

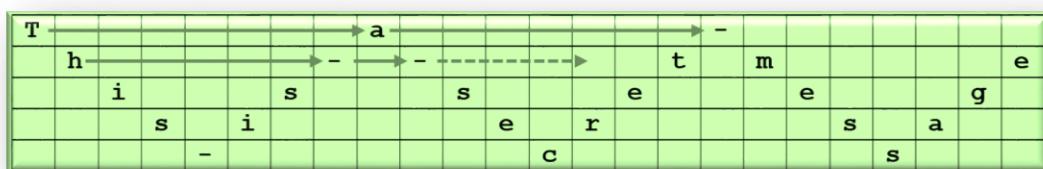
Mục lục ATHENA XII

VW03. MÃ HÓA LƯƠI CUA	Tên chương trình: SAW.CPP	3
VW04. THỬ NGHIỆM	Tên chương trình: CHECKOUT.CPP	6
VW05. LÀM GIÀU QUÄNG	Tên chương trình: ENRICHMENT.CPP	9
VW06. TÊN ĐĂNG NHẬP	Tên chương trình: ACCOUNT.CPP	12
VW07. VITAMIN	Tên chương trình: VITAMIN.CPP	17
VW08. NGHE NHẠC	Tên chương trình: PLAYER.CPP	20
VW09. TẢI FILES	Tên chương trình: DOWNLOAD.CPP	22
VW10. HỢP TÁC	Tên chương trình: PAIRING.CPP	24
VW11. PHỦ SÓNG WIFI	Tên chương trình: WIFI.CPP	27
VW12. BÓC BÀI	Tên chương trình: G21.CPP	30
VW13. ĐÊ CHÂN SÓNG	Tên chương trình: DIKE.CPP	32
VW14. SẮP BÀI	Tên chương trình: SOLITAIRE.CPP	35
VW15. TINH DẦU OÀI HƯƠNG	Tên chương trình: LAVENDER.CPP	37
VW16. KHÉO TAY	Tên chương trình: DEXTERITY.CPP	39
VW17. MÃ TRÂN PALINDROME	Tên chương trình: MATRIX.CPP	42
VW18. SỐ ĐO	Tên chương trình: METRICS.CPP	45
VW19. GIÁ SÁCH	Tên chương trình: SHELF.CPP	49
VW20. BẢN ĐỒ DU LỊCH	Tên chương trình: TOURMAP.CPP	53
VW21. DÃY CON	Tên chương trình: SUBSEQ.CPP	56
VW22. QUY LUẬT	Tên chương trình: REGUARITY.CPP	60
VW23 SỐ ĐẸP	Tên chương trình: NICE_NUM.CPP	62
VW24. KHẢO SÁT TÂM LÝ	Tên chương trình: PSYCHOLOGY.CPP	66
VW25. TÀI KHOẢN NGÂN HÀNG	Tên chương trình: BANK_AC.CPP	70
VW26. KHÔI PHỤC	Tên chương trình: RESTORE.CPP	74
VW27. GIẢI MÃ	Tên chương trình: DECRYPTION.CPP	77
VW28. TRỒNG RAU	Tên chương trình: GARDENING.CPP	80
VW29. THAY ĐỔI MÃ KHÓA	Tên chương trình: CHANGEWK.CPP	82
VW30. GIAO DỊCH ỐN ĐỊNH	Tên chương trình: TRANSACTION.CPP	85
VW31. ĐOÀN TÀU DU LỊCH	Tên chương trình: TOURISTS.CPP	87
VW32. GẦN PALINDROME	Tên chương trình: NEARLY.CPP	89
VW33. HAI TUYẾN DU LỊCH	Tên chương trình: TOURS.CPP	92
VW34. DẪN ĐƯỜNG	Tên chương trình: AUTOGUIDE.CPP	95
VW35. PHÂN TÍCH CÚ PHÁP	Tên chương trình: PARSER.CPP	97

VW36. CÙNG CÓ GIÁ TRỊ LỚN NHẤT	<i>Tên chương trình: EQ_MAX.CPP</i>	101
VW37. TỔNG FIBONACCI	<i>Tên chương trình: FIBSUM.CPP</i>	106
VW38. BIỂU ĐỒ DICK	<i>Tên chương trình: DICK_DIAG.CPP</i>	109
VW39. SỮA NGÔ	<i>Tên chương trình: JUICE.CPP</i>	111
VW40. CHIA KẸO	<i>Tên chương trình: CANDIES.CPP</i>	113
VW41. PHẦN THƯỞNG	<i>Tên chương trình: PRIZES.CPP</i>	115
VW42. TỪ MỚI	<i>Tên chương trình: NEWWORDS.CPP</i>	117
VW43. DÃY CON 0 1	<i>Tên chương trình: SUB_01.CPP</i>	120
VW44. IPyX	<i>Tên chương trình: IPVX.CPP</i>	124
VW45. VĂN TỰ CỘ	<i>Tên chương trình: ANC_TXT.CPP</i>	129
VW46. KHỞI NGHIỆP	<i>Tên chương trình: STARTUP.CPP</i>	132
VW47. CÔNG THỨC CHÉ BIẾN	<i>Tên chương trình: RECEIPT.CPP</i>	134
VW48. KIM TỰ THÁP	<i>Tên chương trình: PYRAMID.CPP</i>	136
VW49. PHỦ ĐIỂM	<i>Tên chương trình: COVER.CPP</i>	139
VW50. CẢI TIẾN THỨ BẮC	<i>Tên chương trình: UPTURN.CPP</i>	142
VX01. ĐỘ TRÙ MẬT	<i>Tên chương trình: DENSITY.CPP</i>	144
VX02. SẢN XUẤT ĐÁ	<i>Tên chương trình: STONES.CPP</i>	147
VX03. ĐƠN ĐIỀU	<i>Tên chương trình: MONOTONE.CPP</i>	149
VX04. BÁNH KEM XỐP	<i>Tên chương trình: WAFERS.CPP</i>	152
VX05. XỬ LÝ BIT	<i>Tên chương trình: AND_XOR.CPP</i>	155
VX06. DÃY SỐ KHÔNG GIẢM	<i>Tên chương trình: NON_DECR.CPP</i>	161
VX07. XÓA SỐ	<i>Tên chương trình: ABANDON.CPP</i>	163
VX08. MINH HỌA	<i>Tên chương trình: ILLUS.CPP</i>	165
VX09. PHÁO HÓA	<i>Tên chương trình: FIREWORKS.CPP</i>	168
VX10. HỢP NHẤT	<i>Tên chương trình: COALESCE.CPP</i>	171
VX11. TƯỚI NƯỚC	<i>Tên chương trình: IRRIGATION.CPP</i>	173

Máy tính điện tử phổ biến ngày nay có tốc độ vài trăm triệu phép tính/giây, nhưng người ta vẫn phải dạy học sinh lớp 1 phép tính cộng và phải kiên trì chờ đợi 30 giây khi đưa ra câu hỏi 5+6 bằng bao nhiêu. Mọi phương pháp mã hóa thông tin, dù là đơn giản nhất, đều có vai trò và vị trí trong cuộc sống. Có thể bạn muốn gửi một thông báo với nội dung riêng tư và năm, mười phút nữa sẽ không còn tác dụng cho dù người khác biết. Ví dụ, khi tan lớp bạn muốn nhắc người bạn thân của mình “*Hãy chúc mừng sinh nhật bạn ấy đi!*” thì hãy mã hóa một cách đơn giản theo phương pháp cả hai đều biết. Nếu ai đó có cố tìm cách giải mã được thì có thể đã quá muộn để quấy phá – mọi người đã ra về hết.

Một trong số các cách mã hóa đơn giản là phương pháp lưỡi cưa. Thông báo có độ dài m được viết trên lưới ô vuông n dòng và m cột. Các ký tự được điền vào ô, mỗi ô một ký tự, bắt đầu từ ô trên trái, theo đường chéo cho đến khi gặp ô ở hàng cuối cùng – điền tiếp theo các ô đường chéo đi lên, . . . cứ như thế cho đến khi hết thông báo, sau đó tạo xâu mới bằng cách ghi liên tiếp các ký tự ở hàng thứ nhất (trên cùng) từ trái qua phải, sau đó ghi tiếp các ký tự ở hàng thứ 2, tiếp theo – ghi các ký tự ở hàng thứ 3, . . .



Ví dụ, với thông báo “**This-is-a-secret-message**” ta có kết quả mã hóa “**Ta-h--tmeisseegsiersa-cs**”.

Cho số nguyên n và xâu s độ dài không quá 10^6 ký tự, các ký tự đều thuộc loại hiển thị được, không có dấu cách ở đầu hay cuối thông báo. Hãy đưa ra kết quả mã hóa.

Dữ liệu: Vào từ file văn bản SAW.INP:

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^3$),
- ✚ Dòng thứ 2 chứa xâu s .

Kết quả: Đưa ra file văn bản SAW.OUT xâu đã mã hóa.

Ví dụ:

SAW.INP	SAW.OUT
5 This-is-a-secret-message	Ta-h--tmeisseegsiersa-cs

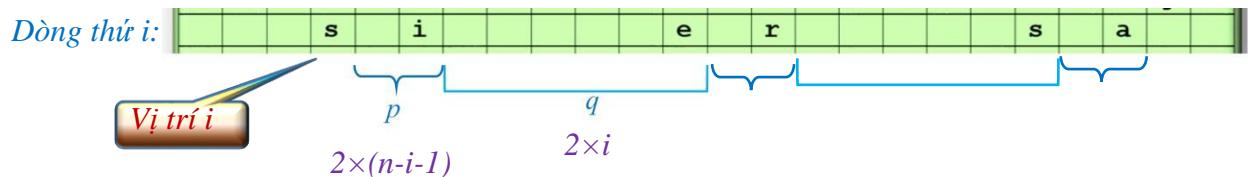


Giải thuật: Phương pháp 2 con trỏ.

Đánh số các dòng từ 0 trở đi từ trên xuống dưới,

Vị trí ký tự đầu tiên ở dòng thứ i là i ,

Các ký tự liên tiếp nhau trên một dòng cách nhau đan xen p vị trí hoặc q vị trí:



Sau khi đưa ra ký tự đầu tiên của dòng lần lượt đưa ra các ký tự ở vị trí p và q theo công thức nêu ở hình trên chừng nào p và q còn trong phạm vi của xâu,

Dòng đầu tiên có $q = 0$, dòng cuối cùng – có $p = 0$ cho biết không có ký tự tương ứng với các tham số này.

Tổ chức dữ liệu: Chỉ cần lưu xâu ban đầu và xâu kết quả.

Xử lý:

Xâu có thể chứa dấu cách (mã 32) vì vậy cần nhập theo chế độ getline,

Khi ghi nhận các ký tự trên một dòng cần tổ chức một chu trình với điều kiện ra là khi p hoặc q chỉ tới vị trí ngoài xâu ban đầu. Thuận tiện hơn cả là tổ chức một vòng lặp vô hạn và thoát ra khi tham chiếu tới vị trí ngoài xâu.

Độ phức tạp của giải thuật: $O(m)$, trong đó m – độ dài của xâu ban đầu.

Chương trình

```
#include<bits/stdc++.h>
#define NAME "saw."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,ls,k=0,p,q,u;
string s,ans;
int main()
{
    fi>>n;
    getline(fi,s);getline(fi,s); ls=s.size();
    ans=s;
    for(int i=0;i<n;++i)
    {
        p=2*(n-i-1); q=i<<1; ans[k++]=s[i]; u=i;
        while(1)
        {
            if(u+p>=ls)break;
            if(p>0)u+=p, ans[k++]=s[u];
            if(u+q>=ls)break;
            if(q>0)u+=q, ans[k++]=s[u];
        }
    }
    fo<<ans;
    Times;
}
```

Chuyển tới dòng
chứa xâu cần nhập



VW04. THỬ NGHIỆM

Tên chương trình: CHECKOUT.CPP

Tuyến đường sắt trên cao bước vào giai đoạn chạy thử nghiệm. Tàu bắt đầu chạy từ ga **A** và tới dừng ở ga **B**. Khoảng cách giữa 2 ga là **d**. Từ ga **A**, với tốc độ **a1** tàu tăng dần tốc độ để đạt tốc độ hành trình nào đó không vượt quá **v**, sau đó chạy với tốc độ này trong một khoảng thời gian không ít hơn **t**, rồi giảm dần tốc độ với tốc độ **a2** để dừng lại tại ga **B**.

Hãy xác định khoảng thời gian ít nhất cần thiết cho chuyến chạy thử.

Dữ liệu: Vào từ file văn bản CHECKOUT.INP gồm một dòng chứa 5 số thực **d, a1, a2, v** và **t** ($0 < d, a1, a2, v < 10^4$, $0 \leq t < 10^4$, các số có không quá 6 chữ số phần thập phân).

Kết quả: Đưa ra file văn bản CHECKOUT.OUT thời gian nhỏ nhất tìm được với độ chính xác không quá 8 chữ số sau dấu chấm thập phân.

Ví dụ:

CHECKOUT.INP	CHECKOUT.OUT
20.5 0.5 0.6 3 7.5	14.37300711



Giải thuật: Kiến thức vật lý, Phương trình bậc 2.

Với chuyển động nhanh dần đều hoặc chậm dần đều ta có các công thức:

$$\text{Tốc độ} = \text{Gia tốc} \times \text{Thời gian} \quad v = at \rightarrow t = \frac{v}{a},$$

Quảng đường đi được từ khi tốc độ ban đầu bằng 0 cho đến khi đạt tốc độ v là

$$s = \frac{at^2}{2} = \frac{av^2}{2a^2} = \frac{v^2}{2a}$$

Có 2 trường hợp xảy ra:

Tàu có thể đạt tốc độ v và *thời gian chạy với tốc độ không đổi v lớn hơn hoặc bằng t* , như vậy thời gian thử nghiệm ngắn nhất sẽ tương ứng *với tốc độ tối đa là v* :



Tàu cần đạt tốc độ tối đa $u < v$ và *thời gian chạy với tốc độ không đổi u lớn hơn hoặc bằng t* , như vậy thời gian thử nghiệm ngắn nhất sẽ tương ứng *với tốc độ tối đa là u* :



Trong trường hợp này ta có phương trình ẩn số u :

$$\frac{u^2}{2a_1} + tu + \frac{u^2}{2a_2} = d$$

Phương trình bậc 2 này có một nghiệm dương và một nghiệm âm. Giá trị cần tìm tương ứng với nghiệm dương.

Thời gian cần tìm sẽ là tổng thời gian đi trên 3 đoạn.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include<bits/stdc++.h>
#define NAME "checkout."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
double d,a1,a2,v,t,t3,s,res;

int main()
{
    fi>>d>>a1>>a2>>v>>t;
    s=v*v*(1.0/a1+1.0/a2)/2;
    t3=(d-s)/v;
    if(t3>=t)res=t3+v/a1+v/a2;
    else
    {
        double u,a=1/a1+1/a2;
        u=(-t+sqrt(t*t+a*a*d))/a;
        s=u*u*(1.0/a1+1.0/a2)/2;
        t3=(d-s)/u;
        res=t3+u/a1+u/a2;
    }
    fo<<fixed<<setprecision(8)<<res;
}
Times;
```

Lưu ý cách đưa ra kết quả



Các nhà địa chất phát hiện một khu mỏ quặng đất hiếm, một thứ rất cần thiết cho công nghiệp chế tạo thiết bị điện tử. Khu mỏ có hình chữ nhật kích thước $n \times m$ ô. Trữ lượng quặng ở ô (i, j) được đánh giá là $a_{i,j}$, $i = 1 \div n$, $j = 1 \div m$. Cần xây dựng một xí nghiệp làm giàu quặng trước khi đưa ra thị trường. Do điều kiện địa hình, xí nghiệp phải xây dựng ngay trên khu mỏ. Xí nghiệp chiếm diện tích 3×3 ô. Dĩ nhiên không thể khai thác quặng dưới nền của xí nghiệp, vì vậy người ta muốn tìm vị trí đặt xí nghiệp sao cho tổng trữ lượng phải để lại là ít nhất.

Hãy xác định tổng trữ lượng nhỏ nhất phải để lại.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($3 \leq n, m; n \times m \leq 10^6$),
- ✚ Dòng thứ i trong n dòng sau chứa m số nguyên $a_{i1}, a_{i2}, \dots, a_{im}$ ($0 \leq a_{ij} \leq 10^5$, $j = 1 \div m$).

Kết quả: Đưa ra file văn bản ENRICHMENT.OUT một số nguyên – tổng trữ lượng nhỏ nhất phải để lại.

Ví dụ:

ENRICHMENT.INP	ENRICHMENT.OUT
<pre> 5 7 10 2 3 7 10 4 8 3 2 1 9 6 2 1 0 3 6 7 8 9 10 5 4 3 0 2 1 8 9 2 3 10 6 4 8 </pre>	<pre> 27 </pre>



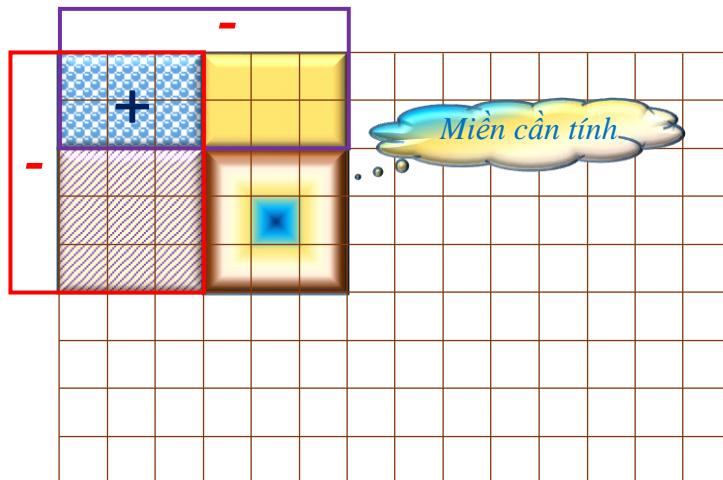
Giải thuật: Tổng tiền tố 2 chiều.

Miền đặt xí nghiệp là một hình vuông kích thước 3×3 phủ lên các ô kề nhau theo hàng và cột trong khu mỏ,

Với 2 mảng hai chiều a và b, mảng b được gọi là mảng tổng tiền tố nếu

$$b_{i,j} = \sum_{p=1}^i \sum_{q=1}^j a_{p,q}$$

Việc tính tổng một miền các phần tử ở các ô kề cạnh liên tục theo hàng và cột chỉ đòi hỏi 3 phép tính số học, không phụ thuộc vào kích thước miền cần tính.



Để tìm lời giải cần duyệt tất cả các vị trí đặt xí nghiệp với góc dưới phải ở vị trí (i, j), $3 \leq i \leq n$, $3 \leq j \leq m$.

Tổ chức dữ liệu:

Mảng `vector<int>` a [$m+1$] – chứa một dòng của ma trận ban đầu,

Mảng 2 chiều `vector<int>` b ($n+1$) – lưu tổng tiền tố.

Xử lý:

Cần tổ chức hàng rào: dòng chứa 0 ở trên và cột chứa 0 ở trái,

Duyệt với mọi i, j thỏa mãn $3 \leq i \leq n$, $3 \leq j \leq m$, tìm min tổng các phần tử ma trận ban đầu trong hình vuông 3×3 .

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

Chương trình

```
#include<bits/stdc++.h>
#define NAME "enrichment."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
typedef vector<int> vi;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n,m,t,ans=10000000000;

int main()
{
    fi>>n>>m;
    vector<int> a[m+1];
    vector<int> b(n+1); b[0]=0;
    for(int j=0;j<=m;++j)a[0].push_back(0);
    for(int i=1;i<=n;++i)a[i].push_back(0);
    for(int i=1; i<=n;++i)
    {
        for(int j=1;j<=m;++j){fi>>t; b[j]=b[j-1]+t;}
        for(int j=1;j<=m;++j) a[i].push_back(a[i-1][j]+b[j]);
    }

    for(int i=3;i<=n;++i)
        for(int j=3;j<=m;++j)
        {
            t=a[i][j]-a[i-3][j]-a[i][j-3]+a[i-3][j-3];
            if(ans>t) ans=t;
        }
    fo<<ans;
    Times;
}
```



VW06. TÊN ĐĂNG NHẬP

Tên chương trình: ACCOUNT.CPP

Trong vụ án triệt phá đường dây buôn lậu chất gây nghiện người ta thu giữ được máy tính của tên cầm đầu đường dây. Sau một thời gian khảo sát máy tính đã bị bẻ khóa và người ta tìm được file lưu giữ tên đăng nhập và mật khẩu vào trang thông tin của các thành viên trong đường dây. File này chứa $2 \times n$ xâu ký tự, mỗi xâu chỉ chứa các ký tự la tinh thường.

Khai báo của các tên đã bị bắt cho biết mật khẩu là tên đăng nhập được viết thêm vào cuối một số (có thể là 0) các ký tự khác. Mỗi xâu có thể sử dụng đúng một lần làm tên đăng nhập hoặc mật khẩu.

Hãy xác định các cặp xâu có thể là tên đăng nhập và mật khẩu tương ứng.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Mỗi dòng trong $2 \times n$ dòng sau chứa một xâu ký tự. Tổng độ dài của các xâu không vượt quá 5×10^5 .

Dữ liệu đảm bảo bài toán có nghiệm. Các xâu được đánh số từ 1.

Kết quả: Đưa ra file văn bản ACCOUNT.OUT các cặp chỉ số xác định Tên đăng nhập và Mật khẩu. Nếu các nhiều cách xác định thì đưa ra cách tùy chọn.

Ví dụ:

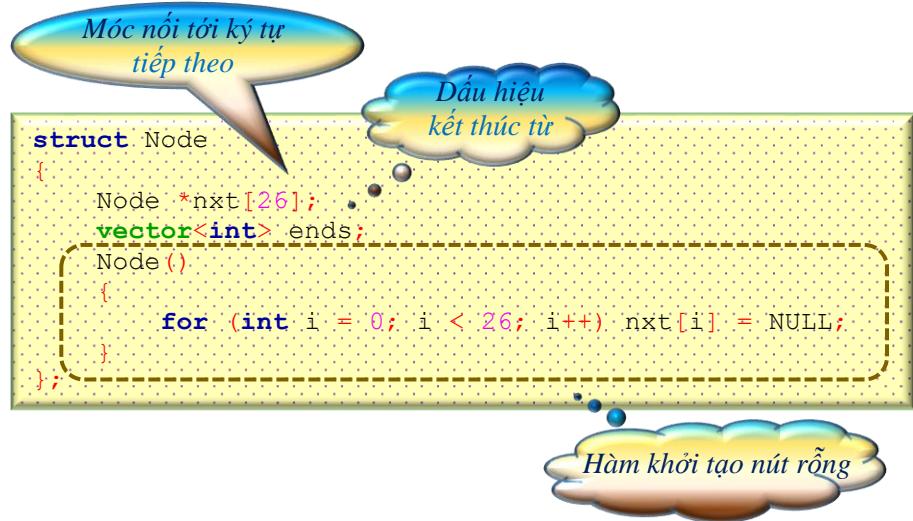
ACCOUNT.INP	ACCOUNT.OUT
2	3 4
abac	1 2
abacab	
aba	
abaa	



Giải thuật: Rừng cây tìm kiếm.

Xây dựng rừng cây lưu các xâu,

Cấu trúc mỗi nút của rừng:



Nạp tất cả các xâu từ input vào rừng.

Tiến hành duyệt rừng theo chiều sâu.

Trong quá trình duyệt: nạp vào stack **st** số thứ tự của xâu khi gặp dấu hiệu kết thúc.

Trên đường đi xuất từ cùng một đỉnh chung chỉ tới, xâu kết thúc trước sẽ là tiền tố của xâu kết thúc sau.

Khi đi hết một nhánh:

- + Đánh dấu các xâu đã gặp kết thúc và chưa được chọn để ghép cặp.
- + Gom từng cặp xâu ghi nhận trong **st** và chưa được sử dụng vào kết quả ghép cặp.
- + Lưu ý: có thể tồn tại trong **st** xâu đã gặp kết thúc nhưng chưa được ghép cặp, xâu này sẽ được ghép cặp với xâu sẽ gặp khi duyệt sang nhánh khác (từ điều kiện *bài toán có nghiêm*),
- + Xóa đánh dấu xâu để phục vụ duyệt nhánh khác.

Tổ chức dữ liệu:

- [R] Rừng cây **Node *root = new Node()** – lưu dữ liệu vào,
- [R] Mảng **vector<int> st** – lưu số thứ tự các xâu đã gặp kết thúc, đóng vai trò stack để ghép cặp,
- [R] Mảng **bool isChild[N]** – đánh dấu các xâu tham gia ghép cặp,
- [R] Mảng **vector<pair<int, int> > ans** – lưu kết quả.

Xử lý:

Ghi nhận dữ liệu vào rừng cây:

```

int n;
fi >> n;
Node *root = new Node();
for (int i = 1; i <= 2 * n; i++)
{
    string str;
    fi >> str;

    Node *node = root;
    for (int j = 0; j < (int)str.size(); j++)
    {
        char letter = str[j] - 'a';

        if (node->nxt[letter] == NULL)
            node->nxt[letter] = new Node();

        node = node->nxt[letter];
        if (j == str.size() - 1) node->ends.push_back(i);
    }
}

```

Xuất phát từ

Tạo đường đi
cho nhánh mới

Ghi nhận kết
thúc xâu

Ghép cặp:

Nạp danh sách các xâu
là kéo dài của xâu trước

```

void dfs(Node *vert)
{
    for (int i = 0; i < (int)(vert->ends.size()); i++)
        st.push_back(vert->ends[i]);
    for (int i = 0; i < 26; i++)
    {
        Node *child = vert->nxt[i];
        if (child != NULL) dfs(child);
    }

    for (int i = 0; i < (int)(vert->ends.size()); i++)
        isChild[vert->ends[i]] = true;
}

while (st.size() > 1 && isChild[st.back()])
{
    int num1 = st.back();
    st.pop_back();
    int num2 = st.back();
    st.pop_back();
    ans.push_back(make_pair(num2, num1));
}

if (st.size() == 1 && isChild[st.back()])
    st.pop_back();

for (int i = 0; i < (int)(vert->ends.size()); i++)
    isChild[vert->ends[i]] = false;
}

```

Đánh dấu các
xâu là mật khẩu

Phục vụ xử lý nhánh
tiếp theo

Độ phức tạp của giải thuật: O(L), L – tổng độ dài các xâu.

Chương trình

```
#include <bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define NAME "account."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

const int N = 300005;
vector<int> st; //stack
bool isChild[N];
vector<pair<int, int> > ans;

struct Node
{
    Node *nxt[26];
    vector<int> ends;
    Node ()
    {
        for (int i = 0; i < 26; i++) nxt[i] = NULL;
    }
};

void dfs(Node *vert)
{
    for (int i = 0; i < (int)(vert->ends.size()); i++)
        st.push_back(vert->ends[i]);
    for (int i = 0; i < 26; i++)
    {
        Node *child = vert->nxt[i];
        if (child != NULL) dfs(child);
    }

    for (int i = 0; i < (int)(vert->ends.size()); i++)
        isChild[vert->ends[i]] = true;
    while (st.size() > 1 && isChild[st.back()])
    {
        int num1 = st.back();
        st.pop_back();
        int num2 = st.back();
        st.pop_back();
        ans.push_back(make_pair(num2, num1));
    }
    if (st.size() == 1 && isChild[st.back()]) st.pop_back();

    for (int i = 0; i < (int)(vert->ends.size()); i++)
        isChild[vert->ends[i]] = false;
}

int main()
{
    int n;
    fi >> n;
    Node *root = new Node();
    for (int i = 1; i <= 2 * n; i++)
    {
        string str;
        fi >> str;
        Node *node = root;
        for (int j = 0; j < (int)str.size(); j++)
        {
            if (str[j] >='A' && str[j] <='Z') node = node->nxt[str[j] - 'A'];
            else if (str[j] >='a' && str[j] <='z') node = node->nxt[str[j] - 'a' + 26];
        }
        node->ends.push_back(i);
    }
}
```

```

        char letter = str[j] - 'a';
        if (node->nxt[letter] == NULL) node->nxt[letter] = new Node();
        node = node->nxt[letter];
        if (j == str.size() - 1) node->ends.push_back(i);
    }
}

dfs(root);

for (int i = 0; i < (int)ans.size(); i++)
    fo << ans[i].first << " " << ans[i].second << endl;
Times;
}

```



Bác sỹ Watson cần tạo một viên thuốc chứa x đơn vị vitamin cho bệnh nhân của mình. Là một bác sỹ có kinh nghiệm, Watson hiểu rằng thiếu vitamin thì sẽ không có tác dụng, nhưng thừa vitamin thì còn nguy hiểm hơn.

Thiết bị chiết xuất vitamin của bác sỹ chưa thật hoàn thiện. Trong n giờ hoạt động liên tục, giờ thứ i máy cho a_i đơn vị vitamin cần thiết, $i = 1 \div n$, bắt đầu từ đó trở đi, mỗi giờ tiếp theo máy cho ổn định a_n đơn vị, nhưng nếu dừng máy một giờ hay nhiều hơn, khi khởi động lại, máy sẽ lại lần lượt cho a_1, a_2, a_3, \dots đơn vị vitamin ở giờ thứ nhất, thứ 2, thứ 3, ...

Hãy xác định khoảng thời gian ít nhất cần thiết để có đúng x đơn vị vitamin hoặc cho biết không thể làm được điều đó.

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

- ⊕ Dòng đầu tiên chứa 2 số nguyên n và x ($1 \leq n \leq 100$, $1 \leq x \leq 10^6$),
- ⊕ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 1000$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản VITAMIN.OUT một số nguyên – thời gian ít nhất cần thiết xác định được hoặc số -1 nếu không thể đạt đúng x đơn vị.

Ví dụ:

VITAMIN.INP
5 12
1 4 2 6 3

VITAMIN.OUT
6



Giải thuật: Quy hoạch động.

Để tiện xử lý, đánh số các a_i bắt đầu từ 0.

Xét riêng trường hợp $n = 1$, cho máy hoạt động liên tục và ta có dãy số a_0, a_0, a_0, \dots . Bài toán có nghiệm khi và chỉ khi x chia hết cho a_0 .

Trường hợp tổng quát: bài toán có mô hình toán học quy hoạch động chuẩn, tương tự bài toán đồi tiền.

Gọi $dp_{j,i}$ là số giờ ít nhất để giảm lượng vitamin còn thiếu xuống j khi có số làm việc liên tục cuối cùng là i .

Ban đầu $dp_{x,0} = 0$.

Dễ dàng nhận thấy rằng, từ $dp_{j,i}$ ta có thể chuyển sang trạng thái một trong 2 trạng thái $dp_{j-a[i], \min(n-1, i+1)}$ hoặc trạng thái $dp_{j-a[0], 1}$ với thời gian nghỉ là 1 giờ.

Do $1 \leq a_i \leq 1\,000$ nên các giá trị dòng mới được dẫn xuất dựa trên không quá 1 000 dòng cũ vì vậy chỉ cần lưu trữ 1001 dòng cuối cùng trong quá trình tính toán và điều chỉnh chỉ số thích hợp.

Kết quả là giá trị nhỏ nhất của $dp_{0,i}$, $i = 0 \div n-1$.

Tổ chức dữ liệu:

- Mảng `vector<int>` $a(n)$ – lưu các giá trị a_i ,
- Mảng `int` $dp[\maxvalue][\maxn]$ – phục vụ sơ đồ quy hoạch động.

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

Chương trình

```
#include <bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define NAME "vitamin."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

const int maxn = 100;
const int maxvalue = 1024;
int dp[maxvalue][maxn];

int main()
{
    int n, x;
    fi>>n>>x;
    vector<int> a(n);
    for (int i = 0; i < n; i++) fi>>a[i];
    if (n == 1)
    {
        fo<<(x % a[0] == 0 ? x / a[0] : -1);
        return 0;
    }
    for (int j = 0; j < maxvalue; j++)
        for (int i = 0; i < maxn; i++) dp[j][i] = INT_MAX;

    dp[x % maxvalue][0] = 0;
    for (int mj = x; mj > 0; mj--)
        for (int i = 0; i < n; i++)
    {
        int j = mj % maxvalue;
        if (dp[j][i] == INT_MAX) continue;
        if (mj >= a[i])
        {
            int ni = min(n - 1, i + 1);
            int nj = (mj - a[i]) % maxvalue;
            dp[nj][ni] = min(dp[nj][ni], dp[j][i] + 1);
        }
        if (mj >= a[0])
        {
            int nj = (mj - a[0]) % maxvalue;
            dp[nj][1] = min(dp[nj][1], dp[j][i] + 2);
        }
        dp[j][i] = INT_MAX;
    }

    int answer = INT_MAX;
    for (int i = 0; i < n; i++)
        answer = min(answer, dp[0][i]);

    fo<<((answer == INT_MAX) ? -1 : answer);
    Times;
}
```



Steve có một danh mục gồm n các bài hát yêu thích và thích nghe những bài này trong khi làm việc với máy tính. Những người lập trình chuyên nghiệp luôn chú ý đảm bảo mọi thứ phải có trình tự chặt chẽ và Steve không phải là một ngoại lệ, các bài hát có thể nghe bắt đầu từ một bài nào đó trong danh sách, nhưng phải theo đúng trình tự liệt kê.

Hôm nay gặp một vấn đề rất khó, Steve ngồi vào bàn, vừa suy nghĩ về nhiệm vụ cần giải quyết vừa chọn ngẫu nhiên một bài hát trong danh sách và kích hoạt player ở chế độ chọn bài liên tục. Chỉ khi bài đầu tiên được chọn vang lên Steve mới sực tỉnh và ngăn người ra – không hiểu vừa rồi mình đã chọn chế độ nào, chọn tiếp liên tục theo danh sách liệt kê hay chọn ngẫu nhiên? Nếu lỡ chọn chế độ trình tự ngẫu nhiên thì phải xác lập lại chế độ phát.

Cho biết trình tự các bài hát chương trình sẽ phát. Hãy xác định Steve có phải xác định chế độ phát hay không và nếu có thì đến khi nghe đến bài hát thứ mấy biết được điều đó.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($3 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n – trình tự các bài hát chương trình sẽ phát, $1 \leq a_i \leq n$, $i = 1 \div n$.

Kết quả: Đưa ra file văn bản PLAYER.OUT đưa ra thông báo **NO** nếu không phải chọn lại chế độ phát và **YES** trong trường hợp ngược lại. Nếu cần chọn lại chế độ thì ở dòng thứ 2 chứa số nguyên xác định bài hát làm cho Steve biết phải chọn lại.

Ví dụ:

PLAYER.INP
3
1 3 2

PLAYER.OUT
YES
2



Giải thuật: Kiểm tra đơn giản.

Nếu các bài hát được phát theo trình tự trong danh sách thì dãy số a được chia thành 2 phần: một phần tăng liên tục cách nhau 1 từ a_1 đến n , phần tiếp theo – tăng liên tục cách nhau 1 từ 1 đến $a_1 - 1$.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define NAME "player."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,k,a;
string ans="NO";

int main()
{
    fi>>n>>a; k=a;
    for(int i=2;i<=n;++i)
    {
        ++k; if(k>n) k=1;
        fi>>a; if(a!=k) {ans="YES\n"; k=i; break;}
    }

    fo<<ans;
    if(ans!="NO") fo<<k;
    Times;
}
```



Steve có thú vui tự mình xây dựng các chương trình phục vụ. Một trong các sản phẩm yêu thích nhất của Steve là chương trình tải đồng thời nhiều files từ Internet. Khi lướt WEB, Steve đánh dấu các files mình quan tâm, chương trình tải file sẽ nạp đường link tới các files đó vào danh sách files cần tải về. Trình tự các files trong danh sách có thể thay đổi trước khi thực hiện quá trình tải. Thông thường Steve đưa lên đầu danh sách file mà mình quan tâm nhiều nhất.

