thể dễ dàng thực hiện bằng cấu trúc dữ liệu **map**.

Ghi nhận kết quả tính tần số dưới dạng các cặp (- *Tần số*, *Xâu*),

Sắp xếp các cặp trên theo thứ tự tăng dần của trường thứ I.

Đưa ra các xâu (trường thứ II) theo trình tự nhận được.

Tổ chức dữ liệu:

- ▀ Bản đồ **map<string, int>** mp – Tính tần số xuất hiện từ,
- ▀ Mảng **vector<pair<int, string>>** res (n) – Truy xuất kết quả.

Độ phức tạp của giải thuật: O(nlogn).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("frequency.inp");
ofstream fo ("frequency.out");
typedef pair<int, string> pis;
int n, k;
map<string, int> mp;
string s;

int main()
{
    while (!fi.eof())
    {
        fi>>s;
        ++mp[s];
    }
    n=mp.size(); k=0;
    vector<pis> res(n);
    for (auto x:mp) res[k++] = {-x.second, x.first};
    sort(res.begin(), res.end());
    for (auto x:res) fo<<x.second<<'\n';
    fo<<"\nTime: "<<clock() / (double)1000<< " sec";
}
```



Bác sỹ Watson thường xuyên kiểm tra danh sách bệnh nhân nhập viện, thống kê số tiền bảo hiểm y tế được chi trả cho bệnh nhân, duyệt quyết định nhập viện, đôi khi ông ra lệnh chuyển bệnh nhân vừa mới nhập viện về sở y tế tuyến dưới và chuyển tiền bảo hiểm cho bệnh nhân đó cho nơi sẽ tiếp nhận bệnh nhân.

Cơ sở dữ liệu của bệnh viện ghi nhận biên bản hoạt động của ông dưới dạng mã hóa, mỗi quyết định ghi trên một dòng thuộc một trong 3 dạng sau:

- + **x** – Nhập viện cho bệnh nhân với số tiền bảo hiểm sẽ được chi trả là **x**,
- - Trả bệnh nhân cuối cùng trong danh sách nhập viện về cơ sở y tế tuyến dưới,
- ? **k** – Thống kê tổng số tiền bảo hiểm của k bệnh nhân cuối cùng còn trong bệnh viện.

Từ đầu năm tài chính đến nay biên bản này đã có **n** dòng. Với mỗi dòng dạng thứ 2 hãy chỉ ra số tiền bảo hiểm cần chuyển về cơ sở tuyến dưới, với mỗi dòng dạng thứ 3 – hãy cho tổng số tiền bảo hiểm tính được.

Dữ liệu: Vào từ file INSURANCE.INP:

- ⊕ Dòng đầu tiên chứa số nguyên **n** ($1 \leq n \leq 10^5$),
- ⊕ Mỗi dòng trong n dòng sau ghi một hoạt động của bác sỹ Watson, trong đó $1 \leq x \leq 10^9$, $1 \leq k \leq 10^5$.

Các hoạt động đều hợp lệ, tức là không có trường hợp chuyển khi trong bệnh viện không có bệnh nhân và số lượng bệnh nhân cần thống kê không vượt quá số lượng bệnh nhân đang có trong bệnh viện

Kết quả: Đưa ra file văn bản INSURANCE.OUT:

- ▣ Với mỗi hoạt động loại thứ 2 – đưa ra số tiền cần chuyển về tuyến cơ sở,
- ▣ Với mỗi hoạt động loại thứ 3 – đưa ra tổng số tiền tính được.

Ví dụ:

INSURANCE.INP	INSURANCE.OUT
7	5
+1	3
+2	2
+3	1
?2	
-	
-	
?1	



Giải thuật: Tổng tiền tố.

Các truy vấn đều liên quan tới một hoặc một số phần tử cuối cùng liên tiếp nhau.

Tổ chức dữ liệu:

Mảng **vector<int64_t>** f – Tích lũy tổng tiền tố số tiền bảo hiểm chi trả.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("insurance.inp");
