

MỤC LỤC ATHENA XIV

VY13. ÔN TẬP	Tên chương trình: REVIEW.CPP	3
VY14. DÒ MÌN	Tên chương trình: DETECTION.CPP.....	6
VY15. BÓC BÀI	Tên chương trình: CARDS.CPP.....	8
VY16. TETRIS	Tên chương trình: TETRIS.CPP.....	10
VY17. ĐĂNG NHẬP	Tên chương trình: LOGIN.CPP	14
VY18. CHUYỀN SÁU	Tên chương trình: SIXPASS.CPP	17
VY19. BỐ VÀ CON	Tên chương trình: DADY.CPP.....	20
B04. XÉP HÀNG	Tên chương trình: TURN.CPP	25
VY20. XÂU CON	Tên chương trình: TWOSTR.CPP	27
VY21. CƠ SỐ 3	Tên chương trình: BASE3.CPP	29
VY22. THI BẢN NHANH	Tên chương trình: RANGE.CPP.....	31
VY23. MÃ SỐ	Tên chương trình: CIPHER.CPP	34
VY24. TRÒ CHƠI JENGA	Tên chương trình: JENGA.CPP.....	37
B09. KHẨM TRẠI	Tên chương trình: MOSAIC.CPP	40
B10. NHIÊN LIỆU SẠCH	Tên chương trình: FUEL.CPP	42
VY25. BÀN CỜ	Tên chương trình: CHESSBOARD.CPP.....	45
VY26. THỦ SỨC	Tên chương trình: ATTEMPT.CPP.....	48
VY27. ĐƯỜNG HÀM	Tên chương trình: TUNNEL.CPP	56
VY28. BẮN CUNG	Tên chương trình: LONGBOW.CPP	60
VY29. ĐĂNG CÁU	Tên chương trình: ISOMRP.CPP.....	62
VY30. CÁ CƯỢC	Tên chương trình: BETTING.CPP	68
VY31. KHÓA CHUNG	Tên chương trình: COMKEY.CPP	71
VY32. XỬ LÝ SỐ LỚN	Tên chương trình: BIGNUM.CPP	73
VY33. GRAFFITI	Tên chương trình: GRAFFITI.CPP	76
VY35. PHONG BÌ	Tên chương trình: ENVELOPE.CPP.....	81
VY36. CHƠI ĐẸP	Tên chương trình: FAIRPLAY.CPP	83
VY37. SỐ LƯỢNG BÀI	Tên chương trình: TASKS.CPP	87
VY38. MÃ KHÓA	Tên chương trình: MAXKEY.CPP	89
B11. ĐOẠN THẮNG DÈO	Tên chương trình: ELASTIC.CPP	93
VY40. SỐ LƯỢNG TESTS	Tên chương trình: UNERASE.CPP.....	95
VY41. BẢNG KẾT QUẢ	Tên chương trình: RES_TAB.CPP	97
VY42. BẮN BIA	Tên chương trình: SHOOTING.CPP	99
VY43. ĐỌC SỐ	Tên chương trình: READNUM.CPP	101
VY44. CỒNG CHIÊNG	Tên chương trình: GONGS.CPP	105

VY45. GIAO HÀNG	<i>Tên chương trình: DELIVERY.CPP</i>	108
VY46. DIỄN TẬP	<i>Tên chương trình: DRONES.CPP</i>	110
VY47. VÁC TRE	<i>Tên chương trình: BAMBOO.CPP</i>	115
VY48. LỰA CHỌN	<i>Tên chương trình: CHOICE.CPP</i>	121
VY49. GẶP MẶT	<i>Tên chương trình: TOGETHER.CPP</i>	124
VY50. HÈM NÚI	<i>Tên chương trình: CANYON.CPP</i>	126
VZ01. NGHỆ THUẬT ĐƯỜNG PHỐ	<i>Tên chương trình: ART.CPP</i>	129
VZ02. GẦN ĐÚNG	<i>Tên chương trình: APPROXIMATION.CPP</i>	132
B12. BỨC TRanh	<i>Tên chương trình: PICTURE.CPP</i>	134
B13. THANG MÁY	<i>Tên chương trình: ELEVATOR.CPP</i>	136
B14. GIẢI ĐẤU CỜ NHANH	<i>Tên chương trình: CHESS.CPP</i>	138
B15. ĐƯỜNG PHỐ	<i>Tên chương trình: STREET.CPP</i>	140
B16. BIỂN QUẢNG CÁO	<i>Tên chương trình: SHIELD.CPP</i>	142
B17. ĐỘI NGŨ	<i>Tên chương trình: PARADE.CPP</i>	144
B18. RÃY SỐ	<i>Tên chương trình: NUMERIC_SQ.CPP</i>	146
B19. ĐƯỜNG ĐI	<i>Tên chương trình: ROUTE.CPP</i>	149
B20. SỐ CÁCH THAY GIÁ TRỊ	<i>Tên chương trình: CHANGE.CPP</i>	152
VZ03. ĐÈN LED	<i>Tên chương trình: LED.CPP</i>	155
VZ04. PHU ÂM	<i>Tên chương trình: CONSONANT.CPP</i>	160
VZ05. ĐO ĐẠC	<i>Tên chương trình: MEASURES.CPP</i>	164
VZ06. SỐ CHÍNH PHƯƠNG	<i>Tên chương trình: SQUARES.CPP</i>	166
VZ07. CHỈNH LÝ	<i>Tên chương trình: CORRECTION.CPP</i>	169
VZ09. CẤP DỮ LIỆU	<i>Tên chương trình: PAIR.CPP</i>	172
VZ10. DIOXIN	<i>Tên chương trình: DIOXIN.CPP</i>	175
VZ11. SẮP XẾP BỘ PHẬN	<i>Tên chương trình: PART_ST.CPP</i>	184
VZ12. HAI NHIỆM VỤ	<i>Tên chương trình: TWOTASKS.CPP</i>	194
VZ13. TAXI BAY	<i>Tên chương trình: TAXI.CPP</i>	197

Việc nhớ từ mới để có thể nói và viết được là rất quan trọng trong quá trình học ngoại ngữ. Ở một lớp học, học viên đã biết **k** từ khác nhau. Để biết học viên có nắm vững cách viết mỗi từ hay không giáo viên viết trên bảng một chữ cái, người được hỏi phải nói từ bắt đầu bằng chữ cái trên bảng, từ này phải có thứ tự từ điển tiếp theo từ có cùng chữ cái bắt đầu và đã được nói trước đó trong quá trình kiểm tra. Việc chọn từ tiếp theo được thực hiện vòng tròn: nếu có **m** từ cùng bắt đầu bằng một chữ cái thì sau khi từ thứ nhất được sử dụng cần nói từ thứ 2, sau khi từ thứ 2 được sử dụng cần nói từ thứ 3, . . . , sau khi từ thứ **m** được sử dụng cần nói từ thứ nhất.

Số lượng chữ cái được lần lượt viết trên bảng là **n**. Với mỗi chữ cái hãy đưa ra từ tương ứng.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên **k** và **n** ($1 \leq k, n \leq 10^5$),
- ✚ Dòng thứ **i** trong **k** dòng tiếp theo chứa xâu ký tự xác định một từ độ dài không quá 21 và chỉ chứa các ký tự la tinh thường,
- ✚ Mỗi dòng trong **n** dòng tiếp theo chứa một ký tự la tinh thường xác định chữ cái viết trên bảng.

Kết quả: Đưa ra file văn bản REVIEW.OUT các từ cần nói, mỗi từ trên một dòng.

Ví dụ:

REVIEW.INP	REVIEW.OUT
4 5	
zagreb	zadar
split	sisak
zadar	split
sisak	zagreb
z	zadar
s	
s	
z	
z	



Giải thuật; Phân loại dữ liệu .

Các bắt đầu cùng một chữ cái được đưa vào cùng một lớp dữ liệu,

Có 26 chữ cái khác nhau vì vậy có không quá 26 lớp cho các từ đã cho.

Sắp xếp các từ theo thứ tự tăng dần,

Các từ cùng một lớp – đứng liên tiếp nhau theo thứ tự từ điển của mình trong dãy,

Với mỗi lớp cần lưu giữ:

- ⊕ Số lượng từ trong lớp,
- ⊕ Con trỏ chỉ tới từ tiếp theo trong lớp.

Với mỗi yêu cầu, sau khi đưa ra từ cần thiết – điều chỉnh con trỏ để chỉ tới từ tiếp trong lớp (điều chỉnh vòng tròn!).

Tổ chức dữ liệu:

- ▀ Mảng `vector<string> w` – Lưu các từ đã học,
- ▀ Mảng `int t[26]={0}` – Lưu số lượng từ trong mỗi lớp,
- ▀ Mảng `int addr[26]={0}` – Lưu con trỏ chỉ tới đầu mỗi lớp,
- ▀ Mảng `int cur[26]={0}` – Lưu con trỏ chỉ tới từ tiếp theo trong lớp.

Độ phức tạp giải thuật: O($n \ln n$).

Chương trình

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

int n,k,vc,pos,p;
char c;
string s;
int t[26]={0},addr[26]={0},cur[26]={0};

int main()
{
    fi>>k>>n;
    vector<string> w;
    for(int i=0; i<k; ++i)
    {
        fi>>s; ++t[s[0]-97];
        w.push_back(s);
    }
    sort(w.begin(),w.end());

    for(int i=0; i<26; ++i) if(t[i]) addr[i]=p, p+=t[i];
    for(int i =0; i<n; ++i)
    {
        fi>>c; vc=c-97;
        pos=addr[vc]+cur[vc];
        cur[vc]=(cur[vc]+1)%t[vc];
        fo<<w[pos]<<'\n';
    }
    Times;
}
```



Sân bay mới dự kiến được xây dựng trên mảnh đất hình chữ nhật kích thước $n \times m$ ô. Việc đầu tiên là phải dò kỹ bom mìn có thể còn sót lại. Máy dò mìn còn thực hiện thêm một số công việc đo đạc khảo sát phụ khác nên khá cồng kềnh. Máy được điều khiển từ xa và chỉ có 2 loại lệnh: Tiến lên phía trước hoặc Quay 90 độ (sang trái hay phải). Tới mỗi ô máy sẽ thực hiện kiểm tra, khảo sát toàn bộ ô. Lệnh Quay 90 độ đòi hỏi rất nhiều năng lượng vì vậy người ta cố gắng hạn chế đến mức thấp nhất việc sử dụng lệnh này.

Ban đầu máy có thể được mang tới đặt ở ô tùy ý. Trong quá trình làm việc máy không được ra khỏi vùng đất xây dựng và khi khảo sát hết các ô – có thể dừng lại ở ô bất kỳ.

Hãy xác định số lượng lệnh Quay ít nhất cần thực hiện.

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

- ✚ Dòng đầu tiên chứa một số nguyên k – số lượng tests ($1 \leq k \leq 50\,000$),
- ✚ Mỗi dòng sau k dòng sau chứa 2 số nguyên n và m ($1 \leq n, m \leq 10^6$).

Kết quả: Đưa ra file văn bản DETECTION.OUT n số nguyên – số lượng lệnh Quay ít nhất cần thực hiện ứng với mỗi test, mỗi số trên một dòng.

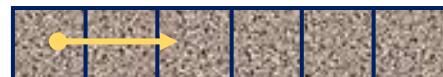
Ví dụ:

DETECTION.INP	DETECTION.OUT
5	0
1 10	4
3 3	4
3 4	8
5 8	6
6 4	



Giải thuật; Kỹ năng phân tích mô hình toán học.

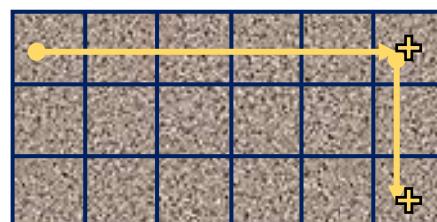
Nếu $n = 1$ hoặc $m = 1$: mảnh đất là một băng và không cần thực hiện một lệnh quay nào.



Với $n > 1$ và $m > 1$:

Với một phép xoay –kích thước mỗi chiều của miền cần khảo sát giảm 1,

Thêm một phép xoay nữa: Đưa về bài toán ban đầu với mỗi kích thước giảm 1.



Như vậy sau $2 \times \min(n-1, m-1)$ ta có miền còn lại cần khảo sát có ít nhất một chiều bằng 1.

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int n,m,k;

int main()
{
    fi>>k;
    for(int i=0; i<k; ++i)
    {
        fi>>n>>m;
        fo<<2*min(n-1,m-1)<<'\n';
    }
}
```



Xét bộ bài 52 quân, mỗi quân bài có 4 chất: Rô, Cơ, Bíc, Chuồn. Giá trị các quân bài có số bằng số ghi trên đó, các quân bài J, Q, K đều có giá trị 10, quân A – có giá trị 11.

Sau khi tráo, cỗ bài được đặt úp trên bàn. Mỗi người chơi, khi đến lượt mình, được lần lượt bốc từng quân bài cho đến khi muốn dừng, nhưng nếu tổng giá trị các quân bài bốc về lớn 21 thì sẽ bị thua.

Tôm là người bốc bài đầu tiên và có trên tay n quân với các giá trị v_1, v_2, \dots, v_n . Tôm đắn đo, không biết có nên bốc tiếp hay không và nhầm tính trong số các quân bài còn lại nếu số lượng quân bài có giá trị không làm Tôm vượt quá 21 nhiều hơn hoặc bằng số quân bài mà nếu bốc được sẽ bị thua thì sẽ bốc tiếp, trong trường hợp ngược lại – dừng, không bốc nữa.

Hãy đưa ra quyết định mà Tôm lựa chọn: “**DRAW**” nếu quyết định bốc tiếp và “**STOP**” trong trường hợp ngược lại.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 52$),
- ✚ Dòng thứ 2 chứa n số nguyên v_1, v_2, \dots, v_n ($2 \leq v_i \leq 11$, $i = 1 \div n$), mỗi giá trị giống nhau gấp không quá 4 lần.

Kết quả: Đưa ra file văn bản CARDS.OUT thông báo ứng với lựa chọn tính được.

Ví dụ:

CARDS.INP	CARDS.OUT
3	
5 6 2	DRAW



Giải thuật; Lập trình cơ sở.

Tạo lập mảng **f** [12] lưu trữ số lượng quân bài ứng với mỗi giá trị trong cỗ bài úp trên bàn,

Ban đầu mảng **f** có giá trị {0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 16, 4}.

Tính tổng **s** giá trị các quân bài đã bốc và cập nhật lại **f_i** tương ứng.

Giá trị còn lại có thể có để không thua là **k=21-s**.

Dễ dàng thấy rằng, nếu **k ≥ 11** thì bốc bài tiếp là cần thiết.

Nếu **k < 11** – đếm số lượng quân bài còn ở cỗ bài trên bàn (có 52 – **n** quân) có giá trị lớn hơn **k**, so sánh với số lượng quân bài còn lại, mỗi quân có giá trị nhỏ hơn hoặc bằng **k** và rút ra kết luận cần thiết.

Tiểu xảo lập trình: Không cần trực tiếp kiểm tra điều kiện **k < 11**.

Độ phức tạp của giải thuật: O(1).

Chương trình

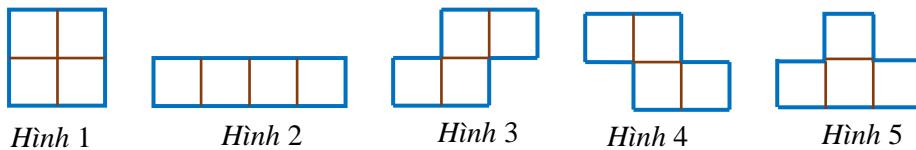
```
#include <bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
#define NAME "cards."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,k,rmd=0,s=0,f[12]={0,0,4,4,4,4,4,4,4,4,16,4};

int main()
{
    fi>>n;
    for(int i=0;i<n;++i)
    {
        fi>>k;
        s+=k;
        --f[k];
    }
    k=21-s;
    for(int i=k+1; i<12; ++i) rmd+=f[i];
    if(52-n-rmd> rmd) fo<<"DRAW"; else fo<<"STOP";
    Times;
}
```



Mỗi chương trình trò chơi là một mô hình trí tuệ nhân tạo thu nhỏ. Câu lạc bộ Tin Học đưa các bạn trẻ vào thế giới trí tuệ nhân tạo bằng yêu cầu tự thiết kế các chương trình trò chơi, bắt đầu từ những trò chơi đơn giản.

Rộn chọn cho mình trò chơi Tetris với một số quy tắc riêng. Năm hình cơ sở được chọn là như sau:



Khi lắp ráp, mỗi hình có thể quay 90° một số lần và tô một màu tùy chọn. Mỗi hình có thể được sử dụng nhiều lần. Rộn chỉ mới hoàn thành xong phần đầu bao gồm việc chọn hình, xoay và tô màu tùy chọn thì đến giờ đi học.

Jimmy, em của Rộn, rất thích thú quan sát anh làm việc và tranh thủ lúc máy rãnh cho chạy thử chương trình. Jimmy dùng các hình cơ sở và các phép xoay, tô màu lắp thành một hình mới, trong đó các hình cơ sở không đè lên nhau nhưng có thể có chung cạnh hoặc đỉnh. Hai hình có chung cạnh hoặc đỉnh được tô bằng những màu khác nhau. Số lượng màu có thể chọn là 26 và được ký hiệu bằng một chữ cái la tinh thường.

Khi đi học về, Rộn nhìn thấy hình ghép mà Jimmy còn để lại và tự hỏi không biết mỗi hình cơ sở phải dùng bao nhiêu lần để có được hình ghép này.

Hãy xác định số lượng mỗi hình cơ sở đã được sử dụng.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m – kích thước hình chữ nhật chứa hình ghép ($1 \leq n, m \leq 10$),
- ✚ Mỗi dòng trong n dòng tiếp theo chứa xâu s độ dài m xác định màu một dòng của hình chữ nhật, xâu s chỉ chứa các chữ cái la tinh thường hoặc dấu chấm, dấu chấm ký hiệu ô trống.

Kết quả: Đưa ra file văn bản TETRIS.OUT 5 số nguyên, mỗi số trên một dòng, số thứ i xác định số lượng sử dụng hình i ($i = 1 \div 5$).

Ví dụ:

TETRIS.INP	TETRIS.OUT
<pre>5 7 .c..... ccddddd. caabbcc aabbacc ...aaa..</pre>	<pre>1 1 2 1 1</pre>



Giải thuật; Bài toán nhận dạng, Kỹ thuật bảng phương án .

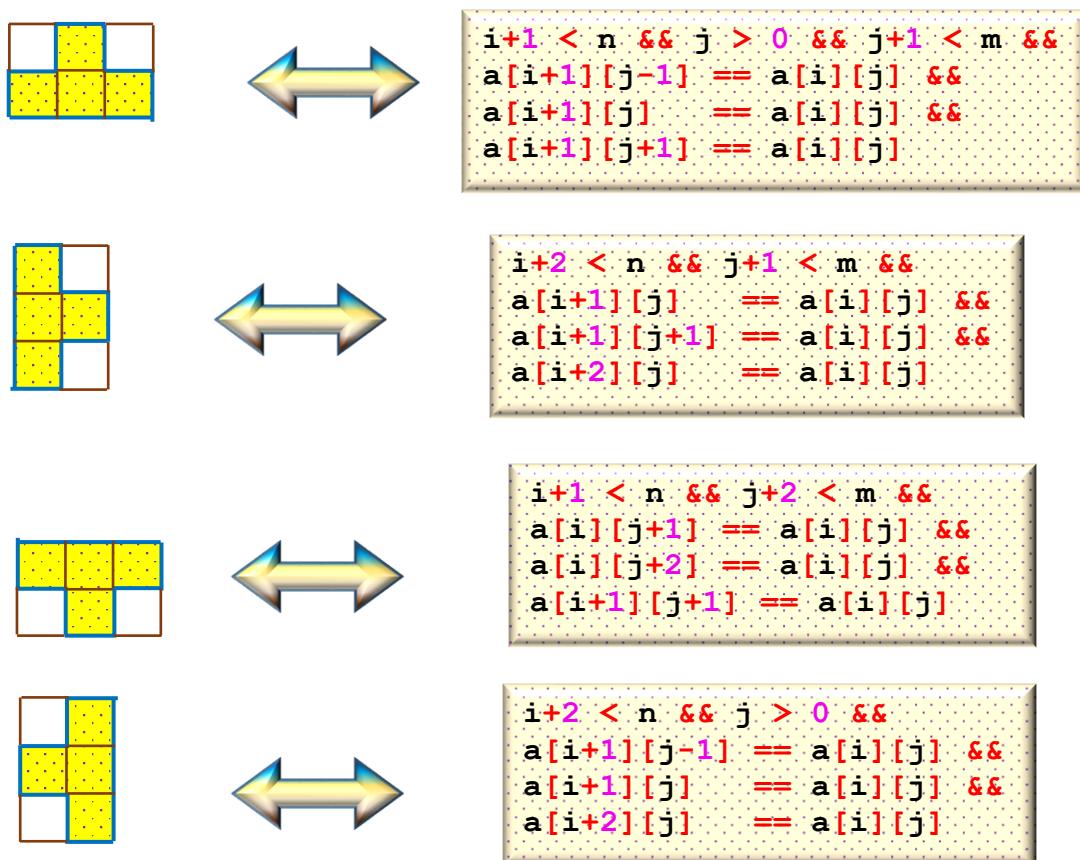
Với các bài toán nhận dạng kỹ thuật bảng phương án (*Decide Table*) là hiệu quả và đơn giản hơn cả.

Để tạo bảng phương án cần liệt kê các khả năng lô gic cho từng trường hợp có thể xảy ra và chỉ ra hành động tương ứng cần thực hiện.

Với bài toán đếm cấu hình: hành động cần thực hiện là tích lũy tần số sử dụng tương ứng.

Gọi **a** – mảng kích thước $n \times m$ lưu các ký tự của dữ liệu vào.

Với hình 5 có 4 câu hình phù hợp:



Các hình khác: Xây dựng các bảng nhận dạng tương tự.

Về nguyên tắc: Sau khi đã nhận dạng được cần **xóa phần đã nhận dạng** (điền dấu ‘.’) Ở đây bài toán có kích thước nhỏ và các hình cần nhận dạng không giao nhau nên có thể không cần xóa.

Độ phức tạp của giải thuật: O(n × m).