Khi kích hoạt quá trình tải, nội dung các files trong danh sách được tải về theo chế độ phục vụ RR (*Round Robin*). Gọi n là số lượng files trong danh sách. Chương trình sẽ làm việc với file thứ nhất trong thời gian 1 giây, sau đó chuyển sang xử lý file thứ 2 với thời gian 1 giây, rồi chuyển sang file thứ 3, . . . cứ như thế đến file thứ n rồi quay lại file thứ nhất. Trong khoảng thời gian 1 giây Đường truyền Steve đang sử dụng cho phép tải về b MB. Nếu dung lượng còn lại của file nhỏ hơn hoặc bằng b MB, sau khi tải phần còn lại, file này sẽ bị loại khỏi danh sách xếp hàng chờ tải về.

Hôm nay Steve cần tải về n files, dung lượng file thứ i là a_i MB, $i = 1 \div n$, trong đó file thứ nhất nói về một thuật toán mới rất hấp dẫn. Steve rất sốt ruột chờ file được tải về đầy đủ để mở file xem nội dung.

Hãy xác định thời gian chờ đợi kể từ khi bắt đầu kích hoạt quá trình tải.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và b ($1 \leq n \leq 10^5$, $1 \leq b \leq 10^9$),
- ✚ 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 DOWNLOAD.OUT một số nguyên – thời gian chờ đợi.

Ví dụ:

DOWNLOAD.INP
4 2
6 2 5 1

DOWNLOAD.OUT
7



VW09 Io20170930 D_AXII

Giải thuật: Xử lý dữ liệu vòng tròn.

Tính thời gian cần thiết để tải về với mỗi file: nếu file có kích thước a_i thì thời gian tải về là $(a_i+b-1)/b$ và đây cũng là số lần duyệt vòng tròn đối với file đang xét.

Với file thứ nhất, số lần duyệt vòng tròn là $t = (a_1+b-1)/b$,

File thứ i tham gia đủ t lần nếu $(a_i+b-1)/b \geq t$, trong trường hợp ngược lại – chỉ tham gia $(a_i+b-1)/b$ lần.

Tích lũy số lần tham gia của mỗi file, ta được kết quả cần tìm.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include<bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
#define NAME "download."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;
int64_t ans,a,b,t;

int main()
{
    fi>>n>>b>>ans;
    ans=(ans+b-1)/b; t=ans-1;
    for(int i=1; i<n; ++i)
    {
        fi>>a;
        //ans+=( (a+b-1)/b < t ? (a+b-1)/b : t );
        ans+=min( (a+b-1)/b, t );
    }
    fo<<ans;
    Times;
}
```



VW10. HỢP TÁC

Tên chương trình: PAIRING.CPP

Hai đoàn nghệ thuật Rạng Đông và Bình Minh phối hợp tổ chức một đêm biểu diễn nghệ thuật. Để chuẩn bị các thành viên trong hai đoàn phải hợp tác thiết kế trang trí sân khấu, chỉnh đao cụ, dự thảo chương trình, v.v. . . Mỗi công việc do 2 người đảm nhiệm. Đoàn Rạng Đông có **a** người chỉ muốn được làm việc với người cùng đoàn và **b** người – muốn được hợp tác với người của Bình Minh. Còn ở Bình Minh có **c** người chỉ muốn được làm việc với người cùng đoàn và **d** người – muốn được hợp tác với người của đoàn kia. Nếu ai đó được ghép cặp không đúng nguyện vọng thì sẽ không vui vẻ và vì vậy năng lực sáng tạo và độ linh hoạt cũng bị giảm sút.

Ban Tổ chức cố gắng đáp ứng tối đa nguyện vọng các cá nhân, tuy vậy vẫn có trường hợp không thỏa mãn được ý thích của tất cả mọi người.

Hãy xác định số lượng người ít nhất không được đáp ứng yêu cầu cá nhân.

Dữ liệu: Vào từ file văn bản PAIRING.INP gồm một dòng chứa 4 số nguyên **a**, **b**, **c** và **d** ($1 \leq a, b, c, d \leq 100$). Dữ liệu đảm bảo **a + b + c + d** là một số chẵn.

Kết quả: Đưa ra file văn bản PAIRING.OUT một số nguyên – số lượng người ít nhất không được đáp ứng yêu cầu cá nhân.

Ví dụ:

PAIRING.INP
2 1 2 1

PAIRING.OUT
0



VW10 Io20170930 E_AXII

Giải thuật: Phân tích tình huống lô gic.

Đầu tiên đáp ứng nguyện vọng những người chỉ muốn làm việc với người cùng đoàn, khi đó số lượng người không thỏa mãn nguyện vọng cá nhân sẽ là $a \% 2 + c \% 2$.

Nếu $b = d$ – ghép những người này theo nguyện vọng.

Xét trường hợp $b > d$:

- ⊕ Sẽ có d cặp hài lòng với cách ghép nhóm,
- ⊕ Còn lại $b-d$ người ở đoàn Rạng Đông,
- ⊕ Tổng số người chưa được ghép cặp và có khả năng sẽ không hài lòng sẽ là

$$a \% 2 + c \% 2 + b - d,$$

- ⊕ Nếu c lẻ thì ghép người đó với một trong số $b-d$ người còn lại trong đoàn Rạng Đông, có cặp trong đó chỉ một người không hài lòng, tổng số người có khả năng sẽ không hài lòng giảm một,
- ⊕ Số người còn lại chưa được ghép cặp là $rm = b - d - c \% 2$,
- ⊕ Nếu a là số lẻ và $rm > 0$ thì ghép một người còn dư với một trong số rm người cùng đoàn muốn là việc với đoàn khác ta được một cặp trong đó chỉ có một người không vui, số người không hài lòng sẽ giảm một và đó là kết quả tốt nhất có thể!

Trường hợp $b < d$: xử lý tương tự.

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int main()
{
    int a, b, c, d;
    fi >> a >> b >> c >> d;
    int ans = a % 2 + c % 2;
    if (b > d)
    {
        ans += b - d;
        if (c % 2)
            ans--;
        if (a % 2 && b - d - c % 2 > 0)
            ans--;
    }
    else if (b < d)
    {
        ans += d - b;
        if (a % 2)
            ans--;
        if (c % 2 && d - b - a % 2 > 0)
            ans--;
    }
    fo << ans;
    return 0;
}
```



VW11. PHỦ SÓNG WIFI

Tên chương trình: WIFI.CPP

Khu Công nghiệp có hình chữ nhật kích thước $n \times m$ ô (n hàng, m cột). Một Hàng sản xuất thiết bị điện tử thuê một số ô trong khu Công nghiệp. Các ô này được đánh dấu ‘*’, các ô còn lại – đánh dấu ‘.’. Để vận chuyển linh kiện từ nơi này sang nơi khác Hàng dùng thiết bị bay không người lái (Drone) điều khiển bằng wifi. Drone có chỉ có thể bay theo tuyến gồm các ô kề cạnh có sóng điều khiển. Các anten phát sóng được định hướng phủ lên một diện tích nhỏ nhất, trong đó bao gồm các ô được thuê và nếu 2 ô bất kỳ cùng hàng hoặc cùng cột có sóng thì những ô ở giữa 2 ô đó cùng hàng (cột) cũng có sóng và đảm bảo drone có thể bay từ ô bất kỳ được thuê tới ô bất kỳ khác được thuê.

Hãy đánh dấu ‘*’ các ô được phủ sóng.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n, m \leq 500$),
- ✚ Mỗi dòng trong n dòng sau chứa xâu độ dài m chỉ chứa các ký tự thuộc tập {*, .}.

Kết quả: Đưa ra file văn bản WIFI.OUT n dòng, mỗi dòng chứa xâu độ dài m chứa các ký tự thuộc tập {*, .} xác định vùng phủ sóng. Trong trường hợp có nhiều lời giải – đưa ra phương án tùy chọn.

Ví dụ:

WIFI.INP	WIFI.OUT
5 5 ...** ...** .**.. .**..**** .****. .**..



VW11 Io20170930 H_AXII

Giải thuật: Tập ô vuông lồi.

Tập ô vuông được gọi là lồi nếu thỏa mãn các điều kiện:

- ⊕ Nếu 2 ô **A** và **B** cùng hàng (hoặc cùng cột) thuộc tập thì các ô cùng hàng (hoặc cùng cột) giữa **A** và **B** cũng thuộc tập,
- ⊕ Nếu **U** và **V** là 2 ô thuộc tập thì bao giờ cũng tồn tại đường đi từ các ô kề cạnh thuộc tập nối **U** và **V**.

Yêu cầu của bài toán: xác định tập lồi diện tích nhỏ nhất chứa các ô cho trước.

Quá trình xây dựng tập bao gồm 3 bước:

B1. Xác định tập đảm bảo giữa 2 ô thuộc tập trên cùng một dòng có đường đi: Với dòng **i**, tìm **lf_i** – cột nhỏ nhất chứa ‘*’, **rt_i** – cột lớn nhất chứa ‘*’, đồng thời xác định **indln** – dòng có **lf_{indln}** = min{**lf_i**}, **indr_x** – dòng có **rt_{indr_x}** = max{**rt_i**},

B2. Xác định tập đảm bảo giữa 2 ô thuộc tập trên cùng một cột có đường đi:

Cập nhật **lf_i**, **lf_i** không tăng từ dòng 0 đến dòng **indln** và không giảm từ dòng **indln** đến dòng **n-1**,

Cập nhật **rt_i**, **rt_i** không giảm từ dòng 0 đến dòng **indr_x** và không tăng từ dòng **indr_x** đến **n-1**,

B3. Liên kết các tập rời nhau, đảm bảo giữa 2 ô bất kỳ thuộc tập có đường đi. Dấu hiệu tồn tại các tập không liên thông: $\exists i | lf_i > rt_{i-1}$ hoặc $rt_i < lf_{i-1}$.

Công thức cập nhật:

lf[i] = max(rt[i-1], 0) – trường hợp 1,

rt[i] = min(lf[i-1], m-1) – trường hợp 2.

Dẫn xuất kết quả: Mỗi dòng có 3 phần: phần đầu và cuối – chứa ‘.’, phần giữa (từ **lf_i** đến **rt_i**) – chứa dấu ‘*’.

Tổ chức dữ liệu:

Các mảng **vector<int>lf(n), rt(n)** chứa biên trái và phải của các dấu ‘*’ trên mỗi dòng.

Độ phức tạp của giải thuật: O(n×m).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "wifi."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
int n,m,l,r,indln,indrx;
string s;

int main()
{
    fi>>n>>m;
    vector<int> lf(n), rt(n);
    int lmin=m, rmax=-1;
    for(int i=0; i<n; ++i)
    {
        fi>>s;
        l=s.find('*'); if(l>=0)r=s.rfind('*'); else l=m, r=-1;
        lf[i]=l; rt[i]=r;
        if(l<lmin) lmin=l, indln=i;
        if(r>rmax) rmax=r, indrx=i;
    }

    for(int i=1;i<indln;++i)
        if(lf[i]>lf[i-1]) lf[i]=lf[i-1];
    for(int i=n-2;i>indln;--i)
        if(lf[i]>lf[i+1]) lf[i]=lf[i+1];
    for(int i=1;i<indrx;++i)
        if(rt[i]<rt[i-1]) rt[i]=max(rt[i-1],lf[i]);
    for(int i=n-2;i>indrx;--i)
        if(rt[i]<rt[i+1]) rt[i]=max(rt[i+1],lf[i]);

    for(int i=1;i<n;++i)
        if(lf[i]<m && rt[i-1]!=-1 && lf[i]>rt[i-1]) lf[i]=max(rt[i-1],0);
        else if(rt[i]!=-1 && lf[i-1]!=m && rt[i]<lf[i-1])
            rt[i]=min(lf[i-1],m-1);

    for(int i=0;i<n;++i)
    {
        if(lf[i]==m) fo<<'.';
        else
        {
            for(int j=0;j<lf[i];++j) fo<<'.';
            for(int j=lf[i];j<=rt[i];++j) fo<<'*';
            for(int j=rt[i]+1;j<m;++j) fo<<'.';
        }
        fo<<'\n';
    }

    Times;
}
```

The diagram illustrates the structure of the C++ program. It features three sections of code, each enclosed in a dashed box and associated with a speech bubble label:

- B1**: This section contains the initial input reading loop where the string `s` is read and processed to find the first and last positions of the asterisk character (*). It also initializes variables `lf` and `rt` and updates `lmin` and `rmax`.
- B2**: This section contains the main update loops for the `lf` and `rt` arrays. It iterates through the array from index 1 to `indln` and from `n-2` down to `indln`. It also iterates through the array from index 1 to `indrx` and from `n-2` down to `indrx`, updating the arrays based on the previous values.
- B3**: This section contains the final output generation loop. It iterates through the array from index 1 to `n-1`. For each element, it prints the appropriate number of dots (if `lf[i] == m`) or asterisks (otherwise). It then prints a new line character (`\n`).



VW12. BÓC BÀI

Tên chương trình: G21.CPP

Trò chơi Blackjak sử dụng bộ bài 52 quân. Các quân bài đánh số 2, 3, ..., 9, 10, J, Q, K và A. Với mỗi số đánh dấu có 4 quân bài (Bích, Cơ, Rô, Chuồn). Các quân J, Q, K có giá trị 10 điểm, quân A có giá trị 11 điểm, các quân còn lại có số điểm bằng số ghi trên quân bài.

Người chơi cần giữ trên tay các quân bài với tổng số điểm không vượt quá 21. Đến lượt mình đi người chơi có thể bốc thêm quân bài mới (**DRAW**) hoặc không bốc (**STOP**).

Steve đang giữ n quân bài, quân thứ i có p_i điểm ($i = 1 \dots n$). Nếu khả năng bốc thêm một quân bài mà tổng số điểm trên tay vẫn không vượt quá 21 lớn hơn hoặc bằng khả năng tổng số điểm vượt quá 21 thì Steve sẽ bốc bài, trong trường hợp ngược lại – không bốc.



Hãy xác định ở nước đi này Steve nên bốc bài hay bỏ qua và đưa ra thông báo tương ứng (**DRAW** hoặc **STOP**).

Dữ liệu: Vào từ file văn bản G21.INP:

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 52$),
- ✚ Dòng thứ i trong n dòng sau chứa số nguyên p_i ($2 \leq p_i \leq 11$).

Dữ liệu đảm bảo hợp lệ.

Kết quả: Đưa ra file văn bản G21.OUT thông báo xác định được.

Ví dụ:

G21.INP
2
6
5

G21.OUT
DRAW



Giải thuật: Thống kê tàn số.

Gọi c_i là số lượng quân bài có i điểm.

Với cỗ bài ban đầu có: $c_0 = c_1 = 0$, $c_{10} = 16$, $c_2 = c_3 = \dots = c_9 = c_{11} = 4$.

Từ dữ liệu vào có thể tính được số lượng còn lại trong cỗ bài với mỗi loại quân bài điểm i ($i = 2 \div 11$).

Gọi **sum** – tổng số điểm các quân bài trên tay,

Tính a – số lượng quân bài còn lại có điểm nhỏ hơn hoặc bằng $t = 21 - sum$ và b – số lượng quân bài còn lại có điểm lớn hơn t .

Nếu $a \geq b$ – bốc tiếp bài, trong trường hợp ngược lại – dừng.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "g21."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,c[12]={0,0,4,4,4,4,4,4,4,20,4},r,sum=0;

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i){fi>>r; sum+=r;--c[r];}
    if(r>=20){fo<<"STOP"; return 0;}
    r=21-sum;
    int a=0,b=0;
    for(int i=2; i<=r; ++i)a+=c[i];
    for(int i=r+1;i<12; ++i)b+=c[i];
    if(a>=b) fo<<"DRAW"; else fo<<"STOP";
}
```



VW13. ĐÊ CHÂN SÓNG

Tên chương trình: DIKE.CPP

Để bảo vệ vùng bờ biển có giá trị kinh tế cao ở một vịnh người ta xây dựng một con đê chắn sóng chắn cửa vịnh. Đoạn thẳng nối 2 bên của cửa vịnh được chia thành n ô cùng độ dài.



Đê gồm nhiều đoạn liên tục được thả “Nhím biển”, mỗi đoạn bao gồm một số ô liên tục. Giữa 2 đoạn phải có ít nhất một ô trống.

Theo thiết kế, đê phải có một đoạn độ dài k ô, 2 đoạn độ dài $k-1$ ô, 3 đoạn độ dài $k-2$ ô, . . . , k đoạn độ dài 1 ô.

Hãy xác định k lớn nhất có thể chọn.

Dữ liệu: Vào từ file văn bản DIKE.INP gồm một dòng chứa số nguyên n ($0 \leq n \leq 10^{18}$).

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

Ví dụ:

DIKE.INP	DIKE.OUT
7	2



VNU HCMC

Giải thuật: Phân tích và biến đổi mô hình toán học, Tìm kiếm nhị phân.

Xét số lượng ô cần thiết để có đoạn đê chắn sóng dài nhất độ dài k.

Số lượng các đoạn chắn sóng là $1 + 2 + 3 + \dots + k = \sum_{i=1}^k i = \frac{k*(k+1)}{2}$,

Số lượng ô cần thiết để trống sẽ là không ít hơn $\frac{k*(k+1)}{2} - 1$.

Số lượng ô thuộc các đoạn đê sẽ là

$$\begin{aligned} & k + 2 \times (k-1) + 3 \times (k-2) + \dots + (k-1) \times 2 + k \times 1 = \\ & = k * (1 + 2 + \dots + k) - (1 \times 2 + 2 \times 3 + 3 \times 4 + \dots + (k-1) \times k) \\ & = k * (1 + 2 + \dots + k) - (1 \times (1+1) + 2 \times (2+1) + 3 \times (3+1) + \dots + (k-1) \times (k-1+1)) \\ & = \frac{k \times k \times (k+1)}{2} - \sum_{i=1}^{k-1} i^2 - (1 + 2 + \dots + (k-1)) \\ & = \frac{k \times k \times (k+1)}{2} - \frac{k \times (k-1) \times (2 \times k - 1)}{6} - \frac{k \times (k-1)}{2} \\ & = \frac{k \times k \times (k+1)}{2} - \frac{k \times (k-1) \times (k+1)}{3} \\ & = \frac{k \times (k+1) \times (k+2)}{6} \end{aligned}$$

Như vậy tổng số lượng ô cần có sẽ là

$$\begin{aligned} f(k) &= \frac{k \times (k+1) \times (k+2)}{6} + \frac{k * (k+1)}{2} - 1. \\ &= \frac{k \times (k+1) \times (k+5)}{6} - 1 \end{aligned}$$

$f(k)$ là hàm đơn điệu tăng trong miền $k \geq 0$.

Giá trị k có thể tìm bằng giải thuật tìm kiếm nhị phân.

Cần lưu ý khi xác định miền tìm kiếm để tránh tràn ô khi tính $f(k)$: với $n \leq 10^{18}$, k không thể lớn hơn 2×10^6 .

Độ phức tạp của giải thuật: O(ln n).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "dike."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n,l,r,m,sum,mx=2000000;

int main()
{
    fi>>n;
    l=0; r=min(n+1, mx);
    while(l<r-1)
    {
        m=(r+l)/2;
        sum=m*(m+1)*(m+5)/6-1;
        if(sum>n) r=m; else l=m;
    }
    fo<<l;
    Times;
}
```



Ở lứa tuổi mẫu giáo phương pháp dạy và học tốt nhất là thông qua các trò chơi. Để dạy các cháu biết phân biệt số lớn hơn, nhỏ hơn, số chẵn và số lẻ cô giáo đã làm một cỗ bài n quân. Trên quân bài thứ i có viết một số nguyên a_i , $i = 1 \div n$. Quân bài có số chẵn gọi là quân chẵn, trong trường hợp ngược lại – quân lẻ.

Cô giáo sắp các bài ra sàn và yêu cầu các cháu chọn các quân bài sắp thành một dãy, trong đó các quân chẵn – lẻ đứng đan xen nhau, tức là quân chẵn, quân lẻ, quân chẵn, . . . hoặc quân lẻ, quân chẵn, quân lẻ, . . . ngoài ra số trên các quân bài phải tăng dần từ trái sang phải. Ai tạo được dãy nhiều quân bài nhất sẽ thắng. Từng tổ một được gọi lên để xây dựng dãy theo yêu cầu. Có một tổ đã tìm được dãy dài nhất và chiến thắng.

Hãy xác định số lượng quân bài tổ chiến thắng đã chọn được.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 100$),
- ✚ 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 SOLITAIRE.OUT một số nguyên – độ dài lớn nhất (số quân bài) của dãy có thể chọn được.

Ví dụ:

SOLITAIRE.INP	SOLITAIRE.OUT
6	4
3 2 8 1 4 3	



VW14_Io20171021_B2_AXII

Giải thuật; Các kỹ năng cơ sở.

Sắp xếp dữ liệu theo thứ tự tăng dần,

Thay đổi các số liên tiếp nhau cùng loại (cùng chẵn hoặc cùng lẻ) bằng một số đại diện,

Số lượng số đại diện là độ dài cần tìm.

Lưu ý: chỉ cần đếm số đại diện, *không phải thực tế loại bỏ số* khỏi dãy.

Độ phức tạp của giải thuật: O(nlnn).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "solitaire."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;

int main()
{
    fi>>n;
    vector<int>a(n);
    for(int i=0; i<n; ++i) fi>>a[i];
    sort(a.begin(), a.end());
    int res=1, p=a[0]&1;
    for(int i=1; i<n; ++i)
        if((a[i]&1)!=p) ++res, p^=1;
    fo<<res;
    Times;
}
```



VW15. TINH DẦU OAI HƯƠNG

Tên chương trình: LAVENDER.CPP

Ở máy chưng cất tinh dầu oai hương, tinh dầu được tụ lại trong bình chứa. Cứ m giây thì áp suất trong bình đủ lớn, đẩy tinh dầu sang bộ phận đóng chai và áp suất lại sút xuống ở mức bình thường – 1A (Atmosphere).

Công nhân vận hành máy chưng cất phát hiện bình chứa có sự cố. Có n chỗ bị rò rỉ, trong quá trình tích áp chỗ thứ i cứ p_i giây nhả một giọt tinh dầu. Chu kỳ này cứ lặp đi lặp lại với mỗi khoảng thời gian tích áp. Nếu đúng thời điểm đủ áp lực đẩy tinh dầu đi, giọt tinh dầu đã hình thành thì nó vẫn rơi xuống trước khi tinh dầu trong bình bị đẩy đi.



Là một người có tinh thần trách nhiệm cao, anh lấy ống nghiệm gắn vào các chỗ dò rỉ để thu các giọt rỉ ra, đồng thời báo cho kỹ sư bảo dưỡng tới khắc phục sự cố.

Sau t giây, kỹ sư bảo dưỡng tới và nhanh chóng khắc phục lỗi. Ban Lãnh đạo nhà máy rất hài lòng với tinh thần trách nhiệm của người công nhân trực và quyết định thưởng cho anh toàn bộ số tinh dầu dò rỉ anh đã tận thu được.

Hãy xác định số giọt tinh dầu đã thưởng cho người công nhân trực.

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

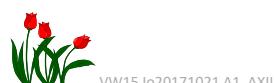
- ✚ Dòng đầu tiên chứa 3 số nguyên n , m và t ($1 \leq n \leq 10^5$, $1 \leq m \leq 10^9$, $0 \leq t \leq 10^9$),
- ✚ Dòng thứ 2 chứa n số nguyên p_1, p_2, \dots, p_n ($1 \leq p_i \leq 10^9$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản LAVENDER.OUT một số nguyên – số giọt tinh dầu đã thưởng cho người công nhân trực.

Ví dụ:

LAVENDER.INP
3 5 17
1 2 3

LAVENDER OUT
4



Giải thuật: Phân tích hoạt động ô tô mát.

Tính số giọt tinh dầu dò rỉ trong một chu kỳ tích áp:

$$\text{drops} = \sum_{i=0}^{n-1} \left(\frac{m}{p_i} \right) \quad (\text{phép chia nguyên})$$

Số chu kỳ tích áp: $\text{pr} = t/m$,

Số tinh dầu dò rỉ ở phần cuối thời gian trước khi sửa máy: $\text{r} = \sum_{i=0}^{n-1} (t \% m) / p_i$.

Từ các tham số này dễ dàng dẫn xuất kết quả cần tìm.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "lavender."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,m,t;

int main()
{
    fi>>n>>m>>t;
    vector<int> p(n);
    int ans,pr,drops=0;
    for(int i=0;i<n;++i) fi>>p[i];
    pr=t/m;
    for(int i=0;i<n;++i) drops+=m/p[i];
    ans=pr*drops; t%=m;
    for(int i=0;i<n;++i) ans+=t/p[i];
    fo<<ans;
    Times;
}
```



Trong thời đại cách mạng khoa học kỹ thuật 4.0 con người dần dần được giải phóng khỏi các công việc chân tay nhảm chán và thay vào đó là các robot. Tuy vậy, robot không những khéo léo mà còn phải có trí tuệ nhân tạo, có khả năng quan sát và xử lý các tình huống lô gic trong công việc.

Để thu hút khách đến dự Lễ hội Bia và Nước giải khát người ta tổ chức một cuộc thi robot phục vụ bàn. Trên một chiếc bàn dài có bày n cốc đầy nước ngọt, bên ngoài cốc thứ i (tính từ trái sang phải) có ghi một số nguyên a_i là giá trị dinh dưỡng của nước trong cốc, $i = 1 \dots n$. Các số khác nhau từng đôi một và nhận giá trị từ 1 đến n . Như vậy ta có một hoán vị các số từ 1 đến n . Robot có thể đổi chỗ 2 cốc đứng cạnh nhau mà không được làm sánh nước ra ngoài và mỗi cốc chỉ được chạm tay vào để di chuyển không quá 2 lần. Mục tiêu của việc di chuyển là nhận được hoán vị có số thứ tự từ điển lớn nhất nếu quan sát từ trái sang phải.

Khán giả sẽ bỏ phiếu bình chọn theo nhiều tiêu chí do Ban Tổ chức đưa ra. Tuy vậy, ở đây chúng ta chỉ quan tâm đến một vấn đề thuần túy kỹ thuật: Dãy số nào sẽ nhận được theo yêu cầu đã nêu.

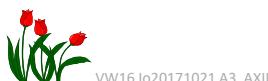
Dữ liệu: Vào từ file văn bản DEXTERITY.INP:

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n là một hoán vị các số từ 1 đến n .

Kết quả: Đưa ra file văn bản DEXTERITY.OUT trên một dòng n số nguyên xác định hoán vị thứ tự từ điển lớn nhất có thể nhận được.

Ví dụ:

DEXTERITY.INP	DEXTERITY.OUT
7	7 3 4 1 2 6 5
7 1 2 3 4 5 6	



VW16 Io20171021 A3_AXII

Giải thuật: Thứ tự từ điển của hoán vị.

Mỗi phần tử của dãy có thể được chuyển dịch sang trái không quá 2 vị trí, vì vậy ta chỉ cần xét 3 vị trí: i , $i + 1$ và $i + 2$,

Gọi cnt_i – số bước chuyển dịch đã thực hiện với phần tử ở vị trí i ,

Lần lượt xét 3 phần tử ở các vị trí i , $i+1$ và $i+2$.

Phần tử ở vị trí $i+2$ sẽ được chuyển dịch sang trái về vị trí i nếu thỏa mãn các điều kiện:

- + $\text{cnt}_{i+2} = 0$ (*trước đó chưa tham gia đổi chỗ*),
- + $p_{i+2} > \max\{p_{i+1}, p_i\}$ (*lớn hơn hai số bên trái*).

Phần tử ở vị trí $i+1$ sẽ được chuyển dịch sang trái về vị trí i nếu thỏa mãn các điều kiện:

- + $\text{cnt}_{i+1} \leq 1$ (*trước đó tham gia đổi chỗ chưa quá một lần*),
- + $p_{i+1} > p_i$.

Khi đổi chỗ cần cập nhật lại các cnt_i tương ứng.

Tổ chức dữ liệu:

Mảng `int` $p[100002]$ – lưu dữ liệu xác định hoán vị,

Mảng `int` $\text{cnt}[100002]$ – ghi nhận số lần đổi chỗ đã áp dụng với mỗi phần tử.

Xử lý:

Tạo hàng rào: $p_n=0$, $\text{cnt}_n=0$;

Duyệt các nhóm 3 $\{i, i+1, i+2\}$ với i từ 0 đến $n-2$ và xét các trường hợp cần đổi chỗ như đã nêu ở trên.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "dexterity."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int n,t;
int p[100002], cnt[100002];

int main()
{
    fi >> n;
    for(int i=0; i<n; ++i) fi>>p[i];
    p[n]=0; cnt[n]=0;

    for(int i=0; i<n-1; ++i)
    {
        if (cnt[i] == 2) continue;
        if (cnt[i+2] == 0 && p[i+2]>p[i] && p[i+2]>p[i+1] && cnt[i+1]<=1)
        {
            cnt[i]++;
            cnt[i+1]++;
            cnt[i+2]+=2;
            t=p[i+2]; p[i+2]=p[i+1]; p[i+1]=p[i]; p[i]=t;
            t=cnt[i+2]; cnt[i+2]=cnt[i+1]; cnt[i+1]=cnt[i]; cnt[i]=t;
            continue;
        }
        if (p[i+1] > p[i] && cnt[i+1] <= 1)
        {
            cnt[i]++;
            cnt[i+1]++;
            swap(p[i+1],p[i]);
            swap(cnt[i+1],cnt[i]);
        }
    }
    for(int i=0; i<n; ++i) fo<<p[i] << ' ';
    Times;
}
```



VW17. MA TRẬN PALINDROME

Tên chương trình: MATRIX.CPP

Steve rất thích thú với tour du lịch mà mình tham gia. Nhưng ngay cả đến mặt trời cũng có vết đen. Mọi thứ đều rất hấp dẫn, mới lạ, trừ thời gian ngồi trên xe chuyển từ địa điểm này tới địa điểm khác. Một cách vô thức Steve cho vào túi tìm kiếm và ngạc nhiên khi rút ra được một tờ giấy gấp tư ghi đầy số. Sau một lúc suy nghĩ, Steve nhớ ra đó là ma trận $n \times m$ của một bài toán quy hoạch tuyến tính đã làm trước khi đi. Có một vài số trên ma trận đã được sửa lại. Steve nảy ra ý nghĩ, tại sao không sửa tiếp một vài số khác để mỗi hàng và mỗi cột đều thành dãy số palindrome, khi đó không cần mở tờ giấy, chỉ nhìn vào một phần tư của nó cũng biết hết các số còn lại.

Hãy xác định số lượng số ít nhất cần sửa để mỗi hàng và mỗi cột của ma trận đều là dãy số palindrome.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n, m \leq 1000$),
- ✚ Mỗi dòng trong n dòng tiếp theo chứa m số nguyên dương xác định một dòng của ma trận.
Các số đều không vượt quá 10^6 .

Kết quả: Đưa ra file văn bản MATRIX.OUT một số nguyên – số lượng ít nhất các số cần sửa.

Ví dụ:

MATRIX.INP
2 4
1 2 2 1
1 2 9 9

MATRIX.OUT
2

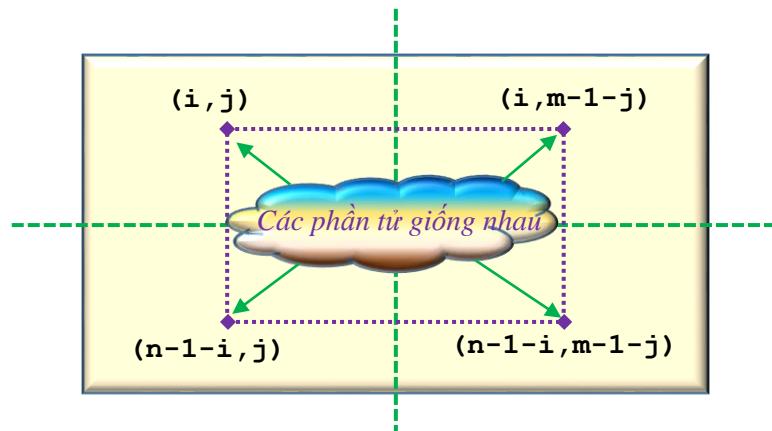


VW17 Io20171021 B6 AXII

Giải thuật; Cơ sở lập trình.