ofstream fo ("insurance.out");
int n, m=0, k, t;
char c;

int main()
{
    fi>>n;
    vector<int64_t> f;
    f.push_back(0);
    for(int i=0; i<n; ++i)
    {
        fi>>c;
        switch(c)
        {
            case '-': fo<<f[m--]-f[m]<<'\n'; f.pop_back(); break;
            case '+': fi>>t; f.push_back(f[m++]+t); break;
            case '?': fi>>k; fo<<f[m]-f[m-k]<<'\n'; break;
        }
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



WA26. PHẢ HỆ

Tên chương trình: PEDIGREE.CPP

Cây phả hệ thể hiện quan hệ huyết thống của một nhóm người. Gốc của cây phả hệ là cụ tổ thuộc thế hệ 0. Mỗi người còn lại trong nhóm có đúng một người là cha (hoặc mẹ). Nếu cha hoặc mẹ có thế hệ là k thì con sẽ có thế hệ $k+1$.

Xét cây phả hệ của nhóm n người. Cây phả hệ sẽ chứa $n-1$ cặp quan hệ huyết thống (*Tên con, Tên Bố mẹ*). Tên là một xâu không quá 25 ký tự và không chứa dấu cách.

Với các cặp quan hệ đã cho, hãy xác định thế hệ của mỗi người và đưa ra theo thứ tự từ điển của tên.

Dữ liệu: Vào từ file PEDIGREE.INP:

- ✚ Dòng đầu tiên chứa số nguyên n ($2 \leq n \leq 10^5$),
- ✚ Mỗi dòng trong $n-1$ dòng tiếp theo chứa 2 tên xác định một cặp quan hệ huyết thống.

Kết quả: Đưa ra file văn bản PEDIGREE.OUT đưa ra n dòng, mỗi dòng chứa tên và một số nguyên thể hiện thế hệ của người đó.

Ví dụ:

PEDIGREE.INP	PEDIGREE.OUT
9	Alexander_I 4
Alexei Peter_I	Alexei 1
Anna Peter_I	Anna 1
Elizabeth Peter_I	Elizabeth 1
Peter_II Alexei	Nicholaus_I 4
Peter_III Anna	Paul_I 3
Paul_I Peter_III	Peter_I 0
Alexander_I Paul_I	Peter_II 2
Nicholaus_I Paul_I	Peter_III 2



Giải thuật: Kỹ thuật tổ chức dữ liệu, Tính bậc các nút của cây.

Sử dụng một cấu trúc map để ánh xạ từ tên sang số:

- Các tên khác nhau tương ứng với số khác nhau,
- Các tên giống nhau tương ứng với cùng một số.

Ghi nhận cấu trúc cây,

Ghi nhận ánh xạ ngược từ số sang tên,

Đánh dấu các nút là con của một nút khác,

Tìm nút tương ứng với cụ tổ trong dòng họ,

Bằng DFS xác định bậc của mỗi nút trong cây,

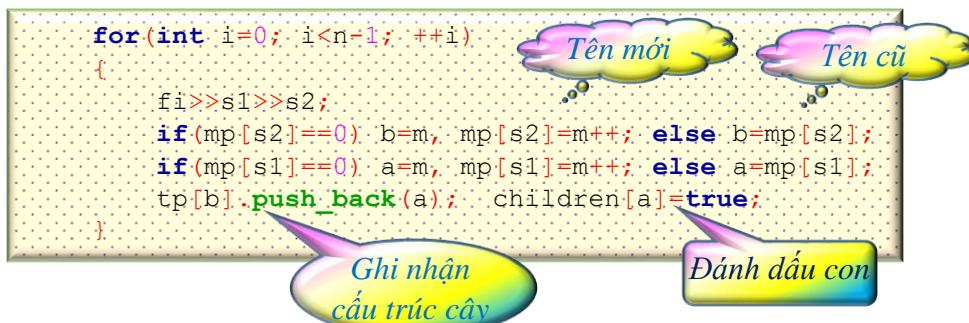
Dựa vào ánh xạ ngược, gắn bậc vào tên và đưa ra kết quả.

Tổ chức dữ liệu:

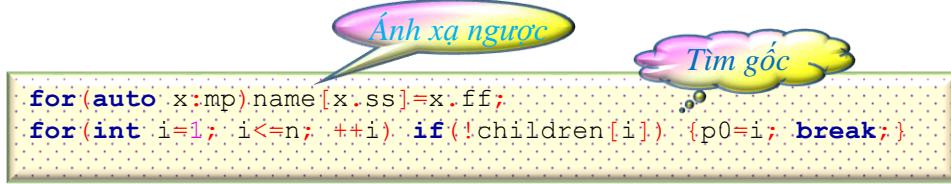
- Bản đồ `map<string, int>` mp – Phục vụ ánh xạ tên sang số,
- Mảng `vector<string>` name (n+1) – Ghi nhận ánh xạ ngược số → tên,
- Mảng `vector<bool>` children – Ghi nhận các nút là con của nút khác,
- Mảng `vector<bool>` visited – Đánh dấu phục vụ loang,
- Mảng `vector<vector<int>>` tp – Lưu trữ cấu trúc cây,
- Mảng `vector<int>` res – Lưu trữ bậc các nút của cây.

Xử lý:

Ánh xạ tên sang số và ghi nhận cấu trúc cây:



Ghi nhận ánh xạ ngược và tìm gốc của cây:



Tính thê hệ của các đối tượng:

```
void dfs(int p, int v)
{
    visited[p]=true;
    res[p]=v;
    for(int x:tp[p])
        if(!visited[x]) dfs(x,v+1);
}
```

Lời gọi: **dfs(p0, 0);**

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
ifstream fi ("pedigree.inp");
ofstream fo ("pedigree.out");
typedef pair<int,int> pii;
int n, m=1, k, a, b, p0;
string s1,s2;
vector<bool> visited, children;
vector<int> res;
vector<vector<int>> tp;

void dfs(int p, int v)
{
    visited[p]=true;
    res[p]=v;
    for(int x:tp[p])
        if(!visited[x]) dfs(x,v+1);
}
int main()
{
    fi>>n;
    map<string,int> mp;
    tp.resize(n+1);
    visited.assign(n+1,0);
    vector<string> name(n+1);
    children.assign(n+1,0);
    res.resize(n+1);

    for(int i=0; i<n-1; ++i)
    {
        fi>>s1>>s2;
        if(mp[s2]==0) b=m, mp[s2]=m++; else b=mp[s2];
        if(mp[s1]==0) a=m, mp[s1]=m++; else a=mp[s1];
        tp[b].push_back(a); children[a]=true;
    }

    for(auto x:mp) name[x.ss]=x.ff;
    for(int i=1; i<=n; ++i) if(!children[i]) {p0=i; break;}
    dfs(p0,0);
    for(int i=1; i<=n; ++i) mp[name[i]]=res[i];
    for(auto x:mp) fo<<x.ff<<' '<<x.ss<<'\n';

    fo<<"\nTime: "<<clock() / (double)1000<<" second";
}
```



WA27. SINH BA

Tên chương trình: TRINES.CPP

Một gia đình có 3 con sinh ba. Các con còn nhỏ nên cái gì bố mẹ cũng phải chia đều cho 3 anh em. Hôm nay nhà trường tổ chức đi tham quan và đề nghị bố mẹ cho kẹo vào túi cho các cháu để học sinh có thể ăn trên đường đi.

Khi mở túi các con, mẹ thấy ở một túi còn a chiếc kẹo, túi thứ 2 còn b chiếc và túi còn lại – còn c chiếc. Mẹ rất vội vì sắp tới giờ đưa con đi học nên bà bóc kẹo cả 2 tay, bỏ đồng thời vào hai túi, mỗi túi một viên.

Hãy xác định số lần bỏ thêm kẹo ít nhất để có số kẹo ở ba túi bằng nhau.

Dữ liệu: Vào từ file TRINES.INP gồm một dòng chứa 3 số nguyên a, b, c ($1 \leq a, b, c \leq 5 \times 10^8$).

Kết quả: Đưa ra file văn bản TRINES.OUT một số nguyên – số lần bỏ thêm kẹo tính được.

Ví dụ:

TRINES.INP	TRINES.OUT
1 2 3	3



WA27 StP 20192710 A XVI

Giải thuật: Nguyên lý bù trừ.

Số lèn bỏ sung thêm 1 vào hai túi tương đương với số lèn bót một ở một túi để cuối cùng số kẹo 3 túi là bằng nhau.

Số lèn lấy bót ít nhất sẽ làm số kẹo ở mỗi túi bằng số lượng kẹo nhỏ nhất ở 3 túi ban đầu.