Chương trình

```
#include <bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
#define NAME "tetris."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,m,ans[5]={0};
int main()
{
    fi>>n>>m;
    vector<string>a(m);
    for(int i=0; i<n; ++i) fi>>a[i];
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
    {
        if (a[i][j] == '.') continue;

        // square H1
        if (i+1 < n && j+1 < m &&
            a[i+1][j] == a[i][j] &&
            a[i][j+1] == a[i][j] &&
            a[i+1][j+1] == a[i][j])
        {
            ++ans[0]; a[i][j]='.'; a[i+1][j]='.';
            a[i][j+1]='.'; a[i+1][j+1]='.';
        }

        // line H2
        if (j+3 < m &&
            a[i][j+1] == a[i][j] &&
            a[i][j+2] == a[i][j] &&
            a[i][j+3] == a[i][j])
        {
            ++ans[1]; a[i][j]='.'; a[i][j+1]='.';
            a[i][j+2]='.'; a[i][j+3]='.';
        }

        if (i+3 < n &&
            a[i+1][j] == a[i][j] &&
            a[i+2][j] == a[i][j] &&
            a[i+3][j] == a[i][j])
        {
            ++ans[1]; a[i][j]='.'; a[i+1][j]='.';
            a[i+2][j]='.'; a[i+3][j]='.';
        }

        // S H3
        if (i+1 < n && j > 0 && j+1 < m &&
            a[i+1][j-1] == a[i][j] &&
            a[i+1][j] == a[i][j] &&
            a[i][j+1] == a[i][j])
        {
            ++ans[2]; a[i][j]='.'; a[i+1][j-1]='.';
            a[i+1][j]='.'; a[i][j+1]='.';
        }

        if (i+2 < n && j+1 < m &&
            a[i+1][j] == a[i][j] &&
            a[i+1][j+1] == a[i][j] &&
            a[i+2][j+1] == a[i][j])
        {
            ++ans[2]; a[i][j]='.'; a[i+1][j]='.';
            a[i+1][j+1]='.'; a[i+2][j+1]='.';
        }
    }
}
```

```

    {
        ++ans[2]; a[i][j]='.'; a[i+1][j]='.';
        a[i+1][j+1]='.'; a[i+2][j+1]='.';
    }

    // Z H4
    if (i+1 < n && j+2 < m &&
        a[i][j+1] == a[i][j] &&
        a[i+1][j+1] == a[i][j] &&
        a[i+1][j+2] == a[i][j])
    {
        ++ans[3]; a[i][j]='.'; a[i][j+1]='.';
        a[i+1][j+1]='.'; a[i+1][j+2]='.';
    }

    if (i+2 < n && j > 0 &&
        a[i+1][j-1] == a[i][j] &&
        a[i+1][j] == a[i][j] &&
        a[i+2][j-1] == a[i][j])
    {
        ++ans[3]; a[i][j]='.'; a[i+1][j-1]='.';
        a[i+1][j]='.'; a[i+2][j-1]='.';
    }

    // T H5
    if (i+2 < n && j+1 < m &&
        a[i+1][j] == a[i][j] &&
        a[i+1][j+1] == a[i][j] &&
        a[i+2][j] == a[i][j])
    {
        ++ans[4]; a[i][j]='.'; a[i+1][j]='.';
        a[i+1][j+1]='.'; a[i+2][j]='.';
    }

    if (i+1 < n && j > 0 && j+1 < m &&
        a[i+1][j-1] == a[i][j] &&
        a[i+1][j] == a[i][j] &&
        a[i+1][j+1] == a[i][j])
    {
        ++ans[4]; a[i][j]='.'; a[i+1][j+1]='.';
        a[i+1][j]='.'; a[i+1][j-1]='.';
    }

    if (i+2 < n && j > 0 &&
        a[i+1][j-1] == a[i][j] &&
        a[i+1][j] == a[i][j] &&
        a[i+2][j] == a[i][j])
    {
        ++ans[4]; a[i][j]='.'; a[i+1][j-1]='.';
        a[i+1][j]='.'; a[i+2][j]='.';
    }

    if (i+1 < n && j+2 < m &&
        a[i][j+1] == a[i][j] &&
        a[i][j+2] == a[i][j] &&
        a[i+1][j+1] == a[i][j])
    {
        ++ans[4]; a[i][j]='.'; a[i][j+1]='.';
        a[i][j+2]='.'; a[i+1][j+1]='.';
    }

    for (int i:ans) fo<<i<<'\n';
    Times;
}

```



Một mạng liên kết nội bộ được xây dựng để các học sinh trong trường trao đổi thông tin, bài vở với nhau. Mạng có giao diện đẹp và quản lý được nhiều loại thông tin khác nhau phù hợp với nhu cầu của học sinh nên mọi người rất thích.

Tuy nhiên, sau một thời gian khai thác, người ta phát hiện ra một lỗi bảo mật nghiêm trọng. Nếu thành viên có mật khẩu đăng nhập là xâu **P** thì khi gõ một xâu **X** có chứa **P** như một xâu con các ký tự liên tiếp, hệ thống cũng chấp nhận và cho vào hệ thống với các quyền của người có mật khẩu **P**. Ví dụ **P** = “**abc**”, thì khi gõ “**abc**” hay “**abcd**” hoặc “**imabcom**” hệ thống sẽ chấp nhận và cho vào, còn nếu gõ “**abxc**” – sẽ bị từ chối.

Hiện nay đang có **n** người đăng ký sử dụng, người thứ **i** có mật khẩu đăng nhập là **p_i**, **i** = 1 ÷ **n**, mỗi mật khẩu là một xâu khác rỗng, độ dài không quá 10 và chỉ chứa các ký tự la tinh thường.

Người quản trị mạng muốn xác định có bao nhiêu cặp người sử dụng có thể vào hệ thống với quyền của người khác.

Hãy đưa ra số lượng cặp người sử dụng, người thứ nhất trong cặp có thể vào hệ thống và có quyền sử dụng của người thứ hai.

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

- ✚ Dòng đầu tiên chứa một số nguyên **n** ($1 \leq n \leq 2 \times 10^4$),
- ✚ Dòng thứ **i** trong **n** dòng sau chứa xâu ký tự **p_i**.

Kết quả: Đưa ra file văn bản LOGIN.OUT một số nguyên – số cặp xác định được.

Ví dụ:

LOGIN.INP	LOGIN.OUT
5 mir mirta ta ir t	6



Giải thuật; Kỹ thuật tổ chức dữ liệu.

Với mỗi xâu p_i độ dài k :

- Trích xâu con các ký tự liên tiếp nhau độ dài 1, 2, ..., $k-1$, k ,
- Với các xâu con giống nhau: chỉ giữ một đại diện,

Tích lũy tàn số xuất hiện của mỗi xâu con nhận được khi xử lý p_1, p_2, \dots, p_n .

Số lượng các xâu con khác nhau trích được từ một p_i nhiều nhất là

$$k + (k-1) + (k-2) + \dots + 2 + 1 \leq 55$$

vì vậy có thể lưu giữ được các xâu con khác nhau cho mọi mật khẩu.

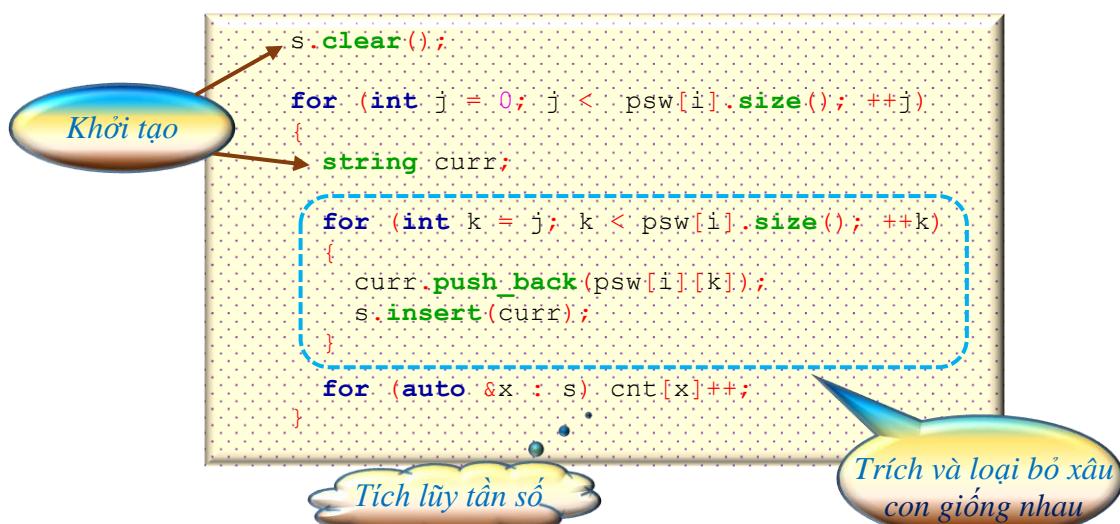
Số lượng xâu con bằng p_i là số người có thể truy nhập vào mạng và có đặc quyền của người thứ i (*kể cả chính người i*). Từ đây dễ dàng suy ra kết quả cần tìm.

Tổ chức dữ liệu:

- Mảng `vector<string> psw(n)` – Lưu các mật khẩu,
- Tập `unordered_set<string> s` – Lưu các xâu con khác nhau của một mật khẩu,
- Bảng `unordered_map<string, int> cnt` – Lưu tàn số xuất hiện các xâu con.

Xử lý:

Trích và lưu xâu con khác nhau của một mật khẩu:



Độ phức tạp của giải thuật: $O(m \log m)$, trong đó $m \approx 10 \times n$.

Chương trình

```
#include <bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define NAME "login."
using namespace std;
ifstream fi (NAME"inp");
//ifstream fi ("20");
ofstream fo (NAME"out");

int n;
unordered_set<string> s;
unordered_map<string, int> cnt;

int main()
{
    fi >> n;
    vector<string> psw(n);
    for (int i = 0; i < n; ++i)
    {
        fi >> psw[i];
        s.clear();

        for (int j = 0; j < psw[i].size(); ++j)
        {
            string curr;
            for (int k = j; k < psw[i].size(); ++k)
            {
                curr.push_back(psw[i][k]);
                s.insert(curr);
            }

            for (auto &x : s) cnt[x]++;
        }
    }

    int ans = 0;
    for (auto &x : psw) ans += cnt[x];

    fo << ans - n << '\n';
    Times;
}
```



Chuyền sáu là trò chơi chuyền bóng không đòi hỏi sân bãi đặc biệt. Có thể có 2 hay nhiều đội cùng tham gia. Mỗi đội có 6 người. Cầu thủ một đội bắt được bóng phải cố gắng chuyền bóng cho người khác của đội mình. Nếu sau nhiều đường chuyền qua lại giữa các cầu thủ trong cùng một đội cả 6 người đều có lần bắt được bóng thì được một điểm và bóng được chuyền cho đội khác. Trong quá trình chuyền cầu thủ các đội khác cố gắng tranh cướp bóng để chuyền cho người của mình. Trò chơi có thể kéo dài bao lâu tùy ý và trong quá trình chơi các đội cũng có quyền thay người.

Ở một trại hè các bạn trẻ tổ chức chơi Chuyền sáu. Cuộc chơi sẽ kéo dài trong m phút. Đội đang xét có n cầu thủ, người thứ i có chỉ số hiệu quả là k_i và thể lực cho phép chỉ có thể chơi trong vòng $1k_i$ phút ($i = 1 \dots n$). Một người, khi đã bị thay ra, sẽ không được vào lại. Hiệu quả của đội ở mỗi phút là tổng hiệu quả các cầu thủ của đội trong phút đó. Hiệu quả của đội trong toàn cuộc chơi là tổng hiệu quả mỗi phút trong toàn quá trình chơi.

Đi nhiên chỉ có thể 6 người ra sân nên huấn luyện viên phải chọn đội hình xuất phát và sau đó – ở thời điểm cần thiết, cho thay người.

Hãy xác định hiệu quả lớn nhất có thể đạt được trong toàn cuộc chơi, những người ra sân trong đội hình xuất phát, số lần cần thay người, các thời điểm thay người và cặp cầu thủ của phép thay người.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên m và n ($1 \leq m \leq 5 \times 10^5$, $6 \leq n \leq 5 \times 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên k_i và $1k_i$ ($1 \leq k_i \leq 10^5$, $1 \leq 1k_i \leq m$).

Dữ liệu đảm bảo có lời giải.

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

- ✿ Dòng thứ nhất chứa một số nguyên – hiệu quả lớn nhất có thể đạt được,
- ✿ Dòng thứ 2 chứa 6 số nguyên xác định số thứ tự của các cầu thủ ở đội hình xuất phát,
- ✿ Dòng thứ 3 chứa số nguyên s – số lần thay người,
- ✿ Mỗi dòng trong s dòng sau chứa 3 số nguyên – thời điểm thay, người ra và người vào.

Nếu có nhiều phương án cùng cho hiệu quả lớn nhất – đưa ra phương án tùy chọn.

Ví dụ:

SIXPASS.INP	SIXPASS.OUT
3 9	1610
100 3	1 2 3 4 5 6
100 3	2
100 3	1 6 8
100 3	2 5 7
100 2	
100 1	
50 1	
30 2	
1 1	



Giải thuật: Nguyên lý cực trị.

Mỗi cầu thủ tương ứng với một bản ghi:

<i>Chỉ số hiệu quả</i>	<i>Thời gian chơi</i>	<i>Id</i>
<i>k_i</i>	<i>lk_i</i>	<i>i</i>

Sắp xếp các bản ghi theo thứ tự giảm dần.

Chọn đội hình xuất phát:

Ban đầu: Đội hình xuất phát là 6 người có hiệu cao nhất, với những người cùng hiệu quả – chọn người có thời gian chơi liên tục dài nhất.

Với mỗi người: Xét hai lần:

- ★ Thời điểm đưa vào đội hình,
- ★ Thời điểm cần thay thế.

Xác định người vào thay thế: người còn lại với hiệu quả cao nhất.

Tổ chức dữ liệu:

Mảng **vector<int>** player (n) – ghi nhận dữ liệu vào,

Mảng **vector<int>** team – ghi nhận đội hình xuất phát,

Mảng **vector<pii>** res [500001] – ghi nhận cặp người trong phép thay ở mỗi thời điểm của trận đấu.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define NAME "sixpass."
#define ff first
#define ss second
using namespace std;
typedef tuple<int,int,int> t3i;
typedef pair<int,int> pii;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int m,n,msb,k,lk,id;
vector<int> team;
vector<pii> res[500001];
int64_t ans=0;
int main()
{
    fi>>m>>n;
    vector<t3i> player(n);
    for(int i=0; i<n; ++i)
    {
        fi>>k>>lk;
        player[i]=make_tuple(k,lk,i+1);
    }
    sort(player.rbegin(),player.rend());
    int it=0,tsum=0,numsub=0,prev=-1;
    for(int i=0;i<n && it<6; ++i)
    {
        tie(k,lk,id)=player[i];
        if(lk==0) continue;
        int tmp=min(lk,m-tsum);
        if(tsum==0) team.push_back(id);
        else
            if(it!=5 && lk==m)
            {
                ++it;
                team.push_back(id);
                ans+=(int64_t) k*m;
                continue;
            }
            else {
                ++numsub;
                res[tsum].push_back({prev,id});
            }
        tsum+=tmp;
        ans+=(int64_t) k*tmp;
        player[i]=make_tuple(k,lk-tmp,id);
        prev=id;
        if(tsum==m) {++it; tsum=0;}
        --i;
    }
    fo<<ans<<'\n';
    for(int ii:team) fo<<ii<<' ' ; fo<<'\n';
    fo<<numsub<<'\n';
    for(int i=1; i<m;++i)
    {
        reverse(res[i].begin(),res[i].end());
        for(int j=0; j<res[i].size(); ++j)
            fo<<i<<' '<<res[i][j].ff<<' '
                            <<res[i][j].ss<<'\n';
    }
}
```



Marina ngồi với máy tính cạnh bô tải, về chương trình làm phim hoạt hình 3D và xây dựng một phim hoạt hình đơn giản: có một đoàn tàu chở n người xuất phát từ ga số 0, lần lượt chạy qua các ga 1, 2, 3, . . . Hành khách được đánh số từ 1 đến n . Sau đó Marina làm phong phú thêm nội dung bằng việc lần lượt đưa vào các yêu cầu mới, mỗi yêu cầu có dạng "**M x a**" – tàu dừng ở ga **x** để hành khách **a** xuống. Dĩ nhiên, không có hành khách nào xuống 2 lần.

Rất thích thú với kết quả của mình Marina thường gọi bô nhìn xem cảnh tàu dừng và hành khách xuống xe. Để tránh việc bị gián đoạn trong công việc, thỉnh thoảng bô Marina đưa ra một câu hỏi làm Marina phải suy nghĩ, tính toán hồi lâu và để cho bô được yên. Câu hỏi đưa ra có dạng "**D y b**" – trong số các hành khách có số không nhỏ hơn **b** hãy xác định người có số nhỏ nhất đã xuống và đi không quá **y** ga. Nếu không có ai như vậy thì đưa ra số -1. Những câu hỏi này đã không làm giảm sự hào hứng của Marina đồng thời bô cũng được tập trung vào công việc của mình trong một thời gian dài.

Với mỗi câu hỏi của bô, hãy đưa ra câu trả lời của Marina.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m , trong đó m – số truy vấn ($2 \leq n, m \leq 2 \times 10^5$),
- ✚ Mỗi dòng trong m dòng sau chứa một truy vấn, các tham số đảm bảo hợp lệ.

Kết quả: Đưa ra file văn bản DADY.OUT các câu trả lời của Marina dưới dạng số nguyên, mỗi số trên một dòng.

Ví dụ:

DADY.INP	DADY.OUT
10 10	-1
M 20 10	-1
D 1 9	3
M 2 3	2
D 17 10	9
M 20 2	
D 8 2	
M 40 1	
D 25 2	
M 33 9	
D 37 9	



Giải thuật: Ứng dụng Treap với khóa tương minh .

Mỗi truy vấn được đặc trưng bằng một cặp dữ liệu (**x**, **pr**), trong đó **x** – ga, **pr** – người,

Cấu trúc Treap **t** cần xây dựng phải quản lý người như một *cây nhị phân* và quản lý ga như một *Heap min*,

Để thuận tiện sử dụng các công cụ chuẩn, thay vì **x** ta sẽ lưu trữ **-x** và sẽ có một *Treap chuẩn* với Heap max.

Treap **t** sử dụng **pr** như khóa và **-x** – mức ưu tiên.

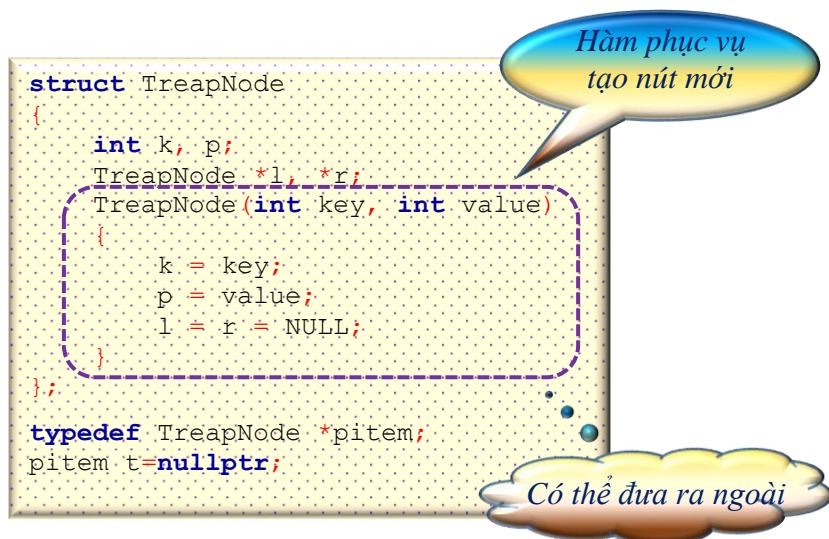
Mỗi truy vấn dạng “**M x pr**” tương ứng với việc nạp phần tử (**pr**, **-x**) vào cây.

Mỗi truy vấn dạng “**D x pr**” tương ứng với việc trích ra cây con chỉ chứa các phần tử có khóa không nhỏ hơn **pr**, xác định phần tử có khóa nhỏ nhất trong cây con với mức ưu tiên không vượt quá **x**.

Theo giải thuật này, giá trị **n** không đóng vai trò quan trọng, chỉ phục vụ xác định kiểu dữ liệu trong khai báo biến.

Tổ chức dữ liệu:

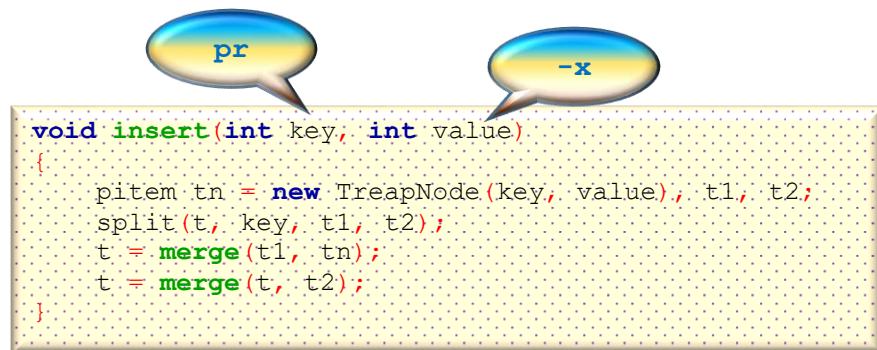
Phân tử của cây có dạng:



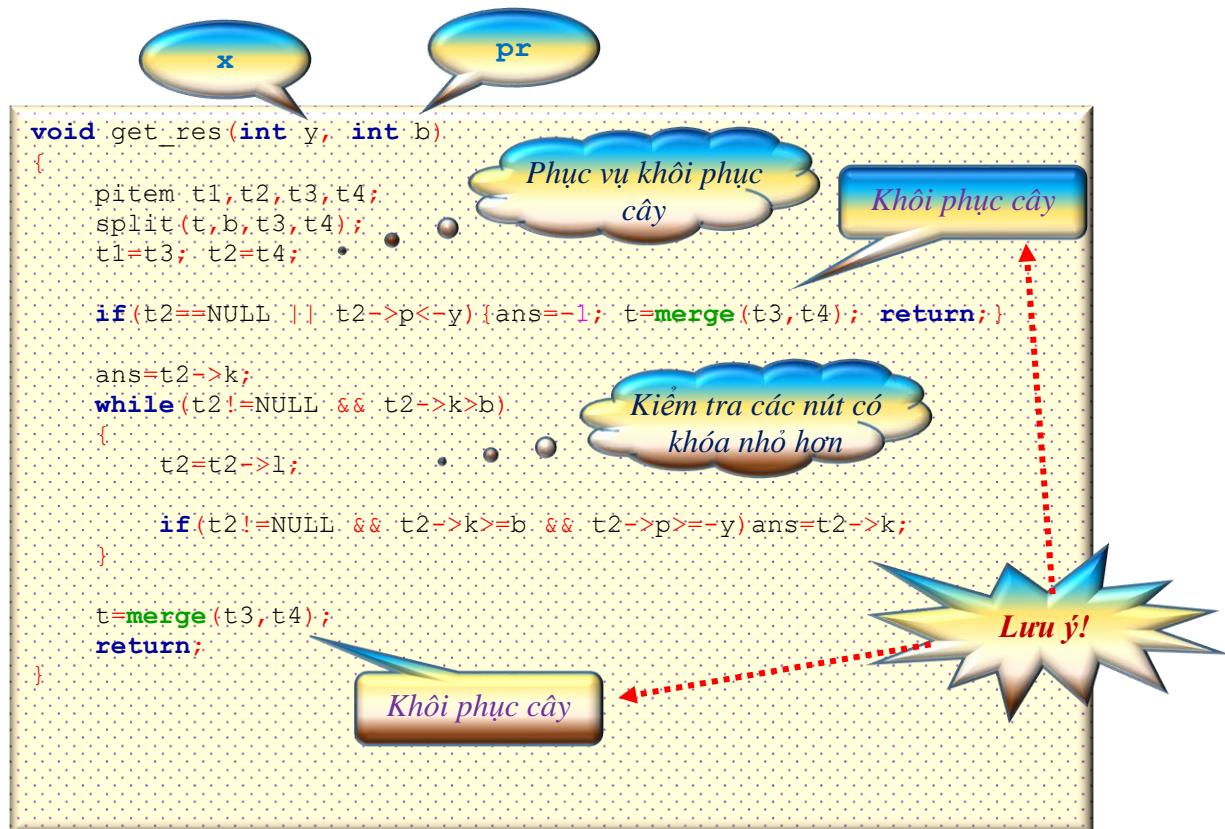
Xử lý:

Sử dụng các hàm chuẩn **merge** và **split** của cấu trúc,

Xuất phát từ cây rỗng, gấp truy vấn dạng **M x pr** → bô sung thêm nút mới:



Xử lý truy vấn **D x pr**:



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

Chương trình