Xét ma trận n dòng và m cột, trong đó mỗi hàng và mỗi cột đều là dãy số palindrome, các hàng và cột đánh số bắt đầu từ 0,

Với $0 \leq i < (n+1)/2$ và $0 \leq j < (m+1)/2$ có:



Nếu 4 phần tử được đánh dấu khác nhau từng đôi một thì cần thay đổi 3 phần tử,

Nếu có 2 phần tử giống nhau – thay đổi 2 phần tử,

Nếu có 3 phần tử giống nhau – thay đổi một phần tử.

Cần xét riêng dòng $n/2$ nếu n lẻ và cột $m/2$ nếu m lẻ.

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "matrix."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

const int N = 1111;
int n, m, c[N][N];

int main()
{
    fi>>n>>m;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j) fi>>c[i][j];

    int ans = 0;
    for (int i = 0; i < (n + 1) / 2; ++i)
        for (int j = 0; j < (m + 1) / 2; ++j)
            if (i * 2 + 1 != n && j * 2 + 1 != m)
            {
                int a[4]={c[i][j],c[n-1-i][j],c[i][m-1-j],c[n-1-i][m-1-j]};
                int cans = 3;
                for (int x : a)
                {
                    int w = 0;
                    for (int y : a)
                        w += (x != y);
                    cans = min(cans, w);
                }
                ans += cans;
            } else if (i * 2 + 1 != n && j * 2 + 1 == m)
                ans += (c[i][j] != c[n-1-i][j]);
            else if (i * 2 + 1 == n && j * 2 + 1 != m)
                ans += (c[i][j] != c[i][m-1-j]);
            else
                ans += (c[i][j] != c[n-1-i][m-1-j]);

    fo<<ans;
    Times;
}
```



Có nhiều loại số đo khác nhau trong toán học để đánh giá sự khác biệt của 2 đối tượng như số đo Euclid, số đo Hamming, số đo rời rạc, ... Các đối tượng tham gia tính số đo phải có cùng số chiều.

Trong công trình nghiên cứu khoa học của mình Steve đã cải tiến số đo rời rạc, đưa ra khái niệm “Số đo rời rạc có trọng số” cho phép xác định khoảng cách giữa 2 vec tơ hoặc mảng không cùng số chiều. Cụ thể là với với 2 mảng số $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$ và $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_m)$ khoảng cách có trọng số $d(\mathbf{A}, \mathbf{B})$ được tính theo công thức sau:

$$d(\mathbf{A}, \mathbf{B}) = \sum_{i=1}^n \sum_{j=1}^m |a_i - b_j|$$

Là một nhà toán học, Steve không mấy quan tâm đến việc tính $d(\mathbf{A}, \mathbf{B})$ cụ thể sẽ được thực hiện như thế nào nhưng đối với các nhà tin học thì là một chuyện khá đau đầu!

Với \mathbf{A} và \mathbf{B} cho trước, hãy tính $d(\mathbf{A}, \mathbf{B})$.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^4$, $i = 1 \div n$),
- ✚ Dòng thứ 3 chứa số nguyên m ($1 \leq m \leq 10^5$),
- ✚ Dòng thứ 4 chứa m số nguyên b_1, b_2, \dots, b_m ($1 \leq b_j \leq 10^4$, $j = 1 \div m$).

Kết quả: Đưa ra file văn bản METRICS.OUT một số nguyên – $d(\mathbf{A}, \mathbf{B})$.

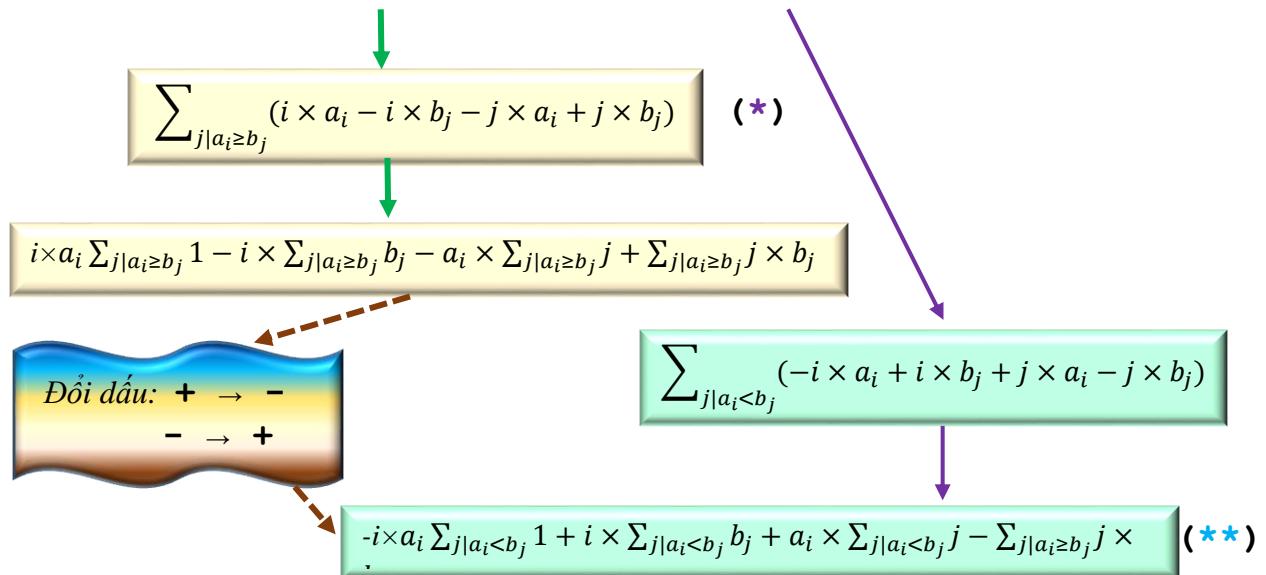
Ví dụ:

METRICS.INP	METRICS.OUT
4 1 4 3 6 3 8 1 1	34



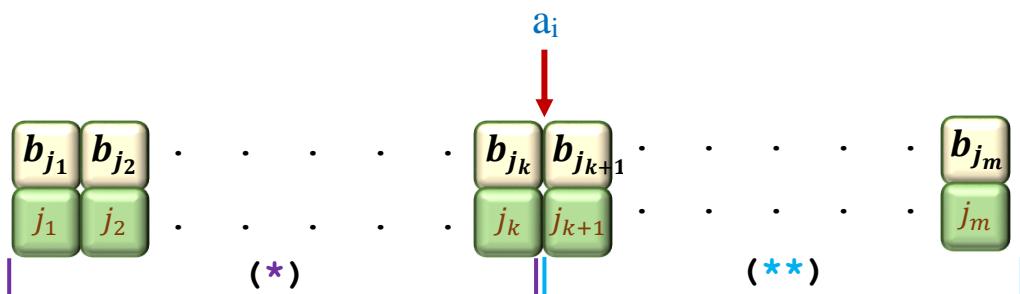
Giải thuật: Tổng tiền tố, Tìm kiếm nhị phân..

Cố định một chỉ số i và từ đó – kéo theo việc cố định giá trị a_i , ta xác định cách tính $\sum_j (i - j) |a_i - b_j| = \sum_{j|a_i \geq b_j} (i - j)(a_i - b_j) + \sum_{j|a_i < b_j} (i - j)(b_j - a_i)$



Để tính các tổng trên ta cần lưu mảng thứ 2 (mảng \mathbf{B}) dưới dạng các cặp $(\mathbf{b}_j, \mathbf{j})$, sắp xếp chúng theo thứ tự tăng dần và, với mảng đã sắp xếp, chuẩn bị các mảng tổng tiền tố với \mathbf{j} , \mathbf{b}_j và $\mathbf{j} \times \mathbf{b}_j$.

Bằng phương pháp tìm kiếm nhị phân, với mỗi a_i có thể xác định được \mathbf{k} chia mảng chứa $(\mathbf{b}_j, \mathbf{j})$ thành hai phần, mỗi phần ứng với một công thức đã nêu.



Tổ chức dữ liệu:

- ▀ Mảng `vector<int> a (n+1)` – lưu giá trị a_i , $i = 1 \div n$,
- ▀ Mảng `vector<pii> b (m+1)` – lưu cặp giá trị $(\mathbf{b}_j, \mathbf{j})$, $j = 1 \div m$,
- ▀ Mảng `vector<int64_t> sb (m+1)` – lưu tổng tiền tố \mathbf{b}_j ,
- ▀ Mảng `vector<int64_t> sj (m+1)` – lưu tổng tiền tố \mathbf{j} ,
- ▀ Mảng `vector<int64_t> sjb (m+1)` – lưu tổng tiền tố $\mathbf{j} \times \mathbf{b}_j$.

Để tiện tính toán, các phần tử dữ liệu được *danh số từ 1*.

Xử lý:

Sắp xếp và tính các tổng tiền tố cần thiết:

```
sort(b.begin(), b.end());  
  
sb[0]=sjb[0]=sj[0]=0;  
for(int j=1; j<=m; ++j)  
{  
    sb[j]=sb[j-1]+b[j].first;  
    sj[j]=sj[j-1]+b[j].second;  
    sjb[j]=sjb[j-1]+b[j].first*b[j].second;  
}
```

Sơ đồ xử lý với mỗi a_i :

Tìm điểm phân chia

```
if(a[i]<b[0].first) idx=0;  
else if(a[i]>=b[m].first) idx=m;  
else  
    idx=upper_bound(b.begin(), b.end(), make_pair(a[i], -1))-b.begin()-1;  
  
{ans+=((int64_t)a[i]*i*idx+sjb[idx]); ← (*)  
ans-=((int64_t)i*sb[idx]+(int64_t)a[i]*sj[idx]);  
ans-=(int64_t)(a[i]*i*(m-idx)+sjb[m]-sjb[idx]); ← (**)  
ans+=(int64_t)i*(sb[m]-sb[idx])+(int64_t)a[i]*(sj[m]-sj[idx]); ← (***)}
```

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "metrics."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
int n,m;

int main()
{
    fi>>n;
    vector<int>a(n+1);
    for(int i=1; i<=n; ++i) fi>>a[i];
    fi>>m;
    vector<pii>b(m+1); b[0]={0,0};
    vector<int64_t> sb(m+1),sjb(m+1),sj(m+1);
    for(int i=1;i<=m;++i){fi>>b[i].first; b[i].second=i;}

    sort(b.begin(),b.end());

    sb[0]=sjb[0]=sj[0]=0;
    for(int j=1;j<=m;++j)
    {
        sb[j]=sb[j-1]+b[j].first;
        sj[j]=sj[j-1]+b[j].second;
        sjb[j]=sjb[j-1]+b[j].first*b[j].second;
    }
    int64_t ans=0,idx;
    for(int i=1;i<=n;++i)
    {
        if(a[i]<b[0].first) idx=0;
        else if(a[i]>=b[m].first) idx=m;
        else
            idx=upper_bound(b.begin(),b.end(),make_pair(a[i],-1))-b.begin()-1;

        ans+=(int64_t)a[i]*i*idx+sjb[idx];
        ans-=(int64_t)i*sb[idx]+(int64_t)a[i]*sj[idx];

        ans-=(int64_t)(a[i]*i*(m-idx)+sjb[m]-sjb[idx]);
        ans+=(int64_t)i*(sb[m]-sb[idx])+(int64_t)a[i]*(sj[m]-sj[idx]);
    }

    fo<<ans;
    Times;
}
```



VW19. GIÁ SÁCH

Tên chương trình: SHELF.CPP

Trên một giá sách của thư viện lưu trữ sách của n chuyên đề, chuyên đề i có b_i cuốn, $i = 1 \dots n$. Các cuốn cùng chuyên đề được đặt liên tiếp nhau và theo thứ tự tăng dần của chuyên đề từ trái sang phải.

Trong ngày có m độc giả đến trả sách, mỗi người trả một cuốn sách. Độc giả thứ j trả sách chuyên đề t_j .

Mỗi lần có người trả, thủ thư phải chuyển dịch sang trái hoặc sang phải một số cuốn trên giá, lấy chỗ để xếp sách mới vào đúng vị trí sao cho các cuốn cùng chuyên đề được đặt liên tiếp nhau, sau đó đẩy chúng sít lại.



Hãy xác định số lượng ít nhất các cuốn sách cần di chuyển mỗi khi có độc giả trả sách.

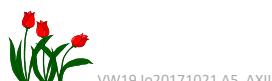
Dữ liệu: Vào từ file văn bản SHELF.INP:

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$, $i = 1 \dots n$),
- ✚ Dòng thứ 2 chứa n số nguyên b_1, b_2, \dots, b_n ($0 \leq b_i \leq 10^5$),
- ✚ Dòng thứ 3 chứa số nguyên m ($1 \leq m \leq 10^5$),
- ✚ Dòng thứ 4 chứa m số nguyên t_1, t_2, \dots, t_m ($1 \leq t_j \leq n$, $j = 1 \dots m$).

Kết quả: Đưa ra file văn bản SHELF.OUT m số nguyên – số lượng ít nhất các cuốn sách cần di chuyển sau mỗi lần độc giả trả sách.

Ví dụ:

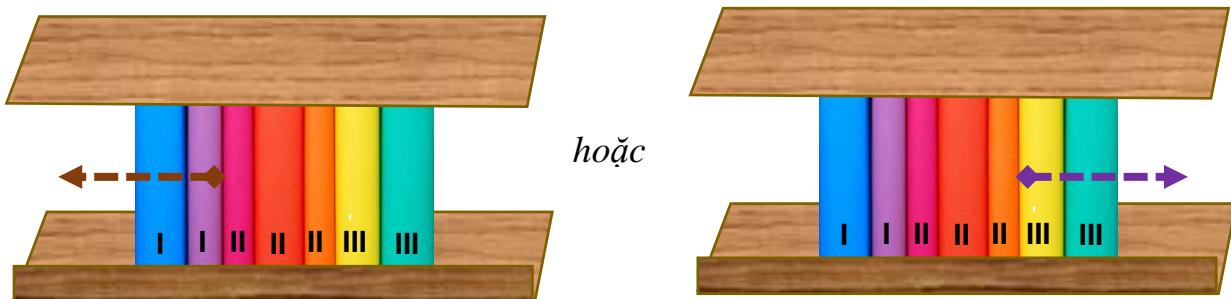
SHELF.INP	SHELF.OUT
8	2 0 2 0 3 0
2 3 2 0 0 0 0 0	
6	
2 1 2 3 2 5	



VW19.la20171021.A5_AXII

Giải thuật: Tổng tiền tố, Cây Fenwick.

Để đưa cuốn sách chuyên đề t_j vào đúng vị trí cần phải di chuyển các cuốn sách từ chuyên đề 1 đến chuyên đề t_{j-1} sang trái hoặc chuyển các cuốn từ chuyên đề t_{j+1} đến chuyên đề n sang phải:



Bổ sung cuốn sách chuyên đề 2

Giữa 2 lựa chọn cần thực hiện phương án có số lượng sách di chuyển nhỏ hơn.

Để xác định số lượng sách phải di chuyển cần có *hàm tính tổng tiền tố* bởi vì số lượng sách mỗi loại thay đổi trong quá trình xử lý. Cây Fenwick là một sự lựa chọn thích hợp vì không đòi hỏi nhiều bộ nhớ và các hàm xử lý đơn giản.

Tổ chức dữ liệu:

Mảng `int64_t fw[100002] = {0}` – lưu tổng tiền tố,

Mảng `vector<int64_t> ans(m)` – lưu kết quả cần tìm.

Xử lý:

Các hàm bổ sung phần tử và tính tổng trên cây Fenwick:

```

void insert_t(int ii, int64_t x)
{
    while(ii <= n) { fw[ii] += x; ii += (ii & (-ii)); }
}

int64_t sum_t(int ii)
{
    int64_t r;
    r = 0;
    while(ii > 0) { r += fw[ii]; ii -= (ii & (-ii)); }
    return(r);
}

```

Xử lý một lần xếp sách:

Cây Fenwick quản lý
dữ liệu từ 1

```
fi>>t;  
if(t==1) rl=0; else rl=sum_t(t-1);  
if(t==n) rr=0; else rr=total-sum_t(t);  
ans[i]=min(rl,rr);  
insert_t(t,1LL); ++total;
```

Cập nhật cây và
tổng số sách

Độ phức tạp của giải thuật: O(nlnn).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "shelf."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int n,m;
int64_t fw[100002]={0},total=0,rl,rr,t;

void insert_t(int ii,int64_t x)
{
    while(ii<=n){fw[ii]+=x;ii+=(ii&(-ii));}
}

int64_t sum_t(int ii)
{
int64_t r;
r=0;
while(ii>0){r+=fw[ii];ii&=(ii-1);}
return(r);
}

int main()
{
    fi>>n;
    for(int i=1;i<=n;++i){fi>>t; insert_t(i,t); total+=t;}

    fi>>m;
    vector<int64_t>ans(m);
    for(int i=0;i<m;++i)
    {
        fi>>t;
        if(t==1) rl=0; else rl=sum_t(t-1);
        if(t==n) rr=0; else rr=total-sum_t(t);
        ans[i]=min(rl,rr);
        insert_t(t,1LL); ++total;
    }
    for(auto &i:ans) fo<<i<<' ';
    cout<<Times;
}
```



Khu vui chơi giải trí có n khu vực rời nhau, đánh số từ 1 đến n . Kinh phí dự trù còn lại chỉ đủ xây dựng m con đường, mỗi đường nối 2 khu vực khác nhau và là đường hai chiều. Giữa 2 khu vực khác nhau có không quá một đường nối trực tiếp.

Ban Quản lý muốn thiết kế các con đường sao cho khách du lịch xuống xe ở một khu vui chơi nào đó, có thể đi thăm từ nơi này đến nơi khác chưa thăm, mỗi khu thăm một lần và trở về đúng nơi mình đã xuống xe.

Hãy xác định có thể thiết kế mạng đường đi như vậy hay không, nếu có thể, hãy chỉ ra một cách thiết kế các đường đi.

Dữ liệu: Vào từ file văn bản TOURMAP.INP gồm một dòng chứa 2 số nguyên n và m ($1 \leq n \leq 13$, $0 \leq m \leq \frac{n(n-1)}{2}$).

Kết quả: Đưa ra file văn bản TOURMAP.OUT trên một dòng thông báo **Possible** hoặc **Impossible** ứng với trường hợp có cách thiết kế hoặc không. Nếu có cách thiết kế thì ở dòng thứ i trong m dòng tiếp theo chứa 2 số nguyên a_i , b_i xác định một con đường, $a_i < b_i$ và $a_i \leq a_j$ với $i < j$.

Ví dụ:

TOURMAP.INP	TOURMAP.OUT
4 4	Possible 1 4 1 2 2 3 3 4

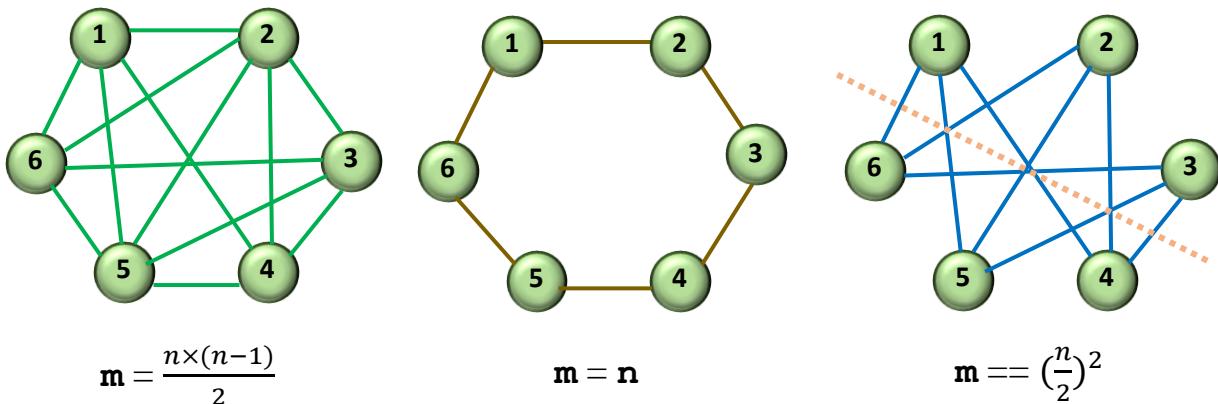


Giải thuật: Chu trình Hamilton.

Tùy lý thuyết đồ thị, với n đỉnh có 3 trường hợp có thể xây dựng một đồ thị m cạnh theo các tính chất đã nêu:

- ✚ Đồ thị đầy đủ: $m = \frac{n \times (n-1)}{2}$;
- ✚ Một chu trình: $m = n$,
- ✚ Đồ thị 2 phía đầy đủ đối xứng: $m = (\frac{n}{2})^2$.

Ví dụ, với $n = 6$, ta có 3 đồ thị thỏa mãn yêu cầu đã nêu:



Nếu m không thỏa mãn một trong 3 điều kiện trên thì không tồn tại đồ thị thỏa mãn yêu cầu đầu bài.

Việc dẫn xuất các đường đi là không khó khăn. Ở trường hợp thứ 3, các đỉnh từ 1 đến $\frac{n}{2}$ được xếp vào phía thứ nhất, các đỉnh còn lại – phía thứ 2.

Độ phức tạp của giải thuật: $O(n^2)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "tourmap."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
//ifstream fi ("27");
ofstream fo (NAME"out");

using namespace std;

int main()
{
    int n, m;
    fi >> n >> m;

    if (n == m)
    {
        fo << "Possible" << endl;
        fo << "1 " << n << endl;
        for (int i = 1; i < n; ++i)
            fo << i << " " << (i + 1) << endl;
        Times; return 0;
    }

    if (m == n * (n - 1) / 2)
    {
        fo << "Possible" << endl;
        for (int i = 1; i <= n; ++i)
            for (int j = i + 1; j <= n; ++j)
                fo << i << " " << j << endl;
        Times; return 0;
    }

    if (n % 2 == 0 && n / 2 * n / 2 == m)
    {
        fo << "Possible" << endl;
        for (int i = 1; i <= n / 2; ++i)
            for (int j = n / 2 + 1; j <= n; ++j)
                fo << i << " " << j << endl;
        Times; return 0;
    }

    fo << "Impossible" << endl;
    Times; return 0;
}
```

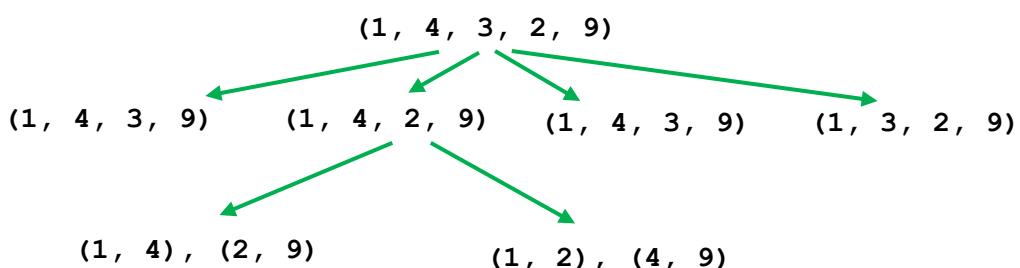


Cho dãy số nguyên dương $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$. Bằng cách gạch bỏ một số phần tử của \mathbf{A} ta nhận được một dãy con \mathbf{B} . Số phần tử trong dãy con được gọi là độ dài của nó.

Với mỗi số nguyên k từ 1 đến n hãy xác định độ dài lớn nhất của dãy con \mathbf{B} nhận được từ \mathbf{A} , sao cho dãy \mathbf{B} có thể tách thành các dãy con không giao nhau độ dài k và mỗi dãy con đều là tăng dần.

Dãy con $\mathbf{C} = (\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_k)$ là tăng dần nếu $\mathbf{c}_1 < \mathbf{c}_2 < \dots < \mathbf{c}_k$.

Ví dụ, với $\mathbf{A} = (1, 4, 3, 2, 9)$ và $k = 2$ ta có thể trích từ \mathbf{A} dãy con \mathbf{B} độ dài 4 thỏa mãn điều kiện đã nêu. Có nhiều cách xác định \mathbf{B} , một trong các dãy con độ dài 4 phù hợp là $(1, 4, 2, 9)$ vì dãy này có thể các dãy con tăng dần độ dài 2:



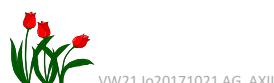
Dữ liệu: Vào từ file văn bản SUBSEQ.INP:

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 300$),
- ✚ 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 SUBSEQ.OUT trên một dòng n số nguyên – các độ dài lớn nhất tìm được.

Ví dụ:

SUBSEQ.INP	SUBSEQ.OUT
5	5 4 3 0 0
1 4 3 2 9	



Giải thuật: Quy hoạch động.

Quá trình giải bài toán được chia thành 2 bước :

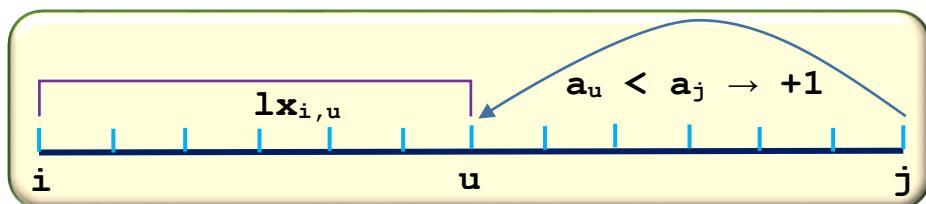
- ⊕ Bước 1: Tính $1x_{i,j}$ – độ dài lớn nhất của dãy con tăng dần từ vị trí i đến vị trí j và có *sự tham gia* của phần tử a_j ,
- ⊕ Bước 2: Tính $f_{k,r}$ – số lượng dãy con tăng dần có độ dài lớn hơn hoặc bằng k trong r phần tử đầu tiên của dãy \mathbf{A} .

Việc tính $1x_{i,j}$ được thực hiện bằng phương pháp quy hoạch động.

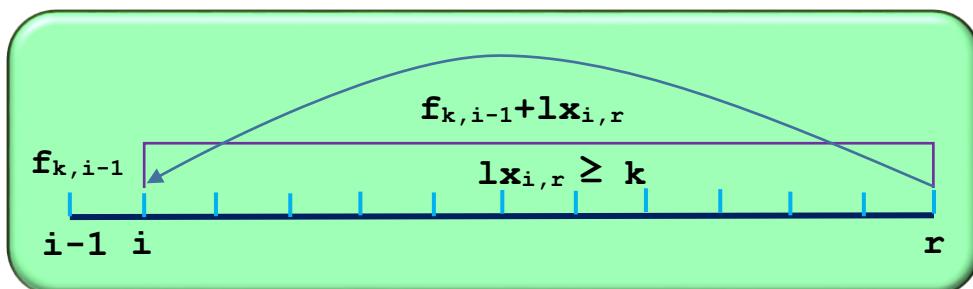
Gọi m là độ dài của đoạn cần tính. Tồn tại giải thuật tính $1x_{i,j}$ với độ phức tạp $O(mlnm)$, nhưng với m bé ($m \leq 300$), để đơn giản lập trình ta có thể tính theo giải thuật độ phức tạp $O(m^2)$:

Khởi tạo: $1x_{i,j} = 1$,

Công thức lặp: $1x_{i,j} = \max\{1x_{i,j}, 1x_{i,u} + 1\}$ nếu $a_u < a_j$, $u = i, i+1, \dots, j-1$.



Tính $f_{k,r}$:



Nếu chưa xét đoạn từ i đến r ($i < r$) thì $f_{k,r} = \max\{f_{k,r}, f_{k,i-1}\}$.

Xét sự tham gia của đoạn từ vị trí i đến vị trí r : Nếu $1x_{i,r} \geq k$ thì giá trị cần tìm *có thể* là $f_{k,i-1} + 1x_{i,r}$ nếu nó lớn hơn $f_{k,r}$ đang có.

Cần duyệt với mọi i thay đổi từ 1 đến n và với mỗi i – duyệt r từ i đến n .

Kết quả: Với mỗi k kết quả cần tìm là giá trị $f_{k,n}$.

Tổ chức dữ liệu:

- ▀ Mảng **int** a [N] lưu dữ liệu vào,
- ▀ Mảng **int** l× [N] [N] lưu giá trị cần tìm ở bước 1,
- ▀ Mảng **int** f [N] [N] lưu giá trị cần tìm ở bước 2.
- ▀ Mảng **int** ans [N] lưu kết quả.

Độ phức tạp của giải thuật: $O(n^2)$.

Lưu ý: Có thể dừng kết xuất kết quả với các **f_m** (**m** > **k**) khi gặp **f_k** = 0.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "subseq."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

const int N = 333;

int n,a[N],ans[N],f[N][N],lx[N][N];

int main()
{
    fi>>n;
    for (int i = 1; i <= n; ++i) fi>>a[i];

    for (int i = 1; i <= n; ++i)
    {
        for (int j = 0; j < i; ++j) lx[i][j] = 0;
        for (int j = i; j <= n; ++j)
        {
            lx[i][j] = 1;
            for (int u = i; u < j; ++u)
                if (a[u] < a[j]) lx[i][j] = max(lx[i][j],lx[i][u]+1);
        }
    }

    for (int k = 1; k <= n; ++k)
    {
        for (int i = 1; i <= n; ++i)
            for (int j = i; j <= n; ++j)
            {
                f[k][j] = max(f[k][j], f[k][j - 1]);
                if (lx[i][j]>=k) f[k][j] = max(f[k][j], f[k][i-1]+lx[i][j]);
            }
        ans[k] = f[k][n];
    }
    for (int i = 1; i <= n; ++i) fo<<ans[i]<<' ';
}

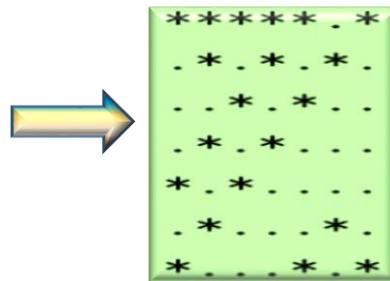
Times; return 0;
}
```



Thông tin dưới dạng bảng gợi mở cho ta nhiều ý tưởng độc đáo. Jimmy muốn tìm đề tài cho buổi sinh hoạt seminaire thường kỳ của Câu lạc bộ Tin Học và quyết định khảo sát mối quan hệ giữa một số tự nhiên với các ước của nó.

Jimmy tạo một bảng vuông kích thước $n \times n$ và điền các số từ 1 đến n^2 theo từng dòng, bắt đầu từ ô trên trái, mỗi ô chứa một số. Với một số nguyên k chọn trước Jimmy tô đậm các ô chứa các số có không quá k ước và sau đó tạo bản đồ, trong đó các ô tô đậm được đánh dấu ‘*’, các ô còn lại – dấu ‘.’. Ví dụ, với $n = 7$ và $k = 3$, bảng đồ nhận được có dạng:

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49



Để có thể rút ra quy luật nào đó Jimmy cần tạo rất nhiều bản đồ khác nhau.

Cho n và k . Hãy xác định bản đồ nhận được.

Đữ liệu: Vào từ file văn bản REGUARITY.INP gồm một dòng chứa 2 số nguyên n và k ($1 \leq n \leq 40$, $1 \leq k \leq n^2$).

Kết quả: Đưa ra file văn bản REGUARITY.OUT bản đồ nhận được gồm n xâu, mỗi xâu trên một dòng, độ dài n và chỉ chứa các ký tự trong tập {*, .}.

Ví dụ:

REGUARITY.INP
7 3

REGUARITY.OUT
*****.*
.*.**.*.
..*.*...*
.**.*....
......
.**....*
.....*



Giải thuật: Tính các ước số.

Mỗi số tự nhiên có ít nhất 2 ước: 1 và chính nó,

Với mỗi số a : các ước còn lại được tìm bằng cách duyệt và kiểm tra trực tiếp từ 2 đến \sqrt{a} . Nếu j là một ước thì cần ghi nhận thêm ước a/j nếu $j^2 < a$.

Dùng mảng đánh dấu các số có số lượng ước không ít hơn k và dựa vào đó – đưa ra bản đồ cân dã xuất.

Độ phức tạp của giải thuật: $O(n^{2.5})$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "regularity."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n, k, dvr,;

int main()
{
    fi>>n>>k;
    vector<bool>b (n*n+1, 0);
    b[1]=true;
    for(int i=2;i<=n*n;++i)
    {
        dvr=2;
        for(int j=2;j*j<=i;++j)
            if(i%j==0) if(j*j==i) dvr+=1; else dvr+=2;
        b[i]=dvr<=k;
    }
    for(int i=1;i<=n;++i)
    {
        for(int j=1;j<=n;++j) if(b[(i-1)*n+j]) fo<<'*'; else fo<< '.';
        fo<<'\n';
    }
    Times;
}
```



Số chỉ chứa một loại chữ số bao giờ cũng dễ nhớ và mơ ước của nhiều người khi đi đăng ký biển số xe, chọn số của SIM điện thoại, mã tài khoản v.v... Vì vậy trên quan điểm của nhiều người, những số chỉ chứa một chữ số là số đẹp.

Steve nhớ được nhiều nhiều số liệu khô khan khác nhau cũng dựa trên cơ sở đó: phân tích mỗi số thành tổng của một hoặc nhiều số hạng, mỗi số hạng là một số đẹp. Ví dụ, nếu bạn có nhiệm vụ lập kế hoạch dài hạn cho tương lai thì cần phải nhớ rằng năm 3328 có tháng 2 với 30 ngày! Làm thế nào để nhớ số 3328? Không khó khăn lắm nếu để ý rằng $3328 = 2222 + 999 + 99 + 8$.

Chính vì vậy cần phải học cách phân tích một số bất kỳ thành tổng các số đẹp.

Dữ liệu: Vào từ file văn bản NICE_NUM.INP gồm một dòng chứa số nguyên n ($0 \leq n < 10^{100}$).

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

- ✚ Dòng đầu tiên chứa số nguyên k – số lượng số hạng trong tổng,
- ✚ Dòng thứ 2 chứa k số nguyên đẹp theo trình tự không tăng – các số hạng tìm được.

Nếu có nhiều cách phân tích thì đưa ra cách tùy chọn.

Ví dụ:

NICE_NUM.INP	NICE_NUM.OUT
3328	4 2222 999 99 8



VW23 IO20171111 G AXII

Giải thuật: Xử lý số lớn.