Như vậy số lèn bỏ sung cần tìm sẽ là $a+b+c-3 \times \min\{a, b, c\}$.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("trines.inp");
ofstream fo ("trines.out");

int main()
{
    int a,b,c;
    fi>>a>>b>>c;
    fo<<(a+b+c)-3*min(a,min(b,c));
}
```



WA28. THIẾU HỤT

Tên chương trình: DEFICIENCY.CPP

Trong quỹ của một Công ty còn s đồng. Trong m ngày tới sẽ có n giao dịch liên quan tới thu chi: một số công việc đã hoàn thành và đối tác sẽ chuyển khoản vào quỹ của công ty một số tiền trong khoảng các ngày từ a đến b (kể cả b) hoặc phải trả một khoản tiền thanh toán cho vật tư đã mua, yêu cầu chi trả sẽ đến trong khoảng các ngày từ x đến y (kể cả y).

Khi có yêu cầu chi trả Công ty phải trả ngay trong ngày, ngay lúc nhận được thông báo. Việc chuyển tiền vào tài khoản của Công ty có thể được thực hiện ở ngày bất kỳ trong phạm vi từ a đến b .

Nếu trong một ngày xuất hiện nhiều giao dịch – các giao dịch có thể xuất hiện theo trình tự bất kỳ.

Hãy xác định có thể xảy ra tình trạng ở một ngày nào đó Công ty không có đủ tiền đáp ứng yêu cầu chi trả hay không.

Dữ liệu: Vào từ file DEFICIENCY.INP:

- ✚ Dòng đầu tiên chứa 3 số nguyên n, m và s ($1 \leq n, m \leq 1000, 1 \leq s \leq 10^6$),
- ✚ Mỗi dòng sau sau chứa 3 số nguyên $t, d1, d2$, trong đó nếu $t > 0$ là số tiền sẽ được chuyển tới, $t < 0$ – cần chi trả, $|t| \leq 10^6, 1 \leq d1 \leq d2 \leq m$ – phạm vi ngày có thể xuất hiện giao dịch.

Kết quả: Đưa ra file văn bản DEFICIENCY.OUT thông báo **YES** nếu có khả năng xảy ra thiếu tiền hoặc **NO** trong trường hợp ngược lại.

Ví dụ:

DEFICIENCY.INP	DEFICIENCY.OUT
4 3 100 100 1 3 -100 1 2 1 2 3 -1 2 2	YES



WA28 StP 20192710 B XVI

Giải thuật: *Nguyên lý cực trị, Tổng tiền tố.*

Tình trạng khó khăn nhất có thể xảy ra khi:

Việc chuyển khoản tiền tới Công ty diễn ra ở ngày cuối cùng trong thời hạn đã nêu,

Việc chi trả phải thực hiện ngay trong ngày đầu tiên ở thời hạn đã nêu.

Khoảng thời gian cần xét không lớn ($n \leq 1\ 000$) vì vậy có thể xây dựng hàm tổng tiền tố về số tiền còn lại từng ngày trong quỹ ứng với tình huống khó khăn nhất.

Cần ghi nhận số tiền cần chi trả ở đầu mỗi ngày và số tiền có thể nhận được ở cuối mỗi ngày.

Mỗi ngày cần kiểm tra khả năng chi trả ở đầu ngày và tổng số tiền có trong quỹ vào cuối ngày.

Nếu không có ngày nào tổng tiền tố ở đầu ngày nhận giá trị âm thì tình trạng thiếu hụt không xảy ra.

Tổ chức dữ liệu:

- Mảng `int64_t fp[1001]={0}` – Ghi nhận số tiền nhận thêm được ở cuối mỗi ngày,
- Mảng `int64_t fn[1001]={0}` – Ghi nhận số tiền cần chi trả ở đầu mỗi ngày.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("deficiency.inp");
ofstream fo ("deficiency.out");
int64_t fp[1001]={0},fn[1001]={0};
int n,m,s,t,d1,d2,dmx=0,remain;
string ans="NO";

int main()
{
    fi>>n>>m>>s; remain=s;
    for(int i=0; i<n; ++i)
    {
        fi>>t>>d1>>d2;
        if(t>0) {fp[d2]+=t; dmx=max(dmx,d2);}
        else {fn[d1]+=t; dmx=max(dmx,d1);}
    }

    for(int i=1; i<=dmx; ++i)
    {
        remain+=fn[i];