```
#include <bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
#define NAME "dady."
using namespace std;
ifstream fi (NAME"inp");
//ifstream fi ("07");
ofstream fo (NAME"out");
int ans;
struct TreapNode
{
    int k, p;
    TreapNode *l, *r;
    TreapNode(int key, int value)
    {
        k = key;
        p = value;
        l = r = NULL;
    }
};
typedef TreapNode *pitem;
pitem t=nullptr;

pitem merge(pitem a, pitem b)
{
    if (!a || !b)
        return a ? a : b;
    if (a->p > b->p)
    {
        a->r = merge(a->r, b);
        return a;
    } else
    {
        b->l = merge(a, b->l);
        return b;
    }
}

void split(pitem t, int k, pitem &a, pitem &b)
{
    if (!t)
        a = b = NULL;
    else if (t->k < k)
    {
        split(t->r, k, t->r, b);
        a = t;
    } else
    {
        split(t->l, k, a, t->l);
        b = t;
    }
}

void insert(int key, int value)
{
    pitem tn = new TreapNode(key, value), t1, t2;
    split(t, key, t1, t2);
    t = merge(t1, tn);
    t = merge(t, t2);
}

void get_res(int y, int b)
```

```

{
    pitem pr,t1,t2,t3,t4;
    split(t,b,t3,t4);
    t1=t3; t2=t4;

    if(t2==NULL || t2->p<-y) {ans=-1; t=merge(t3,t4); return;}
    ans=t2->k;
    while(t2!=NULL && t2->k>b)
    {
        t1=t2;t2=t2->l;

        if(t2!=NULL && t2->k>=b && t2->p>=-y) ans=t2->k;
    }
    t=merge(t3,t4);
    return;
}

int main()
{
    int n,m,pr,x,y;
    char c;
    fi>>n>>m;
    for(int i=0;i<m;++i)
    {
        fi>>c>>x>>pr;
        if(c=='M') insert(pr,-x);
        else
        {
            get_res(x,pr);
            fo<<ans<<'\\n';
        }
    }
    Times;
}

```



Trong chuyến đi du lịch nước ngoài đoàn của Steve được đưa tới một cửa hàng đồ lưu niệm. Sau khi chọn xong một số thứ ưng ý Steve ra quầy thanh toán. Đã có một dòng dài người xếp hàng chờ đến lượt. Cũng may là có một số người thanh toán riêng lẻ, nhưng cũng có nhóm thanh toán chung một lượt, vì vậy thời gian chờ cũng không quá lâu. Steve nhận thấy là những người thanh toán chung là cùng một đoàn và mặc áo phông màu giống nhau, đứng liên tiếp trong hàng. Những người thanh toán riêng có màu áo không giống người đứng trước (nếu có) và người đứng sau.

Từ những quan sát trên Steve nhầm tính được lượt thanh toán của mình.

Hãy xác định lượt thanh toán của Steve,

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

- ✚ Dòng đầu tiên chứa số nguyên n – số người trong hàng trước Steve ($1 \leq n \leq 10^6$),
- ✚ Dòng thứ i trong n dòng sau chứa một chữ cái la tinh in hoa xác định màu áo của người thứ i trong hàng.

Kết quả: Đưa ra file văn bản TURN.OUT một số nguyên xác định lượt thanh toán của Steve.

Ví dụ:

TURN.INP	TURN.OUT
6	5
C	
C	
P	I
C	
Z	
Z	



Giải thuật; Cơ sở lập trình.

Lần lượt nhập từng ký tự, so sánh với ký tự trước đó, nếu khác nhau → tăng kết quả lên 1 và biến ký tự mới thành ký tự trước.

Chuẩn bị: gán kết quả bằng 1 và ký tự bất kỳ khác chữ cái la tinh in hoa cho ký tự trước.

Độ phức tạp giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define NAME "turn."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,ans=1;
char c,c1;

int main()
{
    fi>>n; c1='#';
    for(int i=0; i<n; ++i)
    {
        fi>>c;
        if(c!=c1)++ans, c1=c;
    }
    fo<<ans;
    Times;
}
```



Cho xâu s chỉ chứa các ký tự là tinh thường độ dài n không quá 10^5 . Ký hiệu $s[i..j]$ là xâu con các ký tự liên tiếp nhau của s từ vị trí i đến vị trí j . Hai xâu con $x = s[a..b]$ và $y = s[c..d]$ được gọi là cùng một lớp tương đương nếu có cách đổi chỗ các ký tự trong một xâu để nhận được xâu kia. Các vị trí được đánh số từ 1 trở đi.

Ví dụ với $s = “abcbacaac”$, $s[1..3]$ tương đương với $s[4..6]$, còn $s[1..3]$ không tương đương với $s[5..7]$.

Cho q truy vấn, mỗi truy vấn là 4 số nguyên a, b, c, d xác định các xâu con x và y . Với mỗi truy vấn hãy xác định hai xâu con này có cùng một lớp tương đương hay không và đưa ra câu trả lời tương ứng “YES” hoặc “NO”.

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

- ✚ Dòng đầu tiên chứa xâu s ,
- ✚ Dòng thứ 2 chứa số nguyên q ($1 \leq q \leq 10^5$),
- ✚ Mỗi dòng trong q dòng sau chứa 4 số nguyên a, b, c, d ($1 \leq a \leq b \leq n, 1 \leq c \leq d \leq n$).

Kết quả: Đưa ra file văn bản TWOSTR.OUT các câu trả lời xác định được cho mỗi truy vấn, mỗi câu trả lời trên một dòng.

Ví dụ:

TWOSTR.INP	TWOSTR.OUT
abcbacaac 2 1 3 4 6 1 3 5 7	YES NO



Giải thuật; Kỹ thuật hàm băm, Tổng tiền tố.

Chọn **h** nguyên dương đủ lớn,

Xây dựng hàm tổng tiền tố **f** theo xâu **s**, thay ‘**a**’ bằng **h¹**, ‘**b**’ bằng **h²**, . . . , ‘**z**’ bằng **h²⁶**.

Xác định 2 xâu con **s[a..b]** và **s[c..d]** tương đương: kiểm tra điều kiện

$$f_b - f_{a-1} = f_d - f_{c-1}$$

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define NAME "twostr."
using namespace std;
ifstream fi (NAME"inp");
//ifstream fi ("10");
ofstream fo (NAME"out");
const int64_t h=50003;
int n,q,a,b,c,d;
int64_t p[27],x,y;
string s;

int main()
{
    fi>>s;
    n=s.size();
    vector<int64_t> f(n+1,0);
    p[0]=1;
    for(int i=1; i<=26; ++i) p[i]=p[i-1]*h;
    for(int i=1; i<=n; ++i) f[i]=f[i-1]+p[s[i-1]-96];
    fi>>q;
    for(int i=0; i<q; ++i)
    {
        fi>>a>>b>>c>>d;
        x=f[b]-f[a-1]; y=f[d]-f[c-1];
        fo<<(x==y? "YES\n" :"NO\n");
    }
    Times;
}
```



Một số nguyên n có thể biểu diễn ở cơ số 3 với các chữ số 0, 1 và 2. Ví dụ $6_{10} = 20_3$. Trong một số trường hợp, để thuận tiện xử lý, người ta có thể biểu diễn số n bằng hệ cơ số đối xứng (cơ số $3a$) với các chữ số a , 0 và 1, trong đó $a = -1$. Ví dụ $6_{10} = 20_3 = 1a0_{3a}$.

Yêu cầu: cho số nguyên n ở hệ cơ số 10. Hãy đưa ra n ở các dạng biểu diễn cơ số 3 và $3a$.

Dữ liệu: Vào từ file văn bản BASE3.INP gồm một dòng chữ số nguyên n ($|n| \leq 10^{18}$).

Kết quả: Đưa ra file văn bản BASE3.OUT các dạng biểu diễn của n ở cơ số 3 và $3a$, mỗi dạng trên một dòng.

Ví dụ:

BASE3.INP	BASE3.OUT
6	20 1a0



Giải thuật; Cơ sở lập trình.

Gọi **x** – xâu biểu diễn số n ở cơ số 3, **y** – xâu biểu diễn số n ở cơ số 3a.

Xâu **y** nhận được từ **x** bằng cách duyệt từ cuối về đầu:

$$y_i = \begin{cases} x_i & \text{nếu } 0 \text{ hoặc } 1, \\ a & \text{và } ++x_{i-1} \text{ (theo cơ số 3) nếu } x_i = 2. \end{cases}$$

Dộ phức tạp của giải thuật: O(log₃n)

Chương trình

```
#include <bits/stdc++.h>
#define NAME "base3."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
typedef tuple<int,int,int> tii;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n;
int r;
bool flg;
string x,y;

int main()
{
    fi>>n; x=""; y=x; flg=n<0;
    if(n==0) {fo<<"0\n0"; return 0;}
    if(n<0) n=-n;
    while(n>0)
    {
        r=n%3; n/=3;
        x+=char(r+48);
    }
    if(flg) fo<<'-'; for(int i=x.size()-1; i>=0; --i) fo<<x[i];
    fo<<'\n';
    x+='0';
    for(int i=0; i<x.size()-1; ++i)
    {
        if(x[i]=='2') {x[i]='a'; ++x[i+1]; continue;}
        if(x[i]=='3') x[i]='0', ++x[i+1];
    }
    r=x.size(); if(x[r-1]=='0') --r;
    if(flg) fo<<'-'; for(int i=r-1; i>=0; --i) fo<<x[i]; fo<<'\n';
}
```



VY22. THI BẮN NHANH

Tên chương trình: RANGE.CPP

Để kiểm tra tốc độ phản xạ mỗi thí sinh phải bắn n phát đạn vào bia di động trong thời gian nhanh nhất có thể. Súng được dùng có hộp băng đạn chứa được m viên. Thời gian nạp đạn vào băng và lắp băng vào súng để bắn tiếp là a , không phụ thuộc vào việc có định lắp đạn đầy băng hay không. Người bắn có thể lắp từng viên đạn vào nòng với thời gian b giây cho một lần lắp. Thí sinh còn mất một giây để ngắm và bắn.

Ban đầu súng không có đạn.

Hãy xác định thời gian ngắn nhất hoàn thành bài thi của thí sinh.

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

Kết quả: Đưa ra file văn bản RANGE.OUT thời gian ngắn nhất xác định được.

Ví dụ:

RANGE.INP	RANGE.OUT
3 2 1 1	5



VY22 Io20181014 A A XIV

Giải thuật; Cơ sở lập trình.

Thời gian cần tìm bằng tổng thời gian nạp đạn và thời gian bắn.

Thời gian bắn cố định là n , như vậy thời gian hoàn thành bài thi chỉ phụ thuộc vào thời gian nạp đạn,

Nếu $a \geq b \times m \rightarrow$ không sử dụng cách nạp đạn vào băng,

Trong trường hợp ngược lại:

- Sử dụng cách nạp đạn vào băng n/m lần,
- Phần còn lại ($n \% m$):
 - ✿ Nếu nạp đạn vào băng: mất a giây,
 - ✿ Nếu nạp đạn trực tiếp vào nòng: mất $(n \% m) \times b$,
- Trong hai cách: lựa chọn cách ít thời gian hơn.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "range."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,m,a,b,k,ans;

int main()
{
    fi>>n>>m>>a>>b;
    if(a>= m*b) ans=n*(b+1);
    else
    {
        k = n%m;
        k = min(a,k*b);
        ans = (n/m)*a + k + n;
    }
    fo<<ans;

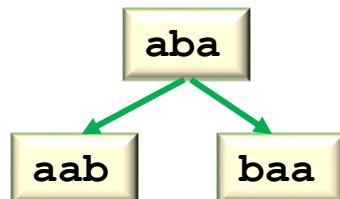
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Steve đăng nhập vào nhiều trang WEB khác nhau để tham gia vào các kỳ thi Tin học trên mạng. Mỗi trang WEB đều cần một mật khẩu đăng nhập. Ngoài ra, các trang WEB này đều định kỳ đề nghị người dùng thay đổi mật khẩu để bảo mật và loại bỏ những người không còn tham gia.

Steve đặt mật khẩu theo quy tắc gồm tên mình, sau đó là một số nguyên. Để khỏi quên, Steve lưu trong phần trợ giúp một xâu **s** chỉ chứa các ký tự la tinh thường. Số nguyên cần viết sau tên sẽ là số xâu khác nhau có thể nhận được từ **s** bằng cách đổi chỗ 2 ký tự của xâu.

Ví dụ, **s** = “**aba**”, ta có 3 xâu khác nhau có thể nhận được theo quy tắc trên:



Cho xâu **s**. Hãy xác định số nguyên cần viết sau tên ở mật khẩu đăng nhập.

Dữ liệu: Vào từ file văn bản CIPHER.INP gồm một dòng chứa xâu **s** khác rỗng độ dài không quá 10^5 .

Kết quả: Đưa ra file văn bản CIPHER.OUT một số nguyên – số cần tìm.

Ví dụ:

CIPHER.INP	CIPHER.OUT
abacaba	15



Giải thuật; Cơ sở lập trình.

Xét ký tự thứ i trong xâu s ($i = 0 \div s.size()$).

Có bao nhiêu $j < i$ và $s_j \neq s_i$ thì có bấy nhiêu các đổi chỗ để nhận được *xâu mới khác s*.

Gọi $f[s_i]$ là số lượng ký tự s_i có mặt trong đoạn $[0 \dots i]$.

Số lượng xâu mới nhận được bằng các đổi chỗ ký tự s_i với các ký tự trước đó của xâu là $i - (f[s_i] - 1)$.

Như vậy chỉ cần tích lũy giá trị nói trên với mỗi s_i ta sẽ nhận được kết quả cần tìm.

Lưu ý: bản xâu s ban đầu cũng thuộc tập các xâu khác nhau nhận được từ s .

Tổ chức dữ liệu:

- ▀ Xâu **string** s – lưu dữ liệu vào,
- ▀ Mảng **int64_t** $f[26] = \{0\}$ – lưu tần số xuất hiện các ký tự trong quá trình duyệt.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "cipher."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,k;
int64_t ans=1,f[26]={0};
string s;

int main()
{
    fi>>s;
    for(int i=0; i<s.size(); ++i)
    {
        k=s[i]-97;
        ++f[k];
        ans += i-f[k]+1;
    }
    fo<<ans;

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



VY24. TRÒ CHƠI JENGA

Tên chương trình: JENGA.CPP

Từ các thanh gỗ hình hộp chữ nhật giống nhau có mặt đáy kích thước 1×3 người ta xếp thành các khối kích thước 3×3 bằng cách ghép dọc hoặc ghép ngang chúng. Các khối 3×3 được xếp chồng đan xen lớp dọc/lớp ngang thành một hình trụ, mỗi khối 3×3 được gọi là một lớp. Các người chơi, khi đến lượt mình, rút một miếng gỗ ra khỏi hình trụ từ lớp tùy chọn, trừ lớp trên cùng và lớp dưới cùng. Hình trụ bị đổ nếu 2 thanh kè cạnh ở cùng một lớp bị rút ra. Ai làm hình trụ đổ là thua.

Johnny và Lorna xây một hình trụ n lớp và bắt đầu rút các thanh gỗ. Johnny được đi trước.

Hãy xác định ai là người thắng cuộc.

Dữ liệu: Vào từ file văn bản JENGA.INP gồm một dòng chứa số nguyên n ($3 \leq n \leq 10^7$).



Kết quả: Đưa ra file văn bản JENGA.OUT tên người thắng cuộc (**Johnny** hoặc **Lorna**).

Ví dụ:

JENGA.INP	JENGA.OUT
3	Johnny



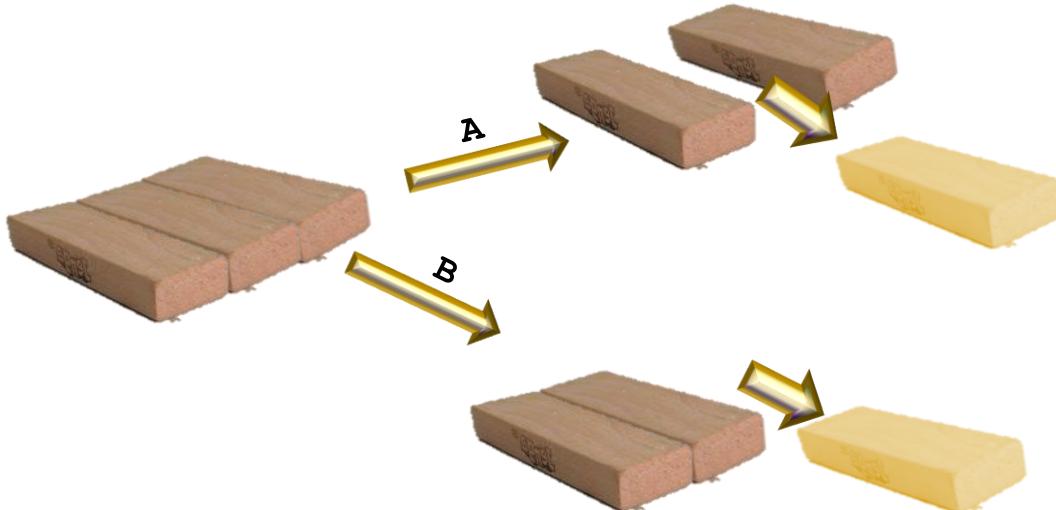
VY24_Io20181014_D_A XIV

Giải thuật; Cơ sở lập trình, Lý thuyết trò chơi, Định lý Grundy.

Số lớp cho phép rút các thanh gỗ: $n-2$.

Từ *trạng thái ban đầu của lớp* người chơi có 2 cách đi:

- ⊕ Cách A : Làm giảm số lớp được phép lựa chọn 1,
- ⊕ Cách B: Không làm giảm số lớp được phép lựa chọn.



Trường hợp n lẻ: Người đi trước thực hiện nước đi đầu tiên theo cách **A**, để lại một số lượng chẵn các lớp có thể được chọn, tương ứng với một số chẵn cách đi. Nếu người thứ 2 còn cách đi thì người thứ nhất cũng còn cách đi và để lại một chẵn cách đi. Nếu hết cách đi – người thứ 2 bị hết trước và thua.

Cách đi của người thứ nhất sau nước đi đầu tiên: lặp lại cách đi của người thứ 2 ở một lớp khác.

Trường hợp n chẵn: Sau cách đi bất kỳ của người thứ nhất luôn còn lại một số lẻ cách đi để kết thúc trò chơi: Người thứ 2 lặp lại cách đi của người thứ nhất ở lớp khác!

Như vậy, nếu n lẻ - đưa ra thông báo **Johnny**, trong trường hợp ngược lại – đưa ra thông báo **Lorna**.

Độ phức tạp của giải thuật: $O(1)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "jenga."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;
int64_t ans=1,f[26]={0};
string s;

int main()
{
    fi>>n;
    fo<<((n&1) ? "Johnny" : "Lorna");

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



Đồ gỗ khám trai là một mặt hàng mỹ nghệ xuất khẩu rất được ưa chuộng trên thế giới. Một nghệ nhân có đứa cháu cũng rất ham thích kỹ nghệ khám và ông quyết định dành nhiều thời gian để truyền nghề cho cháu.

Lô hàng cành khám nhận được lần này là $2 \times k$ chiếc lọ hoa gỗ cành khám tranh. Hai ông cháu cùng làm việc và Ông quyết định mình sẽ làm k cái, k cái còn lại sẽ do cháu làm. Người ông bắt tay vào làm việc trước, vừa làm Ông vừa giảng giải cho cháu các tiểu xảo nghệ thuật. Sau khi khám xong một lọ hoa hoặc đang có ngẫu hứng sáng tạo ông làm liền khám liền 2 lọ hoa sau đó mới dừng lại giảng giải cho đứa cháu, còn người cháu thì là xong một hoặc 2 lọ hoa lại nghĩ tay xin nhận xét, góp ý của ông. Mỗi lần dừng lại trao đổi hai người lại viết vào sổ số lọ hoa vừa hoàn thành. Kết quả trong sổ có n số nguyên a_1, a_2, \dots, a_n ghi số lọ hoa hoàn thành sau mỗi khoảng thời gian làm việc liên tục.

Tồn tại một khoảng thời gian giữa 2 lần nghỉ số lọ hoa đã khám của người ông vượt trội hơn hẳn số lọ hoa đã làm của cháu.

Hãy xác định sự chênh lệch tối đa có thể xảy ra.

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

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

Kết quả: Đưa ra file văn bản MOSAIC.OUT một số nguyên – chênh lệch tối đa xác định được.

Ví dụ:

MOSAIC.INP	MOSAIC.OUT
<pre>3 2 1 2 1</pre>	<pre>1</pre>



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

Các số **a_i** chỉ nhận giá trị 1 hoặc 2 nên chênh lệch lớn nhất xảy ra khi người ông khám hết **k** lọ hoa của mình mà người cháu chưa bắt đầu hoặc mới khám được một lọ.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include<bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define NAME "mosaic."
using namespace std;
ifstream fi (NAME"int");
ofstream fo (NAME"out");
int n,k,x,s=0;

int main()
{
    fi>>n>>k;
    while(s<k)
    {
        fi>>x;
        s+=x;
    }
    if(s==k) fo<<k; else fo<<k-1;
    Times;
}
```



B10. NHIÊN LIỆU SẠCH

Tên chương trình: FUEL.CPP

Một nhóm nhà khoa học đã tìm ra cách chế tạo nhiên liệu sạch hiệu năng cao, mỗi lít nhiên liệu cho phép ô tô chạy được 100 km. Để chế tạo nguyên liệu một số hỗn hợp thành phần được bơm thẳng vào thùng chứa nhiên liệu của xe, một số hỗn hợp khác – lưu trong các bình đặc biệt ở phòng thí nghiệm. Vì vậy xe phải được đưa vào phòng thí nghiệm mới sản xuất được nhiên liệu. Quy định an toàn cháy nổ chỉ cho phép tổng hợp mỗi lần một lít nhiên liệu – vừa đúng bằng bình chứa của xe. Tuy vậy, khi ra khỏi phòng thí nghiệm nhiên liệu có thể được chiết sang bình khác đặt ở ven đường để đổ vào xe nếu bình nhiên liệu trên xe còn có khả năng chứa. Xe cũng không được chở theo các bình nhiên liệu dự trữ.

Các nguyên vật liệu hiện có chỉ cho phép sản xuất **a** lít nhiên liệu. Bằng cách chiết nguyên liệu từ xe sang các bình dự trữ ở ven đường, xe có thể quay về phòng thí nghiệm tạo nhiên liệu mới hoặc đổ tiếp nhiên liệu từ các bình đã chiết trước đó ven đường để đi tiếp.

Hãy xác định xe có thể đi cách xa phòng thí nghiệm tối đa bao nhiêu km.

Dữ liệu: Vào từ file văn bản FUEL.INP gồm một dòng chứa số thực **a** với 6 chữ số sau dấu chấm thập phân ($1.0 \leq a \leq 2.0$).

Kết quả: Đưa ra file văn bản FUEL.OUT một số thực với độ chính xác 6 chữ số sau dấu chấm thập phân – khoảng cách xa nhất khỏi phòng thí nghiệm có thể đi được.

Ví dụ:

FUEL.INP	FUEL.OUT
1.000000	100



B10.Io20181020 A

A XIV

Giải thuật; Cơ sở lập trình, Khả năng phân tích giải thuật .

Nếu $a \leq 1$ thì không cần san chiết để dành và khoảng cách xa nhất đi được sẽ là $100 \times a$.