Tìm số nguyên đẹp **x** lớn nhất và nhỏ hơn hoặc bằng **n**,

Lặp lại quá trình tìm kiếm trên với **n-x** cho đến khi nhận được kết quả bằng 0.

Các số cần lưu trữ dưới dạng xâu hoặc mảng số.

Tổ chức dữ liệu:

- Mảng **vector<int>** **v** – lưu các chữ số của **n**,
- Mảng **vector<vector<int>>** **ans** – lưu kết quả.

Xử lý:

Ghi nhận dữ liệu:

Nhập xâu dữ liệu và ghi nhận các chữ số của **n** theo trình tự “ngược” – bắt đầu từ hàng đơn vị, trong quá trình xử lý số chữ số của **n** đang xét giảm dần nhưng **vị trí các chữ số còn lại không thay đổi** trong mảng.

```

string s;
fi >> s;
int n = s.size();
vector<int> v;
vector<vector<int>> ans;
for (int i = n - 1; i >= 0; i--) v.push_back(s[i] - '0');
    
```

Tìm chữ số của số đẹp lớn nhất không vượt quá **n**:

Số đẹp cần tìm có thể có thể có số lượng chữ số bằng hoặc nhỏ hơn 1 số với số chữ số của **n**,

```

int d = v[v.size() - 1];
for (int i = (b) v.size() - 2; i >= 0; i--)
{
    if (v[i] < d)
    {
        d--; break;
    }
    if (v[i] > d) break; Dánh dấu trường hợp số đẹp có ít chữ số hơn
}
int t = 1;
if (d == 0) {d = 9; t = 0;}
    
```

Gọi chữ số bậc cao nhất (chữ số cuối cùng trong mảng) là **d**,

Xét số đẹp hình thành từ chữ số **d** và độ dài bằng độ dài của **n**,

So sánh số này với **n**, nếu nó lớn hơn thì giảm **d** đi 1,

Nếu kết quả có **d = 1** thì số đẹp cần tìm là số chứa chữ số 9 với độ dài nhỏ hơn 1 so với độ dài của **n**.

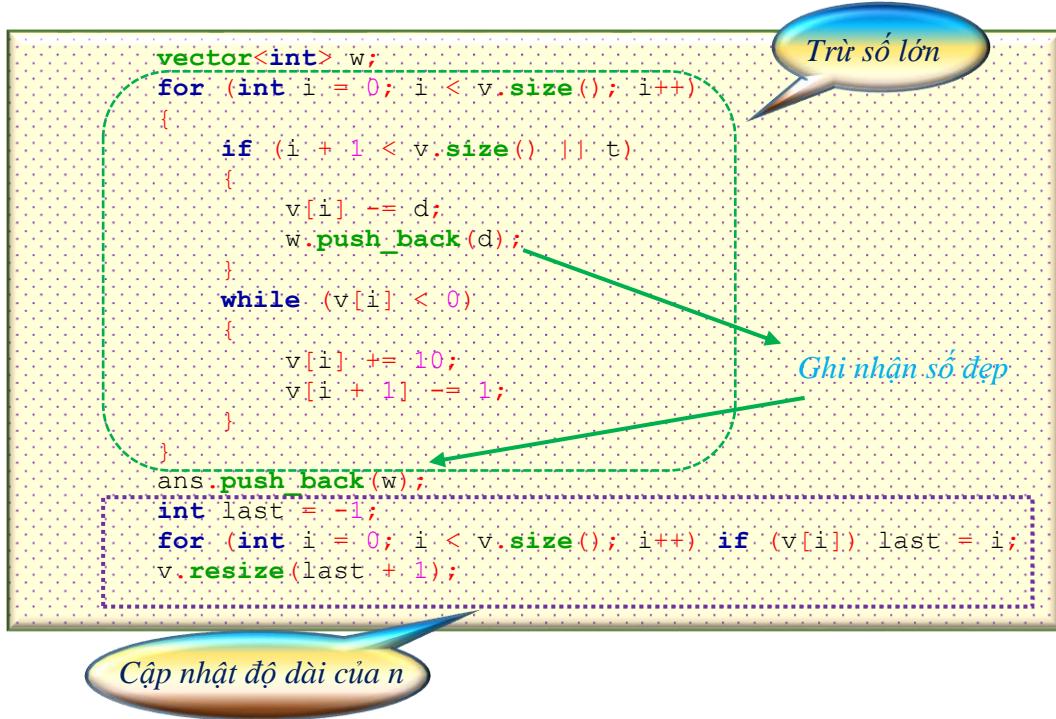
Cập nhật **n** và lưu kết quả số đẹp tìm được:

Cần phân biệt trường hợp số đẹp có độ dài bằng hoặc nhỏ hơn độ dài số **n**,

Thực hiện phép trừ số lớn theo 2 mảng số,

Ghi nhận số đẹp: vì số lượng số đẹp cần dẫn xuất không nhiều nên có thể ghi nhận mảng chứa các chữ số của số đẹp hoặc ghi nhận dưới dạng cặp (*chữ số, độ dài*),

Sau khi thực hiện phép trừ cần cập nhật lại độ dài của n.



Dẫn xuất kết quả: phụ thuộc vào cách ghi nhận kết quả trước đó.

Độ phức tạp của giải thuật: O(n²).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "nice_num."
#define Times fo<<"\n*** Time: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    string s;
    fi >> s;
    int n = s.size();
    vector<int> v;
    vector<vector<int> > ans;
    for (int i = n - 1; i >= 0; i--) v.push_back(s[i] - '0');
    while (v.size())
    {
        int d = v[v.size() - 1];
        for (int i = (int)v.size() - 2; i >= 0; i--)
        {
            if (v[i] < d)
            {
                d--; break;
            }
            if (v[i] > d) break;
        }
        int t = 1;
        if (d == 0) {d = 9; t = 0;}
        vector<int> w;
        for (int i = 0; i < v.size(); i++)
        {
            if (i + 1 < v.size() || t)
            {
                v[i] -= d;
                w.push_back(d);
            }
            while (v[i] < 0)
            {
                v[i] += 10;
                v[i + 1] -= 1;
            }
        }
        ans.push_back(w);
        int last = -1;
        for (int i = 0; i < v.size(); i++) if (v[i]) last = i;
        v.resize(last + 1);
    }
    fo << ans.size() << endl;
    for (auto w: ans)
    {
        for (auto x: w) fo << x;
        fo << ' ';
    }
    Times;
}
```



VW24. KHẢO SÁT TÂM LÝ

Tên chương trình: PSYCHOLOGY.CPP

Các toa của tàu hành khách tốc độ cao được đánh số bắt đầu từ 1, ở mỗi toa các ghế ngồi cũng được đánh số từ 1 trở đi. Hành khách khi lên tàu, tự tìm chỗ ngồi mình thích. Một khảo sát về tâm lý hành khách cho thấy, nếu còn nhiều chỗ trống mà hành khách chọn chỗ ngồi là toa n và ghế k trong toa thì sẽ có hệ số tâm lý là

$$\beta(n, k) = \begin{cases} 1 & n = 0 \\ k \cdot \frac{\beta(0,k)+\beta(1,k)+...+\beta(n-1,k)}{n} & n \geq 1 \end{cases}$$

Từ camera an ninh người ta thấy người thứ i trong số q hành khách lên tàu đầu tiên chọn ghế k_i trong toa n_i .

Hãy xác định hệ số tâm lý của các hành khách này và đưa ra theo mô đun 998244353.

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

- ✚ Dòng đầu tiên chứa số nguyên q ($1 \leq q \leq 2 \times 10^5$),
- ✚ Dòng thứ i trong q dòng sau chứa 2 số nguyên n_i và k_i ($1 \leq n_i, k_i \leq 2 \times 10^5$).

Kết quả: Đưa ra file văn bản PSYCHOLOGY.OUT q số nguyên – các hệ số tìm được, mỗi số trên một dòng.

Ví dụ:

PSYCHOLOGY.INP
2
5 2
6 3

PSYCHOLOGY.OUT
6
28



Giải thuật: Bảng phương án, số nghịch đảo.

Xét công thức tính hệ số tâm lý:

$$\beta(n, k) = \begin{cases} 1 & n = 0 \\ k \cdot \frac{\beta(0, k) + \beta(1, k) + \dots + \beta(n-1, k)}{n} & n \geq 1 \end{cases}$$

Từ đây ta có thể dẫn xuất công thức truy hồi:

$$\beta(n, k) = \frac{n-1+k}{n} \cdot \beta(n-1, k)$$

Triển khai đệ quy về phải, ta có:

$$\beta(n, k) = \frac{(n-1+k) \cdot (n-1+k-1) \cdot \dots \cdot k}{n \cdot (n-1) \cdot \dots \cdot 2 \cdot 1} \cdot \beta(0, k) = \frac{(n-1+k)!}{n! \cdot (k-1)!}$$

Với số nguyên dương u và số nguyên tố m , theo định lý Fermat nhỏ:

$$u^{m-1} \equiv 1 \pmod{m}$$

Từ đó suy ra u^{m-2} sẽ là số nghịch đảo của u theo mod m .

Như vậy, nếu ta có bảng giá trị $u!$ và bảng các giá trị nghịch đảo tương ứng, việc tính mỗi hệ số tâm lý sẽ có độ phức tạp $O(1)$.

Tổ chức dữ liệu:

- ─ Mảng `ll fc[400001]` – lưu bảng giá trị giai thừa $f_{c,i} = i!$ ($\text{mod } 998244353$),
- ─ Mảng `ll rfc[400001]` – lưu các giá trị nghịch đảo.

Xử lý:

Tính các giai thừa:

```

f[0] = 1;
for(int i=0; i<400000; ++i)
    f[i+1] = (f[i] * ll(i+1)) % mod;

```

Tính số nghịch đảo:

Nếu có v là nghịch đảo của $u!$ ta dễ dàng tính nghịch đảo của $(u-1)!, (u-2)!, (u-3)!, \dots$

$$\begin{aligned}
v \times u! &= v \times u \times (u-1) \times (u-2) \times (u-3) \times \dots \times 3 \times 2 \times 1 = 1 \\
&\quad \boxed{(u-1)!} = 1 \\
&\quad \boxed{(u-2)!} = 1 \\
&\quad \boxed{(u-3)!} = 1
\end{aligned}$$

```

ll powl(ll a, ll st)
{
    ll ans = 1;
    while(st)
    {
        if(st&1) ans = (ans * a) % mod;
        a = (a * a) % mod;
        st>>=1;
    }
    return ans;
}

```

Tính nhanh lũy thừa

```

rfc[400000] = powl(fc[400000], mod - 2);
for (int i = 400000 - 1; i >= 0; i--)
    rfc[i] = (rfc[i + 1] * ll(i + 1)) % mod;

```

Xử lý truy vấn: tra cứu các giá trị trong công thức từ 2 bảng đã lập.

Dộ phức tạp của giải thuật: O(q).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "psychology."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

typedef long long ll;
const ll mod = 998244353;

ll q, n, k, fc[400001], rfc[400001];

ll powl(ll a, ll st)
{
    ll ans = 1;
    while(st)
    {
        if(st&1) ans = (ans * a) % mod;
        a = (a * a) % mod;
        st>>=1;
    }
    return ans;
}

int main()
{
    fi >> q;
    fc[0] = 1;
    for(int i=0; i<400000; ++i)
        fc[i + 1] = (fc[i] * ll(i + 1)) % mod;

    rfc[400000] = powl(fc[400000], mod - 2);
    for (int i = 400000 - 1; i >= 0; i--)
        rfc[i] = (rfc[i + 1] * ll(i + 1)) % mod;
    for(int i=0; i<q; ++i)
    {
        fi >> n >> k;
        k--;
        ll ans = ((fc[n + k] * rfc[k]) % mod) * rfc[n] % mod;
        fo << ans << "\n";
    }
    Times;
    return 0;
}
```



VW25. TÀI KHOẢN NGÂN HÀNG

Tên chương trình: BANK_AC.CPP

Một đối tượng đang bị điều tra về tội tham nhũng. Đối tượng có tài khoản tín dụng ở ngân hàng nước ngoài. Tài khoản tín dụng là loại tài khoản cho phép người sở hữu vay tiêu trước rồi trả sau.

Theo nguyên tắc bảo mật ngân hàng không cung cấp thông tin về các giao dịch của chủ tài khoản. Tuy vậy, tập hợp thông tin từ Mail, Facebook và một số nguồn khác người ta xác định được n phiên giao dịch lần lượt theo trình tự tăng dần của thời gian, phiên giao dịch thứ i có số tiền là f_i , $i = 1 \div n$, nhưng không biết không biết đó là lần rút tiền hay nạp vào tài khoản. Ngoài ra, người ta cũng biết thêm là tài khoản được mở với số dư ban đầu là 0 và trong suốt quá trình tồn tại số dư tài khoản không ít hơn a và không lớn hơn b ($a \leq b$).

Hãy xác định mỗi phép giao dịch là rút tiền hay nạp vào tài khoản và đưa ra xâu s độ dài n , trong đó $s_i = '0'$ nếu phép giao dịch thứ i là rút tiền và bằng ' 1 ' – là nạp tiền. Nếu thông tin thu được không phù hợp với khoảng biến động số dư $[a, b]$ thì đưa ra thông báo **Impossible**.

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

- ✚ Dòng đầu tiên chứa 3 số nguyên n , a và b ($1 \leq n \leq 10^4$, $-100 \leq a \leq b \leq 100$),
- ✚ Dòng thứ 2 chứa n số nguyên f_1, f_2, \dots, f_n ($1 \leq f_i \leq 100$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản BANK_AC.OUT xâu s xác định được hoặc thông báo **Impossible**. Nếu tồn tại nhiều xâu s khác nhau phù hợp thì đưa ra xâu tùy chọn.

Ví dụ:

BANK_AC.INP
5 -2 7
2 4 3 8 1

BANK_AC.OUT
01010



VW25_1020171126_BB_AXII

Giải thuật: Quy hoạch động, giá trị hàm mục tiêu và phương án.

Xét $dp_{i,j}$ với giá trị **bool**, $dp_{i,j} = \text{true}$ nếu ở phiên giao dịch thứ i có thể tạo ra số dư trong tài khoản là j và $dp_{i,j} = \text{false}$ trong trường hợp ngược lại,

$dp_{0,f[i]} = \text{true}$ nếu $f[i] \in [a, b]$,

$dp_{0,-f[i]} = \text{true}$ nếu $-f[i] \in [a, b]$.

Cập nhật $dp_{i,j}$ với $i = 1 \div n$ và với $j = a \div b$:

Nếu $dp_{i-1,j} = \text{true}$ (ở bước trước đã có cách đạt tới số dư trong tài khoản là j) thì giá trị số dư mới sẽ là $j+f[i]$ hoặc $j-f[i]$. Với các số dư j mới – dễ dàng xác định $dp_{i,j}$ tương ứng.

Nếu sau bước n có ít nhất một giá trị $dp_{n,j} = \text{true}$ – bài toán có nghiệm.

Để truy ngược vết xác định các giao dịch cần lưu các giá trị j (số dư tài khoản) trong quá trình xét, vì vậy $dp_{i,j}$ là cặp giá trị (*Khả năng hợp lệ*, Giá trị số dư trong tài khoản). Giá trị số dư tài khoản chỉ cần lưu khi khả năng hợp lệ là **true**.

Lần ngược vết từ cuối về đầu, ta dễ dàng xác định thể loại mỗi phiên giao dịch.

Lưu ý là C++ chỉ làm việc với chỉ số không âm nên phải tịnh tiến các giao dịch thêm **abs(a)** khi biên trái **a** âm!

Tổ chức dữ liệu:

- ▀ Mảng **int** $f[1 \dots MAX]$ – lưu dữ liệu giao dịch,
- ▀ Mảng **vector<bool>** ans – lưu kết quả cần đưa ra,
- ▀ Mảng **pair<bool, int>** $dp[1 \dots MAX][1 \dots 4*MAX2]$ – phục vụ sơ đồ quy hoạch động.

Độ phức tạp của giải thuật: $O(n(b-a))$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "bank_acc."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
//ifstream fi ("26");
ofstream fo (NAME"out");

const int MAX = 1e4 + 5;
const int MAX2 = 1e2 + 5;
const int INF = (int)1e9;
int f[MAX];
pair<bool, int> dp[MAX] [4 * MAX2];

int main()
{
    int n, l, r;
    fi >> n >> l >> r;
    for (int i = 0; i < n; ++i) fi >> f[i];

    int lneg = 0;
    if (l < 0) lneg = -1;

    int x = f[0];
    if ((x > r || x < l) && (-x > r || -x < l))
    {
        fo << "Impossible" << endl;
        return 0;
    }

    if (x >= l && x <= r)
    {
        dp[0][x + lneg].first = true;
        dp[0][x + lneg].second = 0;
    }
    if (-x >= l && -x <= r)
    {
        dp[0][-x + lneg].first = true;
        dp[0][-x + lneg].second = 0;
    }
    for (int i = 1; i < n; ++i)
        for (int j = l; j <= r; ++j)
            if (dp[i - 1][j + lneg].first)
            {
                if (j + f[i] <= r)
                {
                    dp[i][j + f[i] + lneg].first = true;
                    dp[i][j + f[i] + lneg].second = j;
                }
                if (j - f[i] >= l)
                {
                    dp[i][j - f[i] + lneg].first = true;
                    dp[i][j - f[i] + lneg].second = j;
                }
            }
}
```

```

        }

int sum = 0;
vector<bool> ans;
for (int i = l; i <= r; ++i)
    if (dp[n - 1][i + lneg].first)
    {
        sum = dp[n - 1][i + lneg].second;
        if (sum + f[n - 1] == i) ans.push_back(1);
        else if (sum - f[n - 1] == i) ans.push_back(0);
        break;
    }

if ((int)ans.size() == 0)
{
    fo << "Impossible" << endl;
    return 0;
}

for (int i = n - 2; i > 0; --i)
{
    int sum2 = dp[i][sum + lneg].second;
    if (sum2 + f[i] == sum) ans.push_back(1);
    else if (sum2 - f[i] == sum) ans.push_back(0);
    sum = sum2;
}
if (n - 1 > 0)
    if (sum == f[0]) ans.push_back(1);
    else ans.push_back(0);

for (int i = (int)ans.size() - 1; i >= 0; --i) fo << ans[i];

Times;
}

```



Viện Bảo tàng Nghệ thuật thu hồi được một bức tranh quý bị đánh cắp nhiều năm trước. Bức tranh được khảo sát cẩn thận và phục chế các chỗ hư hỏng nếu có trước khi đưa ra trưng bày trở lại. Tác giả bức tranh có thói quen ký tên bằng dòng số nguyên là hoán vị vòng tròn của các số từ 1 đến n . Nhóm trộm tranh đã thay đổi một hoặc vài số trong dãy bằng cách ghi số mới đè lên với giá trị lớn hoặc nhỏ hơn số cũ 1 để làm như đây là một bản sao chứ không phải bản gốc, nhằm chạy tội nếu bị phát hiện.

Bộ phận phục chế cần xác định vị trí của số 1 trong dãy số ban đầu để khôi phục nguyên trạng của bức tranh. Cũng không loại trừ khả năng bức tranh đã qua tay nhiều nhóm tội phạm khác nhau và quá trình sửa dãy số đã được thực hiện nhiều lần. Khi đó cần các công cụ soi chiếu tối tân hơn mới giải quyết được vấn đề.

Hãy xác định, dựa vào dãy số hiện thấy, người ta có xác định được vị trí số 1 ở đâu trong nguyên bản, nếu được – đưa ra thông báo **YES** và vị trí trong dãy của số 1. Trong trường hợp ngược lại – đưa ra thông báo **NO**.

Dữ liệu: Vào từ file văn bản RESTORE.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 ($0 \leq a_i \leq n+1, i = 1 \div n$).

Kết quả: Đưa ra file văn bản RESTORE.OUT trên một dòng thông báo **YES** và ở dòng thứ 2 – một số nguyên xác định vị trí của số 1 trong dãy hoặc đưa ra thông báo **NO** nếu không thể xác định được vị trí của số 1 trong dãy ban đầu.

Ví dụ:

RESTORE.INP
3
2 3 4

RESTORE.OUT
YES
1



VW26.l020171126.BC_AXII

Giải thuật: Phân tích hoạt động ô tô mát.

Để xử lý dữ liệu vòng tròn cần lưu mảng **a** 2 lần liên tiếp nhau.

Giả thiết trong mảng ban đầu số 1 đứng ở vị trí **i**.

Khi đó dãy số từ vị trí **i** đến vị trí **n+i-1** phải là các số 1, 2, 3, . . . , **n**.

Điều này có nghĩa $|a_{i+j-1} - j| \leq 1$.

Vị trí tiềm năng của số 1 là không quá 3 trong dãy số có thể sản sinh ra hoán vị vòng tròn các số từ 1 đến n đứng liên tiếp, vì vậy việc kiểm tra toàn bộ dãy số được thực hiện không quá 3 lần.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "restore."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    int n;
    fi >> n;
    vector<int>a(2*n);
    for (int i = 0; i < n; i++)
    {
        fi >> a[i];
        a[n + i] = a[i];
    }
    int cnt = 0;
    for (int i = 0; i < n; i++)
    {
        if (abs(a[i] - 1) <= 1)
        {
            cnt++;
            if (cnt <= 3)
            {
                bool d = 1;
                for (int j = 0; j < n; j++)
                {
                    if (abs(a[i + j] - j - 1) > 1)
                    {
                        d = 0;
                        break;
                    }
                }
                if (d)
                {
                    fo << "YES\n" << i + 1 << endl;
                    Times; return 0;
                }
            }
        }
    }
    fo << "NO" << endl;
    Times;
    return 0;
}
```



Một thông điệp mật được truyền đi dưới dạng file âm thanh, mỗi nốt nhạc tương ứng với một số nguyên thể hiện cường độ. Để làm cho giống một bản nhạc tự nhiên, mỗi số nguyên trong file ban đầu được cộng thêm một số nguyên chọn ngẫu nhiên trong khoảng $[-r, r]$. Bộ phận an ninh đã chặn được file truyền đi, xác định được r và cố gắng giải mã thông điệp.

Việc đầu tiên cần làm là xác định tối đa có bao nhiêu số nguyên trong file ban đầu chưa bị mã hóa.

File chặn được có n nốt nhạc, nốt thứ i có cường độ a_i , $i = 1 \div n$ và giá trị r xác định khoảng phục vụ chọn số để biến đổi.

Hãy đưa ra m – số lượng tối đa các số khác nhau có thể có trong file ban đầu và dãy số b_1, b_2, \dots, b_n có đúng m số khác nhau thỏa mãn điều kiện $|a_i - b_i| \leq r$.

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

- ⊕ Dòng đầu tiên chứa 2 số nguyên n và r ($1 \leq n \leq 10^5$, $1 \leq r \leq 10^9$),
- ⊕ 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 DECRYPTION.OUT, dòng thứ nhất chứa số nguyên m , dòng thứ 2 chứa n số nguyên b_1, b_2, \dots, b_n thỏa mãn điều kiện đã nêu. Nếu tồn tại nhiều dãy số khác nhau cùng thỏa mãn – đưa ra dãy tùy chọn.

Ví dụ:

DECRYPTION.INP
3 1
1 1 1

DECRYPTION.OUT
3
0 1 2



Giải thuật: Phân tích tình huống lô gic.

Sắp xếp dữ liệu theo thứ tự tăng dần cùng với chỉ số của mỗi phần tử,
 Việc lưu trữ chỉ số là cần thiết để dẫn xuất dãy số kết quả,
 Giả thiết đã xử lý $i-1$ phần tử đầu tiên (trong dãy đã sắp xếp),
 Gọi y là giá trị phần tử cuối cùng đã xác định, x là giá trị phần tử thứ i cần xử lý.



Nguyên tắc xử lý: Nếu có thể thay x bằng số mới, khác biệt với các số đã có thì thay x bằng số nhỏ nhất có thể.

Có 3 trường hợp xảy ra:

- + $x - y > r \rightarrow$ ghi nhận $x-r$ là kết quả và có thêm số mới khác các số trước đó,



đó,

- + $x - y < -r \rightarrow$ không tạo số mới vì mọi số $\in [x-r, x+r]$ đã xuất hiện trước đó,

Không tạo giá trị mới



- + trường hợp còn lại: ghi nhận $y+1$ là kết quả và có thêm số mới khác các số trước đó.

Tổ chức dữ liệu:

- ☰ Mảng `vector<pair<int, int>>` a(n) – lưu dữ liệu input,
- ☰ Mảng `vector<int>` ansr(n) – lưu dãy giá trị mới.

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "decryption."
#define Times fo<<"\n*** Time: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    int n, r;
    fi >> n >> r;
    vector<pair<int, int>> a(n);
    for (int i = 0; i < n; i++)
        fi >> a[i].first, a[i].second = i;
    sort(a.begin(), a.end());
    long long y = -2e9 - 100;
    int ans = 0;
    vector<int> ansr(n);
    for (int i = 0; i < n; i++)
    {
        int j = a[i].second;
        int x = a[i].first;
        if (x - y > r)
        {
            y = x - r + 1;
            ansr[j] = x - r;
            ans++;
        }
        else if (x - y < -r)
            ansr[j] = x;
        else
        {
            ansr[j] = y++;
            ans++;
        }
    }
    fo << ans << endl;
    for (int i = 0; i < n; i++)
        fo << ansr[i] << ' ';
    Times;
    return 0;
}
```



VW28. TRỒNG RAU

Tên chương trình: GARDENING.CPP

Trạm Nghiên cứu khoa học ở Nam cực được trang bị một nhà kính trồng rau đảm bảo nhu cầu vitamin tự nhiên cho các cán bộ trong trạm. Người ta mang theo hạt giống của n loại rau quả, loại thứ i cho phép bắt đầu thu hoạch sau a_i ngày kể từ khi trồng.

Do lịch nghiên cứu khảo sát khá dày đặc, mỗi ngày người ta chỉ có thể tạo ra một luồng trồng một loại rau nào đó trong số các loại hạt giống mang theo.

Với trình tự trồng thích hợp, hãy xác định số ngày tối thiểu mà kể từ đó các nhân viên của Trạm có thể thưởng thức được mọi loại rau quả từ vỏ hạt giống mang theo.

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

- ✚ Dòng đầu tiên chứa một 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 GARDENING.OUT một số nguyên – số ngày tối thiểu tìm được.

Ví dụ:

GARDENING.INP	GARDENING.OUT
5	6
1 5 3 5 4	



VW28_Io20171126_BE_AXII

Giải thuật; Cơ sở lập trình.

Nguyên tắc: Những rau nào chậm thu hoạch cần trồng trước.

Tổ chức dữ liệu:

Mảng **vector<int>** **a(n)** – lưu dữ liệu input.

Xử lý:

Sắp xếp mảng **a** theo trình tự không tăng,

Xác định kết quả: tìm giá trị lớn nhất **a_i+i** với mọi **i**.

Độ phức tạp của giải thuật: **O(nlgn)**.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "gardening."
#define Times fo<<"\n*** Time: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    int n;
    fi>>n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) fi>>a[i];

    sort(a.rbegin(), a.rend());
    int ans = 0;
    for (int i = 0; i < n; i++)
        ans = max(ans, a[i] + i);

    fo<<ans;
    Times;
}
```



VW29. THAY ĐỔI MÃ KHÓA

Tên chương trình: CHANGEWK.CPP

Bác sỹ Watson lưu các mẫu virus nguy hiểm phục vụ nghiên cứu chế tạo vắc xin trong két sắt ở phòng thí nghiệm. Két sắt có ổ khóa chữ. Mã mở két là xâu ký tự **s** chỉ chứa các ký tự la tinh thường. Theo quy tắc an toàn và bảo mật, định kỳ cần thay đổi mã khóa.

Tham khảo ý kiến của các chuyên gia tin học Bác sỹ hiểu rằng một mã khóa có tính an toàn cao nếu nó không chứa 3 hay nhiều hơn các ký tự liên tiếp giống nhau và không một ký tự nào xuất hiện nhiều hơn một nữa độ dài xâu.

Nếu mã mở khóa hiện tại thỏa mãn các điều kiện trên thì tạm thời không thay đổi. Trong trường hợp cần thay đổi Bác sỹ sẽ tìm cách thay đổi ít nhất số ký tự hiện có bằng ký tự mới.

Hãy xác định số ít nhất các ký tự cần thay đổi.

Dữ liệu: Vào từ file văn bản CHANGEWK.INP gồm một dòng chứa xâu **s** độ dài không quá 25 và chỉ chứa các ký tự la tinh thường.

Kết quả: Đưa ra file văn bản CHANGEWK.OUT xâu mã khóa mới. Nếu có nhiều cách chọn mã khóa mới thì đưa ra mã tùy chọn.

Ví dụ:

CHANGEWK.INP	CHANGEWK.OUT
coooooode	cooaooode



Giải thuật: Xử lý xâu.

Độ dài xâu đã cho không vượt quá 25 vì vậy tồn tại ít nhất một ký tự có tần số xuất hiện trong xâu bằng 0, gọi ký tự này là **c0**,

Gọi **n** là độ dài xâu **s**,

Việc xử lý có thể chia thành 2 giai đoạn:

- ✚ Khắc phục các có nhiều hơn 2 ký tự liên tiếp giống nhau,
- ✚ Biến đổi để không có ký tự nào có tần số xuất hiện lớn hơn nữa độ dài của xâu.

Việc xử lý được tiến hành dựa trên cơ sở tần số xuất hiện của ký tự trong xâu.

Giai đoạn 1:

Với mỗi $i \geq 2$ kiểm tra s_i có trùng với s_{i-1} và s_{i-2} hay không , nếu đúng thì thay s_i bằng **c0** và cập nhật lại tần số xuất hiện của các ký tự tương ứng.

Giai đoạn 2:

Nếu tìm thấy ký tự s_i có tần số xuất hiện lớn hơn $n/2$ thì thay thế mỗi ký giống s_i kể từ đầu xâu bằng ký tự có tần số xuất hiện nhỏ nhất và cập nhật tần số xuất hiện của các ký tự liên quan, việc thay thế được tiến hành cho đến khi tần số xuất hiện của s_i không vượt quá **n/2**.

Nhận xét: *Do kích thước xâu s rất nhỏ nên không cần dùng các cấu trúc dữ liệu phức tạp phục vụ tìm ký tự có tần số xuất hiện nhỏ nhất.*

Tổ chức dữ liệu:

Mảng **int** `cnt[30]={0}` – lưu tần số xuất hiện các ký tự trong xâu.

Độ phức tạp của giải thuật: $O(n^2)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "changewk."
#define Times fo<<"\n*** Time: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
char c;
int cnt[30]={0};
string s;

int main()
{
    fi >> s;
    int n = s.size();

    for (int i = 0; i < n; ++i) ++cnt[s[i] - 'a'];
    for (int i = 0; i < 26; ++i)
        if (cnt[i] == 0) {c = i + 'a'; break;}

    for (int i = 2; i < n; ++i)
        if (s[i - 2] == s[i - 1] && s[i - 1] == s[i])
    {
        --cnt[s[i] - 'a'];
        s[i] = c;
        ++cnt[s[i] - 'a'];
    }

    for (int i = 0; i < 26; ++i)
        if (cnt[i] > n / 2)
    {
        c = 'a';
        for (int j = 0; j < n; ++j)
            if (s[j] - 'a' == i)
            {
                for (int q = 0; q < 26; ++q)
                    if (cnt[c - 'a'] > cnt[q]) c = q + 'a';
                s[j] = c;
                ++cnt[c - 'a'];
                --cnt[i];
                if (cnt[i] <= n / 2) break;
            }
    }

    fo << s;
    Times;
    return 0;
}
```



Trong hội nghị cổ đông cuối năm Phó Giám đốc phụ trách tài chính có nhiệm vụ báo cáo về tình hình kinh doanh của Công ty trong năm.

Có tất cả n giao dịch được thực hiện trong năm, giao dịch thứ i có giá trị a_i , $i = 1 \dots n$. Cổ đông sẽ ấn tượng tốt về sự ổn định trong hoạt động của Công ty nếu các giao dịch đều có giá trị xấp xỉ nhau, vì vậy tỷ số giữa giá trị giao dịch nhỏ nhất với giá trị giao dịch lớn nhất sẽ là chỉ số đánh giá mức ổn định trong điều hành kinh doanh. Phó Giám đốc yêu cầu bộ phận Tài chính – Kế toán sửa lại số liệu một chút để cải thiện chỉ số ổn định. Cụ thể ông yêu cầu với một số giao dịch tách một giao dịch thành 2 giao dịch với giá trị tùy chọn sao cho tổng giá trị giao dịch vẫn giữ nguyên, nhưng chỉ số ổn định tăng. Ví dụ, một giao dịch giá trị 100 USD có thể tách thành 2 giao dịch với các giá trị tương ứng là 25.55 và 74.45 USD.

Tuy nhiên, để tránh mọi nghi ngờ có thể xảy ra, các giao dịch mới, xuất hiện sau khi tách sẽ không được tách tiếp.

Hãy xác định chỉ số ổn định lớn nhất có thể đạt được sau khi sửa dữ liệu.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^4$, $i = 1 \dots n$).

Kết quả: Đưa ra file văn bản TRANSACTION.OUT một số thực với độ chính xác 8 chữ số sau dấu chấm thập phân – chỉ số ổn định lớn nhất có thể đạt.

Ví dụ:

TRANSACTION.INP	TRANSACTION.OUT
3	1.00000000
1 2 1	



Giải thuật: Làm việc với số thực.

Nhận xét:

- + Với $a_i < a_j$, nếu chỉ phân chia a_i và không phân chia a_j chỉ số cần tính có thể giảm,
- + Nếu cần phân chia a_j thì tốt nhất là chia thành 2 phần bằng nhau.

Như vậy cần phải sắp xếp các giá trị theo thứ tự tăng dần, duyệt với mọi i từ 2 đến $n-1$, tính

```
min_ans = min{a1, ai/2}
max_ans = max{ai-1, an/2}
```

và tìm $\max\{\min_ans/\max_ans\}$.

Độ phức tạp của giải thuật: O(nlnn).

Lưu ý: Để chống tích lũy sai số làm tròn cần **tránh thực hiện phép chia trong quá trình xử lý**. Chỉ thực hiện phép chia một lần khi dẫn xuất kết quả cuối cùng tìm được.

Chương trình

```
#include <bits/stdc++.h>
#include <bits/stdc++.h>
#define NAME "transaction."
#define Times fo<<"\n*** Time: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int main()
{
    int n;
    fi >> n;
    vector<double> a(n);
    for (int i = 0; i < n; i++) fi >> a[i];
    sort(a.begin(), a.end());
    double ans_min = a.front();
    double ans_max = a.back();
    for (int i = 1; i < n; i++)
    {
        double cur_min = min(a.front(), a[i] / 2);
        double cur_max = max(a[i - 1], a.back() / 2);
        if (ans_min * cur_max < cur_min * ans_max)
            ans_min = cur_min, ans_max = cur_max;
    }
    fo.precision(8);
    fo << fixed << ans_min / ans_max << endl;
    Times;
    return 0;
}
```



VW31. ĐOÀN TÀU DU LỊCH

Tên chương trình: TOURISTS.CPP

Đoàn tàu hỏa chở khách tới du lịch ở một địa điểm nổi tiếng có n toa, toa thứ i có a_i người, $i = 1 \dots n$. Khi đến nơi mọi người đều nóng lòng muốn ra.

Để tránh ùn tắc và gây lộn xộn trên sân ga cứ mỗi đơn vị thời gian người ta có thể cho tất cả hành khách ở một toa xuống hoặc cho mỗi toa một người xuống.

Hãy xác định thời gian tối thiểu để mọi hành khách xuống được ga.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($0 \leq a_i \leq 10^9$, $i = 1 \dots n$).

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

Ví dụ:

TOURISTS.INP	TOURISTS.OUT
3	
1 2 1	2



Giải thuật; Khả năng phân tích giải thuật.

Sắp xếp dữ liệu theo trình tự không tăng,

Các toa đánh số từ 0 đến $n-1$,

Khi đó nếu giải phóng i toa đầu tiên theo từng toa, các toa còn lại – từng người ở mỗi toa thời gian mọi hành khách xuống hết khỏi tàu là $i+a_i$.

Lời giải sẽ là $\min\{i+a_i\}$ với mọi i từ 0 đến $n-1$.

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "tourists."
#define Times fo<<"\n*** Time: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int main()
{
    int n;
    fi>>n;
    vector<int> a(n+1);
    for (int i = 0; i < n; i++) fi>>a[i];
    sort(a.rbegin(), a.rend()); a[n]=0;
    int ans = n;
    for (int i = 0; i < n-1; i++)
        ans = min(ans, i + a[i]);
    fo<<ans;
    Times;
    return 0;
}
```



VW32. GẦN PALINDROME

Tên chương trình: NEARLY.CPP

Một vụ án mạng xảy ra. Trên hiện trường người ta tìm thấy một mảnh giấy nhỏ ghi dòng chữ chứa các ký tự la tinh thường. Đây là manh mối để thu hép phạm vi tìm kiếm nghi can.