        if(remain<0) {ans="YES"; break;}
        remain+=fp[i];
    }
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Trong khu vực có n tỉnh, mỗi tỉnh có một trường chuyên. Giữa một số cặp 2 tỉnh (a, b) có tuyến xe tốc hành nối a với b và ngược lại. Ban Giám hiệu của một số trường chuyên có ký thỏa thuận hợp tác trao đổi kinh nghiệm với nhau. Hiện nay đã có m thỏa thuận được ký và có k tuyến tốc hành khác nhau. Giữa 2 cặp tỉnh có không quá một tuyến tốc hành.

Hàng năm một trường sẽ đăng cai tổ chức trại hè. Những trường có quan hệ hợp tác với trường đăng cai sẽ cử giáo viên và học sinh của mình tới dự nếu từ đó có thể tới tỉnh có trường đăng cai, trực tiếp hoặc qua một số tỉnh trung gian bằng xe tốc hành.

Theo thời gian, một số tuyến tốc hành mới được xác lập hòa vào mạng lưới tốc hành hiện có, một số cặp trường có thể ký thỏa thuận hợp tác, các quan hệ hợp tác cũ vẫn được giữ nguyên.

Thông tin về cặp quan hệ mới được đưa dưới dạng thông báo $F \ a \ b$ – hai trường a và b ký thỏa thuận hợp tác. Thông tin về tuyến tốc hành mới được đưa dưới dạng $T \ a \ b$ – có tuyến nối tỉnh a với tỉnh b và ngược lại.

Nếu trường đăng cai ở tỉnh v thì người ta cần biết trước sẽ có bao nhiêu trường từ các tỉnh bạn có thể tới tham dự và truy vấn sẽ có dạng $? \ v$.

Với nơi đăng cai cho trước hãy xác định số trường bạn tới dự.

Dữ liệu: Vào từ file SUM_CAMP.INP:

- ⊕ Dòng đầu tiên chứa số nguyên n, m và k ($1 \leq n \leq 10^5, 0 \leq m, k \leq 10^5$),
- ⊕ Mỗi dòng trong m dòng tiếp theo chứa 2 số nguyên a và b xác định 2 trường a và b có quan hệ hợp tác ($1 \leq a, b \leq n, a \neq b$), không có 2 dòng nào giống nhau,
- ⊕ Mỗi dòng trong k dòng tiếp theo chứa 2 số nguyên a và b xác định giữa 2 tỉnh a và b có tuyến tốc hành ($1 \leq a, b \leq n, a \neq b$), không có 2 dòng nào giống nhau,
- ⊕ Dòng tiếp theo chứa số nguyên q – số truy vấn cần xử lý ($1 \leq q \leq 10^5$),
- ⊕ Mỗi dòng trong q dòng sau chứa một truy vấn thuộc một trong các dạng đã nêu.

Kết quả: Đưa ra file văn bản SUM_CAMP.OUT, với mỗi truy vấn dạng $? \ v$ – đưa ra số trường bạn tới dự.

Ví dụ:

SUM_CAMP.INP
4 2 2
1 2
1 3
1 2
1 4
5
? 1
F 4 1
? 1
T 4 3
? 1

SUM_CAMP.OUT
1
2
3



Giải thuật: DSU.

Xây dựng đồ thị liên kết các trường có quan hệ hợp tác,

Từ mạng lưới tốc hành xác định các nhóm trường đã hợp tác có thể đi tới nhau bằng kỹ thuật DSU,

Xử lý truy vấn:

- ❖ Truy vấn loại **T from to** : Chính lý lại tập không liên thông.
- ❖ Truy vấn loại **F from to** :
 - Bổ sung thêm cạnh vào đồ thị quan hệ các trường,
 - Nếu 2 trường đã nêu có chung đại diện – cập nhật số lượng trường tới được.
- ❖ Truy vấn loại **? v** : Dẫn xuất số lượng trường tới được.

Tổ chức dữ liệu:

- ❑ Mảng **vector<int>** g [N] – Ghi nhận cấu trúc đồ thị,
- ❑ Mảng **int sz [N]** – Lưu số đỉnh trong thành phần liên thông,
- ❑ Mảng **int p [N]** – Lưu các đỉnh đại diện,
- ❑ Mảng **set<int> s [N]** – Lưu các đỉnh thuộc một đại diện,
- ❑ Mảng **int ans [N]** – Lưu kích thước miền liên thông của mỗi đỉnh.

Xử lý:

Khởi tạo DSU:

```
void dsuInit()
{
    for (int i = 0; i < N; ++i)
    {
        p[i] = i;
        sz[i] = 1;
        s[i].insert(i);
    }
}
```



Tìm đỉnh đại diện:

```
int dsuGet(int v)
{
    if (v == p[v]) return v;
    return p[v] = dsuGet(p[v]);
}
```

Hợp nhất 2 miền liên thông:

The diagram shows the `dsuUnite` function with several annotations:

- A yellow speech bubble labeled "Tìm đỉnh đại diện" points to the first part of the code where the root of each node is found.
- A blue speech bubble labeled "Tìm đỉnh liên thông" points to the `if (a == b) return;` statement.
- A pink cloud-like shape labeled "Gắn miền nhỏ vào miền lớn" points to the `swap(a, b)` statement.
- A yellow speech bubble labeled "Điều chỉnh kích thước miền liên thông" points to the `++ans[i];` and `++ans[to];` lines.
- A purple speech bubble labeled "Mở rộng danh sách của đại diện" points to the `s[b].insert(i);` line.
- A small green dot is positioned between the "Gắn miền nhỏ vào miền lớn" and "Điều chỉnh kích thước miền liên thông" annotations.

```
void dsuUnite(int a, int b)
{
    a = dsuGet(a);
    b = dsuGet(b);

    if (a == b) return;
    if (sz[a] > sz[b]) swap(a, b) Tìm đỉnh liên thông

    for (auto i : s[a])
    {
        for (auto to : g[i])
            if (dsuGet(to) == b)
            {
                ++ans[i];
                ++ans[to];
            }
        s[b].insert(i); Mở rộng danh sách của đại diện
    }
    p[a] = b;
    sz[b] += sz[a];
}
```

Xử lý truy vấn F:

```
if (type == 'F')  
{  
    int from, to;  
    fi>>from>>to;  
    --from; --to;  
  
    g[from].push_back(to);  
    g[to].push_back(from);  
  
    if (dsuGet(from) == dsuGet(to))  
    {  
        ++ans[from];  
        ++ans[to];  
    }  
}
```

Chỉnh lý đồ thị

*Chỉnh lý số phần tử
thuộc tập liên thông*

Xử lý truy vấn T:

```
if (type == 'T')  
{  
    int from, to;  
    fi>>from>>to;  
    --from; --to;  
  
    dsuUnite(from, to);  
}
```

*Mở rộng miền
liên thông*

Dộ phức tạp của giải thuật: $\approx O((q+m+(k+q)\log n))$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("sum_camp.inp");
ofstream fo ("sum_camp.out");

const int N = 100100;
set<int> s[N];
int p[N], sz[N];
vector<int> g[N];
int ans[N];

void dsuInit()
{
    for (int i = 0; i < N; ++i)
    {
        p[i] = i;
        sz[i] = 1;
        s[i].insert(i);
    }
}

int dsuGet(int v)
{
    if (v == p[v]) return v;
    return p[v] = dsuGet(p[v]);
}

void dsuUnite(int a, int b)
{
    a = dsuGet(a);
    b = dsuGet(b);

    if (a == b) return;
    if (sz[a] > sz[b]) swap(a, b);

    for (auto i : s[a])
    {
        for (auto to : g[i])
            if (dsuGet(to) == b)
            {
                ++ans[i];
                ++ans[to];
            }
        s[b].insert(i);
    }
    p[a] = b;
    sz[b] += sz[a];
}

int main()
```

```

{
    int n, m, k;
    fi>>n>>m>>k;
    while (m--)
    {
        int from, to;
        fi>>from>>to;
        --from; --to;
        g[from].push_back(to);
        g[to].push_back(from);
    }

    dsuInit();

    while (k--)
    {
        int from, to;
        fi>>from>>to;
        --from; --to;
        dsuUnite(from, to);
    }

    int q;
    fi>>q;

    while (q--)
    {
        char type;
        fi>>type;

        if (type == 'T')
        {
            int from, to;
            fi>>from>>to;
            --from; --to;

            dsuUnite(from, to);
        }
        else
            if (type == 'F')
            {
                int from, to;
                fi>>from>>to;
                --from; --to;

                g[from].push_back(to);
                g[to].push_back(from);

                if (dsuGet(from) == dsuGet(to))
                {
                    ++ans[from];
                    ++ans[to];
                }
            }
    }
}

```

```
        }
    }
else
{
    int v;
    fi>>v;
    fo<<ans[v-1]<<'\\n';
}
fo<<"\\nTime: "<<clock() / (double)1000<<" sec";
}
```