Trường hợp $1 < a \leq 2$:

Cần san chiết để dành vì ngay từ đầu không thể đổ hết nhiên liệu vào xe,

Số lần quay trở về càng ít càng tốt để tránh tốn nhiên liệu đi nhiều lần trên cùng một quãng đường → chỉ quay về một lần,

Gọi d – quãng đường cần đi từ phòng thí nghiệm tới vị trí dừng san chiết nhiên liệu, x là chi phí nhiên liệu đi tới đó.

Sau khi tiếp nhiên liệu dọc đường ta cần có đầy bình để đi được xa nhất.



Như vậy dễ dàng thấy rằng $x = (a-1)/3$ và khoảng cách xa nhất có thể đạt được là $100(1 + (a-1)/3)$.

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int main()
{
    double a;
    fi >> a;
    fo<<fixed<<setprecision(6);
    if (a <= 1)    fo << 100 * a ;
    else
        fo << 100 + 100 * (a - 1) / 3 ;
    Times;
}
```



VY25. BÀN CỜ

Tên chương trình: CHESSBOARD.CPP

Một nhóm bạn trẻ định muốn tạo một kỷ lục để được ghi tên vào sách Kỷ lục Guinness với thành tích làm được bàn cờ lớn nhất thế giới. Bàn cờ phải là một hình vuông kích thước $k \times k$, mỗi ô trong bàn cờ có kích thước 1×1 , có một trong hai màu: trắng hoặc đen, 2 ô kề cạnh phải có màu khác nhau.

Các bạn đã chuẩn bị được n ô màu trắng và m ô màu đen, mỗi ô có kích thước 1×1 .

Hãy xác định kích thước k lớn nhất của bàn cờ mà các bạn trẻ có thể làm được.

Dữ liệu: Vào từ file văn bản CHESSBOARD.INP gồm một dòng chứa 2 số nguyên n và m ($0 \leq n, m \leq 10^9, 0 < n+m$).

Kết quả: Đưa ra file văn bản CHESSBOARD.OUT một số nguyên – giá trị k tìm được.

Ví dụ:

CHESSBOARD.INP	CHESSBOARD.OUT
8 9	4



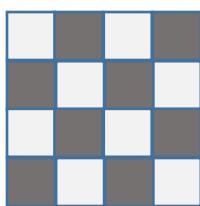
VY24_Io20181020_A1_A XIV

Giải thuật; Duyệt vét cạn .

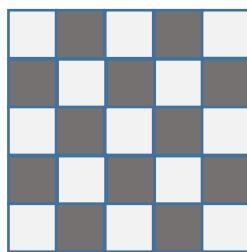
Nếu $n = m = 0 \rightarrow$ không tạo được bàn cờ.

Nếu k chẵn thì số ô đen bằng số ô trắng và bằng $k^2/2$.

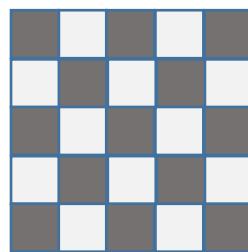
Nếu k lẻ thì có thể có 2 loại bàn cờ với số ô trắng và ô đen tương ứng là $(k^2+1)/2$ và $(k^2-1)/2$ hoặc $(k^2-1)/2$ và $(k^2+1)/2$.



Số ô trắng: $k^2/2$
Số ô đen: $k^2/2$



Số ô trắng: $(k^2+1)/2$
Số ô đen: $(k^2-1)/2$



Số ô trắng: $(k^2-1)/2$
Số ô đen: $(k^2+1)/2$

Lần lượt kiểm tra tính khả thi với kích thước $1, 2, \dots, 2 \times \max(n, m)$.

Độ phức tạp giải thuật: lớp $O(n^{1/2})$.

Chương trình

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

int main()
{
    int n, m;
    fi >> n >> m;
    int x = max(n, m);
    if (x == 0)
    {
        fo<<"-1";
        return 0;
    }
    int ans = 0;
    for (int i = 1; i * i <= 2 * x; i++)
    {
        int64_t sq = (int64_t)(i) * i;
        if ((i % 2 == 0 && n >= sq / 2 && m >= sq / 2) ||
            (i % 2 == 1 && ((n >= sq / 2 && m >= sq / 2 + 1) ||
                           (n >= sq / 2 + 1 && m >= sq / 2)))) {
            ans = i;
        }
    }
    fo<<ans;
    return 0;
}
```



Trong hội xuân có chương trình thi đấu giữa đo vật vô địch tinh với các trai làng. Có n chàng trai muốn thử sức với nhà vô địch, đứng xếp hàng trước xới đấu. Với kinh nghiệm dày dạn của mình nhà vô địch chỉ nhìn lượt qua là đánh giá được tiềm lực của mỗi người, cụ thể, người thứ i có tiềm lực s_i , $i = 1 \dots n$.

Với tiềm lực T của mình, sau khi thi đấu với người thứ i , tiềm lực của nhà vô địch sẽ giảm xuống còn $\lfloor T/s_i \rfloor$. Nếu tiềm lực bị giảm xuống bằng 0 nhà vô địch sẽ thua và bị loại khỏi cuộc chơi, không thi đấu tiếp nữa.

Các chàng trai không phải là những vận động viên chuyên nghiệp, vì vậy sự chờ đợi sót ruột và căng thẳng đã làm cho các người từ vị trí $1f$ đến rt bị giảm đi 1, cụ thể là $s_i = \max\{s_{i-1}, 1\}$, $i = 1f \dots rt$.

Nhà vô địch quan sát thấy sự suy giảm tiềm lực ở một số chàng trai và nhầm tính với tiềm lực hiện tại x của mình, nếu chọn thi đấu với các chàng trai từ vị trí 1_j đến vị trí x_j và lần lượt đấu từ trái sang phải với những người trong đọa đó thì có bị thua hay không, nếu bị thua thì là ở dưới tay ai.

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

- ⊕ Dòng đầu tiên chứa 2 số nguyên n và q , trong đó q – số truy vấn cần xử lý ($1 \leq n, q \leq 5 \times 10^5$),
- ⊕ Dòng thứ 2 chứa n số nguyên s_1, s_2, \dots, s_n ($1 \leq s_i \leq 10^9$, $i = 1 \dots n$),
- ⊕ Mỗi dòng trong q dòng sau chứa thông tin ở một trong 2 dạng:
* $1 \ 1f \ rt \ x$ – tiềm lực các người từ $1f$ đến rt bị giảm đi 1,

2 1f rt x – nhà vô địch có tiềm lực x và chọn thi đấu lần lượt với các người từ $1f$ đến rt .

Kết quả: Đưa ra file văn bản ATTEMPT.OUT: Với mỗi truy vấn loại 2 đưa ra trên một dòng thông báo **-1** – nhà vô địch thắng hoặc một số nguyên – người hạ đo ván nhà vô địch.

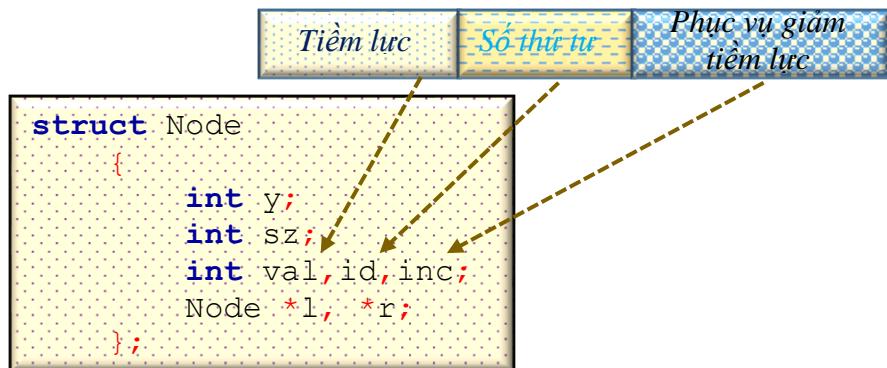
Ví dụ:

ATTEMPT.INP	ATTEMPT.OUT
6 4	-1
1 2 3 2 3 1	3
2 1 6 61	-1
2 1 3 2	
1 1 3	
2 1 3 2	



Giải thuật: Treap khóa ẩn và cây Fenwick .

Thông tin về mỗi đo vật được lưu trữ dưới dạng bản ghi:



Khi nhập thông tin chỉ lưu lại những người có tiềm lực lớn hơn 1.

Những người có tiềm lực 1 sẽ không thay đổi trong quá trình xử lý và không làm thay đổi kết quả xử lý vì vậy không được lưu lại trong mảng thông tin về đồ vật và được đánh dấu trong cây Fenwick.

Xử lý truy vấn loại 1:

Từ **lf** và **rt** dựa vào cây Fenwick tính **a**, **b** – các số thứ tự tương ứng trong treap,

Gọi hàm **increase (tp, a, b, -1)** giảm tiềm lực các phần tử từ **a** đến **b**.

Xử lý truy vấn loại 2:

Nếu tồn tại các bản ghi có tiềm lực **val** ≤ 1 nhận dạng ở phép xử lý loại 2 trước đó: xóa khỏi treap,

Tính a và b tương ứng với lf, rt,

Tách cây chứa đoạn a ÷ b,

Duyệt các phần tử của cây, nếu nút có val $\leq 1 \rightarrow$ đưa số thứ tự vào danh sách xóa nút,

Nếu nút có val > 1 : cập nhật x=val, phép cập nhật này không xảy ra quá 20 lần cho mỗi truy vấn (vì $x \leq 10^9$),

Nếu có x=0 \rightarrow ngừng duyệt và đưa ra id tương ứng.

Trong trường hợp sau khi duyệt vẫn có x > 0 \rightarrow đưa ra -1.

Tổ chức dữ liệu:

- ▀ Mảng **vector<int>** **fw** – phục vụ tổ chức cây Fenwick,
- ▀ Cây Node ***tp=nullptr** – phục vụ tổ chức treap,
- ▀ Mảng **vector<pair<int, int>>** **del** – lưu trữ các nút cần xóa phát hiện khi xử lý truy vấn loại 2.

Độ phức tạp của giải thuật: O(nlogn).

Chương trình I

```
#include <bits/stdc++.h>
#pragma comment(linker, "/STACK:66777216")
#pragma comment(linker, "/HEAP:66777216")
#define ff first
#define ss second
using namespace std;
ifstream fi ("attempt.inp");
ofstream fo ("attempt.out");
vector<int> fw;
vector<pair<int,int> >del;
pair<int,int> p;
int n,q,x,lf,rt,a,b,t,flg;

void insert_fw(int k)
{
    while (k<=n) {++fw[k]; k+=(k&(-k)) ;}
}

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

struct Node
{
    int y;
    int sz;
    int val,id,inc;
    Node *l, *r;
};
Node *new_node(int val,int id)
{
    Node *result = new Node;
    result->y = rand();
    result->sz = 1;
    result->val = val;
    result->id = id;
    result->inc = 0;
    result->l = result->r = nullptr;
    return result;
}
Node *tp=nullptr,*ta,*tb,*tab,*tc,*td;

void get_Inc(Node *t)
{
    if(t==nullptr) return;
    t->val += t->inc;
    if (t->l != NULL) t->l->inc += t->inc;
    if (t->r != NULL) t->r->inc += t->inc;
    t->inc = 0;
}

int get_sz(Node *t)
{
    if (t == nullptr) return 0;
    return t->sz;
}

void upd_sz(Node *t)
```

```

    {
        if (t == nullptr) return;
        t->sz = 1 + get_sz(t->l) + get_sz(t->r);
    }

void push(Node *t)
{
    get_Inc(t);
}

void split(Node *t, int x, Node *&t1, Node *&t2)
{
    if (t == nullptr)
    {
        t1 = t2 = nullptr;
        return;
    }
    push(t);
    if (get_sz(t->l) < x)
    {
        split(t->r, x - get_sz(t->l) - 1, t->r, t2);
        t1 = t;
    }
    else
    {
        split(t->l, x, t1, t->l);
        t2 = t;
    }
    upd_sz(t);
}
Node *merge(Node *t1, Node *t2)
{
    push(t1); push(t2);
    if (t1 == nullptr) return t2;
    if (t2 == nullptr) return t1;
    push(t1); push(t2);
    if (t1->y > t2->y)
    {
        t1->r = merge(t1->r, t2);
        upd_sz(t1);
        return t1;
    }
    else
    {
        t2->l = merge(t1, t2->l);
        upd_sz(t2);
        return t2;
    }
}

Node *add(Node *t, int pos, int val, int id)
{
    Node *t1, *t2;
    split(t, pos, t1, t2);
    Node* new_tree = new_node(val, id);
    return merge(merge(t1, new_tree), t2);
}

Node* remove(Node *t, int pos, int num)
{
    Node *t1, *t2, *t3, *t4;
    split(t, pos, t1, t2);
    split(t2, num, t3, t4);
}

```

```

        t = merge(t1, t4);
        delete t3;
        return t;
    }

int get_value(Node *t, int pos)
{
    int my_idx = get_sz(t->l);
    if (pos < my_idx)
        return get_value(t->l, pos);
    else if (pos == my_idx) {p={t->val,t->id}; return 0;}
    else
        return get_value(t->r, pos - my_idx - 1);
}

void increase(Node *t, int l, int r, int val)
{
    Node *t1, *t2, *t3;
    split(t, l, t1, t2);
    split(t2, r-l+1, t2, t3);
    t2->inc=val;
    t = merge(t1, t2);
    t= merge(t, t3);
}

void check(Node *t)
{
    flg=0;
    for(int i=0; i< b; ++i)
    {
        int q=get_value(t,i);
        if(p.ff<=1) del.push_back({i,p.ss});
        else {x/=p.ff; if(x==0) {flg=1; fo<<p.ss<<'\\n'; break;}}
    }
    if(flg==0) fo<<"-1\\n";
}

int main()
{
    fi>>n>>q;
    fw.assign(n+2, 0);
    for(int i=1; i<=n; ++i)
    {
        fi>>x;
        if(x==1) {insert_fw(i); continue;}
        Node* new_tree = new_node(x,i);
        tp=merge(tp,new_tree);
    }

    for(int i=0; i<q; ++i)
    {
        fi>>t>>lf>>rt;
        a=lf-sum_fw(lf); if(a>0)--a;
        b=rt-sum_fw(rt); if(b>0)--b;
        if(t==1) increase(tp,a,b,-1);
        else
        {
            fi>>x;
            if(!del.empty())
            {
                for(int j=del.size()-1; j>=0; ++j)
                    {remove(tab,del[j].ff,1); insert_fw(del[j].ss);}
                del.clear();
            }
        }
    }
}

```

```

        }
        split(tp,a,ta,tab);
        b=b-a+1;
        split(tab,b,tab,tc);
        flg=0; check(tab);
        tp=merge(ta,merge(tab,tc));
    }
}
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
return 0;
}

```

Có thể tổ chức treap dưới dạng cây kiểm nhị phân với việc cài đặt các phép loại bỏ nút tương tự như ở Treap với khóa ẩn (cơ chế này thường được gọi là Cây quản lý đoạn với Lazy Update).

Chương trình II

```

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int dd = (int)1e6 + 7;

struct dsu
{
    int r[dd], p[dd], ind[dd];

    dsu() { for (int i = 0; i < dd; i++) ind[i] = p[i] = i, r[i] = 1; }

    int get(int v) { return v == p[v] ? v : p[v] = get(p[v]); }
    int next(int v) { return ind[get(v)]; }
    void un(int a, int b)
    {
        a = get(a), b = get(b);
        if (a != b)
        {
            if (r[b] > r[a]) swap(a, b);
            p[b] = a, r[a] += r[b], ind[a] = max(ind[a], ind[b]);
        }
    }
} J;

struct SegmentTree
{
    pair<int, int> T[4 * dd];
    int P[4 * dd];

    int N;

    void assign(int n)
    {
        N = 1;
        while (N < n) N *= 2;
    }

    void build(int v, int tl, int tr, vector<int> &A)
    {
        if (tl == tr) T[v] = { A[tl], tl };
        else

```

```

    {
        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, A);
        build(v * 2 + 1, tm + 1, tr, A);
        T[v] = min(T[v * 2], T[v * 2 + 1]);
    }
}

void upd(int v, int tl, int tr, int l, int r, int val)
{
    if (tl == l && tr == r)
        {T[v].first += val, P[v] += val;}
    else
    {
        int tm = (tl + tr) / 2;
        if (r <= tm) upd(v * 2, tl, tm, l, r, val);
        else if (l > tm) upd(v * 2 + 1, tm + 1, tr, l, r, val);
        else upd(v * 2, tl, tm, l, tm, val), upd(v * 2 + 1, tm + 1, tr,
tm + 1, r, val);

        T[v] = min(T[v * 2], T[v * 2 + 1]);
        T[v].first += P[v];
    }
}

pair<int, int> mer(pair<int, int> a, int b)
{
    a.first += b;
    return a;
}

pair<int, int> get(int v, int tl, int tr, int l, int r)
{
    if (tl == l && tr == r) return T[v];
    int tm = (tl + tr) / 2;
    if (r <= tm) return mer(get(v * 2, tl, tm, l, r), P[v]);
    if (l > tm) return mer(get(v * 2 + 1, tm + 1, tr, l, r), P[v]);
    return mer(min(get(v * 2, tl, tm, l, tm), get(v * 2 + 1, tm + 1,
tr, tm + 1, r)), P[v]);
}

int getPos(int pos)
{
    pos += N;
    int res = T[pos].first;
    pos /= 2;
    while (pos) {
        res += P[pos];
        pos /= 2;
    }
    return res;
}

} T;

int main()
{
    freopen("wrestling.inp", "r", stdin);
    // freopen("57", "r", stdin);
    freopen("wrestling.out", "w", stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(0);
}

```

```

int n, q;
cin >> n >> q;

T.assign(n);
int len = T.N;

vector<int> A(len);
for (int i = 0; i < n; i++)
{
    cin >> A[i];
    if (A[i] == 1) J.un(i, i + 1), A[i] = (int)1e9;
}

T.build(1, 0, len - 1, A);

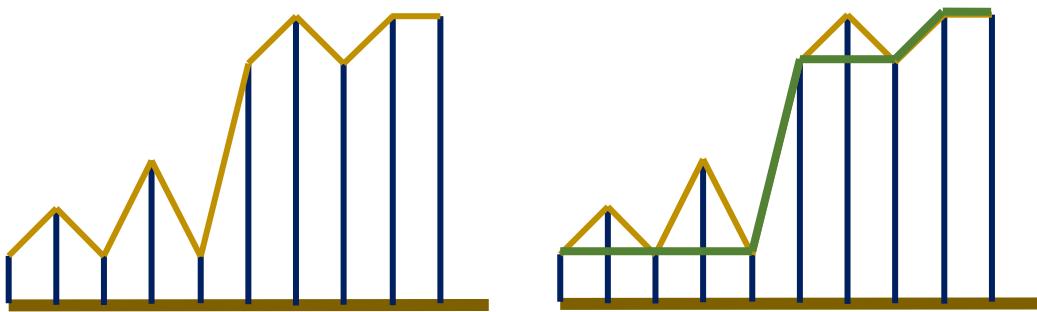
while (q--) {
    int tp, l, r;
    cin >> tp >> l >> r;
    l--, r--;

    if (tp == 1)
    {
        T.upd(1, 0, len - 1, l, r, -1);
        pair<int, int> tmp;
        while ((tmp = T.get(1, 0, len - 1, l, r)).first == 1)
        {
            T.upd(1, 0, len - 1, tmp.second, tmp.second, (int)1e9);
            J.un(tmp.second, tmp.second + 1);
        }
    } else
    {
        int x; cin >> x;
        int t = 0;
        for (int i = J.next(l); i <= r; i = J.next(i + 1))
        {
            int c = T.getPos(i);
            x /= c;
            if (x == 0)
            {
                cout << i + 1 << "\n";
                t = 1;
                break;
            }
        }
        if (!t) cout << "-1\n";
    }
}
cout << "\nTime: " << clock() / (double)1000 << " sec";
}

```



Nhà máy thủy điện có hai lớp. Nước từ hồ chứa được xả ra làm quay turbin lắp ở lớp I. Lợi dụng độ cao của nhà máy ở lớp I, nước xả xuống sẽ làm quay turbin ở lớp II ở dưới chân núi.



Địa hình từ nơi đặt turbin lớp II đến lớp I là không bằng phẳng, được đặc trưng bởi n độ cao $h_1, h_2, \dots, h_{n-1}, h_n$. Turbin lớp II được đặt ở điểm độ cao h_1 , turbin lớp I - ở vị trí độ cao h_n .

Đường dẫn nước từ turbin lớp I xuống turbin lớp II phải bao gồm các đoạn nằm ngang hoặc dốc xuống. Nếu $h_i \leq h_{i+1}$ – đường dẫn nước đặt nổi theo sườn núi. Nếu tồn tại $h_i = h_j$ ($i < j$) – người ta có thể đào đường hầm (nếu cần) hoặc bắc nối trên không dẫn nước nằm ngang nối từ j tới i . Các đoạn dẫn nước đó được gọi là các đoạn đặc biệt. Đoạn đặc biệt có thể đi qua điểm k với $h_k = h_i$.

Hãy xác định một cách dẫn nước để có số đoạn đặc biệt là ít nhất, chỉ ra số lượng đoạn đặc biệt và vị trí hai đầu của mỗi đoạn đặc biệt.

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

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^6$),
- ✚ Dòng thứ 2 chứa n số nguyên h_1, h_2, \dots, h_n ($1 \leq h_i \leq 10^6$, $i = 1 \div n$).

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

- ✿ Dòng đầu tiên chứa số nguyên m – số ít nhất các đoạn đặc biệt,
- ✿ Mỗi dòng trong m dòng tiếp theo chứa 2 số nguyên xác định các điểm đầu và cuối của một đoạn đặc biệt, các đoạn được đưa ra theo thứ tự tăng dần của điểm đầu.

Ví dụ:

TUNNEL.INP	TUNNEL.OUT
10	2
1 2 1 3 1 5 6 5 6 6	1 5 6 8



Giải thuật: Quy hoạch động.

Đặc điểm của bài toán:

- Phạm vi giá trị cần xét không lớn ($1 \leq h_i \leq 10^6$),
- Yêu cầu dẫn xuất giá trị tối ưu của hàm mục tiêu và phương án tối ưu.
- Việc dẫn xuất phương án tối ưu có thể dựa vào kết quả phần đầu để kéo dài phạm vi xét.

Từ các đặc điểm trên ta thấy:

- Có thể dùng sơ đồ quy hoạch động để giải bài toán,
- Cần lưu các tham số liên quan tới mỗi độ cao đã xét để có thể tham chiếu tới kết quả tốt nhất đã đạt được ở cùng độ cao.

Quá trình giải bài toán gồm 2 phần:

- Tìm giá trị tối ưu (số ít nhất các đoạn đặc biệt),
- Dẫn xuất phương án tối ưu (đưa ra các đoạn đặc biệt).

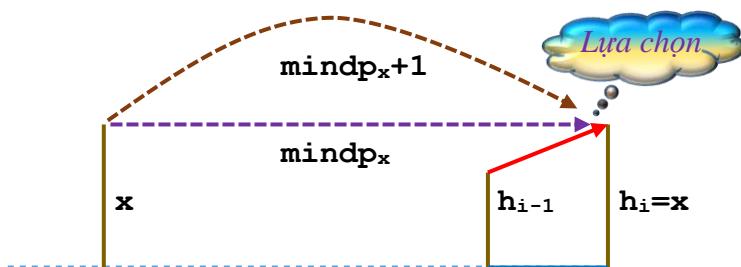
Sơ đồ quy hoạch động:

Gọi dp_i – số ít nhất các đoạn đặc biệt từ đầu tới vị trí i ,

Khai báo: `vector<int> dp(n, INT_MAX)`,

Công thức lặp:

$$dp_0 = 0,$$



Sau khi lựa chọn: lưu lại kết quả tối ưu ứng với độ cao $x = h_i$.