Trước hết cần phải kiểm tra xem dòng chữ tìm được có phải là một palindrome hay gần palindrome không vì có một băng tội phạm thường mã hóa tin của mình dưới dạng palindrome hoặc gần palindrome.

Một xâu được gọi là gần palindrome nếu đổi chỗ một số cặp ký tự đứng liền tiếp nhau ta sẽ được một palindrome và mỗi ký tự trong xâu tham gia vào quá trình đổi chỗ không quá một lần.

Hãy xác định dòng chữ tìm được có phải là một xâu palindrome hay gần palindrome hay không.

Dữ liệu: Vào từ file văn bản NEARLY.INP gồm một dòng chứa xâu chỉ có các ký tự la tinh thường và có độ dài không quá 10^5 .

Kết quả: Đưa ra file văn bản NEARLY.OUT thông báo **YES** hoặc **NO**.

Ví dụ:

NEARLY.INP
baaaccba

NEARLY.OUT
YES

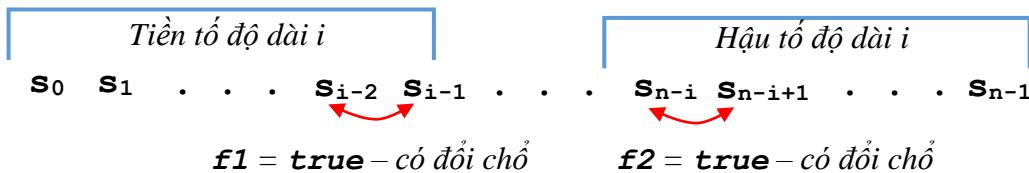


Giải thuật: Quy hoạch động.

Tù điệu kiện đã cho ta chỉ được phép đổi chỗ 2 ký tự cạnh nhau và mỗi ký tự tham gia vào quá trình đổi chỗ không quá 1 lần.

Lời giải bài toán được xây dựng theo sơ đồ quy hoạch động.

Xét $d_{i,f1,f2}$ nhận giá trị **bool**, trong đó $i \leq n/2$, $f1, f2 \in \{\text{true}, \text{false}\}$.



$d_{i,f1,f2} = \text{true}$ nếu *tiền tố độ dài i* với trạng thái **f1** trùng với *đảo ngược hậu tố độ dài i* với trạng thái **f2**.

Ban đầu tất cả $d_{i,f1,f2}$ nhận giá trị **false**.

Xét riêng trường hợp độ dài xâu bằng 1: kết quả luôn là **YES**.

$d_{1,f1,f2}$ được tính trực tiếp bằng cách so sánh các cặp ký tự tương ứng với trạng thái có đổi chỗ hay không.

Tính $d_{i,f1,f2}$:

Dẫn xuất cặp ký tự **c1, c2** tương ứng cuối tiền tố độ dài **i** và đầu hậu tố độ dài **i** cho các trường hợp có thể xuất hiện,

Gọi các trạng thái để nhận được **c1** và **c2** tương ứng phù hợp với **f1, f2** là **nf1** và **nf2**,

$d_{i,f1,f2} = d_{i,f1,f2} \text{ or } d_{i-1,nf1,nf2}$ nếu **c1 == c2**.

Trong trường hợp **n** chẵn kết quả là $d_{n/2,0,0}$.

Trường hợp **n** lẻ: kết quả là $d_{n/2,0,0} \text{ or } d_{n/2,1,0} \text{ or } d_{n/2,0,1}$

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "nearly."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec";
using namespace std;
ifstream fi (NAME"inp");
//ifstream fi ("63");
ofstream fo (NAME"out");
const int N = 100777;

string s;
int n;
bool d[N/2][2][2]={0};

void solve(int n1)
{
    for (int i = 1; i <= n1; ++i)
        for (int w1 = 0; w1 <= 1; ++w1)
            for (int w2 = 0; w2 <= 1; ++w2)
            {
                if (i == 1)
                {
                    d[i][w1][w2] = (s[w1] == s[n-1 - w2]);
                    continue;
                }
                for (int nw1 = 0; nw1 <= 1 - w1; ++nw1)
                    for (int nw2 = 0; nw2 <= 1 - w2; ++nw2)
                    {
                        char c1;
                        if (w1) c1 = s[i];
                        else if (nw1) c1 = s[i-2];
                        else c1 = s[i-1];
                        char c2;
                        if (w2) c2 = s[n-1 - i];
                        else if (nw2) c2 = s[n-1 - (i-2)];
                        else c2 = s[n-1 - (i-1)];
                        if (c1 == c2) d[i][w1][w2] |= d[i-1][nw1][nw2];
                    }
            }
}

int main()
{
    fi >> s;
    n = (int)s.size();
    if (n == 1) {fo << "YES\n"; return 0;}
    solve(n / 2);
    bool res = d[n/2][0][0];
    if (n % 2 == 1 && n != 1) res |= (d[n/2][1][0] | d[n/2][0][1]);
    if (res) fo << "YES\n";
    else fo << "NO\n";
    Times;
}
```



Tuyến đường du lịch tự do (đi phượt) từ bắc vào nam lần lượt đi qua các điểm được đánh số lần lượt từ 1 đến n , trong đó có một số địa điểm được mọi người đặc biệt yêu thích. Hiệp hội những người du lịch tự do quyết định xuất một cẩm nang hướng dẫn hỗ trợ các hội viên. Cẩm nang có 2 phiên bản, một in ở miền bắc và phiên bản kia – in ở miền nam. Để tránh mọi thắc mắc, dị nghị có thể xảy ra người ta quyết định 2 phiên bản giới thiệu 2 đoạn khác nhau các địa điểm liên tục trên đường đi, cả 2 đoạn đều có số lượng địa điểm như nhau và cùng chứa số lượng địa điểm được đặc biệt yêu thích như nhau.

Theo nguyên vong các hội viên, đoạn đường được giới thiệu ở mỗi phiên bản phải dài nhất có thể, tức là chứa nhiều địa điểm nhất.

Hãy xác định điểm đầu, điểm cuối $s1, t1$ nằm trong cẩm nang thứ nhất và điểm đầu, điểm cuối $s2, t2$ nằm trong cẩm nang thứ hai.

Dữ liệu: Vào từ file văn bản TOURS.INP gồm một dòng chứa xâu ký tự độ dài n , mỗi ký tự thuộc tập {0, 1} tương ứng với một điểm trên đường đi, trong đó ký tự 1 đánh dấu điểm được đặc biệt yêu thích. Độ dài n không nhỏ hơn 3 và không vượt quá 10^6 . Dữ liệu đảm bảo tồn tại lời giải.

Kết quả: Đưa ra file văn bản TOURS.OUT bốn số nguyên $s1, t1, s2$ và $t2$.

Ví dụ:

TOURS.INP
10101

TOURS.OUT
1 4 2 5

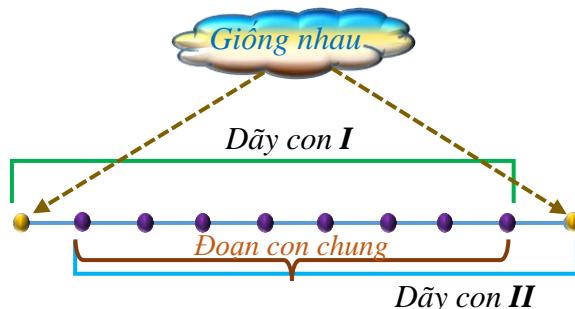


Giải thuật: Phân tích mô hình toán học.

Mô hình toán học:

- ✚ Cho dãy số $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n, \mathbf{a}_i = 0$ hoặc $1, i = 1 \dots n$,
- ✚ Dãy con là các dãy phần tử liên tiếp,
- ✚ Tìm 2 dãy con khác nhau dài nhất chứa cùng số lượng phần tử 1.

Nhận xét: Nếu $\mathbf{a}_u = \mathbf{a}_v$ ($u < v$) thì 2 dãy con $[u \dots v-1]$ và $[u+1 \dots v]$ có cùng độ dài và chứa số lượng phần tử 1 như nhau:



Để có dãy con dài nhất ta cần tìm đoạn con chung dài nhất. Điều này tương đương với việc tìm dãy con dài nhất có phần tử đầu và cuối cùng giá trị.

Nếu $\mathbf{a}_1 = \mathbf{a}_n \rightarrow$ Ta có ngay đoạn cần tìm.

Nếu $\mathbf{a}_1 \neq \mathbf{a}_n$:

- ✚ Tìm p nhỏ nhất thỏa mãn $\mathbf{a}_p = \mathbf{a}_n$,
- ✚ Tìm q lớn nhất thỏa mãn $\mathbf{a}_1 = \mathbf{a}_q$,
- ✚ Chọn đoạn dài hơn trong 2 đoạn $[p \dots n]$ và $[1 \dots q]$.

Với đoạn được chọn – dễ dàng dẫn xuất lời giải cần tìm.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "tours."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    string s;
    fi >> s;
    int n = s.size();
    vector<int> v[2];
    for (int i = 0; i < n; i++)
    {
        v[s[i] - '0'].push_back(i);
    }
    int a1 = 0, a2 = -1, b1 = 0, b2 = -1;
    for (int i = 0; i < 2; i++)
    {
        if (v[i].size() > 1)
        {
            if (v[i][v[i].size() - 1] - v[i][0] > a2 - a1 + 1)
            {
                a1 = v[i][0];
                a2 = v[i][v[i].size() - 1] - 1;
                b1 = v[i][0] + 1;
                b2 = v[i][v[i].size() - 1];
            }
        }
    }
    fo << a1 + 1 << ' ' << a2 + 1 << ' ' << b1 + 1 << ' ' << b2 + 1;
    Times;
    return 0;
}
```



VW34. DẪN ĐƯỜNG

Tên chương trình: AUTOGUIDE.CPP

Ngày nay người lái xe thường sử dụng hệ thống chỉ đường tự động để đến các nơi mình không quen thuộc. Nhưng hệ thống chung không đủ chi tiết cho từng vùng cụ thể. Vì vậy người ta tạo các hệ thống dẫn đường chi tiết riêng cho từng vùng và lắp thiết bị phát dọc theo đường trực giao thông.

Đường trực giao thông có độ dài n km. Có m cột mốc thiết bị phát, cột thứ i ở vị trí cột cây số a_i , $i = 1 \dots m$. Mỗi thiết bị có bán kính hoạt động k , tức là nếu thiết bị đặt ở cột cây số x thì các xe đang ở trên đoạn đường từ cột cây số $x-k$ cho đến cột cây số $x+k$ đều nhận được sóng.

Với cách lắp đặt thiết bị như vậy có thể có những đoạn đường chưa được phủ sóng.

Hãy xác định cần đặt thêm bao nhiêu thiết bị dẫn đường nữa để toàn bộ đường được phủ sóng.

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

- ✚ Dòng đầu tiên chứa 3 số nguyên n , m và k ($1 \leq n \leq 10^5$, $1 \leq m, k \leq n$),
- ✚ Dòng thứ 2 chứa m số nguyên a_1, a_2, \dots, a_m ($0 \leq a_i \leq n$, $a_i < a_j$ với $i < j$).

Kết quả: Đưa ra file văn bản AUTOGUIDE.OUT một số nguyên – số lượng thiết bị cần lắp thêm.

Ví dụ:

AUTOGUIDE.INP
26 3 3
3 19 26

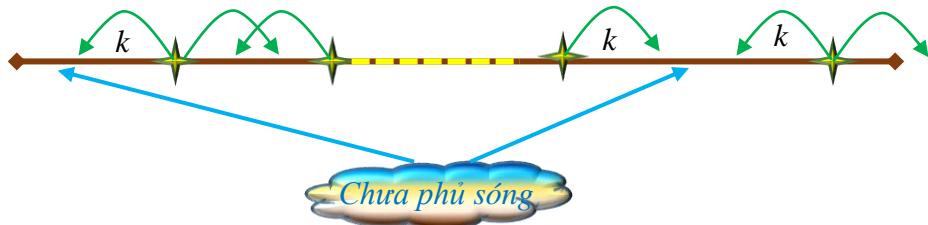
AUTOGUIDE.OUT
3



VW34 Mc20171216 A AXII

Giải thuật; Kiến thức cơ sở.

Cần lưu ý xử lý riêng các điểm đầu/cuối: thiết bị đầu tiên chỉ phủ được điểm đầu khi $a_1 \geq k$, thiết bị cuối cùng chỉ phủ nốt phần còn lại khi $n - a_m \leq k$,



Phần còn lại: tồn tại đoạn chưa phủ sóng nếu $a_i - a_{i-1} > 2 \times k$.

Mỗi trạm phát thêm vào sẽ phủ sóng đoạn đường độ dài $2 \times k$.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "autoguide."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    int n,m,k,ans=0;
    fi>>n>>m>>k;
    int k2=2*k;
    vector<int>a(m);
    for(int i=0;i<m;++i) fi>>a[i];
    if(a[0]-k>0) ans+=(a[0]-k+k2-1)/k2;
    if(n-a[m-1]>k) ans+=(n-a[m-1]+k-1)/k2;
    for(int i=1; i<m; ++i)
        if(a[i]-a[i-1]>k2) ans+=(a[i]-a[i-1]-1)/k2;
    fo<<ans;
    Times;
}
```



Steve có nhiệm vụ xây dựng mô đun phân tích cú pháp xử lý file chứa các phép gán. File có thể chứa các khối. Các khối có thể rời nhau hoặc lồng nhau. Mỗi khối bắt đầu bằng dòng chứa ký tự '{' và kết thúc bằng dòng chứa ký tự '}'. Toàn bộ chương trình được coi như một khối.

Phép gán có dạng **<variable>=<taget>**, trong đó **<variable>** là tên biến. Tên biến có độ dài không quá 10 và chỉ chứa các ký tự la tinh thường. **<taget>** có thể là số nguyên với giá trị tuyệt đối không quá 10^9 hoặc tên biến. Ban đầu các biến nhận giá trị 0.

Chương trình xử lý theo từng dòng, mỗi dòng chứa một phép gán hoặc dấu hiệu đầu khối hay cuối khối. Biến xuất hiện lần đầu ở về trái phép gán trong chương trình ở khối nào chỉ tồn tại trong khối đó và các khối lồng trong nó. Ra khỏi một khối biến cùng tên lại khôi phục giá trị trước khi vào khối.

Cho file thỏa mãn các điều kiện trên. Với mỗi phép gán hãy đưa ra giá trị được gán.

Dữ liệu: Vào từ file văn bản PARSER.INP chứa các dòng nội dung của file cần xử lý. Không tồn tại dấu cách trong file.

Kết quả: Đưa ra file văn bản PARSER.OUT các giá trị được gán, mỗi giá trị trên một dòng.

Ví dụ:

PARSER.INP	PARSER.OUT
a=b	
b=123	0
var=b	123
b=-34	-34
{	1000000000
c=b	1000000000
b=1000000000	123
d=b	-34
{	
a=b	
e=var	
}	
}	
b=b	



Giải thuật: Tổ chức quản lý dữ liệu.

Mỗi biến được lưu trữ dưới dạng cặp dữ liệu **<đại lượng, giá trị>** và được quản lý bằng **map<string, int>**,

Mỗi khối có một bản đồ giá trị riêng để khôi phục giá trị biến khi ra khỏi một khái niệm. Ngoài ra, còn cần có một bản đồ riêng lưu trữ giá trị hiện tại của các biến.

Có 2 phép xử lý chính:

- ⊕ Xử lý cuối khái niệm: khi ra khỏi một khái niệm,
- ⊕ Xử lý phép gán.

Xử lý cuối khái niệm:

- ⊕ Khôi phục giá trị trước khi vào khái niệm cho các biến,
- ⊕ Xóa bản đồ giá trị của các khái niệm vừa thoát ra.

Xử lý phép gán:

- ⊕ Tách tên biến ở bên trái phép gán,
- ⊕ Tính giá trị tương ứng với bên phải phép gán,
- ⊕ Đưa ra kết quả và cập nhật bản đồ ghi nhận giá trị.

Tổ chức dữ liệu:

Bản đồ **map<string, int>** ma – lưu giá trị hiện tại của các biến tương ứng với thời điểm xử lý,

Các bản đồ **map<string, int>** ma2 [(int) 1e5 + 10] – lưu giá trị biến ở các khái niệm ngoài.

Xử lý:

Xử lý kết thúc khái niệm: biến **b** quản lý độ lồng của khái niệm.



Xác định biến được gán giá trị:

```

int i=s.find('=') ;
string t=s.substr(0,i);
if (ma.count(t) == 0) ma[t] = 0;

```

Tính giá trị hàng số về phải:

```
int64_t a;
int64_t qw = 1;
if (s[i] == '-') {qw = -1; i++;}
if ('0' <= s[i] && s[i] <= '9')
{
    a = 0;
    for (; i < s.size(); i++)
        a = 10 * a + (s[i] - '0');
    a *= qw;
}
```

Tách tên biến về phải và xác định giá trị:

```
string q = s.substr(i, s.size() - i);
a = ma[q];
```

Dộ phức tạp của giải thuật: O(nlnn).

Chương trình

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

map<string, int> ma, ma2[(int)1e5 + 10];

int main()
{
    string s;
    int b = 0;
    while (fi >> s)
    {
        if (s[0] == '{') b++;
        else if (s[0] == '}')
        {
            for (auto p: ma2[b]) ma[p.first] = p.second;
            ma2[b].clear();
            b--;
        }
        else
        {
            int i=s.find('=');
            string t=s.substr(0,i);
            if (ma.count(t) == 0) ma[t] = 0;
            i++;
            int64_t a;
            int64_t qw = 1;
            if (s[i] == '-') {qw = -1; i++;}
            if ('0' <= s[i] && s[i] <= '9')
            {
                a = 0;
                for (; i<s.size(); i++)
                    a = 10 * a + (s[i] - '0');
                a *= qw;
            }
            else
            {
                string q = s.substr(i,s.size()-i);
                a = ma[q];
                fo << a << '\n';
            }
            if (ma2[b].count(t) == 0) ma2[b][t] = ma[t];
            ma[t] = a;
        }
    }

    cerr << "Time: " << clock() / (double)1000 << endl;
    return 0;
}
```



VW36. CÙNG CÓ GIÁ TRỊ LỚN NHẤT

Tên chương trình: EQ_MAX.CPP

“Văn ôn, võ luyện” là cảm nang dẫn đến thành công trong học tập. Steve hiểu rất rõ điều đó và không bao giờ chênh mảng trong ôn tập, rèn luyện kỹ năng.

Bài ôn tập hôm nay là xây dựng cây quản lý đoạn để trả lời các truy vấn “Tìm \max trong đoạn từ phần tử i đến phần tử j của dãy số nguyên dương a_1, a_2, \dots, a_n ”, tức là yêu cầu dẫn xuất $\max\{a_i, a_{i+1}, \dots, a_j\}$. Dĩ nhiên, Steve giải quyết vấn đề rất nhanh và hết sức hiệu quả.

Tuy vậy, trong quá trình hiệu chỉnh, chạy thử nghiệm với các truy vấn khác nhau Steve nhận thấy tồn tại rất nhiều đoạn có cùng giá trị \max như nhau. Thực tế đó đã làm nảy sinh một vấn đề hết sức tự nhiên: có bao nhiêu cách chọn 2 đoạn khác nhau không giao nhau cùng chứa giá trị \max như nhau. Nói một cách khác, có bao nhiêu bộ 4 số nguyên i, j, k, m thỏa mãn các điều kiện $1 \leq i \leq j < k \leq m \leq n$ và $\max\{a_i, a_{i+1}, \dots, a_j\} = \max\{a_k, a_{k+1}, \dots, a_m\}$.

Hãy xác định số cách chọn 2 đoạn không giao nhau thỏa mãn điều kiện đã nêu và đưa ra theo mô đun $10^9 + 7$.

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

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

Kết quả: Đưa ra file văn bản EQ_MAX.OUT một số nguyên – số cách có thể chọn (theo mô đun $10^9 + 7$).

Ví dụ:

EQ_MAX.INP	EQ_MAX.OUT
6	16
3 3 4 4 3 2	

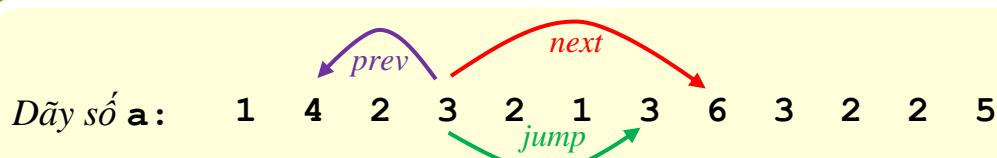


Giải thuật: Phân tích cấu hình, Tổ chức dữ liệu.

Với mỗi a_i ($i = 0, 1, \dots, n-1$) cần biết 3 thông tin:

- + prev_i – vị trí phần tử gần nhất bên trái lớn hơn a_i , $\text{prev}_i = -1$ nếu a_i lớn hơn tất cả các phần tử đứng bên trái,
- + next_i – vị trí phần tử gần nhất bên phải lớn hơn a_i , $\text{next}_i = n$ nếu a_i lớn hơn tất cả các phần tử đứng bên phải,
- + jump_i – vị trí phần tử gần nhất bên phải bằng a_i , $\text{jump}_i = -1$ nếu không có phần tử bên phải nào bằng a_i .

Ví dụ:



Việc tính prev_i và next_i được thực hiện bằng cách tổ chức *stack*, còn tính jump_i – bằng bản đồ dữ liệu ghi nhận vị trí xuất hiện lần đầu tiên của các số khác nhau trong dãy đã cho.

Với mỗi giá trị khác nhau $t = a_i$:

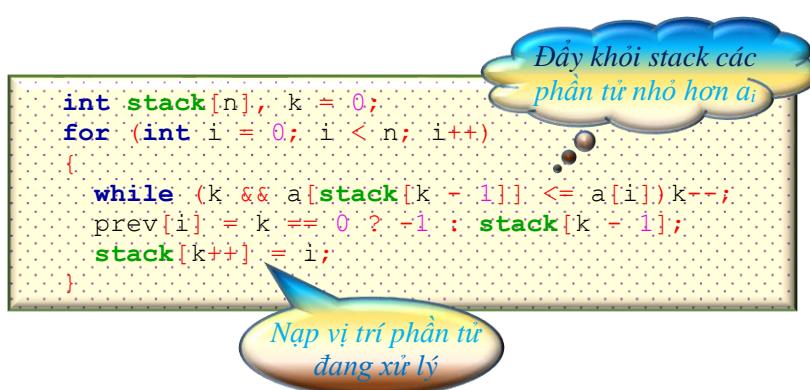
- + Xác định số lượng đoạn có *điểm cuối là i* và có *max* trong đoạn là t ,
- + Xác định số lượng đoạn có *điểm đầu lớn hơn i* và có *max* trong đoạn là t ,
- + Tích lũy vào kết quả tích của 2 giá trị nhận được.

Tổ chức dữ liệu:

- ─ Mảng `int a[n]` – lưu mảng dữ liệu vào,
- ─ Các mảng `int next[n], prev[n], jump[n]` – ghi nhận các móc nối đã nêu,
- ─ Mảng `int stack[n]` – phục vụ tạo móc nối các phần tử giống nhau,
- ─ Bản đồ `unordered_map <int, int> pos` – ghi nhận các giá trị khác nhau của dãy và vị trí đầu trong dãy của mỗi giá trị.

Xử lý:

Tính prev :



Tính **next**:

Duyệt từ cuối về đầu

```
k = 0;
for (int i = n - 1; i >= 0; i--)
{
    while (k && a[stack[k - 1]] <= a[i]) k--;
    next[i] = k == 0 ? n : stack[k - 1];
    stack[k++] = i;
}
```

Tính **jump**:

Duyệt từ cuối về đầu

```
unordered_map<int, int> pos;
for (int i = n - 1; i >= 0; i--)
{
    if (pos.count(a[i])) jump[i] = pos[a[i]];
    else jump[i] = -1;
    pos[a[i]] = i;
}
```

Ghi nhận mốc nói

*Ghi nhận vị trí mới
của a_i*

Tính số lượng cặp đoạn có cùng max :

Với mỗi i trong tập các giá trị khác nhau của dãy ban đầu xét các cặp đoạn có $max = t$, trong đó $t = a_i$,

Thông tin cần tích lũy từ 2 phần:

Số cặp đoạn mà cả 2 đều nằm trong khoảng từ $\text{prev}_{a[i]}$ đến $\text{next}_{a[i]}$,

Số cặp trong đó một đoạn nằm ngoài khoảng trên và thuộc phần đã duyệt qua.

Chương trình

```
#include "bits/stdc++.h"
#define NAME "eq_max."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int p = (int)1e9 + 7;

int mul ( int a, int b ){return ((int64_t)a * b) % p;}
int add ( int a, int b ){return (a + b)%p;}

int main ()
{
    int n;
    fi>>n;
    int a[n];
    for (int i = 0; i < n; i++) fi>>a[i];
    int next[n], prev[n], jump[n];
    int stack[n], k = 0;
    for (int i = 0; i < n; i++)
    {
        while (k && a[stack[k - 1]] <= a[i]) k--;
        prev[i] = k == 0 ? -1 : stack[k - 1];
        stack[k++] = i;
    }
    k = 0;
    for (int i = n - 1; i >= 0; i--)
    {
        while (k && a[stack[k - 1]] <= a[i]) k--;
        next[i] = k == 0 ? n : stack[k - 1];
        stack[k++] = i;
    }
    unordered_map <int, int> pos;
    for (int i = n - 1; i >= 0; i--)
    {
        if (pos.count (a[i])) jump[i] = pos[a[i]];
        else jump[i] = -1;
        pos[a[i]] = i;
    }

    int ans = 0;

    for (auto it: pos)
    {
        int sum = 0;
        for (int i = it.second; jump[i] != -1; i = jump[i])
        {
            int j = jump[i];
            if (next[i] >= j)
            {
                int t = i - prev[i];
                int tm=(int64_t) (j-i)*(j-i+1)/2%p;
                t = mul(t,tm);
                t = mul (t, next[j] - j);
                ans = add (ans, t);
            } else
            {
                int t = i - prev[i];
                t = mul (t, next[i] - i);
                t = mul (t, j - prev[j]);
                t = mul (t, next[j] - j);
            }
        }
    }
}
```

```

        ans = add (ans, t);
    }

    int r = sum;
    r = mul (r, j - max (i, prev[j]));
    r = mul (r, next[j] - j);
    ans = add (ans, r);
    int t = i - prev[i];
    t = mul (t, min (next[i], j) - i);
    sum = add (sum, t);
}
}

fo<<ans;
cerr << "Time: " << clock() / (double)1000 << endl;
return 0;
}

```



VW37. TỔNG FIBONACCI

Tên chương trình: *FIBSUM.CPP*

Dãy số Fibonacci được xác định bằng công thức:

$$f_0 = f_1 = 1,$$

$$f_{m+1} = f_m + f_{m-1}, m > 1.$$

Cho số nguyên n . Hãy xác định các cách biểu diễn n dưới dạng tổng các số, mỗi số hạng thuộc dãy số f_1, f_2, \dots , các số hạng giống nhau không gặp quá k lần.

Dữ liệu: Vào từ file văn bản FIBSUM.INP gồm một dòng chứa 2 số nguyên n và k ($1 \leq n \leq 100$, $1 \leq k \leq 20$).

Kết quả: Đưa ra file văn bản FIBSUM.OUT, mỗi dòng chứa một biểu thức tổng tìm được và không chứa dấu cách.

Ví dụ:

FIBSUM.INP	FIBSUM.OUT
6 2	1+1+2+2 1+2+3 1+5 3+3



VW37 Ok_Stp20171111 C_AXII

Giải thuật: Lập trình đệ quy.

Các số fibonacci tham gia tính tổng không nhiều, vì vậy có thể xác định trước,

Lần lượt sử dụng các số trong dãy Fibonacci từ nhỏ đến lớn,

Sử dụng bộ đếm cho số nhỏ nhất đang dùng,

Nếu số nhỏ nhất đang sử dụng là **x** thì lặp lại bài toán phân tích thành tổng cho số **n-x**, giữ nguyên số nhỏ nhất nếu số lần đã dùng nhỏ hơn **k** trong trường hợp ngược lại – tăng số nhỏ nhất.

Như vậy, tham số cho hàm đệ quy sẽ là:

- **n** – số cần phân tích ra tổng,
- **p** – số thứ tự của toán hạng sẽ xác định trong biểu thức,
- **last** – số nhỏ nhất trong dãy fibonacci đang sử dụng,
- **cnt** – số lần đã dùng số nhỏ nhất.

Khi **n = 0** – đưa ra kết quả phân tích.

Các toán hạng – lưu vào mảng.

Lời gợi ý ban đầu: **gen(n, 0, 0, 0)** ;

Độ phức tạp của giải thuật: O(n^2).