Để phục vụ dẫn xuất phương án tối ưu: cần lưu lại chỉ số i vào vector p , nơi đạt min khi có đoạn đặc biệt (dp_i tăng).

Tính dp_i với $i = 1 \div n-1$.

Kết quả tối ưu của hàm: giá trị dp_{n-1} . Nếu $dp_{n-1} = \infty \rightarrow$ Vô nghiệm.

Dẫn xuất phương án tối ưu:

Duyệt từ $n-1$ về 0, ghi nhận các cặp chỉ số khác nhau liên tiếp trong p và đưa ra theo trình tự từ 0 đến $n-1$.

Tổ chức dữ liệu:

- Mảng `vector<int>` `a(n)` – Ghi nhận dữ liệu vào,
- Mảng `vector<int>` `dp(n, INT_MAX)` – Lưu giá trị hàm mục tiêu,
- Mảng `vector<int>` `mindp(MAX_VALUE, INT_MAX)` – Lưu giá trị hàm mục tiêu đã có theo độ cao,
- Mảng `vector<int>` `minindex(MAX_VALUE, INT_MAX)` – Lưu chỉ số độ cao ứng với giá trị hàm mục tiêu đã ghi nhận.
- Mảng `vector<int>` `p(n, -1)` – Lưu vết đường đi.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "tunnel."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int MAX_VALUE = 1e6 + 10;

int main()
{
    int n;
    fi >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) fi >> a[i];
    vector<int> dp(n, INT_MAX);
    vector<int> p(n, -1);
    vector<int> mindp(MAX_VALUE, INT_MAX);
    vector<int> minindex(MAX_VALUE, INT_MAX);
    dp[0] = 0;
    mindp[a[0]] = 0;
    minindex[a[0]] = 0;
    for (int i = 1; i < n; i++)
    {
        if (a[i - 1] <= a[i])
        {
            dp[i] = dp[i - 1];
            p[i] = i - 1;
        }
        if (mindp[a[i]] != INT_MAX && dp[i] > mindp[a[i]] + 1)
        {
            dp[i] = mindp[a[i]] + 1;
            p[i] = minindex[a[i]];
        }
        if (dp[i] < mindp[a[i]])
        {
            mindp[a[i]] = dp[i];
            minindex[a[i]] = i;
        }
    }
    if (dp[n - 1] == INT_MAX) {

        fo<<"-1\n";
        return 0;
    }

    fo<<dp[n-1]<<' \n';
    vector<pair<int, int>> ans;
    int cur = n - 1;
    while (cur != -1) {
        int pr = p[cur];
        if (pr != cur - 1) {
            ans.push_back({pr, cur});
        }
        cur = pr;
    }
    reverse(ans.begin(), ans.end());
    for (size_t i = 0; i < ans.size(); i++) {

        fo<<ans[i].first + 1<< ' '<<ans[i].second + 1<<' \n';
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Câu lạc bộ bắn cung có n hội viên. Dựa theo các bài giảng về cung nỏ, mỗi hội viên phải tự chế tạo bộ cung tên riêng cho mình. Bộ cung tên của người thứ i có thể bắn tên với tốc độ ban đầu là c_i và cứ bay một đoạn r_i tốc độ bị giảm đi 1, tên sẽ bay cho đến khi tốc độ bằng 0 thì rơi ($i = 1 \dots n$).

Để chuẩn bị cho buổi biểu diễn bắn tên nghệ thuật mọi người đứng thành một hàng ngang, người thứ i đứng ở vị trí tọa độ nguyên x_i , $x_i < x_{i+1}$ với $i = 1 \dots n-1$. Khi phát lệnh bắt đầu, một số người được chỉ định trước sẽ bắn một mũi tên về phía bên phải (phía tọa độ tăng dần). Khi có một mũi tên bay ngang qua hoặc rơi xuống trước mặt ai, người đó phải bắt lấy mũi tên ấy và bắn một phát tên về phía phải từ bộ cung tên của mình.

Hãy xác định số người ít nhất cần chỉ định bắn ban đầu để mỗi người trong số còn lại có dịp bắn ít nhất một phát tên.

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

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^3$),
- ✚ Dòng thứ i trong n dòng sau chứa 3 số nguyên x_i , r_i và c_i ($1 \leq x_i \leq 10^9$, $1 \leq r_i, c_i \leq 100$).

Kết quả: Đưa ra file văn bản LONGBOW.OUT: một số nguyên – số người ít nhất cần chỉ định bắn ban đầu.

Ví dụ:

LONGBOW.INP	LONGBOW.OUT
<pre>5 1 3 3 5 1 2 8 2 3 10 1 2 11 3 2</pre>	<pre>2</pre>



VY28 Io20181020 BE A XIV

Giải thuật; Cơ sở lập trình .

Người thứ nhất luôn luôn là người được chỉ định bắn vì bên trái người đó không có ai,

Người thứ **i** (**i** > 0) sẽ được chỉ định bắn khi tên của người **i-1** không tới được vị trí người **i**, tức là khi **x_{i-1}+r_{i-1}xc_{i-1} < x_i**.

Độ phức tạp của giải thuật: O(n).

Chương trình

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

const int dd = 1001;
int r[dd], x[dd], c[dd];
int main()
{
    int n; fi >> n;
    for (int i = 0; i < n; i++)
        fi >> x[i] >> r[i] >> c[i];

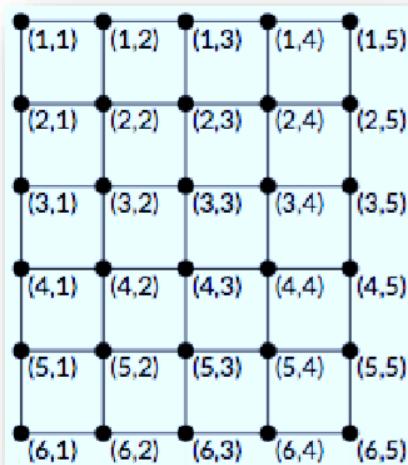
    int ans = 1;
    for (int i = 1; i < n; i++)
        if (x[i - 1] + r[i - 1] * c[i - 1] < x[i]) ans++;

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



Các bạn trong Câu lạc bộ Tin học trẻ được làm quen với các khái niệm cơ bản về đồ thị. Thầy giáo vẽ trên bảng một đồ thị vô hướng dạng lưới ô vuông, mỗi nút giao của lưới là một đỉnh của đồ thị, còn cạnh ô vuông – cạnh của đồ thị. Như vậy, nếu lưới ô vuông có $n \times m$ đỉnh (n hàng, mỗi hàng m đỉnh) thì đồ thị sẽ có $n \times (m-1) + m \times (n-1)$ cạnh. Ký hiệu (i, j) là đỉnh nằm trên giao giữa dòng i với cột j , đồ thị được vẽ có cạnh giữa 2 đỉnh (i, j) và $(i+1, j)$ với mọi $1 \leq i < n$, $1 \leq j \leq m$, có cạnh giữa 2 đỉnh (i, j) và $(i, j+1)$ với mọi $1 \leq i \leq n$, $1 \leq j < m$.

Bài tập về nhà là xây dựng đồ thị vô hướng với $n \times m$ đỉnh, chứa $n \times (m-1) + m \times (n-1)$ cạnh, các đỉnh được đánh số từ 1 đến $n \times m$, không chứa các cạnh dẫn từ một đỉnh tới chính nó, giữa 2 đỉnh có không quá một cạnh và đăng cấu với đồ thị dạng lưới ô vuông, tức là có cách đặt tương ứng một – một các đỉnh của đồ thị này với đỉnh đồ thị dạng lưới ô vuông.



Jimmy chọn n , m và liệt kê các cạnh đồ thị của mình dưới dạng cặp số (u_i, v_i) , $u_i \neq v_i$ và không có cặp cạnh nào trùng nhau.

Hãy xác định đồ thị của Jimmy có thỏa mãn yêu cầu của bài tập hay không.

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

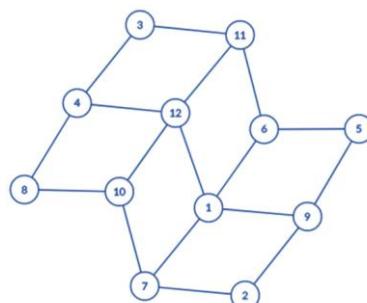
- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($2 \leq n, m, n \times m \leq 2 \times 10^5$),
- ✚ Mỗi dòng trong $n \times (m-1) + m \times (n-1)$ dòng sau chứa một cặp số nguyên xác định cạnh của đồ thị.

Kết quả: Đưa ra file văn bản ISOMRP.OUT thông báo kết quả kiểm tra **Yes** hoặc **No**.

Ví dụ:

ISOMRP.INP
4 3
2 9
2 7
5 9
5 6
7 10
1 7
1 6
1 9
1 12
6 11
10 12
11 12
8 10
4 8
4 12
3 4
3 11

ISOMRP.OUT
Yes



Giải thuật: Kỹ thuật bảng phương án.

Với các bài toán nhận dạng:

- ⊕ Liệt kê các điều kiện cần kiểm tra (các *mẫu nhận dạng*) và kết quả tương ứng cần có với một câu hình cuối cố định (*câu hình chuẩn*),
- ⊕ Xử lý các trường hợp đặc biệt,
- ⊕ Duyệt các khả năng tương ứng giữa đối tượng cần nhận dạng với câu hình chuẩn.

Với bài toán đang xét:

- ⊛ Có đúng 4 đỉnh bậc 2 (4 đỉnh góc),
- ⊛ Các đỉnh trên biên – bậc 3,
- ⊛ Các đỉnh trong – bậc 4.
- ⊛ Trường hợp riêng: $n = 2$ hoặc $m = 2 \rightarrow$ không có đỉnh trong.
- ⊛ Điều kiện cần: Có 4 đỉnh bậc 2.

Đánh số các dòng và cột bắt đầu từ 0.

Khoảng cách từ đỉnh (i, j) trong câu hình chuẩn về đỉnh góc trên trái và trên phải là $\mathbf{x} = i+j$ và $\mathbf{y} = i+(m-j)$ (khoảng cách Manhattan).

Như vậy nếu trong câu hình cần nhận dạng, có định 2 đỉnh bậc 2 làm đỉnh trên trái và trên phải, một đỉnh có khoảng cách tới 2 đỉnh góc này là \mathbf{x} và \mathbf{y} thì nó phải tương ứng với $i = (x+y-m)/2$ và $j = (x-y+m)/2$.

Có $4! = 24$ cách chọn 2 đỉnh trên trái và trên phải. Với mỗi cách – kiểm tra tồn tại các cặp (i, j) riêng biệt.

Tổ chức dữ liệu:

- ▀ Mảng `vector<set<int>>` graph($n * m$) – ghi nhận danh sách đỉnh kề,
- ▀ Mảng `set<pii>` edges – ghi nhận cạnh,
- ▀ Mảng `vector<bool>` used($n * m$) – đánh dấu đỉnh đã duyệt,
- ▀ Mảng field = `vector<vector<int>>(2, vector<int>(m, -1))` – ghi nhận cạnh biên,
- ▀ Mảng `vector<int>` corners – ghi nhận đỉnh góc,
- ▀ Mảng `vector<int>` path; – ghi nhận đường đi trong quá trình duyệt,
- ▀ Mảng `vector<bool>` good($n * m$) – đánh dấu kết quả nhận dạng,
- ▀ Hàng đợi `queue<pii>` qu – phục vụ loang.

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

Chương trình

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

#define szof(_x) ((int) (_x).size())
typedef pair<int, int> pii;
int dx[4] = {1, 0, -1, 0};
int dy[4] = {0, 1, 0, -1};

void solve()
{
    int n, m;
    fi >> n >> m;
    vector<set<int>> graph(n * m);
    set<pii> edges;
    vector<vector<int>> field;

    for (int i = 0; i < n * (m - 1) + m * (n - 1); ++i)
    {
        int a, b;
        fi >> a >> b;
        --a; --b;
        edges.insert({a, b});
        edges.insert({b, a});
        graph[a].insert(b);
        graph[b].insert(a);
    }

    if (n == 2 || m == 2)
    {
        m = n + m - 2;
        n = 2;
        int any1 = -1, any2 = -1;
        int cnt_corner = 0;
        for (int i = 0; i < n * m; ++i) {
            if (szof(graph[i]) == 1 || szof(graph[i]) > 3)
            {
                fo << "No\n";
                return;
            }
            if (szof(graph[i]) == 2)
            {
                ++cnt_corner;
                if (any1 == -1) any1 = i;
                else if (any2 == -1 && graph[i].count(any1)) any2 = i;
            }
        }

        if (cnt_corner != 4 || any1 == -1 || any2 == -1)
        {
            fo << "No\n";
            return;
        }

        field = vector<vector<int>>(2, vector<int>(m, -1));
        field[0][0] = any1;
        field[1][0] = any2;
    } else
    {
```

```

vector<bool> good(n * m);

int cnt_good = 0;
int any_good = -1;
int cnt_corner = 0;
for (int i = 0; i < n * m; ++i)
{
    if (szof(graph[i]) == 1 || szof(graph[i]) > 4)
    {
        fo << "No\n";
        return;
    }
    if (szof(graph[i]) < 4)
    {
        good[i] = true;
        ++cnt_good;
        any_good = i;
    }
    if (szof(graph[i]) == 2) ++cnt_corner;
}

if (cnt_good != n * 2 + m * 2 - 4 || cnt_corner != 4)
{
    fo << "No\n";
    return;
}

for (int i = 0; i < n * m; ++i) {
    if (!good[i]) continue;

    int cnt = 0;
    for (int to : graph[i])
        if (good[to]) ++cnt;
    if (cnt != 2)
    {
        fo << "no\n";
        return;
    }
}
}

vector<bool> used(n * m);
vector<int> path;

function<void(int)> dfs = [&] (int v)
{
    used[v] = true;
    path.push_back(v);
    for (int to : graph[v])
        if (!used[to] && good[to]) dfs(to);
};

dfs(any_good);
for (int i = 0; i < n * m; ++i)
    if (good[i] && !used[i])
    {
        fo << "No\n";
        return;
    }
vector<int> corners;
for (int i = 0; i < szof(path); ++i)
{
    int next = (i + 1) % szof(path);

```

```

        if (!edges.count({path[i], path[next]}))
        {
            fo << "No\n";
            return;
        }
        if (szof(graph[path[i]]) == 2) corners.push_back(i);
    }

    assert(szof(corners) == 4);

    if (corners[1]-corners[0]==corners[3]-corners[2]
        && corners[2]-corners[1]==corners[0]+szof(path)-corners[3])
    {
        if (corners[1] - corners[0] == n - 1)
            rotate(path.begin(), path.begin() + corners[0], path.end());
        else if (corners[1] - corners[0] == m - 1)
            rotate(path.begin(), path.begin() + corners[1], path.end());
        else
        {
            fo << "No\n";
            return;
        }
    } else
    {
        fo << "No\n";
        return;
    }

    field = vector<vector<int>>(n, vector<int>(m, -1));
    int c = 0;
    for (int i = 0; i < n; ++i) field[i][0] = path[c++];
    for (int i = 1; i < m; ++i) field[n - 1][i] = path[c++];
    for (int i = n - 2; i >= 0; --i) field[i][m - 1] = path[c++];
    for (int i = m - 2; i >= 1; --i) field[0][i] = path[c++];
}

for (int x = 0; x < n; ++x)
for (int y = 0; y < m; ++y)
{
    if (field[x][y] == -1) continue;
    for (int d = 0; d < 4; ++d)
    {
        int nx = x + dx[d];
        int ny = y + dy[d];
        if (0 <= nx && nx < n && 0 <= ny && ny < m && field[nx][ny] != -1)
        {
            graph[field[x][y]].erase(field[nx][ny]);
            graph[field[nx][ny]].erase(field[x][y]);
        }
    }
}

queue<pii> qu;

for (int i = 0; i < n; ++i)
    for (int j = 0; j < m; ++j)
        if (field[i][j] != -1 && szof(graph[field[i][j]]) == 1) qu.push({i, j});

while (szof(qu))
{
    int x, y;
    tie(x, y) = qu.front();
    assert(field[x][y] != -1);
}

```

```

qu.pop();
if (szof(graph[field[x][y]]) == 0) continue;
int memx = -1, memy = -1;
for (int d = 0; d < 4; ++d)
{
    int nx = x + dx[d];
    int ny = y + dy[d];
    if (0 <= nx && nx < n && 0 <= ny && ny < m && field[nx][ny] == -1)
    {
        memx = nx; memy = ny;
        break;
    }
}
assert(memx != -1);
field[memx][memy] = *graph[field[x][y]].begin();
for (int d = 0; d < 4; ++d)
{
    int nx = memx + dx[d];
    int ny = memy + dy[d];
    if (0 <= nx && nx < n && 0 <= ny && ny < m && field[nx][ny] != -1)
    {
        if (!graph[field[nx][ny]].count(field[memx][memy]))
        {
            fo << "No\n";
            return;
        }
        graph[field[nx][ny]].erase(field[memx][memy]);
        if (szof(graph[field[nx][ny]]) == 1) qu.push({nx, ny});

        graph[field[memx][memy]].erase(field[nx][ny]);
        if (szof(graph[field[memx][memy]]) == 1) qu.push({memx, memy});
    }
}
for (int i = 0; i < n; ++i)
{
    for (int j = 0; j < m; ++j)
    {
        if (field[i][j] == -1)
        {
            fo << "No\n";
            return;
        }
        for (int d = 0; d < 4; ++d)
        {
            int nx = i + dx[d];
            int ny = j + dy[d];
            if (0 <= nx && nx < n && 0 <= ny && ny < m)
                if (!edges.count({field[i][j], field[nx][ny]}))
                {
                    fo << "No\n";
                    return;
                }
        }
    }
}
fo << "Yes\n";
}

int main()
{
    solve();
    fo << "\nTime: " << clock() / (double)1000 << " sec";
}

```



Đội bóng của lớp được vào chung kết giải bóng đá của trường. Cả lớp rất phấn khởi và quyết định tổ chức cá cược vui để gây quỹ liên hoan với đội bóng sau trận chung kết. Quy tắc đánh cược là như sau: Nếu bạn đặt cược a đồng cho đội nhà thắng, thì nếu đội nhà thắng bạn sẽ được hoàn lại tiền đặt cược và được thêm xxa đồng. Nếu bạn đặt cược b đồng cho đội nhà thua, thì nếu đội nhà thua bạn sẽ được hoàn lại tiền đặt cược và được thêm yxb đồng. Các hệ số x, y do bộ phận tổ chức quyết định và thông báo trước khi bắt đầu đăng ký cửa cá cược. Mỗi bạn chỉ được đặt một cửa.

Một cặp bạn thân quyết định chơi trò lầu cá. Cả hai góp chung lại được n đồng, một người sẽ đặt cược a đồng cho cửa đội nhà thắng, người kia đặt $n-a$ đồng còn lại cho cửa đội nhà thua. Hai người tính toán chọn a sao cho số tiền thu về lớn hơn vốn n đồng ban đầu càng nhiều càng tốt không phụ thuộc vào kết quả trận chung kết.

Hãy xác định tổng số tiền lớn nhất 2 người bạn sẽ có thể nhận được ứng với trường hợp kết trận đấu là có lợi nhất cho 2 bạn và các cặp giá trị (a, b) cần chọn để được tổng thu nhập đó.

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

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^9$),
- ✚ Dòng thứ 2 chứa 2 số thực x và y có không quá 5 chữ số sau dấu chấm thập phân ($10^{-5} \leq x, y \leq 10^4$).

Kết quả: Đưa ra file văn bản BETTING.OUT nếu không có cách đặt cược để luôn thu về nhiều hơn n đồng thì đưa ra số **-1**, trong trường hợp ngược lại:

- ✿ Dòng đầu tiên đưa ra tổng số tiền lớn nhất có thể thu về được ứng với trường hợp kết trận đấu là có lợi nhất cho 2 bạn với độ chính xác 10^{-6} ,
- ✿ Dòng thứ 2 chứa số nguyên k – số cách đặt cược,
- ✿ Mỗi dòng trong k dòng sau chứa 2 số nguyên a, b xác định một cách đặt cược, thông tin đưa ra theo thứ tự tăng dần của a .

Ví dụ:

BETTING.INP	BETTING.OUT
8	12 . 000000
3 1	1 3 5



VY30_Io20181020_UF_A XIV

Giải thuật; Phân tích mô hình toán học.

Xét các điều kiện để thu về được nhiều tiền hơn tổng số tiền đặt cược.

Trường hợp đội nhà thắng:

$$a \cdot x + a > n \Rightarrow a \cdot (x+1) > n \Rightarrow a > \frac{n}{x+1}$$

Trường hợp đội nhà thua:

$$(n-a) \cdot y + (n-a) > n \Rightarrow (n-a) \cdot y > a \Rightarrow a < \frac{n \cdot y}{y+1}$$

Từ hai bất đẳng thức trên suy ra:

$$\frac{n}{x+1} < a < \frac{n \cdot y}{y+1}$$

Nếu không tồn tại a thỏa mãn – bài toán vô nghiệm.

Như vậy khả năng tiền thắng lớn nhất sẽ là một trong hai giá trị:

- ✿ Giá trị cực đại của a nhân với ($x+1$),
- ✿ Giá trị cực tiểu của a nhân với ($y+1$).

Nếu 2 giá trị trên bằng nhau thì ta có nhiều phương án phân chia để có cùng kết quả, trong trường hợp ngược lại – chỉ tồn tại một cách chọn cặp giá trị a, b .

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int main()
{
    freopen ("betting.inp", "r", stdin);
    freopen ("betting.out", "w", stdout);
    ios::sync_with_stdio(false);
    cin.tie(0);      cout.tie(0);
    int64_t n;
    long double x, y;
    cin >> n >> x >> y;
    int64_t le = (int64_t)ceil(n / (x + 1) + 1e-7);
    int64_t ri = (int64_t)floor((n * y) / (y + 1) - 1e-7);
    if (le > ri)
    {
        cout << -1 << endl;
        return 0;
    }
    cout.precision(10);
    long double c1 = (x + 1) * ri;
    long double c2 = (y + 1) * (n - le);
    if (abs(c1 - c2) < (long double)1e-7)
    {
        cout << fixed << c1 << '\n';
        set<int64_t> se;
        se.insert(le);
        se.insert(ri);
        cout << se.size() << '\n';
        for (auto q: se)
            cout << q << ' ' << n - q << endl;
    }
    else if (c1 > c2)
    {
        cout << fixed << c1 << '\n';
        cout << 1 << '\n' << ri << ' ' << n - ri << endl;
    }
    else
    {
        cout << fixed << c2 << '\n';
        cout << 1 << '\n' << le << ' ' << n - le << endl;
    }
    return 0;
}
```



Bob và Alice là đồng minh trong một trò chơi online. Để phối hợp trong một hành động hai người phải gửi cùng một khóa cho hệ thống điều khiển. Khóa là một xâu ký tự độ dài k , được tách ra từ xâu do hệ thống cung cấp cùng với công cụ xử lý tương ứng. Bob nhận được xâu s và công cụ cho phép tách ra một xâu con độ dài k các ký tự liên tiếp nhau, Alice nhận được xâu t với công cụ cho phép tách ra k ký tự bất kỳ và ghép chúng lại theo trình tự tùy ý thành xâu độ dài k .

Nếu Bob và Alice cùng tạo được xâu giống nhau độ dài k thì có thể hợp sức giải quyết vấn đề đang gặp.

Hãy xác định với dữ liệu và công cụ nhận được hai người có thể tạo được khóa chung hay không và đưa ra câu trả lời tương ứng **YES** hoặc **NO**.

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

- ✚ Dòng đầu tiên chứa số nguyên k ($1 \leq k \leq 3 \times 10^5$),
- ✚ Dòng thứ 2 chứa xâu s ,
- ✚ Dòng thứ 3 chứa xâu t .

Các xâu chỉ chứa ký tự la tinh thường và độ dài không quá 3×10^5 .

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

Ví dụ:

COMKEY.INP	COMKEY.OUT
3 aba bbaa	YES



Giải thuật; Thống kê cơ bản.

Xác định tần số xuất hiện các ký tự trong xâu **t**,

Xác định tần số xuất hiện các ký tự trong mỗi xâu con độ dài **k** các ký tự liên tiếp nhau trong **s**,

Bài toán có nghiệm khi tìm được xâu con trong **s** có tần số không vượt quá tần số tương ứng đã tính với xâu **t**,

Với chi phí $O(1)$ từ các tần số đã tính với **s[i..i+k-1]** dễ dàng tính tần số các ký tự trong **s[i+1..i+k]**.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

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

int a[26]={0}, b[26]={0};

void check()
{
    for (int i = 0; i < 26; i++)
        if (a[i] > b[i])
            return;
    cout << "YES" << endl;
    exit(0);
}

int main()
{
    freopen("comkey.inp","r",stdin);
    freopen("comkey.out","w",stdout);
    ios::sync_with_stdio(false);
    cin.tie(0);    cout.tie(0);
    int k;
    string s, t;
    cin >> k >> s >> t;
    for (char c: t)
        b[c - 'a']++;
    int n = s.size();
    if (n < k)
    {
        cout << "NO" << endl;
        return 0;
    }
    for (int i = 0; i < k; i++)
        a[s[i] - 'a']++;
    check();
    for (int i = 0; i + k < n; i++)
    {
        a[s[i] - 'a']--;
        a[s[i + k] - 'a']++;
        check();
    }
    cout << "NO" << endl;
    return 0;
}
```



Cho số nguyên dương n và bộ quy tắc biến đổi:

- Tách chữ số hàng đơn vị của n ,
- Cộng giá trị chữ số này vào n chừng nào chưa nhận được kết quả chẵn chục,
- Chia kết quả cho 10 và gán lại cho n .

Bộ quy tắc biến đổi trên được thực hiện vô hạn lần.

Hãy xác định số nhỏ nhất nhận được trong quá trình biến đổi.

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

- ✚ Dòng đầu tiên chứa số nguyên t ($1 \leq t \leq 10^5$) – số lượng tests,
- ✚ Mỗi dòng trong t dòng sau chứa số nguyên n ($1 \leq n \leq 10^{500\,001}$).

Kết quả: Đưa ra file văn bản BIGNUM.OUT các số nhỏ nhất nhận được, mỗi số trên một dòng.

Ví dụ:

BIGNUM.INP	BIGNUM.OUT
4	1
2	3
3	3
6	1
10	



Giải thuật; Xử lý xâu, Cộng số lớn .

Lưu trữ **n** dưới dạng xâu **s**,

Dễ dàng chứng minh được (bằng phản chứng) nếu $n \geq 10$ thì sau mỗi phép biến đổi **n** mới sẽ nhỏ hơn **n** ban đầu.

Như vậy *min* cần tìm sẽ là số có một chữ số.

Gọi **dig** là chữ số cuối cùng (chữ số hàng đơn vị) của **n**.

Ký hiệu **carry** – giá trị nhớ phát sinh khi thực hiện phép cộng,

Ban đầu **carry** = 0,

Để có số chia hết cho 10 ta cần cộng **n** với $\text{dig} \times (10/\gcd(10, \text{dig}) - 1)$.

Gọi **cur** – tổng chữ số cuối cùng với lượng cộng thêm đã nói ở trên.

dig mới sẽ là **cur%10**,

Sau mỗi lần xóa 0 **carry** = **cur**/10 .

Việc tính **cur** và **carry** sẽ được thực hiện vho đến khi số chỉ còn một chữ số và **carry** bằng 0.

Kết quả cần đưa ra là **cur** đang có.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include "bits/stdc++.h"
#define szof(_x) ((int) (_x).size())
#define NAME "bignum"
using namespace std;
typedef long long ll;

void solve()
{
    string s;
    cin >> s;
    reverse(s.begin(), s.end());
    ll carry = 0;
    for (int i = 0; carry || i < szof(s); ++i)
    {
        ll cur = carry;
        if (i < szof(s)) cur += s[i] - '0';
        int dig = cur % 10;
        if (i >= szof(s) - 1 && cur < 10 && __gcd(10, dig) == 1)
        {
            cout << cur << "\n";
            break;
        }
        int mult = 10 / __gcd(10, dig);
        cur += dig * (mult - 1);
        assert(cur % 10 == 0);
        carry = cur / 10;
    }
}

int main()
{
    freopen(NAME ".inp", "r", stdin);
    freopen(NAME ".out", "w", stdout);
    ios::sync_with_stdio(false);

    int t ;
    cin >> t;
    for (int it = 1; it <= t; ++it) solve();

    return 0;
}
```



Bob và Alice được giao nhiệm vụ dọn dẹp hội trường sau buổi hội thảo “*Năm bắt cơ hội lập nghiệp*”. Công việc cũng nhẹ nhàng, đơn giản và Alice nghĩ chi mất độ mươi phút là xong. Nhưng tình hình bỗng trở nên rắc rối và phức tạp khi hai bạn phải khuân đi cát một tấm bảng hình chữ nhật dán các dòng chữ về chuyên đề hội thảo. Mặt trái của nó là một bàn cờ kích thước $n \times m$ ô, các hàng được đánh số từ trên xuống dưới từ 1 đến n , các cột đánh số từ 1 đến m từ trái qua phải. Giao giữa hàng a và cột b là ô (a, b) . Mỗi ô của bàn cờ được sơn một trong 2 màu trắng hoặc đen, nếu $a+b$ là số chẵn – ô có màu trắng, trong trường hợp ngược lại – màu đen.

Tai họa là ở chỗ Bob bỗng nỗi máu nghệ sỹ, muốn dùng các hộp sơn phun cạnh đó tạo một bức tranh graffiti trùu tượng bằng cách sơn đảo màu một số ô, tức là ô trắng thành ô đen và ngược lại.

Alice, vốn say mê Tin học, rất xa lạ với graffiti, hơn nữa lại là graffiti trùu tượng. Sốt ruột khoanh tay đứng nhìn bạn vẽ, Alice nghĩ thầm, thà sơn sao cho tồn tại một t để các ô ở các dòng trên t có màu đen, các ô còn lại (từ dòng t trở xuống) – có màu trắng. Ít ra như vậy cũng còn dùng như tấm bảng vừa có thể viết phán (ở phần màu đen) và viết bút dạ (ở phần màu trắng).

Cứ mỗi lần sơn lại một ô, Bob lại đứng ngắm nhìn và suy nghĩ chọn ô tiếp theo, còn Alice thì nhầm tính bấy giờ phải sơn đổi màu ít nhất bao nhiêu ô để tạo thành bảng 2 phần như mình muốn. Vốn có trí nhớ tốt, Alice nhớ hết các số đó.

Sau khi Bob sơn lại ô thứ q thì sự kiên nhẫn của Alice cũng cạn kiệt, cô dứt khoát yêu cầu Bob khiêng bảng đi cát. Khi ra về, những số đã nhớ trong đầu tính được vẫn hiện rõ trong đầu của Alice.

Hãy đưa ra dãy số đó,

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n \leq 2 \times 10^5$, $1 \leq m \leq 10$),
- ✚ Dòng thứ 2 chứa số nguyên q ($1 \leq q \leq 2 \times 10^5$),
- ✚ Mỗi dòng trong q dòng tiếp theo chứa 2 số nguyên a và b ($1 \leq a \leq n$, $1 \leq b \leq m$).

Kết quả: Đưa ra file văn bản GRAFFITI.OUT các số Alice đã nhầm tính được theo trình tự tính, mỗi số trên một dòng.

Ví dụ:

GRAFFITI.INP
5 4
4
1 1
5 1
1 3
2 3

GRAFFITI.OUT
9
8
7
8



Giải thuật: Cây phân khúc (Segment tree) quản lý min có cập nhật.

Xét chi phí để tô các dòng trước t thành đen và các dòng từ t đến n là trắng với $t = 0, 1, 2, \dots, n$.

Nếu không có truy vấn cập nhật màu của ô vẫn đề nêu trên đơn giản là bài toán tìm min.

Việc tồn tại truy vấn thay đổi màu của ô dẫn đến phải giải quyết bài toán **RMQ** (Range Minimum Query).

Để giải quyết bài toán **RMQ** với dãy số cần tổ chức cây phân đoạn quản lý min có cập nhật đoạn (tăng hay giảm các phần tử dãy số ở các vị trí trong đoạn $[l, r]$) lên hoặc xuống cùng một số).

Xét việc tô lại ô (x, y) và sự *thay đổi* của *các giá trị đang lưu giữ* trong cây.

Trường hợp ô (x, y) đang là màu đen:

- Màu mới của ô sẽ là trắng,
- Giá trị lưu giữ ứng với các dòng *bắt đầu từ x trở lên* giảm đi 1,
- Giá trị lưu giữ ứng với các dòng *lớn hơn x* tăng thêm 1.

Trường hợp ô (x, y) đang là màu trắng:

- Màu mới của ô sẽ là đen,
- Giá trị lưu giữ ứng với các dòng *bắt đầu từ x trở lên* tăng thêm 1,
- Giá trị lưu giữ ứng với các dòng *lớn hơn x* giảm đi 1.

Như vậy với mỗi truy vấn thay đổi màu ở một ô cần thực hiện 2 phép cập nhật tương ứng với các đoạn $[1, x]$ và $[x+1, n]$.

Để thuận tiện xử lý, các dòng được đánh số từ 0 đến $n-1$.

Tổ chức dữ liệu:

- ▀ Mảng **int** $t[4 * N]$ – tổ chức cây phân đoạn,
- ▀ Mảng **int** $d[4 * N]$ – phục vụ ghi nhận thay đổi cập nhật,
- ▀ Mảng **map** $\langle \text{pair} \langle \text{int}, \text{int} \rangle, \text{int} \rangle$ col – ghi nhận màu ở các ô có thay đổi.

Xử lý:

Hàm truyền hiệu ứng cập nhật xuống các cây con:

```
void push(int v)
{
    t[v * 2 + 1] += d[v];
    d[v * 2 + 1] += d[v];
    t[v * 2] += d[v];
    d[v * 2] += d[v];
    d[v] = 0;
}
```

Hàm cập nhật cây trên đoạn $[l, r]$:

```

void upd(int v, int l, int r, int tl, int tr, int x)
{
    if (tl >= r || tr <= l) return;
    if (tl >= l && tr <= r)
    {
        d[v] += x;
        t[v] += x;
    }
    else
    {
        int tm = (tl + tr) / 2;
        push(v);
        upd(v * 2 + 1, l, r, tl, tm, x);
        upd(v * 2, l, r, tm, tr, x);
        t[v] = min(t[v * 2], t[v * 2 + 1]);
    }
}

```

(Chi tiết về giải thuật cập nhật: Xem phần lý thuyết cây phân đoạn, V 15).

Khởi tạo cây:

```

Số ô đen fi >> n >> m;
Y = (m + 1) / 2;
X = m / 2;
Cập nhật phần tô đen
Số ô trắng for (int i = 0; i < n; i++)
{
    swap(x, y);
    upd(1, i + 1, n + 1, 1, n + 1, x);
    upd(1, 0, i + 1, 1, n + 1, y);
Cập nhật phần tô trắng
}

```

Số lượng ô trắng và đen ở mỗi dòng của bảng ban đầu có quy luật đơn giản:

- Mọi dòng chẵn có số lượng giống nhau,
- Mọi dòng lẻ: số lượng giống nhau và suy được từ kết quả dòng chẵn.

Để tránh dùng mảng ghi nhận dữ liệu ban đầu việc khởi tạo cây có thể thực hiện bằng hàm cập nhật.

Gọi **x** – số ô trắng trong dòng **i**, **y** – số ô đen ở dòng **i**, **i** = 0 ÷ **n**-1.

Kết quả tô đen các dòng từ 0 đến **i** tăng thêm một lượng là **x**, việc tô trắng các dòng còn lại – tăng thêm **y**.

Xử lý một truy vấn đổi màu ô (x, y):

```
fi >> x >> y;
x--, y--;
if (!col.count({x, y})) col[{x, y}] = (x+y) % 2;

if (!col[{x, y}]) upd(1, x+1, n+1, 1, n+1, -1);
else upd(1, 0, x+1, 1, n+1, -1);

col[{x, y}] ^= 1;
if (!col[{x, y}]) upd(1, x+1, n+1, 1, n+1, 1);
else upd(1, 0, x+1, 1, n+1, 1);

fo << t[1] << '\n';
```

Độ phức tạp của giải thuật: $O(4q \log n)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("graffiti.inp");
ofstream fo ("graffiti.out");
const int N = 2e5+7;
int n,m,x,y;
//vector<int>t,d;
int t[4 * N];
int d[4 * N];
void push(int v)
{
    t[v * 2 + 1] += d[v]; d[v * 2 + 1] += d[v];
    t[v * 2 ] += d[v]; d[v * 2 ] += d[v];
    d[v] = 0;
}
void upd(int v, int l, int r, int tl, int tr, int x)
{
    if (tl >= r || tr <= l) return;
    if (tl >= l && tr <= r)
    {
        d[v] += x; t[v] += x;
    }
    else
    {
        int tm = (tl + tr) / 2;
        push(v);
        upd(v * 2 + 1, l, r, tl, tm, x);
        upd(v * 2 , l, r, tm, tr, x);
        t[v] = min(t[v * 2 ], t[v * 2 + 1]);
    }
}
int main()
{
    fi >> n >> m;
    y = (m + 1) / 2;
    x = m / 2;
//    t.resize(4*n); d.resize(4*n);
    for (int i = 0; i < n; i++)
    {
        swap(x,y);
        upd(1, i + 1, n + 1, 1, n + 1, x);
        upd(1, 0, i + 1, 1, n + 1, y);
    }
    map <pair <int, int>, int> col;
    int q;
    fi >> q;
    while (q--)
    {
        fi >> x >> y;
        x--, y--;
        if (!col.count({x, y})) col[{x, y}] = (x + y) % 2;
        if (!col[{x, y}]) upd(1, x + 1, n + 1, 1, n + 1, -1);
        else upd(1, 0, x + 1, 1, n + 1, -1);
        col[{x, y}] ^= 1;
        if (!col[{x, y}]) upd(1, x + 1, n + 1, 1, n + 1, 1);
        else upd(1, 0, x + 1, 1, n + 1, 1);
        fo << t[1] << '\n';
    }
    fo << "\nTime: " << clock() / (double)1000 << " sec";
}
```



Ngày nay hiếm có sinh viên nào lại không có máy tính cá nhân, việc soạn thảo văn bản thì ngay đến học sinh PTCS cũng đã thành thạo. Vì vậy một số nơi trước khi tới phỏng vấn xin vào làm người ta yêu cầu gửi trước đơn xin việc viết tay. “*Nét chữ - Nét người*”, nhìn vào trang giấy viết tay người ta cũng có thể đoán được nhiều tính cách của người xin việc, thậm chí, có một số trường hợp, biết được tình trạng sức khỏe!

Alice phải gửi một đơn xin việc viết tay. Theo quy định, đơn viết trên giấy hình chữ nhật kích thước $n \times m$. Alice chỉ có trong tay phong bì hình chữ nhật kích thước $h \times w$. Nếu tờ giấy không cho vừa vào phong bì Alice gấp đôi lá đơn theo chiều dọc hoặc chiều ngang một hoặc vài lần. Đơn được bỏ vào phong bì với các cạnh song song với các cạnh của phong bì. Đơn bỏ được vào phong bì nếu các cạnh không lớn hơn cạnh tương ứng của phong bì. Đơn có thể được xoay 90° trước khi bỏ vào phong bì.

Là người cẩn thận, Alice không muốn lá đơn bị nhau quá mức và vì vậy cô gắng thực hiện việc gấp một cách ít nhất có thể.

Hãy xác định số lần gấp ít nhất cần thực hiện.

Dữ liệu: Vào từ file văn bản ENVELOPE.INP gồm một dòng chứa 4 số nguyên n, m, h, w ($1 \leq n, m, h, w \leq 10^{18}$).

Kết quả: Đưa ra file văn bản ENVELOPE.OUT một số nguyên – số lần gấp ít nhất cần thực hiện.

Ví dụ:

ENVELOPE.INP	ENVELOPE.OUT
3 3 2 2	2



Giải thuật; Cơ sở lập trình.

Việc gấp đôi tờ giấy sẽ dẫn đến kích thước phải biến đổi dưới dạng số thực và cần chống tích lũy sai số làm tròn.

Thay vì giảm kích thước tờ giấy – phóng to gấp đôi kích thước phong bì!

Kiểu dữ liệu `int64_t` vẫn lưu được giá trị 2×10^{18} – vượt quá kích thước tối đa của tờ giấy, vì vậy sẽ không có hiện tượng tràn ô (overflow).

Xét 2 trường hợp:

- + Không xoay tờ giấy,
- + Xoay tờ giấy 90° .

Kết quả: số phép biến đổi nhỏ nhất trong 2 trường hợp nói trên.

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "envelope."
using namespace std;

int64_t solve(int64_t n, int64_t m, int64_t h, int64_t w)
{
    int ans = 0;
    while (h < n)
    {
        ans++;
        h *= 2;
    }
    while (w < m)
    {
        ans++;
        w *= 2;
    }
    return ans;
}

int main()
{
    freopen(NAME"inp", "r", stdin);
    //freopen("27", "r", stdin);
    freopen(NAME"out", "w", stdout);
    int64_t n,m,h,w;
    cin >> n >> m >> h >> w;
    cout << min(solve(n, m, h, w), solve(m, n, h, w)) << "\n";
}
```



Alice giành được rất nhiều huy chương vàng và vị trí nhất bảng trong các cuộc thi Olympic Tin học. Năm cuối cùng ở nhà trường Alice lại có mặt trong đội tuyển của trường tham dự Olympic. Trong đội cũng có những bạn ở lớp dưới rất giỏi, nhưng họ vượt qua được Alice. Cô muốn tặng lại vị trí đầu bảng cho một trong số họ, để các bạn áy tự tin và đạt được nhiều thành tích cao hơn trong các năm sau.

Alice có thể không nộp bài, nhưng như vậy là không đẹp và thiếu tính trung thực trong thể thao chân chính. Cô quyết định vẫn sẽ làm hết khả năng của mình, nhưng với chiến lược chọn bài thích hợp để tổng số điểm nhận được càng thấp càng tốt.

Đề thi có n bài, đánh số từ 1 đến n , bài thứ i có điểm tối đa là p_i . Đọc lượt qua các bài Alice xác định ngay được giải thuật và ước lượng là bài thứ i cần t_i thời gian cài đặt. Liếc nhìn đồng hồ, cô thấy còn T đơn vị thời gian để làm bài.

Alice chọn một trong số các bài chưa làm, giải và nộp chấm. Thời gian nộp chấm là không đáng kể. Cô nhận được đầy đủ điểm của bài đã nộp. Nếu còn kịp làm bài khác Alice sẽ tiếp tục làm và nộp chấm. Cô không bao giờ làm các bài mà thời gian còn lại không đủ để viết chương trình. Nếu không còn bài nào đủ thời gian làm Alice chỉ đơn thuần là ngồi chờ hết giờ.

Với chiến lược làm bài nêu trên Alice đã nhận được tổng điểm nhỏ nhất một cách trung thực.

Hãy xác định tổng số điểm của Alice.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và T ($1 \leq n, T \leq 2\,000$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên t_i và p_i ($1 \leq t_i \leq 2\,000, 1 \leq p_i \leq 10^6$).

Kết quả: Đưa ra file văn bản FAIRPLAY.OUT một số nguyên – tổng điểm nhỏ nhất Alice nhận được.

Ví dụ:

FAIRPLAY.INP
4 9
4 2
4 5
3 4
2 10

FAIRPLAY.OUT
7



VY36 StP20181028 C AXIV

Giải thuật: Quy hoạch động.

Mô hình toán học: Bài toán Ba lô với yêu cầu chỉ đưa ra giá trị hàm mục tiêu.

Sắp xếp các cặp $\{t_i, p_i\}$ theo thứ tự giảm dần,

Gọi $dp_{i,j}$ – tổng số điểm nhỏ nhất có thể đạt khi chọn các bài trong số i bài đầu (ở đây đã sắp xếp) và có *thời gian thực hiện đúng bằng j .*

Công thức lặp:

$$dp_{i,j} = \min\{dp_{i-1,j}, dp_{i-1,j-t[i]}+p_i\}$$

với $i = 0, 1, \dots, n-1$, $j = 0, 1, \dots, T$.

Duyệt ngược dãy bài từ cuối về đầu, kiểm tra khả năng bổ sung hoặc thay thế thay thế một số bài bằng các bài chưa chọn để có tổng điểm nhỏ hơn.

Tổ chức dữ liệu:

- ▀ Mảng `int t[N]` – lưu thời gian làm mỗi bài,
- ▀ Mảng `int p[N]` – lưu điểm mỗi bài,
- ▀ Mảng `pair<int, int> tmp[n]` – trung gian, phục vụ thuận tiện cho việc sắp xếp dữ liệu vào,
- ▀ Mảng `ll dp[N][N]` – phục vụ tính lặp theo quy hoạch động.

Xử lý:

Nhập và sắp xếp dữ liệu:

```

int n, T;
cin >> n >> T;
pair<int, int> tmp[n];
for (int i = 0; i < n; i++)
    cin >> tmp[i].fi >> tmp[i].se;
sort(tmp, tmp + n, [](pii x, pii y) { return x >= y; });
//sort(tmp, tmp + n, [](auto x, auto y){return x >= y; });

```

Điều kiện sắp xếp
giảm dần

Tính $dp_{i,j}$:

Lưu ý: Gọi
theo địa chỉ

```
inline void setmin(ll &x, ll y) { if (y < x) x = y; }
```

```

dp[0][0] = 0;
for (int i = 0; i < n; i++)
    for (int j = 0; j <= T; j++)
    {
        setmin(dp[i + 1][j], dp[i][j]);
        if (j + t[i] <= T)
            setmin(dp[i + 1][j + t[i]], dp[i][j] + p[i]);
    }

```

*i = 0 – phục vụ
khởi tạo*

Dẫn xuất kết quả:

```
ll ans = big, sumP = 0;
int sumT = 0;
for (int i = n - 1; i >= 0; i--)
{
    for (int j = max(0, T-sumT-t[i]+1); j <= T-sumT; j++)
        setmin(ans, dp[i][j] + sumP);
    sumT += t[i];
    sumP += p[i];
}
if (sumT <= T) setmin(ans, sumP);
```

Bổ sung/Thay thế bằng
các bài có t nhỏ hơn

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

Chương trình

```
#include <bits/stdc++.h>
#define ll long long
#define fi first
#define se second
#define NAME "fairplay."
using namespace std;
typedef pair<int,int> pii;
inline void setmin(ll &x, ll y) { if (y < x) x = y; }
const int N = 2001;
const ll big = (ll)1e18 + 1;
int p[N], t[N];
ll dp[N][N];

int main()
{
    freopen(NAME"inp", "r", stdin);
    freopen(NAME"out", "w", stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int n, T;
    cin >> n >> T;
    pair<int, int> tmp[n];
    for (int i = 0; i < n; i++)
        cin >> tmp[i].fi >> tmp[i].se;
    sort(tmp, tmp + n, [] (pii x, pii y) { return x >= y; });
    //sort(tmp, tmp + n, [] (auto x, auto y) { return x >= y; });
    for (int i = 0; i < n; i++)
    {
        t[i] = tmp[i].fi;
        p[i] = tmp[i].se;
    }
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= T; j++)
            dp[i][j] = big;
    dp[0][0] = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j <= T; j++)
        {
            setmin(dp[i + 1][j], dp[i][j]);
            if (j + t[i] <= T)
                setmin(dp[i + 1][j + t[i]], dp[i][j] + p[i]);
        }
    ll ans = big, sumP = 0;
    int sumT = 0;
    for (int i = n - 1; i >= 0; i--)
    {
        for (int j = max(0, T - sumT - t[i] + 1); j <= T - sumT; j++)
            setmin(ans, dp[i][j] + sumP);
        sumT += t[i];
        sumP += p[i];
    }
    if (sumT <= T)
        setmin(ans, sumP);
    cout << ans << "\n";
    cout << "\nTime: " << clock() / (double)1000 << " sec";
}
```



Để chuẩn bị thi Olympic Tin học mỗi học sinh phải tích lũy nhiều kiến thức về cấu trúc dữ liệu và thuật toán, kỹ năng sử dụng công cụ và nghệ thuật triển khai giải thuật. Các vấn đề đó chỉ có thể giải quyết được khi bạn có một quá trình khổ luyện lâu dài, hay nói một cách khác – phải làm rất nhiều bài tập. Có những bài giống như con kỳ nhông, thoát nhìn có vẻ rất dễ chịu, nhưng càng nghĩ càng thấy răm rối đến nỗi có khi bạn phải muôn rú lên cho nhẹ đầu! 😊😊😊

Alice đang chuẩn bị cho kỳ thi Olympic sắp tới gần và dĩ nhiên cô làm rất nhiều bài tập. Để giải tỏa tâm lý căng thẳng Alice có thói quen viết vào trang cuối của vở học ký tự ‘**A**’ ghi nhận độ phức tạp của bài toán. Bài càng khó thì thời gian suy nghĩ càng dài và số lượng ký tự ‘**A**’ ứng với nó càng nhiều hơn. Ngoài ra, trong quá trình suy nghĩ có những lúc Alice viết thêm một cách vô thức các ký tự la tinh thường.