Chương trình

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

int n,k,fib[10] = {1,2,3,5,8,13,21,34,55,89};
vector<int> ans;

void gen(int n, int p, int last, int cnt)
{
    if (n == 0)
    {
        for (int i = 0; i < p; i++)
        {
            fo << ans[i];
            if (i < p - 1) fo << "+";
        }
        fo << '\n';
        return;
    }

    int f = last;
    if (cnt == k) f++;
    for (int i = f; fib[i] <= n; i++)
    {
        ans[p] = fib[i];
        gen(n - fib[i], p + 1, i, i == last ? cnt + 1 : 1);
    }
}

int main()
{
    fi >> n >> k;
    ans = vector<int>(n, 0);
    gen(n, 0, 0, 0);

    return 0;
}
```



VW38. BIỂU ĐỒ DICK

Tên chương trình: DICK_DIAG.CPP

Biểu đồ Dick là hình ảnh minh họa trực quan biểu thức ngoặc. Biểu đồ là đường đi liên tục, được tạo bởi các đường chéo ô vuông đơn vị, xuất phát từ gốc tọa độ. Ngoặc mở tương ứng với đường chéo đi lên từ trái sang phải, ngoặc đóng – đường chéo đi xuống từ trái sang phải.

Hãy xây dựng biểu đồ Dick cho biểu thức ngoặc đúng có độ dài không quá 100 trên lưới ô vuông có diện tích nhỏ nhất. Các ô trống – diền ký tự ‘.’, ô chứa đường đi lên – ký tự ‘/’, ô chứa đường đi xuống – ký tự ‘\’.

Dữ liệu: Vào từ file văn bản DICK_DIAG.INP gồm một dòng chứa xâu ký tự là biểu thức ngoặc đúng độ dài không quá 100.

Kết quả: Đưa ra file văn bản DICK_DIAG.OUT lưới ô vuông diện tích nhỏ nhất và biểu đồ Dick trên đó.

Ví dụ:

DICK_DIAG.INP
(()) ()

DICK_DIAG.OUT
. / \ . . . / . . \ / \



VW38 Ok_Stp20171111 B_AXII

Giải thuật: Cơ sở lập trình, xử lý xâu.

Độ dài biểu thức: **n**,

Chuẩn bị mảng **a[n][n]** chứa ký tự ‘.’,

Cần quản lý độ sâu **k** của ngoặc đang xét,

- Gặp ‘(‘ – tăng độ sâu lên 1,
- Gặp ‘)‘ – giảm độ sâu 1.

Khởi tạo: **k=-1**,

Xử lý ký tự **s[i]**:

$$s_i = \begin{cases} ') ' \rightarrow ++k; & a[k][i] = '/' ; \\ ')' ' \rightarrow a[k][i] = '\backslash' ; & --k; \end{cases}$$

Cần lưu **h** – độ sâu lớn nhất của biểu thức.

Đưa ra kết quả: đưa ra **a[i]** với **i = h ÷ 0**.

Độ phức tạp của giải thuật: O(n).



Chương trình

```
#include <bits/stdc++.h>
#define NAME "dick_diag."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,h=-1,k=-1,m=0;
string s,a[100],u,v="";
int main()
{
    fi>>s; n=s.size();
    for(int i=0;i<n;++i)v+= '.';
    for(int i=0;i<n;++i)a[i]=v;
    for(int i=0;i<n;++i)
    {
        if(s[i]== '(') a[++k][i]=' / ';
        else a[k--][i]=' \\ ';
        if(h<k) h=k;
    }
    for(int i=h;i>=0;--i) fo<<a[i]<<'\n';
}
```



VW39. SỮA NGÔ

Tên chương trình: JUICE.CPP

Tớp bạn đi du lịch khám phá vùng núi Tây Bắc đã thâm mệt thì thấy một quán nghỉ ven đường. Ở đó người ta bán sữa ngô, một thứ nước uống từ ngô non xay, rất thơm và ngon. Sữa ngô được bán trong các chai 1 lít. Có 2 loại chai: chai nhựa và chai thủy tinh.

Chai nhựa giá **a** đồng/chai, chai thủy tinh – giá **b** đồng/chai. Nếu bạn mua chai thủy tinh, uống xong, trả vỏ chai thì sẽ được nhận lại **c** đồng ($c < b$). Mọi người góp chung tiền lại được **n** đồng và đều muốn mua sao cho uống tại chỗ được nhiều nhất thứ nước giải khát độc đáo này.

Hãy xác định các bạn có thể mua được bao nhiêu chai.

Dữ liệu: Vào từ file văn bản JUICE.INP gồm một dòng chứa 4 số nguyên **n**, **a**, **b** và **c** ($1 \leq n, a, b \leq 10^{18}$, $1 \leq c < b \leq 10^{18}$).

Kết quả: Đưa ra file văn bản JUICE.OUT một số nguyên – số chai mua được.

Ví dụ:

JUICE.INP	JUICE.OUT
30 6 10 8	12



Giải thuật; Cơ sở lập trình.

Nếu $a \leq b - c \rightarrow$ chỉ mua chai nhựa,

Nếu $a > b - c$:

Giữ lại b đồng:

 Nếu số tiền còn lại không nhỏ hơn $b - c$: luôn luôn mua được chai thủy tinh,

 Số chai mua được: $(n - b) / (b - c)$,

Dùng số tiền còn lại: mua thêm một chai thủy tinh,

Với số tiền vẫn còn lại: mua chai nhựa.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "juice."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n,a,b,c,d,ans;

int main()
{
    fi>>n>>a>>b>>c;
    d=b-c;
    if(a<=d) ans=n/a;
    else
    {
        ans=(n-b)/d+1;
        n-=ans*d;
        ans+=n/a;
    }
    fo<<ans;
}
```



VW40. CHIA KẸO

Tên chương trình: CANDIES.CPP

Bob và Alice tới thăm một bệnh viện nhi, nơi đang có n em nhỏ điều trị. Hai bạn dự kiến mua kẹo chia đều cho các em. Mỗi người trong số hai bạn có thể mua ít nhất a chiếc kẹo và nhiều nhất – b chiếc. Nếu không thể chia đều được cho các em nhỏ thì cũng phải mua sao cho sau khi chia đều, số lượng kẹo còn dư là ít nhất.

Nếu không thể cùng mua một số lượng kẹo như nhau thì Bob sẽ mua phần nhiều hơn. Nếu vẫn còn có nhiều cách mua thì Alice sẽ mua nhiều nhất trong các phương án có thể.

Hãy xác định số lượng kẹo mỗi người cần mua.

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

Kết quả: Đưa ra file văn bản CANDIES.OUT trên một dòng 2 số nguyên xác định số kẹo Bob cần mua và số kẹo Alice cần mua.

Ví dụ:

CANDIES.INP
2 6 7

CANDIES.OUT
7 7



Giải thuật: Cơ sở lập trình, nhận dạng tình huống lô gic.

Số kẹo nhiều nhất mỗi em nhỏ có thể nhận được là $2 \times b/n$,

Trong trường hợp này, số kẹo cần có là $c = (2 \times b/n) \times n$.

Nếu c nhỏ hơn số kẹo tối thiểu 2 người mua thì mỗi người chỉ cần mua số lượng tối thiểu (là a),

Nếu một người mua tối đa, người kia mua tối thiểu vẫn chưa đủ c thì Bob cần mua b chiếc, phần còn lại – Alice mua,

Nếu c nằm giữa giới hạn một người mua tối đa, người kia mua tối thiểu thì Alice cần mua a chiếc, phần còn lại – Bob mua.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "candies."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n,a,b,c,x,y;

int main()
{
    fi>>n>>a>>b;
    c=(b<<1)/n*n;
    if(c<(a<<1))x=a, y=a;
    else
        if(c >= a+b)x=b, y=c-b;
        else x=c-a, y= a;

    fo<<x<<' ' <<y;
}
```



Alice đã đi đến bước cuối cùng trong một trò chơi trí tuệ truyền hình. Người ta giới thiệu n phần thưởng mà người chơi có thể chọn. Các phần thưởng được sắp thành một dãy, phần thưởng ở vị trí i có giá trị a_i , $i = 1 \div n$. Lần lượt duyệt từ trái sang phải theo chiều tăng dần của i , Alice có quyền bỏ qua một số phần thưởng và chọn những cái tùy ý, nhưng không được quay lại những cái đã bỏ qua cũng như không được thay đổi quyết định đưa ra trước đó, ngoài ra, phần thưởng chọn tiếp theo phải có giá trị lớn hơn giá trị phần thưởng vừa chọn trước đó.

Nếu tổng giải thưởng được chọn là lớn nhất trong số các cách chọn hợp lệ thì Alice sẽ nhận được mọi thứ đã chọn, nếu không – sẽ phải ra về trống tay.

Nhưng với trí nhớ tốt, trạng thái tâm lý vững vàng và khả năng tính toán tốt Alice đã nhận được phần thưởng của trò chơi.

Hãy xác định tổng giá trị phần thưởng mà Alice đã nhận được.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 1\,000$),
- ✚ 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 PRIZES.OUT một số nguyên – tổng giá trị phần thưởng mà Alice đã nhận được.

Ví dụ:

PRIZES.INP	PRIZES.OUT
5	11
4 2 3 6 6	



Giải thuật: Quy hoạch động.

Kích thước bài toán không lớn vì vậy có thể sử dụng giải thuật độ phức tạp $O(n^2)$ để đơn giản trong lập trình.

Gọi dp_i – tổng giá trị phần thưởng lớn nhất có thể đạt trong số i phần thưởng đầu tiên nếu chọn phần thưởng thứ i .

- $dp_i = a_i$,
- $dp_i = \max\{dp_i, dp_j + a_i\}$ với $j = 0 \div i, a_j < a_i$.

Cần tính dp_i với $i = 0 \div n-1$.

Kết quả cần tìm sẽ là $\max\{dp_i\}$, $i = 0 \div n-1$

dp_i cần phải khai báo với kiểu dữ liệu `int64_t`.

Độ phức tạp của giải thuật: $O(n^2)$.

Chương trình

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

int main()
{
    fi>>n;
    vector<int64_t>a(n);
    for(int i=0;i<n;++i) fi>>a[i];
    vector<int64_t>dp(a);
    for(int i=0;i<n;++i)
    {
        for(int j=0;j<i;++j)
            if(a[j]<a[i]) dp[i]=max(dp[i], dp[j]+a[i]);
        ans=max(ans, dp[i]);
    }
    fo<<ans;
    cerr<<"Time: "<<clock() / (double)1000<<" sec";
}
```



Trong tin học thường xuất hiện những từ mới bằng cách ghép tiền tố từ thứ nhất với hậu tố từ thứ hai, ví dụ, ***treap*** được hình thành từ ***tree*** và ***heap***.

Cho 2 từ ***s*** và ***t***. Hãy xác định có bao nhiêu từ mới khác nhau có thể tạo ra bằng cách ghép tiền tố khác rỗng của ***s*** với hậu tố khác rỗng của ***t***.

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

- ✚ Dòng đầu tiên chứa xâu ***s***,
- ✚ Dòng thứ hai chứa xâu ***t***.

Các xâu chỉ chứa ký hiệu la tinh thường và mỗi xâu có độ dài không vượt quá 10^5 .

Kết quả: Đưa ra file văn bản NEWWORDS.OUT một số nguyên – số lượng từ mới khác nhau có thể tạo ra.

Ví dụ:

NEWWORDS.INP	NEWWORDS.OUT
tree	
heap	14



VNU Ok_Shp20171111 F_AXII

Giải thuật: Đếm cấu hình.

Ký hiệu:

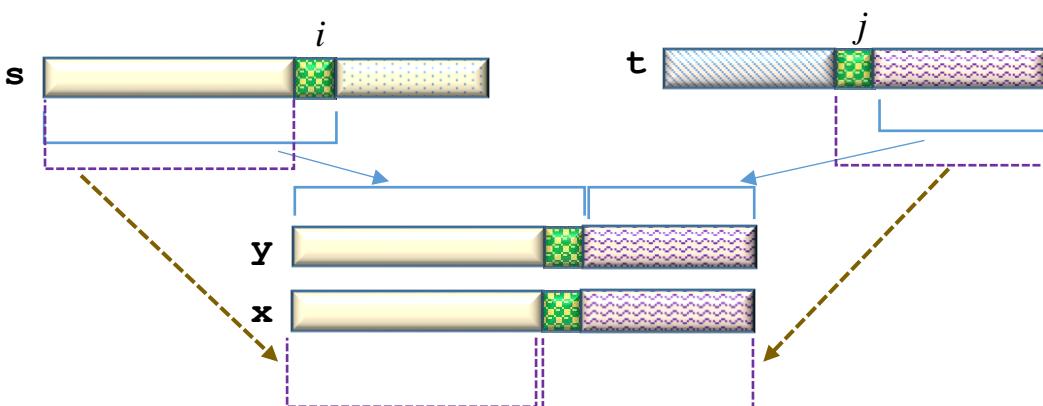
- ls , lt – độ dài các xâu s và t ,
- $s[0..i]$ – tiền tố xâu s kết thúc ở ký tự s_i ,
- $t[i..lt-1]$ – hậu tố xâu t bắt đầu từ ký tự t_i .

Nếu $s_i = t_j$ thì

$$x = s[0..i-1] + t[j..lt-1]$$

$$\text{và } y = s[0..i] + t[j+1..lt-1]$$

sẽ giống nhau.



Như vậy mỗi cặp ký tự giống nhau (s_i, t_j) sẽ làm xuất hiện một cặp xâu trùng nhau.

Do chỉ xét các tiền tố khác rỗng của s và các hậu tố khác rỗng của t nên các ký tự s_0 và t_{lt-1} không tham gia tạo xâu trùng.

Số lượng tất cả các xâu tạo thành từ việc ghép tiền tố khác rỗng của s và các hậu tố khác rỗng của t là $ls \times lt$,

Gọi $fr_{0,i}$ là tần số xuất hiện ký tự thứ i trong bảng chữ cái của s , $fr_{1,i}$ là tần số xuất hiện ký tự thứ i trong bảng chữ cái của t ,

Số lượng các xâu mới tạo ra trùng nhau sẽ là

$$\sum_{i=0}^{25} fr_{0,i} \times fr_{1,i}$$

Từ đây dễ dàng tính ra số lượng từ mới khác nhau.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "newwords."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
string s,t;
int ls,lt;
int64_t ans;

int main()
{
    fi>>s>>t;
    ls=s.size(); lt=t.size();
    vector<int> fr[2];
    fr[0].resize(26);
    fr[1].resize(26);
    for (int i = 1; i < ls; i++) fr[0][s[i]-97]++;
    for (int i = 0; i < lt-1; i++) fr[1][t[i]-97]++;
    ans=(int64_t)ls*lt;
    for (int i = 0; i < 26; i++)
        ans -= (int64_t)fr[0][i] * fr[1][i];

    fo<<ans;
    Times;
}
```



VW43. DÃY CON 0 1

Tên chương trình: SUB_01.CPP

Cho 2 dãy chỉ chứa các số 0 và 1, dãy **a** có độ dài **n**, dãy **b** – độ dài **m**.

Dãy con là dãy nhận được từ dãy ban đầu bằng cách loại bỏ một số (có thể là 0) phần tử.

Hãy xác định độ dài dãy con không giảm dài nhất là dãy con chung của **a** và **b**.

Dữ liệu: Vào từ file văn bản SUB_01.INP:

- ✚ Dòng đầu tiên chứa một số nguyên **n** ($1 \leq n \leq 2 \times 10^5$),
- ✚ Dòng thứ 2 chứa **n** số nguyên **a₁**, **a₂**, ..., **a_n** (**a_i** = 0 hoặc 1, **i** = $1 \div n$),
- ✚ Dòng thứ 3 chứa số nguyên **m** ($1 \leq m \leq 2 \times 10^5$),
- ✚ Dòng thứ 4 chứa **m** số nguyên **b₁**, **b₂**, ..., **b_m** (**b_i** = 0 hoặc 1, **i** = $1 \div m$).

Kết quả: Đưa ra file văn bản SUB_01.OUT một số nguyên – độ dài lớn nhất tìm được.

Ví dụ:

SUB_01.INP
7
0 0 0 1 0 1 1
6
0 0 1 1 0 1

SUB_01.OUT
5



VW43 Io20180121 A AXII

Giải thuật: Tổng hậu tố, Tìm kiếm nhị phân.

Dãy con cần tìm không giảm, vì vậy hoặc chỉ chứa 0, hoặc chỉ chứa 1 hoặc các phần tử đầu là 0, các phần tử còn lại là 1.

Xây dựng các mảng **sufa**, **sufb**, trong đó **sufa_i** là số lượng 1 trong hậu tố của **a** bắt đầu từ vị trí **i**, tương tự như vậy với **sufb_j**.

Đánh số các phần tử 0 trong **a** và **b** bắt đầu từ 1,

Xác định **besta_i** là số phần tử 1 đứng sau phần tử 0 thứ **i** trong dãy **a**,

Xác định **bestb_i** là số phần tử 1 đứng sau phần tử 0 thứ **i** trong dãy **b**.

besta₀ là số phần tử 1 trong toàn dãy **a**,

bestb₀ là số phần tử 1 trong toàn dãy **b**.

Dãy không giảm độ dài **M** với **z** phần tử 0 ở đầu ($0 \leq z \leq M$) nếu:

- **besta_z ≥ M-z** (Sau phần tử 0 thứ **z** trong **a** còn không ít hơn **M-z** phần tử 1),
- **bestb_z ≥ M-z** (Sau phần tử 0 thứ **z** trong **b** còn không ít hơn **M-z** phần tử 1).

Việc xác định **M** lớn nhất có thể thực hiện bằng phương pháp *tìm kiếm nhị phân*.

Tổ chức dữ liệu:

- Các mảng **vector<int>** **a(n)**, **vector<int>** **b(m)** – lưu dữ liệu vào,
- Các mảng **vector<int>** **sufa(n + 1)**, **vector<int>** **sufb(m + 1)** – lưu tổng số lượng 1 của hậu tố,
- Các mảng **vector<int>** **besta(n + 1, -1)**, **bestb(m + 1, -1)** – lưu tổng số lượng 1 sau mỗi phần tử 0.

Xử lý:

Xác định số lượng 1 trong các hậu tố: Với dãy **a** (và tương tự với dãy **b**)

```
vector<int> sufa(n + 1);
for (int i = n - 1; i >= 0; i--)
    sufa[i] = sufa[i + 1] + a[i];
```

Xác định số lượng 1 sau mỗi phần tử 0: Với dãy **a** (và tương tự với dãy **b**)

```
vector<int> besta(n + 1, -1), bestb(m + 1, -1);
int ca = 0;
besta[ca] = max(besta[ca], sufa[0]);
for (int i = 0; i < n; i++)
{
    if (a[i] == 0) ca++;
    besta[ca] = max(besta[ca], sufa[i + 1]);
```

Tìm kiếm nhị phân xác định kết quả:

```
int L = 0;
int R = min(n, m) + 1;
while (R - L > 1)
{
    int M = (L + R) / 2;
    bool ok = false;
    for (int z = 0; z <= M; z++)
    {
        if (besta[z] >= M - z && bestb[z] >= M - z)
        {
            ok = true;
            break;
        }
    }
    if (ok) L = M; else R = M;
}
```

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "sub_01."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    int n;
    fi >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) fi >> a[i];
    int m;
    fi >> m;
    vector<int> b(m);
    for (int i = 0; i < m; i++) fi >> b[i];
    vector<int> sufa(n + 1);
    for (int i = n - 1; i >= 0; i--)
        sufa[i] = sufa[i + 1] + a[i];
    vector<int> sufcb(m + 1);
    for (int i = m - 1; i >= 0; i--)
        sufcb[i] = sufcb[i + 1] + b[i];
    vector<int> besta(n + 1, -1), bestb(m + 1, -1);

    int ca = 0;
    besta[ca] = max(besta[ca], sufa[0]);
    for (int i = 0; i < n; i++)
    {
        if (a[i] == 0) ca++;
        besta[ca] = max(besta[ca], sufa[i + 1]);
    }
    int cb = 0;
    bestb[cb] = max(bestb[cb], sufcb[0]);
    for (int i = 0; i < m; i++)
    {
        if (b[i] == 0) cb++;
        bestb[cb] = max(bestb[cb], sufcb[i + 1]);
    }
    int L = 0;
    int R = min(n, m) + 1;
    while (R - L > 1)
    {
        int M = (L + R) / 2;
        bool ok = false;
        for (int z = 0; z <= M; z++)
        {
            if (besta[z] >= M - z && bestb[z] >= M - z)
            {
                ok = true;
                break;
            }
        }
        if (ok) L = M; else R = M;
    }

    fo << L << endl;
    Times;
    return 0;
}
```



Thời kỳ đầu của Internet địa chỉ của máy nối mạng có dạng 4 chữ số hệ 256. Hệ thống đánh địa chỉ này có tên là IPv4. Với sự phát triển bùng nổ của Internet, IPv4 không còn đáp ứng được nhu cầu đánh địa chỉ và người ta chuyển sang sử dụng hệ thống IPv6. Phòng thí nghiệm của giáo sư Braun đang nghiên cứu phát triển hệ thống đánh địa chỉ mới với tên IPvX. Địa chỉ của máy tính nối mạng cũng có dạng như ở IPv4 nhưng chứa tới x số hệ 256, các số ghi cách nhau một dấu chấm ('.') và có dạng $a_1.a_2 \dots a_x$, trong đó a_i nguyên và $0 \leq a_i \leq 255$, $i = 1 \dots x$. Các địa chỉ được sắp xếp theo thứ tự từ điển. Nói $a_1.a_2 \dots a_x$ nhỏ hơn $b_1.b_2 \dots b_x$ nếu tồn tại k ($0 \leq k < x$), sao cho $a_1 = b_1$, $a_2 = b_2$, ..., $a_k = b_k$ và $a_{k+1} < b_{k+1}$. Ngoài ra còn có khái niệm địa chỉ "tiếp sau" và "ngay trước" là địa chỉ nhỏ nhất lớn hơn địa chỉ đang xét và địa chỉ lớn nhất nhỏ hơn địa chỉ đang xét. Ví dụ, địa chỉ tiếp sau 0.123.123.123 là địa chỉ 0.123.123.124, với địa chỉ 0.123.123.255 tiếp sau là 0.123.124.0, với địa chỉ 0.255.255.255 tiếp sau là 1.0.0.0.

Hiện tại phòng thí nghiệm có n máy tính, máy thứ i có địa chỉ c_i (ở dạng đã nêu), các c_i khác nhau từng đôi một. Các công trình nghiên cứu của giáo sư Braun được đánh giá rất cao và vì vậy ông được tài trợ thêm 2 máy mới. Ở chế độ thử nghiệm, 2 máy này có địa chỉ là \mathbf{A} và \mathbf{B} ($\mathbf{A} < \mathbf{B}$). Rất hài lòng với chất lượng máy giáo sư Braun yêu cầu kết nối chúng với hệ thống máy hiện có sao cho địa chỉ của hai máy mới này phải liên tiếp nhau và nằm trong khoảng từ \mathbf{A} tới \mathbf{B} .

Quy tắc hệ thống xử lý gán địa chỉ cho máy mới là như sau: Ban đầu máy thứ nhất có địa chỉ \mathbf{A} , nếu địa chỉ này bận (đã có máy với địa chỉ này) hệ thống sẽ tìm địa chỉ tiếp sau cho tới khi tìm được địa chỉ trống đầu tiên. Với máy thứ 2 giải thuật cũng tương tự như vậy – bắt đầu từ \mathbf{B} , nhưng việc xác định bằng cách tìm kiếm địa chỉ ngay trước địa chỉ đang xét cho đến khi tìm được địa chỉ trống đầu tiên.

Người trợ lý của Giáo sư đã nghĩ ra phương pháp tạo địa chỉ bận ảo, tức là địa chỉ không gắn với máy thực nào nhưng bị coi là bận để phong tỏa một số địa chỉ trong quá trình xử lý tự động của hệ thống. Tuy nhiên, việc tạo địa chỉ ảo đã làm giảm đáng kể năng suất chung của hệ thống và giáo sư Braun rất không thích việc xuất hiện quá nhiều địa chỉ ảo, ông yêu cầu trợ lý phải phong tỏa ít nhất có thể các địa chỉ.

Hãy xác định số lượng ít nhất các địa ảo cần tạo. Nếu không có cách đánh lại địa chỉ các máy mới theo yêu cầu đã nêu thì đưa ra số -1.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và x ($0 \leq n \leq 10^5$, $1 \leq x \leq 8$),
- ✚ Dòng thứ i trong n dòng sau chứa địa chỉ c_i theo quy cách đã nêu,
- ✚ Dòng thứ $n+1$ chứa địa chỉ \mathbf{A} ,
- ✚ Dòng thứ $n+3$ chứa địa chỉ \mathbf{B} .

Kết quả: Đưa ra file văn bản IPVX.OUT một số nguyên – số lượng ít nhất các địa ảo cần tạo hoặc số -1 nếu không có cách đánh lại địa chỉ các máy mới.

Ví dụ:

IPVX.INP	IPVX.OUT
<pre>2 4 123.123.123.123 123.123.123.125 123.123.123.121 123.123.123.125</pre>	<pre>1</pre>  <p>VW44.Io20180121.B XII</p>

Giải thuật: Số nguyên 64 bít.

Địa chỉ có không quá 8 truwong, mỗi truwong 8 bít, vì vậy có thể biểu diễn địa chỉ dưới dạng số nguyên 64 bít không dấu (**uint64_t**):

$$\text{encode}(\mathbf{a}_1 \cdot \mathbf{a}_2 \dots \cdot \mathbf{a}_x) = \mathbf{a}_1 \times 256^{x-1} + \mathbf{a}_2 \times 256^{x-2} + \dots + \mathbf{a}_{x-1} \times 256 + \mathbf{a}_x$$

Xét đoạn $\text{encode}(\mathbf{A}) \dots \text{encode}(\mathbf{B})$.

Có thể đánh lại địa chỉ cho hai máy mới khi và chỉ khi trên đoạn này có 2 địa chỉ tự do liên tiếp nhau.

Để xác định điều đó cần:

- Sắp xếp các địa chỉ của các máy đã có theo chiều tăng dần,
- Xác định **between** – số địa chỉ nằm trong đoạn đang xét,
- Nếu tồn tại cách đánh lại địa chỉ thì kết quả cần tìm sẽ là

$$\text{encode}(\mathbf{A}) - \text{encode}(\mathbf{B}) - \mathbf{between} - 1$$

Tổ chức dữ liệu:

Mảng **vector<ull>** $a(n)$ – lưu địa chỉ các máy hiện có.

Xử lý:

Nhập địa chỉ dưới dạng **string**,

Chuyển đổi sang dạng **uint64_t**:

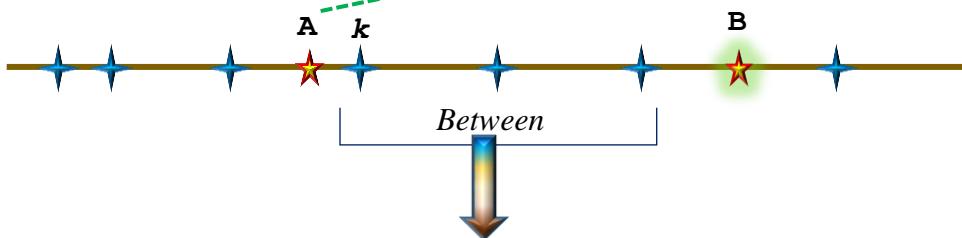
```
ull parse(string k)
{
    ull ans = 0;
    ull s = 0;
    for(int i=0; i<k.size(); ++i)
    {
        if(k[i] == '.')
        {
            ans = ans * 256ULL + s;
            s = 0;
            continue;
        }
        s = s * 10ULL + k[i] - '0';
    }
    ans = ans * 256ULL + s;
    s = 0;
    return ans;
}
```

Lưu ý ép kiểu dữ liệu

Xử lý một truwong

Xác định số thứ tự của địa chỉ đầu trong dãy năm trong đoạn đang xét:

```
int k = lower_bound(a.begin(), a.end(), st) - a.begin();
```



```
for (int j = k; j < n; j++)
{
    if (a[j] <= f)
    {
        ans = ans - 1;
        if (st < a[j] && st + 1 < a[j])
            ok = true;
        else
            if (j+1==n && ld(a[j])+ld(1)<ld(f)
                && ld(a[j])+ld(2)<=ld(f))
                ok = true;
            st = a[j] + 1;
    } else
    {
        if (st + 1 <= f) ok = true;
        break;
    }
}
```

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "ipvx."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

typedef long double ld;
typedef uint64_t ull;
ull n, x;
string s;
vector<ull> a;

ull parse(string k)
{
    ull ans = 0;
    ull s = 0;
    for(int i=0;i<k.size();++i)
    {
        if (k[i] == '.')
        {
            ans = ans * 256ULL + s;
            s = 0;
            continue;
        }
        s = s * 10ULL + k[i] - '0';
    }
    ans = ans * 256ULL + s;
    s = 0;
    return ans;
}

int main()
{
    fi >> n >> x;
    vector<ull> a(n);
    for(int i=0;i<n;++i)
    {
        fi >> s; a[i]=parse(s);
    }
    fi >> s;
    ull st = parse(s);
    fi >> s;
    ull f = parse(s);
    sort(a.begin(), a.end());

    int k = lower_bound(a.begin(), a.end(), st) - a.begin();
    bool ok = false;
    ull ans = f - st - 1;
    if (n == k && ld(st) + ld(1) <= ld(f))
        ok = true;
    for (int j = k; j < n; j++)
    {
        if (a[j] <= f)
        {
            ans = ans - 1;
            if (st < a[j] && st + 1 < a[j])
                ok = true;
            else
                if(j+1==n && ld(a[j])+ld(1)<ld(f) && ld(a[j])+ld(2)<=ld(f))

```

```
        ok = true;
        st = a[j] + 1;
    } else
    {
        if (st + 1 <= f) ok = true;
        break;
    }
}
if (ok) fo << ans; else fo << -1;
Times;
return 0;
}
```



VW45. VĂN TỰ CỎ

Tên chương trình: ANC_TXT.CPP

Các nhà khảo cổ đào được một chiếc khiên hình tròn. Chiếc khiên được viền bởi một dòng văn tự cỏ trong đó có không quá 26 loại ký tự khác nhau. Để tiện xử lý người ta đặt tương ứng mỗi loại ký tự bằng một chữ cái la tinh thường và đưa dòng văn tự vào máy tính dưới dạng xâu ký tự **s** với điểm đầu tùy chọn trên khiên.

Đặc điểm của nền văn minh đang khảo sát là sự sùng bái tính đối xứng. Văn tự này nhất thiết phải chứa một xâu palindrome nào đó chứa nội dung quan trọng.

Hãy xác định độ dài lớn nhất của xâu palindrome có thể chứa trong văn tự cỏ tìm thấy.

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

Kết quả: Đưa ra file văn bản ANC_TXT.OUT một số nguyên – độ dài lớn nhất của xâu palindrome có thể chứa trong văn tự.



Ví dụ:

ANC_TXT.INP	ANC_TXT.OUT
babaca	5



VW45_Io20180121_C_AXII

Giải thuật: Xử lý xâu vòng tròn, Giải thuật Manaker.

Gọi **1s** là độ dài xâu **s**,

Để xử lý xâu vòng tròn cần nhân đôi dữ liệu: **s2=s+s**;

Gọi **n** là độ dài xâu **s2**.

Mọi xâu con palindrome trong **s2** độ dài không quá **1s** cũng là xâu con palindrome trong xâu ban đầu.

Giải thuật Manaker với độ phức tạp $O(n)$ cho phép xác định **d1_i**, **d2_i** – số lượng xâu con palindrome tâm **i** độ dài lớn hơn 1 lẻ, chẵn, từ đó có thể xác định độ dài lớn nhất của xâu con palindrome tâm **i**.

Nếu **d1_i = k** thì xâu palindrome dài nhất tâm **i** sẽ là **2×k+1**,

Nếu **d2_i = k** thì xâu palindrome dài nhất tâm **i** sẽ là **2×k**.

Quá trình tìm **d1_i**, **d2_i** có thể kết thúc khi tìm được xâu con palindrome độ dài lớn hơn hoặc bằng **1s**.

Kết quả cần tìm sẽ là độ dài lớn nhất không vượt quá **1s** của các xâu con palindrome.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "anc_txt."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,ls,ans;
string s;
int calc_1()
{
    vector<int> d(n,0);
    int res=1;
    int l=0, r=-1;
    for(int i=0;i<n;++i)
    {
        int k=0;
        if(i<=r) k=min(r-i,d[r-i+1]);
        while(i+k+1<n && i-k-1>=0 && s[i+k+1] == s[i-k-1]) ++k;
        d[i]=k;
        if(k>res) res=k;
        if(res*2+1>=ls) break;
        if(i+k>r) l=i-k, r=i+k;
    }
    return res;
}
int calc_2()
{
    vector<int> d(n,0);
    int res=1;
    int l=0, r=-1;
    for(int i=0;i<n;++i)
    {
        int k=0;
        if(i<=r) k=min(r-i+1,d[r-i+1]);
        while(i+k<n && i-k-1>=0 && s[i+k] == s[i-k-1]) ++k;
        d[i]=k;
        if(k>res) res=k;
        if(res*2>=ls) break;
        if(i+k-1>r) l=i-k, r=i+k-1;
    }
    return res;
}
int main()
{
    fi>>s; ls=s.size(); s+=s; n=ls<<1;
    ans=2*calc_2();
    if(ans>ls) ans=(ans-ls+1)/2*2;
    if(ans<ls)
    {
        int r1=2*calc_1()+1;
        if(r1>ls) r1=(r1-ls+1)/2*2;
        if(ans<r1) ans=r1;
    }
    fo<<ans;
    Times;
}
```



Steve sắp tốt nghiệp và có dự định cùng nhóm bạn mở một công ty mới. Mỗi người sẽ đóng góp n đồng vào vốn đầu. Để có tiền góp cổ phần Steve quyết định bắt đầu từ ngày 1/1/2018 (ngày Thứ Hai) sẽ tích lũy tiền theo nguyên tắc sau: hàng ngày sẽ tiết kiệm chi tiêu, dành một số tiền chuyển vào quỹ tiết kiệm, thứ hai – góp vào quỹ x đồng, thứ ba – góp $x+k$ đồng, thứ tư – góp $x+2k$ đồng, . . . , chủ nhật – góp $x+6k$. Việc tiết kiệm sẽ kéo dài đến hết ngày 30/12/2018 để hôm sau cả nhóm sẽ tập hợp cổ phần để làm hồ sơ đăng ký kinh doanh.

Hãy xác định x và k để Steve có đúng n đồng đóng góp. Nếu có nhiều cách thực hiện khác nhau thì đưa ra x lớn nhất có thể với điều kiện $0 \leq x < 100$, $k > 0$.

Dữ liệu: Vào từ file văn bản STARTUP.INP gồm một dòng chứa số nguyên n ($1456 \leq n \leq 145600$). Dữ liệu đảm bảo bài toán có lời giải.

Kết quả: Đưa ra file văn bản STARTUP.OUT trên một dòng hai số nguyên x và k .

Ví dụ:

STARTUP.INP	STARTUP.OUT
1456	1 1



Giải thuật: Phương trình Diophantine.

Ngày thứ 2 tiết kiệm \mathbf{x} đồng,

Ngày thứ 3 tiết kiệm $\mathbf{x} + \mathbf{k}$,

Ngày thứ 4 tiết kiệm $\mathbf{x} + 2\mathbf{k}$,

Ngày thứ 5 tiết kiệm $\mathbf{x} + 3\mathbf{k}$,

Ngày thứ 6 tiết kiệm $\mathbf{x} + 4\mathbf{k}$,

Ngày thứ 7 tiết kiệm $\mathbf{x} + 5\mathbf{k}$,

Ngày chủ nhật tiết kiệm $\mathbf{x} + 6\mathbf{k}$,

Một tuần tiết kiệm được $7\mathbf{x} + (1+2+3+4+5+6)\mathbf{k} = 7\mathbf{x} + 21\mathbf{k}$.

Trung bình một ngày tiết kiệm $\mathbf{x} + 3\mathbf{k}$.

Một năm, trừ ngày cuối cùng, còn lại 364 ngày.

Dữ liệu đảm bảo bài toán có nghiệm, do đó \mathbf{n} chia hết cho 364.

Đặt $\mathbf{p} = \mathbf{n}/364$.

Có phương trình nghiệm nguyên $\mathbf{x} + 3\mathbf{k} = \mathbf{p}$ ($0 < \mathbf{x} < 100, 0 < \mathbf{k}$).