Kết thúc một đợt tập huấn Alice tự thưởng cho mình một ngày nghỉ, ngắm nhìn tuyệt tác lập trình mình đã sáng tạo trong thời gian qua. Cũng có đôi chỗ bỗng nhiên trở nên khó hiểu, Alice phải lật lại vở học để xem và ánh mắt của cô bắt gặp xâu ký tự ghi nhận độ khó của các bài. Alice tự nhắc “Đây là lúc tổng kết. Cần xem mình đã làm được nhiêu nhất bao nhiêu bài!” Chỉ có một điều duy nhất bây giờ cô biết chắc chắn là không có hai bài đã làm nào có cùng độ khó.

Hãy xác định nhiêu nhất có bao nhiêu bài đã giải.

Dữ liệu: Vào từ file văn bản TASKS.INP gồm một dòng chứa xâu **s** độ dài không quá 10^6 chỉ bao gồm các ký tự la tinh thường và ký tự ‘**A**’. Trong xâu có ít nhất một ký tự ‘**A**’.

Kết quả: Đưa ra file văn bản TASKS.OUT một số nguyên – số lượng các bài đã làm lớn nhất có thể.

Ví dụ:

TASKS.INP	TASKS.OUT
dfsAAfftaAbcdAAtoshaAtoAApA	3



Giải thuật; Cơ sở lập trình .

Gọi **A** – số lượng ký tự ‘**A**’ trong **s**, **k** – số bài đã làm,

Để số lượng bài làm được là nhiều nhất, độ khó của các bài phải thỏa mãn các điều kiện:

- Độ khó bài thứ **i** là **i**, **i** = 1, 2, . . . , **k-1**,
- Độ khó bài thứ **k** lớn hơn độ khó bài **k-1**,
- Tổng độ khó tất cả các bài bằng **A**.

Như vậy **k** là số nguyên dương lớn nhất có tổng các số từ 1 đến **k** không vượt quá **A**.

Độ phức tạp của giải thuật: lớp O(n).

Chương trình

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

int main()
{
    freopen("tasks.inp", "r", stdin);
    freopen("tasks.out", "w", stdout);
    iosstream::sync_with_stdio(false);
    cin.tie(nullptr); cout.tie(nullptr);

    string s;
    cin>>s;
    int A = count(s.begin(), s.end(), 'A');

    int k = 0;
    while ((k + 1) * int64_t(k + 2) / 2 <= A)
        ++k;

    cout << k << "\n";
    return 0;
}
```



VY39. MÃ KHÓA

Tên chương trình: MAXKEY.CPP

Cửa vào kho lưu trữ tài liệu mật được bảo vệ bằng khóa điện tử. Có 10 nút bấm tương ứng với các chữ số từ 0 đến 9. Dưới mỗi nút có đèn hiển thị số lần tối đa có thể bấm nút đó. Muốn vào được cần bấm các nút để tạo ra một số nguyên lớn nhất thỏa mãn các tính chất sau:

- ✿ Không có các chữ số 0 không có nghĩa ở đầu,
- ✿ Ba chữ số liên tiếp nhau trong số đã tạo ra phải làm thành một số chia hết cho 3.

Không nhất thiết phải bấm hết số lần tối đa cho phép ở mỗi nút.

Hãy xác định số cần tạo ra để mở cửa.

Dữ liệu: Vào từ file input chuẩn của hệ thống gồm một dòng chứa 10 số nguyên c_0, c_1, \dots, c_9 , trong đó c_i – số lần tối đa được bấm nút i , $0 \leq c_i \leq 10^5$, $i = 1 \div 9$.

Kết quả: Đưa ra file output chuẩn của hệ thống số nguyên xác định được.

Ví dụ:

INPUT	OUTPUT
1 2 3 0 0 0 0 0 0 0	21021



Giải thuật; Lý thuyết số, Kỹ thuật duyệt tìm kiếm quay lui .

Nếu 2 chữ số đầu tiên đã xác định được thì các chữ số tiếp theo có thể xác định đơn trị với độ chính xác $mod\ 3$.

Gọi **a, b** – 2 chữ số đầu tiên, **c** – chữ số tiếp theo. Ta có

$$(a+b+c) \equiv 0 \pmod{3}$$

Trong số các **c** thích hợp: chọn **c** lớn nhất.

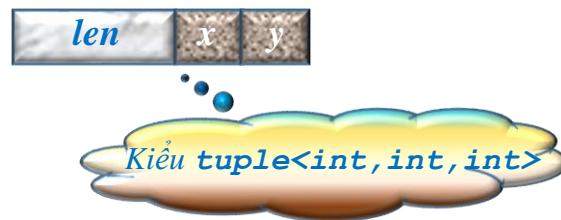
Việc chọn 2 chữ số đầu tiên: Vết cạn các khả năng (các cặp chữ số có thể).

Cặp số đầu tiên sẽ xác định dãy các chữ số tiếp theo: chữ số lớn nhất phù hợp và có số lượng còn lạ khác 0.

Số lượng chữ số có nghĩa càng (gọi tắt là độ dài) nhiều thì số càng lớn.

Khóa để điều khiển tìm kiếm là cặp số đầu tiên (**x, y**) và độ dài tối đa có thể đạt được xuất phát từ cặp số đầu tiên này.

Biểu diễn khóa:



Trong quá trình tìm kiếm chỉ cần lưu lại khóa.

Quá trình giải bài toán gồm hai bước:

- ✿ Tìm độ dài số cần dẫn xuất,
- ✿ Dẫn xuất số tìm được.

Sau khi đã tìm được khóa tối ưu – dẫn xuất lại số cần đưa ra.

Khởi tạo giá trị đầu cho khóa: độ dài bằng 1 và **x = y** = chữ số xuất phát.

Tổ chức dữ liệu:

☰ Mảng **vector<int>** c (10) – lưu dữ liệu vào.

Xử lý:

Tìm độ dài số cần dẫn xuất:

```
triple go(int x, int y, vector<int> c, bool print)
{
    int xx = x, yy = y;
    if (print) cout << x << y;
    int len = 2;
    while (true)
    {
        int next = -1;
        for (int d = (99 - x - y) % 3; d < n; d += 3)
            if (c[d] > 0) next = d;

        if (next >= 0)
        {
            c[next] -= 1;
            len += 1;
            x = yy;
            y = next;
            if (print) cout << next;
        } else break;
    }
    return triple(len, xx, yy);
}
```

Độ phức tạp của giải thuật: lớp O(m), trong đó $m = \max\{c_i\}$

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
typedef tuple<int, int, int> triple;
int const n = 10;

triple go(int x, int y, vector<int> c, bool print)
{
    int xx = x, yy = y;
    if (print) cout << x << y;
    int len = 2;
    while (true)
    {
        int next = -1;
        for (int d = (99 - x - y) % 3; d < n; d += 3)
            if (c[d] > 0) next = d;
        if (next >= 0)
        {
            c[next] -= 1;
            len += 1;
            x = y;
            y = next;
            if (print) cout << next;
        } else break;
    }
    return triple(len, xx, yy);
}

int main()
{
    //freopen("inp", "r", stdin);
    //freopen("maxkey.out", "w", stdout);
    ios::sync_with_stdio(false);
    vector<int> c(n);
    for (int i = 0; i < n; ++i) cin >> c[i];
    triple ans(0, 0, 0);
    for (int i = 0; i < n; ++i)
        if (c[i] > 0) ans = max(ans, triple(1, i, i));
    for (int i = 1; i < n; ++i) {
        if (c[i] > 0)
        {
            c[i] -= 1;
            for (int j = 0; j < n; ++j)
                if (c[j] > 0)
                {
                    c[j] -= 1;
                    ans = max(ans, go(i, j, c, false));
                    c[j] += 1;
                }
            c[i] += 1;
        }
    }
    int len, x, y;
    tie(len, x, y) = ans;
    if (len == 1) cout << x << '\n';
    else
    {
        c[x] -= 1;
        c[y] -= 1;
        go(x, y, c, true);
        cout << '\n';
    }
    cout << "\nTime: " << clock() / (double) 1000 << " sec";
}
```



B11. ĐOẠN THẮNG DẺO

Tên chương trình: ELASTIC.CPP

Nhà toán học Brawn có nhiều công trình nghiên cứu về số học, đặc biệt là về tính chất các số tự nhiên. Một trong số các công trình đó là đoạn thẳng dẻo.

Ông gọi đoạn $[1f, rt]$ chứa các số nguyên dương $1f, 1f+1, \dots, rt$ là đoạn dẻo độ dài $rt - 1f + 1$ nếu mỗi số trên đoạn này có thể thay đổi một đơn vị (thêm hoặc bớt 1) để tích các số trên đoạn thẳng không thay đổi. Nói một cách khác, tồn tại dãy số $a_{1f}, a_{1f+1}, \dots, a_{rt}$ thỏa mãn các điều kiện:

- $a_k = k \pm 1$,
- $1f \times (1f+1) \times \dots \times rt = a_{1f} \times a_{1f+1} \times \dots \times a_{rt}$.

Ông đã chứng minh được với n cho trước, trong nhiều trường hợp có thể tìm ra đoạn dẻo độ dài n và ra bài tập cho sinh viên với n đã cho, hãy xác định có tồn tại đoạn dẻo độ dài n hay không, nếu có – đưa ra giá trị $1f$ và chỉ ra các thay đổi để nhận được các số a_k tương ứng, dấu $+$ cho trường hợp $a_k = k+1$ và dấu trừ cho trường hợp $a_k = k-1$.

Dữ liệu: Vào từ file INPUT chuẩn của hệ thống gồm một dòng chứa số nguyên n ($1 \leq n \leq 10000$).

Kết quả: Đưa ra file OUTPUT chuẩn của hệ thống:

- ✿ Dòng đầu tiên chứa thông báo **YES** hoặc **NO**,
- ✿ Trong trường hợp **YES**:
 - ✿ Dòng thứ 2 chứa một số nguyên $1f$ xác định điểm đầu tùy chọn,
 - ✿ Dòng thứ 3 chứa xâu chỉ bao gồm các ký tự từ tập $\{+, -\}$ xác định cách biến đổi số.

Ví dụ:

INPUT	OUTPUT
4	YES 2 +-+-



B11.StP20181028_G

A XIV

Giải thuật; Kỹ năng phân tích mô hình toán học .

Nếu đoạn $[lf, rt]$ là dẻo thì đoạn $[lf, rt+2]$ cũng là đoạn dẻo: Trong dãy số biến đổi chỉ cần xác định thêm $a_{rt+1} = rt+2$ và $a_{rt+2} = rt + 1$.

Các đoạn $[1, 2]$ và $[2, 3]$ đều là các đoạn dẻo, từ đó suy ra mọi đoạn độ dài chẵn đều là đoạn dẻo!

Với n lẻ:

- $n = 1$ – Vô nghiệm,
- $n > 1$: đoạn $[3, 5]$ là đoạn dẻo vì $3 \times 4 \times 5 = (3-1) \times (4+1) \times (5+1) = 60$.

Như vậy có thể dễ dàng xác định đoạn dẻo độ dài lẻ bất kỳ!

Bài toán có vô số nghiệm với $n > 1$.

Bắt đầu với $lf = 3$ ta có thể chỉ ra cách dẫn xuất đoạn dẻo có độ dài bất kỳ:

- ✿ Với n chẵn: lặp lại xâu “ $+ -$ ” $n/2$ lần,
- ✿ Với n lẻ: Bổ sung vào xâu “ $- + +$ ” $(n-3)/2$ lần xâu “ $+ -$ ”.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include<<bits/stdc++.h>
using namespace std;
int n;
string s;

int main()
{
    cin >> n;
    if (n==1) {cout << "NO"; return 0;}
    cout << "YES\n3\n";
    if(n&1)s="-++",n-=3;
    for(int i=0; i<n/2; ++i) s+="+-";
    cout << s;
}
```



Các tests cho một bài thường được đánh số từ 1 đến n . Hệ thống quản lý files thường đưa ra danh sách tên files theo trình tự sắp xếp theo thứ tự tăng dần vì vậy file 10 sẽ đứng trước file 2, file 25 đứng trước file 3, ... Điều này sẽ đưa lại một số rắc rối không cần thiết cho hệ thống xử lý tự động. Chính vì vậy các tên files thường được cân bằng độ dài bằng các chữ số 0 không có nghĩa ở đầu trước, ví dụ 01, 02, 03, ...

Alice cần sử dụng một bài tập cũ của mình. Đáng tiếc, chất lượng đĩa không còn được như lúc đầu. Thời gian đã xóa đi nhiều ký úc trong con người và cũng xóa đi nhiều files trên chiếc đĩa này. Alice chỉ còn đọc được k tên files tests (còn nội dung của những files này thì cũng chưa biết có đọc được không!). Điều đầu tiên Alice nghĩ tới lúc này là không biết ban đầu mình đã soạn tối thiểu và tối đa là bao nhiêu tests.

Hãy xác định số lượng tối thiểu và tối đa tests đã được soạn.

Dữ liệu: Vào từ file INPUT chuẩn của hệ thống:

- ✚ Dòng đầu tiên chứa số nguyên k ($1 \leq k \leq 1000$),
- ✚ Mỗi dòng trong k dòng sau chứa xâu độ dài không quá 9 xác định tên một file test, các ký tự của xâu chỉ là ký tự số, không có xâu rỗng và không có xâu nào chỉ chứa ký tự ‘0’.

Kết quả: Đưa ra file OUTPUT chuẩn của hệ thống, dòng đầu tiên chứa số lượng tối thiểu các tests, dòng thứ 2 chứa số lượng tối đa các tests.

Ví dụ:

INPUT	OUTPUT
3 05 10 08	10 99



Giải thuật: .Kỹ năng phân tích tình huống lô gic .

Số thứ tự các tests đều có cùng độ dài. Gọi **len** là độ dài của số thứ tự.

Số lượng tests tối đa sẽ là $10^{\text{len}-1}$.

Số lượng tests tối thiểu:

- ✳ Nếu mọi số trong input đều có số 0 ở đầu thì số lượng tests tối thiểu là số nhỏ nhất có **len** chữ số có nghĩa, tức là $10^{\text{len}-1}$,
- ✳ Nếu số lớn nhất trong input là số có **len** chữ số có nghĩa thì đó chính là số tests tối thiểu cần tìm.

Độ phức tạp của giải thuật: O(n).

Chương trình

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

int main()
{
    freopen("input", "r", stdin);
    freopen("unerase.out", "w", stdout);
    std::ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    int n, len = -1;
    int ans = 0;
    string s;
    cin >> n;
    for (int i=0; i<n; ++i)
    {
        cin >> s;
        len = s.size();
        ans = max(ans, atoi(s.c_str()));
    }

    int pw = 1;
    for (int i = 0; i < len - 1; ++i)
        pw *= 10;
    ans = max(ans, pw);
    cout << ans << '\n';
    for (int i = 0; i < len; ++i) cout << '9';
    cout << '\n';
    cout<<"\nTime: "<<clock()/(double)1000<<" sec";
    return 0;
}
```



VY41. BẢNG KẾT QUẢ

Tên chương trình: RES_TAB.CPP

Để chuẩn bị cho cuộc thi Olympic Tin học đồng đội sắp tới các bạn trong đội lấy bảng kết quả vòng thi loại ra phân tích rút kinh nghiệm.

Ở vòng loại có n bài, mỗi bài được nộp không quá m lần. Do là vòng loại nên quy tắc có nói rộng đôi chút. Nếu một bài nào đó đã có kết quả đúng thì sẽ được ghi nhận là đã giải được, không phụ thuộc vào việc sau đó có nộp lại và cho kết quả sai!

Biên bản là một bảng n dòng và m cột. Dòng thứ i xác định kết quả các lần nộp bài i . Ký tự ở cột j trong dòng i là dấu ‘+’ nếu kết quả đúng, dấu ‘-‘ – kết quả sai, ‘.’ – chưa nộp lần j .

Dựa vào bảng kết quả hãy xác định đội đã làm đúng được bao nhiêu bài.

Dữ liệu: Vào từ file INPUT chuẩn của hệ thống :

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n, m \leq 100$),
- ✚ Dòng thứ i trong n dòng sau chứa xâu độ dài m chỉ bao gồm các ký tự trong tập {+, -, .}.

Kết quả: Đưa ra file OUTPUT chuẩn của hệ thống một số nguyên – số lượng bài đã làm được.

Ví dụ:

INPUT	OUTPUT
4 5 ----+. +.... -----. +-..	3



VY41 StP20181111 BA AXIV

Giải thuật; Cơ sở lập trình.

Nhận dạng sự tồn tại ký tự ‘+’ trong mỗi xâu.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
int n,m,ans=0;
string s;

int main()
{
    freopen("input","r",stdin);
    freopen("Res_Tab.out","w",stdout);
    ios::sync_with_stdio(false);
    cin>>n>>m;
    for(int i=0; i<n; ++i)
    {
        cin>>s;
        ans+=s.find('+') != std::string::npos;
    }
    cout<<ans;
}
```



Trong chương trình huấn luyện các thành viên Đội Đặc nhiệm phải vượt qua bài kiểm tra Tiêu diệt mục tiêu di động. Có n mục tiêu lần lượt xuất hiện, học viên phải phát hiện và tiêu diệt mục tiêu bằng một phát đạn. Càng về sau độ khó của việc phát hiện mục tiêu càng tăng. Tiêu diệt mục tiêu đầu tiên học viên được 1 điểm, tiêu diệt mục tiêu thứ 2 – được 2 điểm, tiêu diệt mục tiêu thứ 3 – được 4 điểm, . . . Số điểm nhận được khi tiêu diệt mục tiêu tiếp theo là gấp đôi so với mục tiêu trước. Tổng điểm chỉ được thông báo cho học viên vào cuối buổi tập

Một học viên, sau khi hoàn thành bài kiểm tra, báo cáo với đội trưởng là mình được a điểm, nhưng nói thêm là có đúng một mục tiêu có khả năng mình không đánh giá đúng, đã bị tiêu diệt nhưng nghĩ là không hoặc ngược lại.

Hãy xác định số điểm tối thiểu và tối đa có thể nhận được của học viên đó.

Dữ liệu: Vào từ file INPUT chuẩn của hệ thống gồm một dòng chứa 2 số nguyên n và a ($1 \leq n \leq 63, 0 \leq a \leq 2^{63}-1$).

Kết quả: Đưa ra file OUTPUT chuẩn của hệ thống trên một dòng hai số nguyên xác định điểm tối thiểu và tối đa xác định được.

Ví dụ:

INPUT	OUTPUT
5 0	0 16



Giải thuật; Xử lý bít .

Tính giá trị *min*:

- ★ Xác định bít 1 trái nhất của **a** trong **n** bít cuối cùng,
- ★ Gọi **p** là vị trí bít 1 tìm được → đưa bít thứ **p** của **a** về 0 (tắt bít thứ **p**):

$$\text{min_a} = \text{a}^{\wedge} (1LL<<\text{p})$$

- ★ Nếu không tìm được **p**: **min_a=a**.

Tính giá trị *max*:

- ★ Xác định bít 0 trái nhất của **a** trong **n** bít cuối cùng,
- ★ Gọi **p** là vị trí bít 0 tìm được → đưa bít thứ **p** của **a** về 1 (bật bít thứ **p**):

$$\text{max_a} = \text{a} \mid (1LL<<\text{p})$$

- ★ Nếu không tìm được **p**: **max_a=a**.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
int64_t n, a, u=0, v, mn, mx, k=1;

int main()
{
    freopen("input", "r", stdin);
    freopen("Shooting.out", "w", stdout);
    ios::sync_with_stdio(false);
    cin>>n>>a;
    if(a)
    {
        v=n-1;
        for(int i=n-1; i>=0; --i)
            if((a&(k<<i))==0) {u=i; break;}
        for(int i=n-1; i>=0; --i)
            if(a&(k<<i)) {v=i; break;}
        mn=a^ (k<<v); mx=a | (k<<u);
    }
    else {mn=0; mx=1<<(n-1);}
}

cout<<mn<< ' ' <<mx;
}
```



Một chương trình nhận dạng tự động cho phép đọc số trên căn cước công dân, thẻ ngân hàng, mã số thuế, . . . đã giải phóng người dùng phải tạo thêm mã QR phức tạp.

Trong quá trình hoạt động, do sự bất cẩn của quản trị viên hệ thống bị nhiễm vi rút và chương trình không nhận dạng được k chữ số trong các số hệ thập phân, trở thành các chữ số bị hỏng. Những số nào không chứa chữ số bị hỏng vẫn được nhận dạng đúng.

Hãy xác định trong các số từ 1 đến n ở hệ thập phân có bao nhiêu số vẫn còn được nhận dạng đúng.

Dữ liệu: Vào từ file INPUT chuẩn của hệ thống:

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^{18}$),
- ✚ Dòng thứ 2 chứa số nguyên k – số lượng chữ số bị hỏng ($1 \leq k \leq 9$),
- ✚ Dòng thứ 3 chứa k số nguyên d_1, d_2, \dots, d_k – các chữ số bị hỏng ($0 \leq d_i \leq 9$, $d_i \neq d_j$ với $i \neq j$, $i, j = 1 \div k$).

Kết quả: Đưa ra file OUTPUT chuẩn của hệ thống một số nguyên – số lượng số còn được nhận dạng đúng.

Ví dụ:

INPUT	OUTPUT
100000 8 0 1 2 5 6 7 8 9	62



Giải thuật: Tổ chức dữ liệu, Kỹ thuật bảng phương án.

Giải thuật đơn giản: kiểm tra tất cả các số nguyên từ 1 đến n , với mỗi số - kiểm tra có tồn tại chữ số bị hỏng hay không, đếm các số nhận dạng được.