\mathbf{x} nhận giá trị lớn nhất khi \mathbf{k} nhỏ nhất $\rightarrow \mathbf{x} = \mathbf{p} - 3, \mathbf{k} = 1$.

Nếu $\mathbf{p} - 3 \geq 100$ thì cần chọn $\mathbf{x} = 99$ hay 98 hoặc 97 sao cho $\mathbf{p} - \mathbf{x}$ chia hết cho 3.

$\mathbf{k} = (\mathbf{p} - \mathbf{x})/3$.

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int main()
{
    fi>>n;
    int t=52*7,p,q;
    p=n/t;
    if(p-3<100) x=p-3, k=1;
    else
    {
        q=3-p%3;
        x=99-(3-q); k=(p-x)/3;
    }
    fo<<x<<' ' <<k;
}
```



VW47. CÔNG THỨC CHẾ BIẾN

Tên chương trình: RECEIPT.CPP

Alice dự định là một chiếc bánh đặc biệt chiêu đãi bạn bè nhân dịp họp mặt đầu năm. Quá trình chế biến gồm n công đoạn, ở công đoạn i cần cho phụ gia a_i , $i = 1 \dots n$. Các phụ gia được đánh số từ 1 đến m . Thật xui xẻo, nhà Alice được dọn dẹp gọn gàng sạch sẽ và công thức làm bánh bị thất lạc đâu đó. Việc bới lại đống giấy tờ để tìm công thức sẽ tốn rất nhiều thời gian và sức lực.

Alice có gắng hình dung những lần đã là trước đó và nhớ ra số lượng công đoạn n , số chất phụ gia m , dãy a_i là không giảm, tức là với $i < j$ có $a_i \leq a_j$ và nhớ dãy số b_1, b_2, \dots, b_m , trong đó b_i là số lượng các số trong dãy a nhỏ hơn hoặc bằng i .

Bob, bạn của Alice khẳng định có thể giúp Alice khôi phục lại công thức chế biến.

Hãy xác định công thức chế biến. Dữ liệu đảm bảo tồn tại công thức thỏa mãn.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n, m \leq 10^5$),
- ✚ Dòng thứ 2 chứa m số nguyên b_1, b_2, \dots, b_m ($0 \leq b_i \leq n$).

Kết quả: Đưa ra file văn bản RECEIPT.OUT trên một dòng n số nguyên xác định công thức cần khôi phục.

Ví dụ:

RECEIPT.INP	RECEIPT.OUT
3 2	1 2 2
1 3	



Giải thuật; Kỹ năng phân tích dữ liệu.

Dãy cần tìm là không giảm,

Hiệu $b_i - b_{i-1}$ là số lượng phần tử có giá trị i trong dãy cần tìm,

Duyệt tất cả các hiệu nêu trên với mọi i và đưa ra kết quả cần tìm.

Dãy kết quả là đơn trị.

Độ phức tạp của giải thuật: O(n).

Chương trình

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

int main()
{
    int n, m;
    fi >> n >> m;
    vector<int> b(m);
    for (int i = 0; i < m; i++)
        fi >> b[i];
    int prev = 0;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < b[i] - prev; j++)
            fo<<i+1<<' ';
        prev = b[i];
    }

    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}
```



VW48. KIM TỰ THÁP

Tên chương trình: PIRAMID.CPP

Các bạn trẻ trong Câu lạc bộ “Tuổi trẻ sáng tạo” đã chế tạo được một máy in 3D cho ra các cốc uống nước dùng một lần từ nguyên liệu tái chế. Khi uống nước xong, người ta chỉ việc bỏ cốc vào đầu vào của máy. Máy sẽ tự động rửa sạch, nghiền nát và cho ra chiếc cốc mới. Máy có thể tiếp nhận đồ vật bằng nhựa hoặc giấy bất kỳ để tạo ra cốc uống nước.

Máy được đưa ra triển lãm tại hội chợ công nghệ. Để tạo ấn tượng cho khách tham quan các bạn trẻ cho máy chạy liên tục và các cốc được in ra lắp thành một kim tự tháp có đáy hình chữ nhật kích thước $n \times m$ cốc. Trừ các cốc ở đáy, mỗi cốc còn lại được đặt trên đáy của 4 cốc ở dưới. Như vậy, tầng ngay trên đáy có kích thước $(n-1) \times (m-1)$ cốc, kích thước tầng tiếp theo sẽ là $(n-2) \times (m-2)$, ... Quá trình xếp sẽ tiếp tục cho đến khi không thể đặt cốc mới trên đáy của 4 cốc ở dưới.



Người xem rất thích thú và thường hỏi các bạn đã phải in ra bao nhiêu cốc để xây dựng được kim tự tháp này.

Hãy xác định số lượng cốc.

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

- ✚ Dòng đầu tiên chứa một số nguyên t – số lượng tests ($1 \leq t \leq 10^5$),
- ✚ Mỗi dòng trong t dòng sau chứa 2 số nguyên n và m ($1 \leq n, m \leq 10^9$).

Kết quả: Đưa ra file văn bản PIRAMID.OUT, kết quả mỗi test đưa ra trên một dòng dưới dạng số nguyên, xác định số cốc cần thiết để xây dựng kim tự tháp.

Ví dụ:

PIRAMID.INP	PIRAMID.OUT
3	1
1 1	14
2 5	14
3 3	



Giải thuật: Xử lý số lớn, tổ chức vào – ra dữ liệu.

Chuẩn hóa dữ liệu, đưa về trường hợp $n \leq m$,

Số lượng cốc cho một kim tự tháp sẽ là

$$\begin{aligned} & \sum_{i=0}^{n-1} (n-i) \times (m-i) \\ &= \sum_{i=0}^{n-1} (n-i) \times (m-n+n-i) \\ &= (m-n) \times \sum_{i=0}^{n-1} (n-i) + \sum_{i=0}^{n-1} (n-i)^2 \\ &= (m-n) \times n \times (n+1)/2 + n \times (n+1) \times (2n+1)/6 \\ &= n \times (n+1) \times (3m-n+1)/6 \end{aligned}$$

Cần xử lý số lớn để tính giá trị biểu thức trên.

Để tránh thực hiện chia số lớn: cần giản ước trước:

- ➡ Thừa số $n \times (n+1)$ chia hết cho 2 → giản ước,
- ➡ Một trong 2 thừa số $n \times (n+1)$ và $(3m-n+1)$ chia hết cho 3 → giản ước.

Kết quả cần tìm: không vượt quá 10^{27} , vì vậy để giảm thời gian tính: không cần xử lý tổng quát.

Lưu ý cách đưa ra kết quả:

Để chuyển số thành xâu: dùng cơ chế dãy xuất kết quả ra xâu (**ostringstream**),

Cần dãy xuất cả các số 0 không có nghĩa trong từng đoạn của số lớn,

Cần loại bỏ các số 0 không có nghĩa ở kết quả cuối cùng.

Độ phức tạp của giải thuật: O(t).

Chương trình

```
#include<bits/stdc++.h>
using namespace std;
ifstream fi ("piramid.inp");
ofstream fo ("piramid.out");
typedef uint64_t ull;
const ull base = (ull)1e9;
ull n,m,r0,r1,r2,t;
string s;

void mult(ull x,ull y)
{
    ull x0,x1,y0,y1,t; r2=0;
    x0=x%base; x1=x/base;
    y0=y%base; y1=y/base;
    t=x0*y0; r0=t%base; r1=t/base;
    t=x1*y0; r1+=t%base; r2=t/base+r1/base; r1%=base;
    t=x0*y1; r1+=t%base; r2+=t/base+r1/base; r1%=base;
    r2+=x1*y1;
}

int main()
{
    fi>>t;
    for(int i=0;i<t;++i)
    {
        fi>>n>>m;
        if(n>m) swap(n,m);
        ull a= n*(n+1)/2;
        ull b= 3*m-n+1;
        if(b%3==0) b/=3; else a/=3;
        mult(a,b);
        ostringstream res;
        res<<r2<<setw(9)<<setfill('0')<<r1<<setw(9)<<setfill('0')<<r0;
        s=res.str();
        int ls=s.size();
        while(ls>1 && s[0]=='0') {--ls; s.erase(0,1);}
        fo<<s<<endl;
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VW49. PHỦ ĐIỂM

Tên chương trình: COVER.CPP

Cho n điểm trên mặt phẳng, điểm thứ i có tọa độ (x_i, y_i) , $x_i \times y_i \neq 0$, $i = 1 \dots n$.

Người ta dùng các hình chữ nhật tâm ở gốc tọa độ để phủ lên các điểm đã cho. Một điểm gọi là được phủ nếu nó nằm bên trong hay trên biên của một hình chữ nhật trong số các hình chữ nhật được sử dụng.

Hãy xác định tổng nhỏ nhất diện tích các hình chữ nhật có thể chọn.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 5000$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên x_i và y_i ($0 \leq |x_i|, |y_i| \leq 5 \times 10^7$).

Kết quả: Đưa ra file văn bản COVER.OUT một số nguyên – tổng diện tích nhỏ nhất có thể chọn.

Ví dụ:

COVER.INP	COVER.OUT
3 -7 19 9 -30 25 10	2080



VW49 Cr6 20180203 C_AXII

Giải thuật: Stack, Quy hoạch động.

Bằng phép lấy đối xứng qua trục tung và trục hoành: Đưa các điểm đã cho về cung phần tư thứ nhất,

Bài toán ban đầu tương đương với bài toán phủ các điểm nhận được bằng các hình chữ nhật cạnh song song với trục tọa độ và có đỉnh dưới trái là $(0, 0)$.

Diện tích cần tìm sẽ là 4 lần diện tích tìm được ở bài toán tương đương.

Sắp xếp các điểm theo thứ tự tăng dần của x ,

Lọc dãy điểm đã cho để có dãy s các điểm (u_j, v_j) , trong đó u_j tăng dần và v_j giảm dần, $j = 1, 2, \dots, m$.

Việc lọc dữ liệu được thực hiện bằng stack.

Ta chỉ cần xét cách phủ các điểm (u_j, v_j) .

Trong phần lớn các trường hợp: $m < n$.

Gọi f_i – diện tích nhỏ nhất của các hình chữ nhật phủ i điểm đầu tiên của s .

$f_0 = 0$, $f_1 = u_1 \times v_1$,

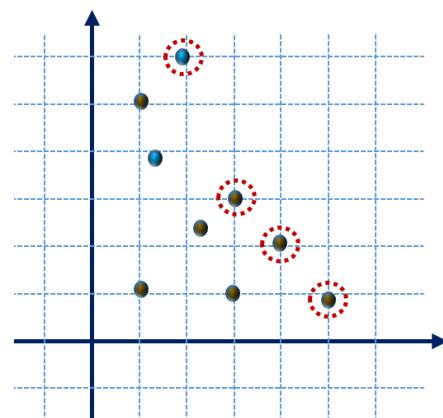
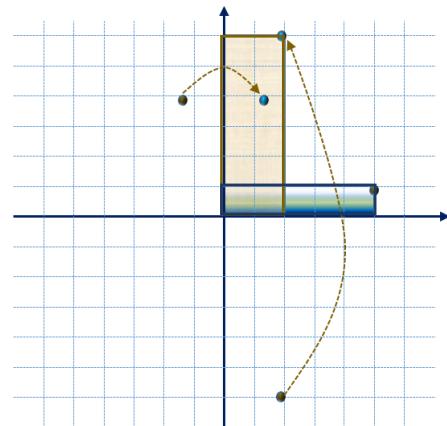
$f_i = \min\{f_{j-1} + v_j \times u_i, j = 1, 2, \dots, i\}, i = 2 \div m$.

Kết quả cần tìm: $4 \times f_m$.

Tổ chức dữ liệu:

- ▀ Mảng `pii p[5001]` – lưu tọa độ các điểm ban đầu,
- ▀ Mảng `pii s[5001]` – lưu $\{(u_j, v_j)\}$,
- ▀ Mảng `int64_t f[5001]` – lưu các tổng diện tích nhỏ nhất.

Độ phức tạp của giải thuật: $\min\{O(nlnn), m^2\}$.



Chương trình

```
#include <bits/stdc++.h>
#define NAME "cover."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
int n,x,y,h,t=0;
int64_t r,rx,f[5001],tr;
pii p[5001],s[5001];

int main()
{
    fi>>n;
    for(int i=0;i<n;++i)
    {
        fi>>x>>y;
        if(x<0) x=-x; if(y<0) y=-y;
        p[i]={x,y};
    }
    sort(p,p+n);
    s[0].second=1000000000;
    for(int i=0;i<n;++i)
    {
        h=p[i].second;
        while(h>=s[t].second) --t;
        s[++t]=p[i];
    }

    f[1]=(int64_t)s[1].first*s[1].second; f[0]=0;
    for(int i=2;i<=t;++i)
    {
        rx=s[i].first;
        r=f[i-1]+rx*s[i].second;
        for(int j=1; j<i;++j)
        {
            tr=f[j-1]+s[j].second*rx;
            if(r>tr) r=tr;
        }
        f[i]=r;
    }
    r=f[t]*4;
    fo<<r;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Việc chọn học sinh vào đội tuyển Tin học của trường được thực hiện dựa trên kết quả giải bài toán trên mạng. Mỗi bài toán có điểm số nguyên từ 0 đến 5. Điểm tổng kết được tính theo điểm trung bình của các bài đã nộp và làm tròn lên hoặc xuống đến số nguyên gần nhất. Nếu phần lẻ là 0.5 thì sẽ được làm tròn lên. Ví dụ, một người giải 3 bài với các điểm từng bài là 2, 3 và 5, điểm trung bình sẽ là $(2+3+5)/3 = 3.33$, điểm tổng kết sẽ là 3. Người khác làm được 4 bài với các điểm số 3, 3, 4 và 4, điểm trung bình sẽ là $(3+3+4+4)/4 = 3.5$ và điểm tổng kết sẽ là 4.

Có một bạn giải được **a** bài với điểm số mỗi bài là 2, **b** bài điểm 3 và **c** bài điểm 4. Bạn đó rất muốn vào đội tuyển và quyết định sẽ giải thêm một số bài nữa (ít nhất là thêm một bài) không khó lầm để nhận được thêm toàn điểm 5, sao cho điểm tổng kết của mình không dưới 4 – điều kiện tiên quyết được chọn vào đội tuyển.

Hãy xác định số lượng bài cần phải giải thêm.

Dữ liệu: Vào từ file văn bản UPTURN.INP gồm một dòng chứa 3 số nguyên **a**, **b** và **c** ($0 \leq a, b, c \leq 10^{15}$, $a+b+c \geq 1$).

Kết quả: Đưa ra file văn bản UPTURN.OUT một số nguyên – số bài ít nhất cần giải thêm.

Ví dụ:

UPTURN.INP	UPTURN.OUT
2 0 0	2



Giải thuật: Phân tích mô hình toán học.

Gọi số bài cần làm thêm là \mathbf{x} ($\mathbf{x} \geq 1$).

Theo yêu cầu ta có:

$$\frac{2 \times a + 3 \times b + 4 \times c + 5 \times x}{a + b + c + x} \geq 3.5$$

$$\Rightarrow 2 \times (2 \times a + 3 \times b + 4 \times c + 5 \times x) \geq 7 \times (a + b + c + x)$$

$$\Rightarrow 3 \times x \geq 3 \times a + b - c$$

$$\Rightarrow x \geq (3 \times a + b - c) / 3$$

Kết quả cần tìm là giá trị biểu thức bên phải làm tròn lên đến số nguyên gần nhất.

Dộ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "upturn."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t a,b,c,x;

int main()
{
    fi>>a>>b>>c;
    x=(3*a+b-c+2)/3;
    if(x<=0)x=1;
    fo<<x;
}
```



VX01. ĐỘ TRÙ MẬT

Tên chương trình: DENSITY.CPP

Trong một phòng nghiên cứu về lý thuyết số người ta tiến hành khảo sát sự phân bố các bình phương và lập phương của các số tự nhiên.

Cho số nguyên không âm k . Xét tập số nguyên thuộc đoạn $[a, b]$. Độ trù mật bậc k của đoạn này là số lượng cặp số x, y thuộc $[a, b]$ và thỏa mãn các điều kiện $a \leq x^2 \leq b$, $a \leq y^3 \leq b$ và $|x^2 - y^3| \leq k$. Ví dụ độ trù mật bậc 2 của $[1, 30]$ là 3:

- ✚ $x = 1, y = 1$ có $|x^2 - y^3| = |1 - 1| = 0$,
- ✚ $x = 3, y = 2$ có $|x^2 - y^3| = |9 - 8| = 1$,
- ✚ $x = 5, y = 3$ có $|x^2 - y^3| = |25 - 27| = 2$.

Với a, b, k cho trước, hãy xác định độ trù mật bậc k của $[a, b]$.

Dữ liệu: Vào từ file văn bản DENSITY.INP gồm một dòng chứa 3 số nguyên a, b, k ($1 \leq a \leq b \leq 10^{18}$, $0 \leq k \leq 10^{18}$).

Kết quả: Đưa ra file văn bản DENSITY.OUT một số nguyên – độ trù mật tìm được.

Ví dụ:

DENSITY.INP	DENSITY.OUT
2 0 0	2



VX01 RegSt20180127 2_AXII

Giải thuật; Kỹ thuật lọc dữ liệu và vét cạn.

Theo điều kiện đầu bài:

$$\mathbf{a} \leq \mathbf{y}^3 \leq \mathbf{b} \leq 10^{18}$$

Suy ra: $1 \leq \mathbf{y} \leq 10^6$.

Như vậy có thể duyệt tất cả các giá trị có thể của \mathbf{y} ,

Với mỗi \mathbf{y} thỏa mãn điều kiện $\mathbf{a} \leq \mathbf{y}^3 \leq \mathbf{b}$: giá trị \mathbf{x}^2 phải thỏa mãn điều kiện:

$$\mathbf{l_f} = \max(\mathbf{y}^3 - \mathbf{k}, \mathbf{a}) \leq \mathbf{x}^2 \leq \min(\mathbf{y}^3 + \mathbf{k}, \mathbf{b}) = \mathbf{r_t}$$

Từ đó có $\sqrt{\mathbf{l_f}} \leq \mathbf{x} \leq \sqrt{\mathbf{r_t}}$.

Ta cần tìm nghiệm nguyên, vì vậy biểu thức bên trái phải làm tròn lên, biểu thức bên phải – làm tròn xuống. Gọi các kết quả làm tròn nhận được là ℓ và r , với \mathbf{y} đang xét số lượng \mathbf{x} thỏa mãn sẽ là $(r - \ell + 1)$.

Vấn đề phức tạp là tính căn bậc 2 và làm tròn lên hoặc xuống. Cần phải duyệt các giá trị nguyên lân cận \sqrt{x} .

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "density."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t a,b,k;

int64_t msqrt(int64_t x)
{
    int64_t y = sqrt(x);
    while (y*y > x)
        --y;
    while ((y+1)*(y+1) <= x)
        ++y;
    return y;
}

int main()
{
    fi >> a >> b >> k;
    int64_t ans = 0;
    for (int64_t y = 1; y*y*y <= b; ++y)
    {
        int64_t yyy = y*y*y;
        if (yyy < a)
            continue;
        ans += max(0LL,msqrt(min(b, yyy+k)) - (msqrt(max(a, yyy-k)-1)+1)+1);
    }
    fo << ans << "\n";
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Thời cổ đá là vật liệu xây dựng chính. Tồn tại những công trường đại thủ công chuyên khai thác đá, đẽo thành các khối vuông hoặc chữ nhật. Số lượng đá làm ra được quản lý chặt chẽ. Cứ mỗi năm người ta ghi lại trên các tấm đất sét nung tổng số lượng đá đã làm được từ khi thành lập công trường cho đến hết năm đó.

Các nhà khảo cổ tìm được một số tấm và giải mã được chúng. Các tấm này liên quan tới thời kỳ n năm liên tục, hết năm thứ i tổng số lượng đá là được là p_i , $i = 1 \div n$. Tuy vậy có một số năm người ta không thấy tấm ghi chép tương ứng. Trong số ghi chép giá trị p_i tương ứng được ghi nhận là -1.

Người ta cũng xác định được mỗi năm đại công trường làm ra không ít hơn m khối.

Hãy xác định số lượng đá làm ra mỗi năm. Không loại trừ việc có một số tấm được giải mã không đúng do đó không thể sản lượng hàng năm. Trong trường hợp này lời giải chỉ chứa một số là -1.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n \leq 1000$, $0 \leq m \leq 100$),
- ✚ Dòng thứ 2 chứa n số nguyên p_1, p_2, \dots, p_n ($-1 \leq p_i \leq 1000$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản STONES.OUT trên một dòng n số nguyên tính được hoặc một số -1.

Ví dụ:

STONES.INP	STONES.OUT
<pre>3 1 1 2 3</pre>	<pre>1 1 1</pre>



Giải thuật: Tổng tiền tố.

Tìm phương án có thứ tự từ điển nhỏ nhất ($\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$).

Nếu không tồn tại dãy số tương ứng với thứ tự từ điển nhỏ nhất – bài toán vô nghiệm.

Giả thiết đã tìm được $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_{i-1}$.

$$\mathbf{a}_i = \begin{cases} \mathbf{m} & \text{nếu } \mathbf{p}_i = -1 \\ \mathbf{p}_i - \mathbf{a}_{i-1} & \text{nếu } \mathbf{p}_i \neq -1 \end{cases}$$

Nếu $\mathbf{a}_i < \mathbf{m}$ – bài toán vô nghiệm.

Cần tính tổng tiền tố ứng với các vị trí có $\mathbf{p}_i = -1$.

Dộ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "stones."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,m,p,q=0,f=0;

int main()
{
    fi>>n>>m;
    vector<int>a(n+1);
    for(int i=1;i<=n;++i)
    {
        fi>>p;
        if(p===-1) a[i]=m, f+=m;
        else
        {
            if(q>=0)
            {
                a[i]=p-f; if(a[i]<m) {a[0]=-1; break;} q=p; f=p; continue;
            }
            a[i]=p-f; f=p; if(a[i]<m) {a[0]=-1; break;}
        }
        q=p;
    }
    if(a[0]===-1) fo<<-1;
    else for(int i=1;i<=n;++i) fo<<a[i]<<' ';
}
```



VX03. ĐƠN ĐIỆU

Tên chương trình: MONOTONE.CPP

Cho xâu s chỉ chứa các ký tự la tinh thường độ dài n . Độ đơn điệu của của xâu con $s[p..q]$ các ký tự liên tiếp nhau của s từ vị trí p đến vị trí q (kể cả p và q) là số cặp (i, j) thỏa mãn các điều kiện:

- ✚ $p \leq i < j \leq q$,
- ✚ $s_i = 'a'$ và $s_j = 'b'$.

Với c cho trước hãy xác định độ dài lớn nhất của xâu con có độ đơn điệu không vượt quá c .

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và c ($1 \leq n \leq 10^6$, $0 \leq c \leq 10^{18}$),
- ✚ Dòng thứ 2 chứa xâu s độ dài n .

Kết quả: Đưa ra file văn bản MONOTONE.OUT một số nguyên – độ dài lớn nhất tìm được.

Ví dụ:

MONOTONE.INP	MONOTONE.OUT
6 2 aabcb	4



Giải thuật: Phương pháp 2 con trỏ.

Xét đoạn $s[i..j]$ ($i \leq j$):

Trong đoạn đang xét, gọi:

- ✚ a – số lượng ký tự ‘ a ’,
- ✚ b – số lượng ký tự ‘ b ’,
- ✚ ab – số lượng cặp ‘ ab ’.

Ban đầu i, j, a, b và ab đều nhận giá trị 0.

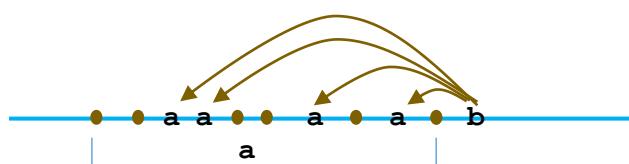
Chừng nào $s[i..j]$ có độ đơn điệu không vượt quá c ($ab \leq c$) – tăng j nếu $j < n$.

Nếu $ab > c$ – tăng i .

Mỗi lần thay đổi biên – cập nhật các tham số a, b, ab và nếu cần – cập nhật kết quả.

Khi tăng j :

- ✚ Nếu $s_j = 'a'$ $\rightarrow ++a,$
- ✚ Nếu $s_j = 'b'$ $\rightarrow ++b$ và $ab+=a,$



Khi tăng i :

- ✚ Nếu $s_i = 'b'$ $\rightarrow --b,$
- ✚ Nếu $s_i = 'a'$ $\rightarrow --a$ và $ab-=b.$

Trong quá trình xét: i và j *tăng dần*.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

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

int main()
{
    int n;
    int64_t c;
    string s;
    fi >> n >> c>>s;
    int j = 0;
    int64_t a = 0, b = 0, ab = 0;
    int ans = 0;

    for (int i = 0; i < n; i++)
    {
        while (j < n && ab <= c)
        {
            ans = max(ans, j - i);
            if (s[j] == 'a') a++;
            else if (s[j] == 'b') b++, ab += a;
            j++;
        }
        if (ab <= c) ans = max(ans, j - i);
        if (s[i] == 'b') b--;
        else if (s[i] == 'a') ab -= b, a--;
    }
    fo << ans << endl;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Câu lạc bộ Tin học có **a** bạn nữ và **b** bạn nam. Để chuẩn bị cho buổi liên hoan 8/3 sắp tới các bạn nam dự định làm một chiếc bánh độc đáo mang đậm phong cách tin học và đáp ứng tâm lý thích đồ ngọt của phái nữ: làm một chiếc bánh kem xốp dài $m \times (a+b)$ ô, trong đó **a** ô đầu tiên có phủ sô cô la, **b** ô tiếp theo – không phủ, **a** ô tiếp có phủ sô cô la, . . . Chiếc bánh sẽ được kéo dài ra bàn từ chiếc thùng bí ẩn, khi phân kéo ra có độ dài **a** thì cắt và trao cho nhóm bạn nữ, phần **b** ô tiếp theo – cắt chia cho nhóm nam, tiếp theo nữa – cắt chia cho nhóm nữ, . . .

Đánh tiếc, các bạn nam của chúng ta không phải là những thợ làm bánh khéo tay, sản phẩm thử nghiệm khi thì lớp xốp cứng như đá thạch anh, khi cháy đen như than mõ dùng để luyện cốc. Thời gian không còn nhiều, các bạn quyết định ra mua bánh làm sẵn ở cửa hàng về chỉnh lại.

Chiếc bánh mua về có độ dài **n** ô, một số ô có phủ sô cô la, một số ô khác – không phủ. Vấn đề tiếp theo là phải cắt một đoạn độ dài $m \times (a+b)$ ô liên tiếp và chỉnh lý cách phủ sô cô la. Không có dụng cụ phun sô cô la nên cách chỉnh duy nhất là hót phần sô cô la trên các ô không cần phủ.

Theo thói quen của những người làm Tin học, thay vì bắt tay vào làm việc ngay, các bạn của chúng ta lại lao vào tranh luận là có bao nhiêu cách cắt khác nhau. Hãy xác định có bao nhiêu cách cắt ra đoạn độ dài $m \times (a+b)$ ô liên tiếp để từ đó có thể chỉnh sửa thành cấu hình chờ đợi. Chiếc bánh khá dài nên không thể xoay ngược được kể cả sau khi cắt. Hai cách cắt gọi là khác nhau nếu tập các ô đưa ra sử dụng là khác nhau.

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

- ✚ Dòng đầu tiên chứa một số nguyên **n** ($1 \leq n \leq 10^6$),
- ✚ Dòng thứ 2 chứa xâu **s** độ dài **n**, ký tự thứ **i** bằng 1 nếu ô **i** có phủ sô cô la và bằng 0 trong trường hợp ngược lại.
- ✚ Dòng thứ 3 chứa 3 số nguyên dương **m**, **a** và **b** ($1 \leq m, a, b, m \times (a+b) \leq n$).

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

Ví dụ:

WAFERS.INP	WAFERS.OUT
10 1111111010 2 1 1	6



Giải thuật: Tổng tiền tố.

Để tính giá trị hàm mục tiêu trong một khoảng liên tục: phương pháp hiệu quả nhất là dùng tổng tiền tố (Prefix Sum),

Đầu tiên cần xây dựng hàm tổng tiền **ps** phục vụ quản lý các đoạn 1 liên tục:

$$ps_0 = 0, ps_i = ps_{i-1} + s_{i-1}, i = 1 \dots n.$$

Bước tiếp theo tính tổng tiền tố các đoạn $d = a+b$ ô có thể chỉnh sửa: dp_i là số lượng các đoạn liên tiếp có thể chỉnh sửa bắt đầu từ s_{i-1} .

$$dp_i = dp_{i+d} + 1 \text{ nếu } ps_{i+a-1} - ps_{i-1} = a, i = n-d+1 \dots 1.$$

Nếu $dp_i \geq m \rightarrow$ có thêm một cách cắt.

Tổ chức dữ liệu:

- ─ Xâu **s** – lưu dữ liệu vào,
- ─ Các mảng **int** **ps** [MX], **dp** [MX] = {0} – phục tính các tổng tiền tố.

Độ phức tạp của giải thuật: O(n).

Ghi chú:

- + Có thể tính dp_i bằng cách duyệt các vị trí của **s** từ đầu về cuối hoặc từ cuối về đầu,
- + Lưu ý cách tính các biên xác định vị trí duyệt.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "wafers."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
constexpr int MX = 1000002;

int ps[MX], dp[MX]={0};

int main()
{
    int n;
    fi >> n;
    string s;
    fi>> s; ps[0]=0;
    for (int i = 1; i <= n; i++) ps[i]=ps[i-1]+s[i-1]-48;
    int ans = 0;
    int m,a,b,d;
    fi>>m>>a>>b;
    d=a+b;
    for (int i=n-d+1; i>=1; i--)
    {
        if (ps[i+a-1]-ps[i-1]== a) dp[i]=dp[i+d]+1;
        if (dp[i] >= m) ans++;
    }

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



VX05. XỬ LÝ BIT

Tên chương trình: AND_XOR.CPP

Sau khi giảng về các phép tính **and** và **xor** xử lý bít thày giáo viết trên bảng dây số không âm $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$ và yêu cầu thực hiện các xử lý tiếp theo sẽ được thông báo. Mỗi phép xử lý tiếp theo có một trong 3 dạng:

- **xor x** – Thay \mathbf{a}_i bằng $\mathbf{a}_i \text{ xor } \mathbf{x}$ với mọi i , \mathbf{x} là số nguyên không âm,
- **and x** – Thay \mathbf{a}_i bằng $\mathbf{a}_i \text{ and } \mathbf{x}$ với mọi i , \mathbf{x} là số nguyên không âm,
- **? lf rt** – Đếm số lượng các phần tử của \mathbf{A} hiện tại thỏa mãn điều kiện $lf \leq \mathbf{a}_i \leq rt$.

x	y	$x \text{ and } y$	$x \text{ xor } y$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Với các phép xử lý dạng “?” hãy đưa ra số lượng tìm được.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và q – số lượng các thông báo ($1 \leq n \leq 10^5$, $0 \leq q \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$ ($0 \leq \mathbf{a}_i < 2^{20}$, $i = 1 \div n$),
- ✚ Mỗi dòng trong q dòng sau chứa một thông báo thuộc dạng đã nêu.

Kết quả: Đưa ra file văn bản AND_XOR.OUT, với thông báo dạng “?” đưa ra trên một dòng một số nguyên – số lượng tìm được.

Ví dụ:

AND_XOR.INP
6 8
1 2 3 4 5 6
and 2
xor 1
xor 4
? 2 5
and 4
? 2 5
and 1
? 2 5

AND_XOR.OUT
3
6
0



VX05 Io20180225 C AXII

Giải thuật: Xử lý bit, Rừng cây số, Cấu trúc dữ liệu.

Xây dựng rừng cây quản lý các số ở dạng nhị phân của dãy \mathbf{A} ,

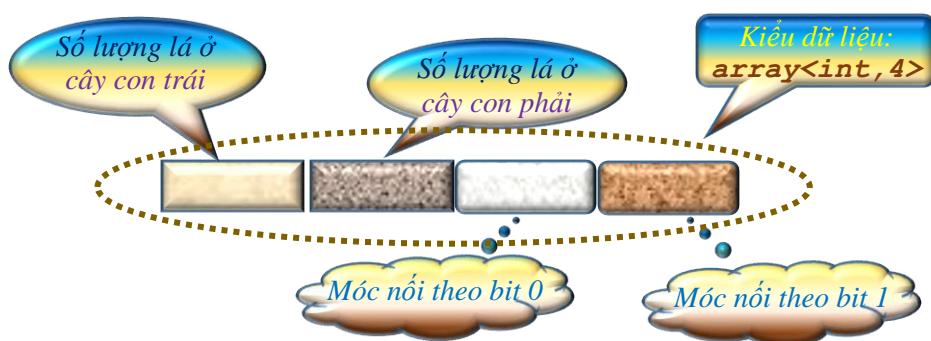
Trong trường hợp *quản lý theo bit*, rừng cây sẽ *suy biến thành cây nhị phân*.

Trong cây quản lý các số theo bits, mỗi số tương ứng với một đường đi từ gốc tới lá, nút gốc tương ứng với bít cao nhất, nút lá - ứng với bít thấp nhất,

Các số đã cho có không quá 20 bits vì vậy có thể xét mỗi số đã cho như một số có đúng 20 bits kể cả các số 0 không có nghĩa,

Mỗi số sẽ tương ứng với một đường đi độ dài 20 (qua 20 nút của cây).

Cấu trúc thông tin ở mỗi nút:

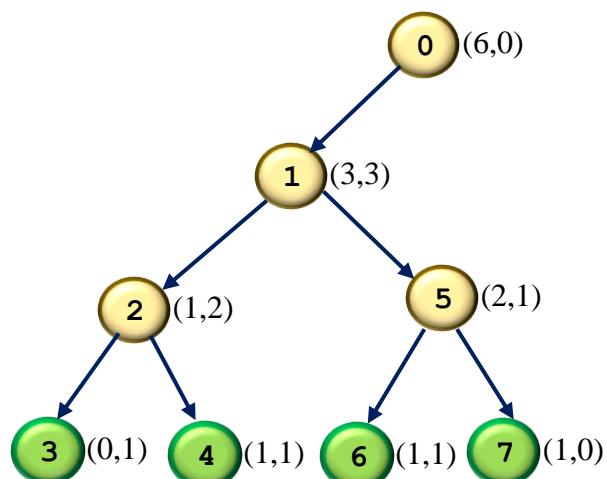


Móc nối có giá trị -1 nếu không có đường đi tiếp theo.

Ví dụ xét $n = 6$ và dãy $\mathbf{A} = (1, 2, 3, 4, 5, 6)$. Các a_i đều nhỏ hơn 2^4 , vì vậy để *đơn giản trong minh họa*, ta xét cây *các đường đi từ gốc tới lá có độ dài 4*.

Gọi \mathbf{B} là cây quản lý dãy \mathbf{A} , ta có:

i	b_{i0}	b_{i1}	b_{i2}	b_{i3}
0 : 6	0	1	-1	-1
1 : 3	3	2	5	
2 : 1	2	3	4	
3 : 0	1	-1	-1	
4 : 1	1	-1	-1	
5 : 2	1	6	7	
6 : 1	1	-1	-1	
7 : 1	0	-1	-1	



Với dãy số \mathbf{A} ban đầu ta có thể xây dựng cây độ dài mỗi nhánh 20 quản lý các giá trị a_i .

Phép xử lý **and** \mathbf{x} sẽ làm bít j trong tất cả các a_i đều bằng 0 nếu $x_j = 0$ và bắt đầu từ đó trở đi *bít j trong tất cả các a_i đều giống nhau*.

Gọi **fix** là mặt nạ các bít giống nhau. Bít $\text{fix}_j = 0$ nếu bít j trong tất cả các a_i giống nhau.

Ban đầu 20 bits thấp nhất của **fix** đều bằng 1.

Mặt nạ **fix** *thay đổi không quá 20 lần* và chỉ có thể thay đổi khi thực hiện phép **and x**.

Với mỗi truy vấn dạng **and x**:

- ⊕ Kiểm tra sự thay đổi của mặt nạ **fix**, nếu có thay đổi – xây dựng lại cây,
- ⊕ Trong mọi trường hợp: cập nhật mặt nạ **d** xác định việc đảo bít và hướng rẽ nhánh khi duyệt cây **B**.

Với mỗi truy vấn dạng **xor x**:

- ⊕ Ghi nhận mặt nạ **d** xác định việc đảo bít và hướng rẽ nhánh khi duyệt cây **B**,
- ⊕ Nếu $d_j = 1$ thì bít j trong các a_i bị đảo 0 thành 1 hoặc ngược lại,
- ⊕ Không cần xây dựng lại cây **B**, nếu $d_j = 1$ – thực hiện rẽ phải khi cần đi theo bít 0 và rẽ trái – khi đi theo bít 1.

Với mỗi truy vấn dạng **? lf rt**:

- ⊕ Kết quả truy vấn dựa trên hàm **calc(x)** xác số lượng phần tử trong **A** lớn hơn **x**,
- ⊕ Kết quả cần tìm sẽ là **calc(lf-1) - calc(rt)**.

Tính **calc(x)** – số lượng phần tử trong **A** lớn hơn **x**:

- ⊕ Gọi **t** là giá trị cần tính,
- ⊕ Ban đầu **t = 0**,
- ⊕ Duyệt cây **B** từ gốc tới lá (**p = 0**),
- ⊕ Ở bước thứ **i**: ký hiệu **v** bít thứ **i** của **d**, **u** – bít thứ **i** của **x**,

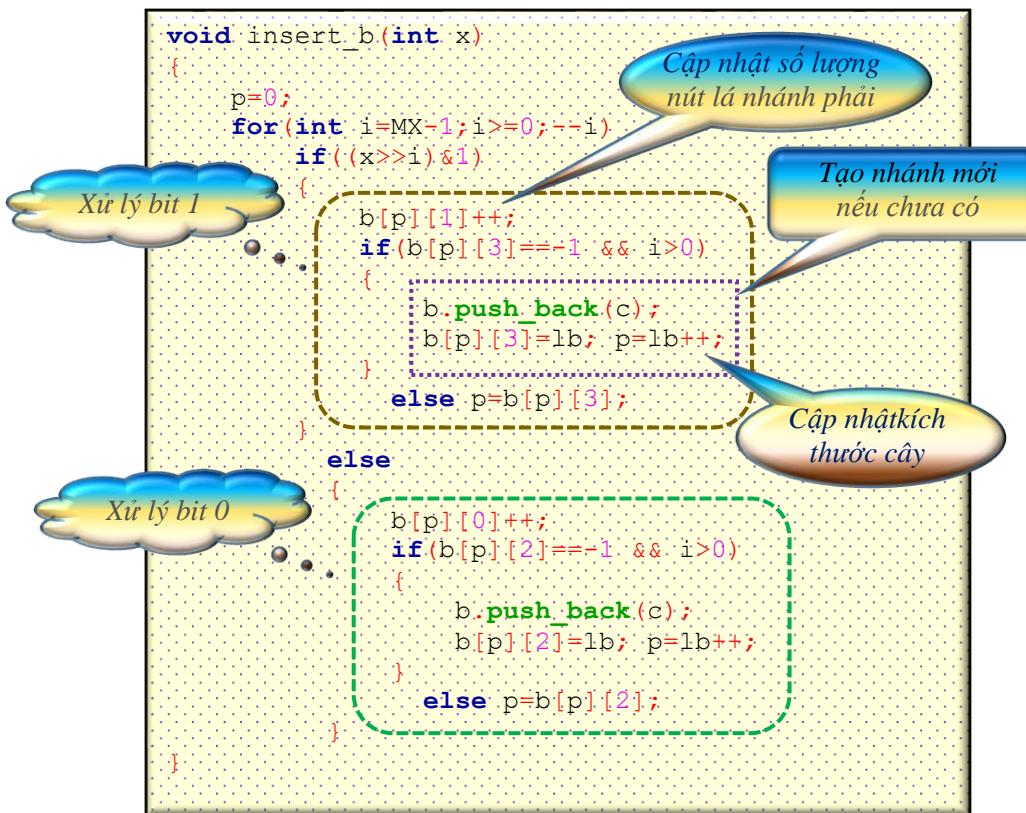
v	u	t	p
0	0	t+=b_{p,1}	p=b_{p,2}
0	1		p=b_{p,3}
1	0	t+=b_{p,0}	p=b_{p,3}
1	1		p=b_{p,2}

Tổ chức dữ liệu:

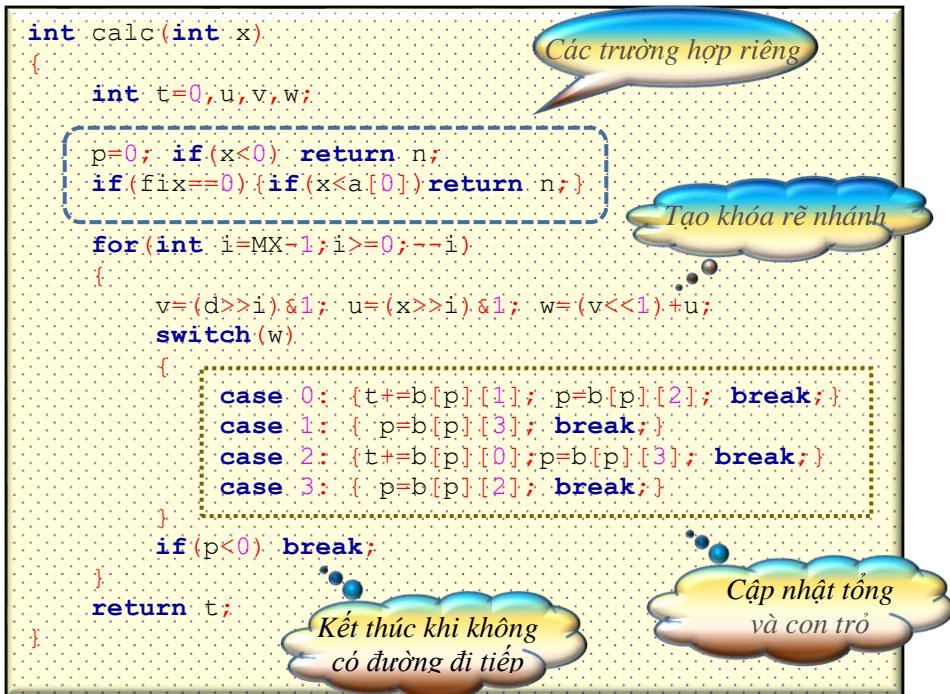
- ─ Mảng **int a[100000]** – lưu dãy số cần xử lý,
- ─ Mảng **vector<array<int, 4>>b** – lưu cây quản lý các số,
- ─ Biến **int fix** – lưu mặt nạ các bít mọi a_i có giá trị giống nhau,
- ─ Biến **int d** xác định việc đảo bít và hướng rẽ nhánh khi duyệt cây **B**.

Xử lý:

Nạp phần tử **x** vào cây – hàm **void insert_b(int x)**:



Tính số lượng:



Dộ phức tạp của giải thuật: Bậc O(nlnn).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "and_xor."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int MX=20;
int a[100000],mask=1024*1024-1;
vector<array<int,4>>b;
array<int,4> c={0,0,-1,-1};
int n,q,p,lb,fix,d=0,x,t,lf,rt;
string s;

void insert_b(int x)
{
    p=0;
    for(int i=MX-1;i>=0;--i)
        if((x>>i)&1)
    {
        b[p][1]++;
        if(b[p][3]==-1 && i>0)
        {
            b.push_back(c);
            b[p][3]=lb; p=lb++;
        }
        else p=b[p][3];
    }
    else
    {
        b[p][0]++;
        if(b[p][2]==-1 && i>0)
        {
            b.push_back(c);
            b[p][2]=lb; p=lb++;
        }
        else p=b[p][2];
    }
}

int calc(int x)
{
    int t=0,u,v,w;
    p=0; if(x<0) return n;
    if(fix==0){if(x<a[0]) return n;}
    for(int i=MX-1;i>=0;--i)
    {
        v=(d>>i)&1; u=(x>>i)&1; w=(v<<1)+u;
        switch(w)
        {
            case 0: {t+=b[p][1]; p=b[p][2]; break;}
            case 1: { p=b[p][3]; break;}
            case 2: {t+=b[p][0];p=b[p][3]; break;}
            case 3: { p=b[p][2]; break;}
        }
        if(p<0) break;
    }
    return t;
}

void init_b()
{
```

```

b.clear();
b.push_back(c); lb=1;
for(int i=0;i<n;++i) insert_b(a[i]);
}
int main()
{
fi>>n>>q;
for(int i=0;i<n;++i) fi>>a[i];
init_b(); fix=mask;
for(int iq=0;iq<q;++iq)
{
    fi>>s;
    if(s[0]=='a')
    {
        fi>>x;
        t=fix&x;
        d&=x;
        if(t!=fix)
        {
            fix=t;
            for(int i=0;i<n;++i) a[i]&=fix;
            init_b(); continue;
        } else continue;
    }
    if(s[0]=='x')
    {
        fi>>x;
        d^=x;
        continue;
    }
    fi>>lf>>rt;

    fo<<calc(lf-1)-calc(rt)<<'\n';
}
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```

Xử lý truy vấn and

Trường hợp mặt nạ
thay đổi



VX06. DÃY SỐ KHÔNG GIẢM

Tên chương trình: NON_DECR.CPP

Cho dãy số nguyên a_1, a_2, \dots, a_n . Với mỗi i có thể giữ nguyên số a_i hay thay nó bằng $-a_i$, $i = 1 \div n$.

Hãy xác định bằng quy tắc biến đổi trên có thể nhận được dãy số không giảm hay không? Nếu không thể nhận được dãy không giảm – đưa ra thông báo “**No**”, trong trường hợp ngược lại – đưa ra thông báo “**Yes**” và ở dòng thứ hai – một dãy số không giảm có thể nhận được.

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

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

Kết quả: Đưa ra file văn bản NON_DECR.OUT kết quả nhận được theo quy cách đã nêu.

Ví dụ:

NON_DECR.INP	NON_DECR.OUT
5	Yes
1 -1 -2 3 6	-1 -1 2 3 6



VX06.la20180319.A_AXII

Giải thuật: Phân tích tình huống lô gic.

Xây dựng dãy không giảm có thứ tự từ điển nhỏ nhất với độ dài lần lượt bằng 1, bằng 2, ...

Nếu xây dựng được dãy không giảm độ dài n – bài toán có nghiệm,

Nếu ở bước thứ i không tìm được phần tử phù hợp khi kiểm tra cả 2 khả năng lựa chọn a_i và $-a_i$ – kết quả là “**No**”,

Ở mỗi bước – giữ lại giá trị nhỏ nhất có thể.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

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

int main()
{
    fi>>n;
    vector<int>a(n);

    for(int i=0;i<n;++i) fi>>a[i];
    if(a[0]>0)a[0]=-a[0];
    for(int i=1;i<n;++i)
    {
        if(a[i]<a[i-1] && -a[i]<a[i-1]) {ans="No"; break;}
        if(-abs(a[i])>=a[i-1]) {a[i]=-abs(a[i]); continue;}
        a[i]=abs(a[i]);
    }

    fo<<ans<<'\n';
    if(ans=="Yes")
        for(int i=0;i<n;++i) fo<<a[i]<<' ';
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VX07. XÓA SỐ

Tên chương trình: ABANDON.CPP

Người ta viết các số từ 1 đến n thành một dãy số tăng dần. Với một số k cho trước, ở mỗi bước người ta xóa khỏi dãy các số ở vị trí $k, 2k, 3k, \dots$. Quá trình trên chấm dứt khi số lượng số trong dãy nhỏ hơn k .

Ví dụ, với $n = 13$ và $k = 2$, ở bước 1 các số 2, 4, 6, 8, 10 và 12 bị xóa và ta có dãy số 1, 3, 5, 7, 9, 11, 13. Ở bước 2 các số 3, 7, 9, 11 bị xóa và dãy số còn lại sẽ là 1, 5, 11, 13. Ở bước 3 các số 5 và 13 bị xóa, dãy còn lại sẽ là 1, 11. Lần cuối cùng – số 11 bị xóa. Trong ví dụ này, số 13 bị xóa ở bước 3.

Hãy xác định số n bị xóa ở bước nào.

Dữ liệu: Vào từ file văn bản ABANDON.INP gồm một dòng chứa 2 số nguyên n và k ($3 \leq n \leq 10^{18}$, $2 \leq k \leq 100$ và $k < n$).

Kết quả: Đưa ra file văn bản ABANDON.OUT một số nguyên xác định bước tìm được.

Ví dụ:

ABANDON.INP
13 2

ABANDON.OUT
3



Giải thuật: Sơ đồ lắp.

Gọi **p** là vị trí số **n** trong dãy,

Ban đầu **p = n**.

Nếu ở một bước nào đó **p** chia hết cho **k** thì số ở vị trí **p** (số **n**) sẽ bị xóa.

Sau mỗi bước xử lý số lượng số trong dãy còn lại sẽ là **p-p/k**.

k < n nên sớm hay muộn cũng sẽ bị xóa!

Độ phức tạp của giải thuật:

- + Sau mỗi bước xử lý độ dài của dãy số kết quả sẽ có bậc **p (1-1.0/k)**,
- + Trường hợp xấu nhất: **k = 100**, tốc độ giảm của độ dài dãy số sẽ là $1.0/0.99$.
- + Như vậy số phép xử lý sẽ xấp xỉ $\log_{1.0/0.99} 10^{18} \approx 4200$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "abandon."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n, k, ans=1;

int main()
{
    fi>>n>>k;
    while (n%k)
    {
        n=n-n/k; ++ans;
    }
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
}
```



Các nhà bác học thời trung cổ thường rất chú ý minh họa khi viết sách. Trong Viện bảo tàng của thành phố còn giữ một cuốn sách cổ viết về thế giới động thực vật thời bấy giờ. Sách có nhiều trang chứa hình ảnh minh họa lý thú đan xen với các trang văn bản thuyết minh và giảng giải. Các trang được đánh số từ 1 trở đi, nhưng số trang chỉ được ghi tường minh ở các trang văn bản. Đáng tiếc là cuốn sách không còn được nguyên vẹn.

Nhóm nghiên cứu muốn tìm hiểu về cuốn sách này, nhưng nó đang được đưa đi trùng tu, bảo dưỡng. Tuy vậy, người ta tìm được một tài liệu khác trong đó có nêu các đánh giá về cuốn sách đang khảo sát. Tác giả cho rằng mỗi trang văn bản chứa toàn bộ thông tin tổng kết từ đầu cuốn sách cho đến trang đó và nó có hàm lượng kiến thức bằng số bằng chính số của trang văn bản. Hàm lượng kiến thức của cuốn sách bằng tổng hàm lượng của các trang văn bản. Qua tài liệu này người ta biết được tổng hàm lượng kiến thức (theo đánh giá của tác giả tài liệu) là s . Ngoài ra, tác giả tài liệu tìm được cũng giành không ít lời ca ngợi k trang đầu tiên của cuốn sách chứa các hình ảnh minh họa hết sức độc đáo.

Nhóm nghiên cứu về cuốn sách muốn xác định xem trong cuốn sách đang khảo sát có ít nhất bao nhiêu trang minh họa.

Hãy xác định số lượng trang minh họa ít nhất có thể trong cuốn sách.

Dữ liệu: Vào từ file văn bản ILLUS.INP gồm 2 số nguyên k và s , các số có thể được ghi trên một hoặc 2 dòng ($0 \leq k \leq 10^9$, $k+1 \leq s \leq 10^{12}$).

Kết quả: Đưa ra file văn bản ILLUS.OUT một số nguyên – số lượng trang minh họa ít nhất có thể trong cuốn sách.

Ví dụ:

ILLUS.INP	ILLUS.OUT
1 8	3



VX08 Reg20180129 6 AXII

Giải thuật; Phương pháp 2 con trỏ.

Gọi **n** – số trang của cuốn sách, **ilp** – số trang minh họa, không kể **k** trang đầu tiên.

Với các trang văn bản đã viết hàm lượng kiến thức sẽ nhỏ nhất khi **i1p** trang minh họa nằm ở cuối và lớn nhất khi các trang minh họa nằm ở đầu, ngay sau **k** trang đầu tiên.

Vì vậy ta có:

$$(k+1) + (k+2) + \dots + (n - ilp) \leq s \leq (k + ilp + 1) + (k + ilp + 2) + \dots + n$$

lp
rp

Xuất phát từ $\mathbf{1lp} = 0$ tìm \mathbf{n} lớn nhất thỏa mãn điều kiện $\mathbf{1lp} \leq \mathbf{s}$.

Nếu với **n** tìm được:

- Nếu $s \leq rp$ thì **ilp** là số trang minh họa thêm ít nhất thỏa mãn điều kiện đầu bài,
 - Trong trường hợp ngược lại: cần tăng **ilp**.

Song song với tăng **i1p**, giá trị **n** cần tìm cũng sẽ tăng theo!

Chương trình

```
#include <bits/stdc++.h>
#define NAME "illus."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t s,k,ans,f,m,t,lp,rp,ilp;

int64_t calc(int64_t l, int64_t r)
{
    return l*(r-l+1)+(r-l)*(r-l+1)/2;
}

int main()
{
    fi>>k>>s;
    ilp = 0;
    for (int64_t n=k+1; ; n++)
    {
        t = calc(k+1,n);
        lp = t - calc(n-ilp+1, n);
        rp = t - calc(k+1, k+ilp);
        if (lp<=s && s<=rp) break;
        if (s < lp) ilp++, n--;
    }
    fo<< ilp + k;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Để chuẩn bị cho Lễ hội Thi pháo hoa Quốc tế người ta nghiên cứu thí nghiệm một loại pháo hoa nổ m lần. Lần thứ nhất, đạn pháo nổ trên bầu trời, các mảnh văng ra, vạch trên bầu trời những vệt lửa nối tiếp nhau và những đốm lóe sáng ở cuối mỗi vệt lửa, tạo thành đồ thị hình cây T^1 . Ở mỗi nút lá của cây T^1 là những quả đạn và mỗi quả đạn này khi nổ lại tạo ra một cây T^1 mới, vẽ lên bầu trời cây T^2 . Các nút lá của cây T^2 lại nổ, mỗi nút lá tạo ra một cây T^1 của mình và vẽ lên bầu trời cây T^3 . Quá trình nổ diễn ra m lần và tạo thành cây T^m .

Xét đường đi giữa 2 nút của cây và không có nút nào trên đường đi bị lặp lại. Độ dài của đường đi là số nút trên đường, kể cả nút đầu và nút cuối.

Độ hấp dẫn của pháo hoa được xác định bằng độ dài đường đi dài nhất sau khi nổ lần thứ m .

Ở hình bên, sau khi nổ lần đầu độ hấp dẫn là 4, sau khi nổ lần thứ 2, độ hấp dẫn là 10, sau khi nổ lần thứ 3 – là 16.

Cho cây T^1 và số nguyên m . Hãy xác định độ hấp dẫn của cây T^m .

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m , trong đó n – số nút của cây T^1 với gốc là nút 1 ($3 \leq n \leq 2 \times 10^5$, $1 \leq m \leq 2 \times 10^5$),
- ✚ Dòng thứ 2 chứa $n-1$ số nguyên p_2, p_3, \dots, p_n , trong đó p_i là cha của nút i , $i = 2 \div n$.

Kết quả: Đưa ra file văn bản FIREWORKS.OUT một số nguyên – độ hấp dẫn tìm được.

Ví dụ:

FIREWORKS.INP	FIREWORKS.OUT
<pre>4 2 1 1 2</pre>	<pre>10</pre>



Giải thuật: Đường kính của cây.

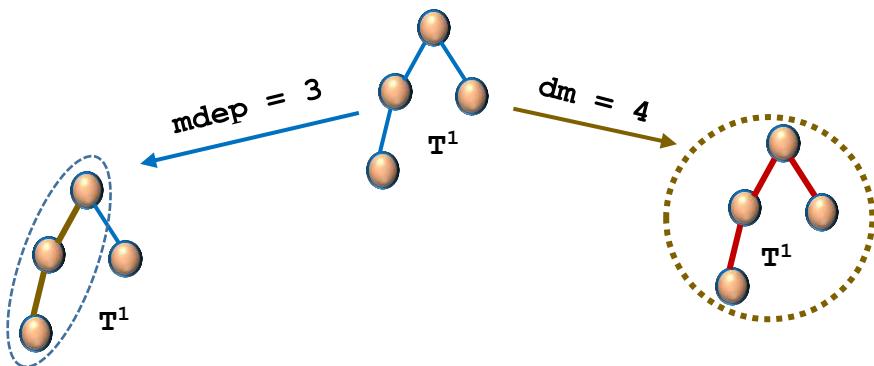
Xét cây T^1 . Đường đi dài nhất nối 2 đỉnh của cây được gọi là *đường kính* của cây.

Số nút trên đường kính (kể cả 2 nút đầu và cuối) là độ dài của đường kính.

Trong bài này ta chỉ quan tâm đến độ dài đường kính, vì vậy, để ngắn gọn, từ “*đường kính*” được dùng để chỉ cả bản thân *đường kính* lẫn *độ dài* của nó.

Khi cây chỉ có một lớp ($m = 1$) độ hấp dẫn cần tìm bằng đường kính dm của cây T^1 .

Gọi $m_{\text{đep}}$ là số đỉnh trên đường đi dài nhất từ gốc tới lá (*độ sâu* của cây).



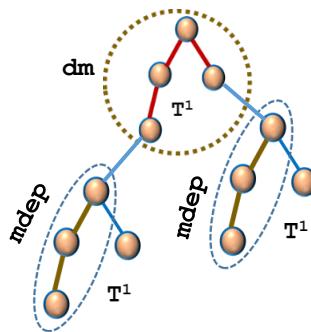
Ở cây 2 lớp ($m = 2$): Đường kính của T^2 bằng đường kính cây T^1 ở gốc cộng hai lần độ sâu của hai nhánh.

Tương tự như vậy, cứ mỗi lần nở, đường kính của cây tăng thêm $2 \times m_{\text{đep}}$.

Kết quả cần tìm sẽ là:

$$dm + 2 \times m_{\text{đep}} \times (m-1)$$

Việc tìm dm được thực hiện bằng loang theo chiều sâu từ đỉnh của cây T^1 .



Lưu ý khi tìm đường kính của cây:

- ⊕ Phân biệt 2 trường hợp: đường kính gồm một nhánh từ đỉnh của cây tới lá và đường kính chứa 2 đường đi từ một nút nào đó của cây tới 2 lá,
- ⊕ Để tìm đường đi dài nhất từ một nút lá tới nút lá khác ta chỉ cần lưu độ dài 2 đường đi dài nhất không có đỉnh chung (trừ đỉnh xuất phát).

Tổ chức dữ liệu:

- ❑ Các mảng **vector < int > e[N]** – lưu danh sách các đỉnh con,
- ❑ Mảng **int dep[N]={0}** – lưu độ sâu từ mỗi đỉnh.

Độ phức tạp của giải thuật: Bằng độ phức tạp của giải thuật tìm đường kính.

Chương trình

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

const int N = 200500;
int n, m;
int dep[N]={0}, mdep;
vector < int > e[N];
int64_t dm=0;

int64_t dfs(int v)
{
    if (e[v].empty()) return 1;
    int64_t d1 = 0, d2 = 0;
    for (int nv : e[v])
    {
        int64_t x = dfs(nv) + 1;
        if (x > d2)
            if (x > d1) d2 = d1, d1 = x;
            else d2 = x;
    }
    dm = max(dm, d1 + d2-1);
    return d1;
}

int main()
{
    fi>>n>>m;
    for (int i = 1; i < n; ++i)
    {
        int x;
        fi>>x;
        --x;
        e[x].push_back(i);
        mdep = max(mdep, dep[i] = dep[x] + 1);
    }

    dfs(0); ++mdep;
    if (dm<mdep) dm=mdep;
    fo<<dm+2*(m-1LL)*mdep;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Có n cửa hàng bán lẻ lớn đang hoạt động, mỗi cửa hàng thuộc một tập đoàn khác nhau. Xu hướng phát triển của nền kinh tế hội nhập đòi hỏi các tập đoàn phải đủ mạnh để tồn tại và vươn xa ảnh hưởng ra phạm vi toàn cầu. Hội nghị hiệp thương các chủ sở hữu quyết định hợp nhất thành một tập đoàn bán lẻ chung. Tuy vậy luật chống độc quyền chỉ cho phép tiến hành hợp nhất 2 công ty bất kỳ thành một. Ngoài ra luật còn quy định, nếu 2 công ty có tổng tài sản giá trị tương ứng là x và y (theo đơn vị tỷ đồng) thì khi hợp nhất một nữa tổng giá trị, tức là $(x+y)/2$ phải chia cho các cổ đông, nửa còn lại mới được giữ để tiếp tục hoạt động. Ví dụ $x = 3$, $y = 4$ thì chỉ được giữ lại 3.5 tỷ cho công ty chung.

Đã là luật thì không thể vi phạm, nhưng có thể vòng tránh! Người ta quyết định thực hiện hợp nhất theo lộ trình $n-1$ bước, ở mỗi bước chọn 2 cửa hàng và hợp nhất thành một chủ sở hữu, số lượng cửa hàng giảm đi một. Thực hiện xong một bước hợp nhất, sau một thời gian bước hợp nhất tiếp theo được tiến hành . . . cứ như vậy cho đến khi chỉ có một chủ sở hữu chung cho cả n cửa hàng. Hiện tại cửa hàng thứ i có tổng tài sản giá trị a_i , $i = 1 \div n$. Các trình tự hợp nhất sẽ cho tổng giá trị tài sản cuối cùng khác nhau. Mọi người đều muốn sau khi hoàn thành hợp nhất n cửa hàng, tổng giá trị tài sản được giữ lại là lớn nhất và những bộ óc hoàn hảo của thế giới kinh doanh đã làm được việc này!

Hãy xác định tổng giá trị tài sản lớn nhất được giữ lại.

Dữ liệu: Vào từ file văn bản COALESCE.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 ($1 \leq a_i \leq 10^4$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản COALESCE.OUT một số thực với độ chính xác 10^{-6} – kết quả tìm được

Ví dụ:

COALESCE S.INP	COALESCE.OUT
3	4.750000
5 5 4	

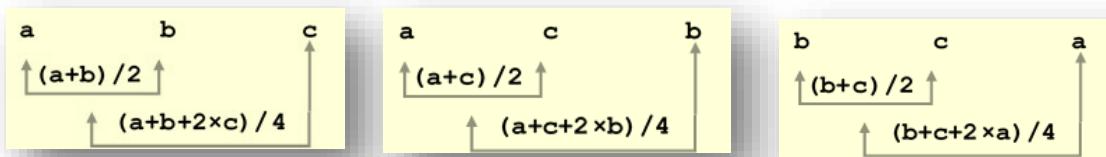


VX10 Cr7 20180310 1 AXII

Giải thuật: Phân tích mô hình toán học.

Xét 3 số nguyên dương a, b, c : $a < b < c$.

Có 3 cách hợp nhất:



Rõ ràng việc để lại số lớn nhất ghép sau cùng sẽ cho kết quả lớn nhất.

Kết quả ghép 2 số sẽ không vượt quá giá trị của số lớn hơn.

Từ những nhận xét trên ta có sơ đồ xử lý:

- ✚ Sắp xếp dãy a_i theo thứ tự tăng dần,
- ✚ Với $i = 1 \div n-1$: thay a_i bằng $(a_{i-1} + a_i) / 2$.

Kết quả cần tìm là a_{n-1} .

Để thuận tiện xử lý, các giá trị a_i nên lưu dưới dạng **float**.

Lưu ý cách đưa ra để thể hiện độ chính xác 10^{-6} .

Độ phức tạp của giải thuật: O(nlnn).

Chương trình

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

int main()
{
    fi>>n;
    vector<float>a(n);
    for(int i=0; i<n; ++i) fi>>a[i];
    sort(a.begin(), a.end());
    for(int i=1; i<n; ++i)a[i]=(a[i-1]+a[i])/2.0;
    fo<<fixed<<setprecision(6)<<a[n-1];
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



VX11. TƯỚI NƯỚC

Tên chương trình: IRRIGATION.CPP

Xe bồn chứa m lit nước chạy dọc theo một tuyến phố tưới cho n cây trồng ở trên giải phân cách. Xuất phát từ cây số 1 ở đầu phố xe lùn lượt dừng lại, tưới cho các cây 1, 2, 3 . . . sau khi dừng và tưới cây thứ n xe quay lại tưới tiếp theo trình tự ngược lại các cây $n-1, n-2, \dots$ cho tới cây 1, rồi lại tưới tiếp các cây 2, 3, . . . cho đến khi hết nước. Mỗi lần dừng xe tưới k lit nước cho cây, trừ ở lần dừng cuối cùng cây nhận số nước còn lại, có thể không đủ k .

Hãy xác định mỗi cây được tưới bao nhiêu nước.

Dữ liệu: Vào từ file văn bản IRRIGATION.INP gồm một dòng chứa 3 số nguyên n, k và m ($2 \leq n \leq 2 \times 10^5, 1 \leq k, m \leq 2 \times 10^9$).

Kết quả: Đưa ra file văn bản IRRIGATION.OUT trên một dòng n số nguyên, số thứ i xác định số nước cây thứ i được tưới, $i = 1 \div n$.

Ví dụ:

IRRIGATION.INP
1 8

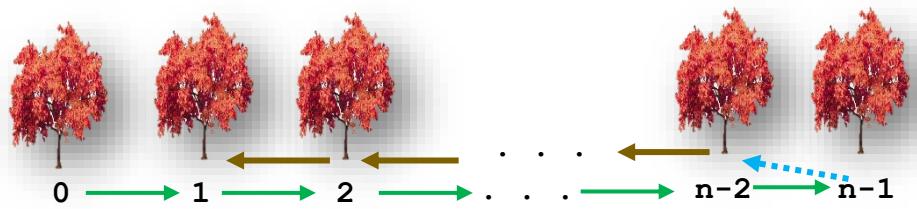
IRRIGATION.OUT
3



VX11 Cr7 20180310_2_AXII

Giải thuật: Phân tích hoạt động ô tô mát.

Với m đủ lớn quá trình tưới cây bao gồm các *chu kỳ*, mỗi chu kỳ độ dài $2 \times n - 2$ và *một chu kỳ thiếu* (ứng với phần nước còn lại sau chu kỳ cuối cùng).



Trong một chu kỳ mỗi cây nhận được lượng nước là $2 \times k$, trừ 2 cây đầu và cuối hàng mỗi cây chỉ nhận được lượng nước k .

Dễ dàng tính được số lượng chu kỳ, kích thước chu kỳ thiếu và số lượng nước ít hơn k mà cây cuối cùng trong quá trình tưới nhận được.

Ở chu kỳ thiếu cần phân biệt kích thước lớn hơn n hoặc không lớn hơn, duyệt các cây theo trình tự xuất hiện trong chu trình và bổ sung k vào lượng nước cây được nhận.

Cập nhật riêng với cây cuối cùng.

Tổ chức dữ liệu:

Mảng – chứa lượng nước được tưới ở mỗi cây.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "irr."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,k,m,p,q,u,v;

int main()
{
    fi>>n>>k>>m;
    vector<int>a(n);
    if(k>=m) a[0]= k;
    else
    {
        p=m/k; q=p/(2*n-2);
        u=m%k; v=p%(2*n-2);
        for(int i=0; i<n; ++i) a[i]=q*k*2;
        a[n-1]-=q*k; a[0]-=q*k;

        if(v>=n)
        {
            for(int i=0; i<n; ++i)a[i]+=k;
            v-=n;
            for(int i=0; i<v; ++i) a[n-2-i]+=k;
            a[n-2-v]+=u;
        }
        else
        {
            for(int i=0; i<v; ++i)a[i]+=k;
            a[v]+=u;
        }
    }
    for(int i=0; i<n; ++i) fo<<a[i]<<' ';
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