Chương trình Giải thuật đơn giản, độ phức tạp cao.

```
#include <bits/stdc++.h>
using namespace std;
int k, ln, gd, d, dd, t=0, p=1, db[10]={1,1,1,1,1,1,1,1,1,1};
string sn;
int64_t n, ans=0, f[2][20], df[20];

bool check(int64_t x)
{
    int tm;
    while(x)
    {
        tm=x%10; if(db[tm]==0) return 0;
        x/=10;
    }
    return 1;
}
int main()
{
    freopen("input", "r", stdin);
    freopen("readnum.out", "w", stdout);
    cin>>n; ln=sn.size();
    cin>>k;
    for(int i=0; i<k; ++i) {cin>>d; db[d]=0; }
    for(int i=1; i<=n; ++i) ans+=check(i);
    cout<<ans;
}
```

Giải thuật tối ưu:

Đánh dấu các số bị hỏng,

Tính tổng tiền tố các chữ số còn tốt: df_i – số lượng các chữ số tốt từ 0 tới i , $i = 0 \div 9$,

Gọi gd – số lượng chữ số tốt, ln – số lượng chữ số có nghĩa của n ,

Lập bảng $\text{pw}_i = gd^i$, $i = 0 \div ln-1$, f_i – tổng tiền tố của pw ,

Đánh số các chữ số tốt theo thứ tự tăng dần từ 0 đến $gd-1$,

Xây dựng mảng dn xác định số tốt lớn nhất không vượt quá n theo cách đánh số lại chữ số đã nêu.

Trường hợp 0 là chữ số tốt:

Mảng dn xác định số lượng số nhận dạng được theo cơ số gd . Kết quả cần tìm là chuyển đổi số do dn xác định sang hệ thập phân.

Trường hợp 0 là chữ số hỏng:

Duyệt các chữ số của n từ trái sang phải (từ bậc cao xuống bậc thấp),

Gọi m là vị trí đầu tiên gặp chữ số bị hỏng.

Nếu mọi chữ số của n đều tốt thì $m = -1$.

Số lượng các số tốt có đúng i chữ số là $\mathbf{pw}_i = \mathbf{gd}^i$, $i = 1, 2, \dots$

Số lượng các số tốt có không quá i chữ số là $\mathbf{gd}^1 + \mathbf{gd}^2 + \dots + \mathbf{gd}^i = \mathbf{f}_i$.

Số lượng các số tốt cần tìm bao gồm:

- 🌟 Số lượng số tốt có không quá $1n$ chữ số,
- 🌟 Số lượng số tốt có $1n$ chữ số và chữ số đầu nhỏ hơn chữ số đầu tiên của n ,
- 🌟 Số lượng số tốt có $1n$ chữ số và chữ số đầu bằng chữ số đầu tiên của n , chữ số thứ 2 nhỏ hơn chữ số thứ 2 của n ,
- 🌟 Số lượng số tốt có $1n$ chữ số và 2 chữ số đầu bằng 2 chữ số đầu tiên của n , chữ số thứ 3 nhỏ hơn chữ số thứ 3 của n ,
-
- 🌟 Số lượng số tốt có $1n$ chữ số và $m-1$ chữ số đầu bằng $m-1$ chữ số đầu tiên của n , chữ số thứ m nhỏ hơn chữ số thứ m của n .

Tổ chức dữ liệu:

- ☒ Xâu **string** sn – lưu số n ,
- ☒ Mảng **int** $db[10]$ – đánh dấu các chữ số tốt,
- ☒ Mảng **int** $df[10]$ – tích lũy tổng tiền tố số lượng chữ số tốt,
- ☒ Mảng **int** $dn[20]$ – ghi nhận số tốt lớn nhất không vượt quá n sau khi đánh số lại các chữ số tốt,
- ☒ Mảng **int64_t** $pw[20]$ – \mathbf{pw}_i lưu số lượng số tốt có đúng i chữ số và 0 là chữ số hỏng,
- ☒ Mảng **int64_t** $f[20]$ – lưu tổng tiền tố của \mathbf{pw} .

Độ phức tạp của giải thuật: bậc O(1) (Vì $n \leq 2^{63}$).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
int k, ln, m=-1, gd, d, dd, t=0, p=1, db[10]={1,1,1,1,1,1,1,1,1,1};
string sn;
int64_t ans=0, f[20], pw[20], df[20], dn[20];

int main()
{
    freopen("input", "r", stdin);
    freopen("readnum.out", "w", stdout);
    cin>>sn; ln=sn.size();
    cin>>k;
    for(int i=0; i<k; ++i) {cin>>d; db[d]=0;}
    df[0]=0;
    for(int i=0; i<=9; ++i) df[i]=df[i-1]+db[i];
    gd=10-k;
    if(db[0]==0)
    {
        f[0]=0; pw[0]=1;
        for(int i=1; i<ln; ++i) pw[i]=pw[i-1]*gd, f[i]=f[i-1]+pw[i];
    }

    for(int i=0; i<ln; ++i)
    {
        d=sn[i]-48;
        t=df[d]; dn[i]=t-1;
        if(db[d]==0) {m=i; for(int j=m+1; j<ln; ++j) dn[j]=gd-1; break;}
    }

    if(db[0]) for(int i=0; i<ln; ++i) ans=ans*gd+dn[i];
    else
    {
        for(int i=0; i<=m; ++i) ans+=dn[i]*pw[ln-1-i];
        if(m>=0) ans+=pw[ln-1-m];
        ans+=f[ln-1];
    }
}

cout<<ans;
}
```



Phòng trưng bày nghệ thuật dân tộc có n^2+1 chiếc cồng và chiêng, một chiếc bày ở giữa phòng, các chiếc còn lại treo trên tường. Để chuẩn bị cho lễ hội cồng chiêng, người ta dự định lấy tất cả n^2+1 chiếc cồng chiêng này sắp thành một hình chữ nhật kích thước $h \times w$ ô trang trí một bên của đường dẫn tới sân khấu sao cho:

- ✿ Mỗi ô có một chiếc cồng/chiêng và không có vị trí nào nào trống,
- ✿ $2 \leq h < w$,
- ✿ Chu vi của hình chữ nhật là lớn nhất.



Hãy xác định có thể bày được như dự kiến hay không, nếu được – đưa ra kích thước h và w .

Dữ liệu: Vào từ file INPUT chuẩn của hệ thống:

- ✿ Dòng đầu tiên chứa số nguyên q – số tests ($1 \leq q \leq 10^6$),
- ✿ Mỗi dòng trong q dòng sau chứa một số nguyên n ($1 \leq n \leq 10^6$).

Kết quả: Đưa ra file OUTPUT chuẩn của hệ thống giá trị -1 nếu không có cách bày hoặc 2 số nguyên h, w trên một dòng.

Ví dụ:

INPUT	OUTPUT
6	-1
1	-1
2	2 5
3	-1
4	2 13
5	5 65
18	



Giải thuật; Kỹ thuật bảng phương án.

Với cùng một diện tích chu vi của hình chữ nhật $h \times w$ sẽ lớn nhất khi h nhỏ nhất.

Với n lẻ, n^2+1 là số chẵn $\rightarrow h = 2$.

Với n chẵn: cần tìm ước số nhỏ nhất khác 1 của n^2+1 .

Xây dựng bảng $a_i =$ ước số nhỏ nhất khác 1 của i^2+1 với $i = 2, 4, \dots, 10^6$.

Xử lý truy vấn: tra bảng để tìm h và từ đó – tính w .

Chương trình Giải thuật đơn giản, độ phức tạp cao.

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

int main() {
    int q;
    freopen("input", "r", stdin);
    freopen("gongs.out", "w", stdout);
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>q;
    for (int i = 0; i < q; i++) {
        int x;
        cin>>x;
        long long y = x * 1LL * x + 1;
        bool found = false;
        for (int d = 2; d <= x; d++) {
            if (y % d == 0) {
                cout<< d<< ' '<< y / d<< '\n';
                found = true;
                break;
            }
        }
        if (!found) {
            cout<<"-1\n";
        }
    }
    cout<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```

Chương trình Giải thuật tối ưu.

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

int main()
{
    freopen("input", "r", stdin);
    freopen("gongs.out", "w", stdout);
    ios::sync_with_stdio(0);
    cin.tie(0);
    const int maxn = 1e6 + 10;
    vector<int64_t> ans(maxn, INT_MAX);
    vector<int64_t> cur(maxn);
    for (int i = 1; i < maxn; i++)
        cur[i] = i * 1LL * i + 1;
    for (int i = 2; i < maxn; i+=2)
    {
        int64_t dvr = cur[i];
        if (dvr == 1) continue;

        for (int64_t j = i; j < maxn; j += dvr)
        {
            while (cur[j] % dvr == 0) cur[j] /= dvr;
            ans[j] = min(ans[j], dvr);
        }
    }
    int q;
    cin>>q;
    for (int i = 0; i < q; i++)
    {
        int x;
        cin>>x; if(x&1)ans[x]=2;
        if (ans[x] == INT_MAX || ans[x] == x * 1LL * x + 1) cout<<"-1\n";
        else
            cout<<ans[x]<<' ' <<(x * 1LL * x + 1) / ans[x]<<'\n';
    }
    cout<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Việc mua bán trực tuyến ngày càng trở nên phổ biến. Khách hàng có thể chọn mua một món hàng cụ thể ở một cửa hàng cụ thể. Hàng hóa sẽ được chở và giao tận nhà cho khách hàng.

Hệ thống trực tuyến mới còn cung cấp dịch vụ, qua đó khách hàng mô tả cụ thể các đặc trưng của thứ mình muốn mua. Hệ thống sẽ giúp khách hàng chọn hàng phù hợp và chuyên chở từ một trong số các đại lý có hàng thuộc công ty tới nhà với tổng chi phí là nhỏ nhất. Như vậy vẫn bán được hàng và quan trọng nhất – chiếm được sự tín nhiệm của khách. Tuy nhiên, khách phải trả tiền xăng cho việc vận chuyển.

Với một yêu cầu mua bộ bàn ghế nội thất hệ thống tìm thấy có n cửa hàng đang có hàng phù hợp, bộ bàn ghế ở cửa hàng thứ i có giá là p_i , v_i lít xăng để giao hàng, chi phí vận chuyển là d_i (kích thước và trọng lượng khác nhau). Giá một lít xăng là c . Hệ thống thông báo tổng chi phí nhỏ nhất khách hàng cần chuẩn bị để thanh toán.

Hãy xác định tổng chi phí mà khách hàng được thông báo.

Dữ liệu: Vào từ file INPUT chuẩn của hệ thống:

- + Dòng đầu tiên chứa 2 số nguyên n và c ($1 \leq n, c \leq 100$),
- + Dòng thứ i trong n dòng sau chứa 3 số nguyên p_i , d_i và v_i ($1 \leq p_i, d_i, v_i \leq 100$).

Kết quả: Đưa ra file OUTPUT chuẩn của hệ thống một số nguyên – tổng chi phí thông báo cho khách hàng.

Ví dụ:

INPUT	OUTPUT
<pre>3 2 1 1 3 3 2 1 5 1 1</pre>	<pre>7</pre>



Giải thuật; Cơ sở lập trình.

Tìm $\min\{\mathbf{p}_i + \mathbf{d}_i + \mathbf{v}_i \times \mathbf{c}\}$

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

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

int main()
{
    freopen("input", "r", stdin);
    freopen("delivery.out", "w", stdout);
    ios::sync_with_stdio(0);
    cin.tie(0); int n, c;
    cin >> n >> c;
    int ans = INT_MAX;
    for (int i = 0; i < n; i++)
    {
        int x, y, z;
        cin >> x >> y >> z;
        ans = min(ans, x + y + c * z);
    }
    cout << ans << '\n';
    return 0;
}
```



Một hội nghị quốc tế quan trọng sắp được tổ chức tại thành phố. Kế hoạch đảm bảo an ninh được soạn thảo hết sức chu đáo.

Toàn bộ thành phố được xét như lưới ô vuông các cột được đánh số từ trái sang phải từ $-\infty$ đến $+\infty$, các hàng được đánh số từ dưới lên trên từ $-\infty$ đến $+\infty$. Khu vực diễn ra hội nghị là ô $(0, 0)$. Mọi thiết bị bay không người lái (drone) đều bị cấm trong thời gian diễn ra hội nghị, nếu xuất hiện sẽ bị bắn hạ. Một khẩu pháo laser được bố trí tại ô $(0, 0)$, cứ mỗi giây có thể phát một chùm laser công suất cao phá hủy một drone ở độ cao và khoảng cách bất kỳ. Để kiểm tra khả năng tác chiến người ta tiến hành một cuộc diễn tập với tình huống giả định là xuất hiện n drones (đánh số từ 1 đến n), drone thứ i ($i = 1 \div n$) xuất hiện ở không phận thuộc ô (x_i, y_i) , cứ sau mỗi giây drone có thể chuyển sang không phận ô kề cạnh hoặc kề đỉnh và hướng về phía ô $(0, 0)$. Trên không phận một ô có thể có đồng thời nhiều drones.

Nhiệm vụ của bộ phận bảo vệ là không để cho drone nào vào được không phận ô $(0, 0)$.

Hãy xác định trình tự các drones cần tiêu diệt. Trường hợp không thể thực hiện được nhiệm vụ đã nêu, đưa ra số -1 .

Dữ liệu: Vào từ thiết bị nhập chuẩn:

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên x_i và y_i ($|x_i|, |y_i| \leq 10^5$).

Kết quả: Đưa ra thiết bị xuất chuẩn số -1 hoặc dãy số nguyên xác định một trình tự drones cần bắn hạ.

Ví dụ:

INPUT	OUTPUT
<pre>3 0 1 -2 3 2 2</pre>	<pre>1 3 2</pre>



VY46 Io20181110 BF AXIV

Giải thuật; Tổ chức dữ liệu .

Sau mỗi giây drone có thể sang ô mới có tọa độ **x** hoặc **y** hay cả hai chênh lệch 1 so với tọa độ cũ.

Từ tọa độ ($\mathbf{x}_i, \mathbf{y}_i$) drone sẽ tới ô $(0, 0)$ sau $\max\{|\mathbf{x}_i|, |\mathbf{y}_i|\}$. Đó là khoảng cách từ drone tới khu vực bảo vệ.

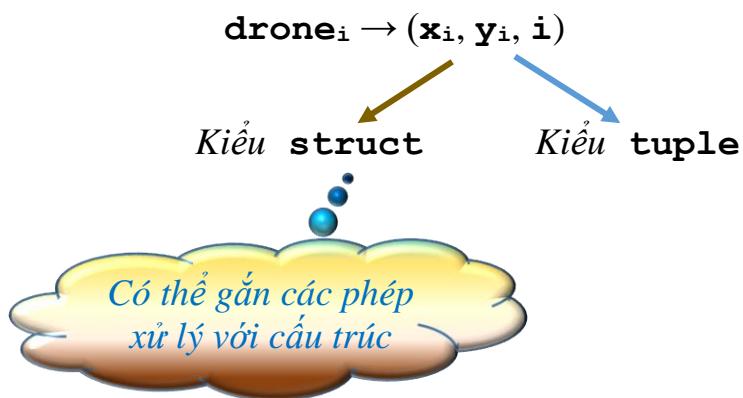
Người ta cần bắn hạ những drones gần mình nhất.

Nếu tồn tại một khoảng cách **d** có số lượng drones lớn hơn hoặc bằng **d** và cách tàu khoảng cách **d** thì không thể bắn hạ hết chúng trước khi tới khung phận ô $(0, 0)$.

Cần sắp xếp drones theo khoảng cách tăng dần tới vùng bảo vệ.

Trình tự sắp xếp trong mảng xác định trình tự cần bắn hạ drones.

Ghi nhận dữ liệu:



Ta chỉ cần quan tâm khoảng cách drone tới tàu nên có thể lưu trữ dữ liệu một cách đơn giản hơn:

$$\boxed{\text{drone}_i \rightarrow (\mathbf{x}_i, \mathbf{y}_i, i) \rightarrow (\max\{|\mathbf{x}_i|, |\mathbf{y}_i|\}, i)}$$

Tổ chức dữ liệu:

`vector<pii> drone(n)`
hoặc
 `vector<t3i> drone(n)` – Lưu thông tin về các drones.

Độ phức tạp của giải thuật: O(nlnn).

Chương trình Sử dụng cấu trúc pair.

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int,int> pii;
int n,x,y,id;

int main()
{
    ios_base::sync_with_stdio(0);
    freopen ("input", "r", stdin);
    freopen ("drones.out", "w", stdout);
    cin.tie(0);
    cout.tie(0);
    cin>>n;
    vector<pii> drone(n);
    for(int i=0; i<n; ++i)
    {
        cin>>x>>y;
        drone[i]={max(abs(x),abs(y)), i+1};
    }
    sort(drone.begin(),drone.end());
    for(int i=0; i<n; ++i)
    {
        if(drone[i].first<=i)
        {
            cout<<"-1";
            return 0;
        }
    }
    for(auto d:drone) cout<<d.second<<' ';
    cout<<"\nTime: "<<clock()/(double)1000<<" sec";
    return 0;
}
```

Chương trình Sử dụng cấu trúc tuple.

```
#include <bits/stdc++.h>
using namespace std;
typedef tuple<int,int,int> t3i;
int n,x,y,id;
bool cmp(t3i a,t3i b) {return
max(abs(get<0>(a)),abs(get<1>(a)))<max(abs(get<0>(b)),abs(get<1>(b)));
int main()
{
    ios_base::sync_with_stdio(0);
    freopen ("input", "r", stdin);
    freopen ("drones.out", "w", stdout);
    cin.tie(0);
    cout.tie(0);
    cin>>n;
    vector<t3i> drone(n);
    for(int i=0; i<n; ++i)
    {
        cin>>x>>y;
        drone[i]=make_tuple(x,y,i+1);
    }
    sort(drone.begin(),drone.end(),cmp);
    for(int i=0; i<n; ++i)
    {
        tie(x,y,id)=drone[i];
        if(max(abs(x),abs(y))<=i)
        {
            cout<<"-1";
            return 0;
        }
    }
    for(auto d:drone) cout<<get<2>(d)<<' ';
    cout<<"\nTime: "<<clock()/(double)1000<<" sec";
    return 0;
}
```

Chương trình Sử dụng cấu trúc **struct**.

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

int solve();

int main()
{
    ios_base::sync_with_stdio(0);
    freopen ("input", "r", stdin);
    freopen ("drones.out", "w", stdout);
    cin.tie(0);
    cout.tie(0);

    solve();
    cout<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}

struct dron
{
    int x, y, num;

    dron(int _x, int _y) : x(_x), y(_y), num(0) {}
    dron() {}

    int way()
    {
        x = abs(x), y = abs(y);
        return max(x, y);
    }

    bool operator< (dron t)
    {
        return (way() < t.way());
    }
};

int solve()
{
    int n;
    cin >> n;
    vector <dron> drons(n);
    for (int i = 0; i < n; ++i)
    {
        cin >> drons[i].x >> drons[i].y;
        drons[i].num = i + 1;
    }

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

    for (int i = 0; i < drons.size(); ++i)
        if (drons[i].way() <= i)
        {
            cout << -1;
            return 0;
        }

    for (auto d : drons)
        cout << d.num << " ";
    return 0;
}
```



Để chuẩn bị làm một cây đàn T'rưng cần tìm được cây tre hoặc vầu cái, có đốt dài và đều, dáng thuôn, không có lỗi như nứt, lỗ sâu, . . . Tóm lại, tìm được một cây ưng ý là rất khó. Một nghệ nhân, khi vào rừng hái lá thuốc tìm một cây tre rất ưng ý, nhưng đường đi ra phải qua một hẻm núi độ rộng n và dài m bước chân.

Có thể coi hẻm núi như lối ô vuông $n \times m$, một số ô có đá cao. Cây tre vác trên vai nên khi đi nó sẽ song song với một cạnh của hẻm núi. Cần phải đi từ trái sang phải. Có thể vào hẻm từ ô bất kỳ bên trái với cây tre song song cạnh trái. Khi cây tre đang ở hướng dọc, chỉ có thể di chuyển dọc lên trên hoặc xuống dưới, nếu cây tre đang ở hướng ngang – chỉ có thể di chuyển sang trái hoặc phải. Trong trường hợp cần chuyển hướng có thể chống một đầu cây tre xuống đất, dựng thẳng đứng và cho đó sang hướng song song với cạnh kia. Dĩ nhiên tre phải được nằm trên các ô không có đá cao. Để ra khỏi hẻm núi phải tới được ô ở cạnh phải với cây tre nằm song song cạnh này. Cần chặt cây tre thành các đoạn ngắn để mang về. Các đoạn đó phải càng dài càng tốt để có nhiều lựa chọn cắt khúc khi làm đàn.



Trạng thái hẻm núi được thể hiện bằng bảng ký tự kích thước $n \times m$, ký tự ‘.’ chỉ vị trí trống, ‘#’ chỉ ô có đá.

Hãy xác định độ dài lớn nhất của đoạn tre có thể mang qua (đơn vị độ dài là cạnh của ô).

Dữ liệu: Vào từ file INPUT chuẩn của hệ thống:

- ⊕ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n, m \leq 300$),
- ⊕ Mỗi dòng trong n dòng sau chứa xâu s độ dài m mô tả trạng thái một dòng của hẻm núi.

Kết quả: Đưa ra file OUTPUT chuẩn của hệ thống một số nguyên – độ dài lớn nhất của đoạn tre có thể mang qua.

Ví dụ:

INPUT	OUTPUT
3 5 ...## .#.#. . ##...	2

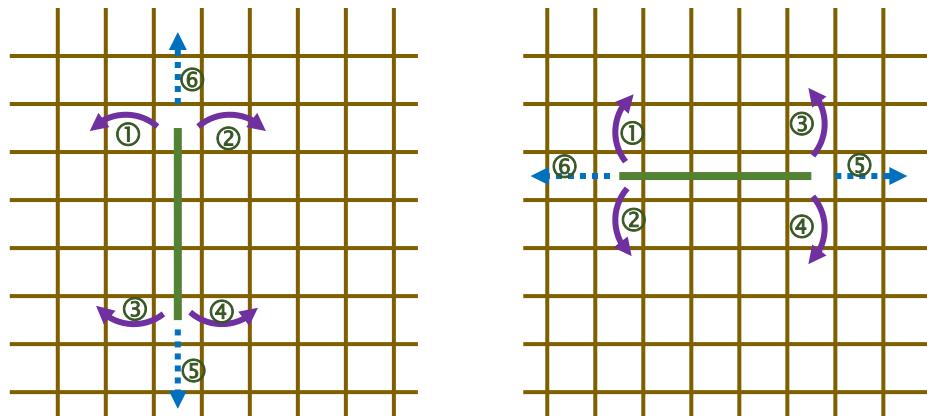


VY47 Io20181110 BG AXIV

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

Nếu có thể mang thanh tre độ dài k qua được thì có thể mang thanh độ dài nhỏ hơn qua, vì vậy có thể tìm độ dài lớn nhất bằng *phương pháp tìm kiếm nhị phân*.

Mỗi trạng thái của thanh tre có *6 cách di chuyển*:



Để quản lý thanh tre cần *hai tham số*:

- + **Tọa độ điểm đầu trên trái:** (x, y),
- + **Hướng** thanh tre: q , $q = 0$ – nằm dọc, $q = 1$ – ngang.

Để xác định khả năng thực hiện phép lật chuyen hướng thanh tre cần hai mảng tích lũy số lượng các ô trống liên tiếp theo hàng và cột.

Việc kiểm tra một thanh tre độ dài k có thể đưa qua khe núi hay không có thể thực hiện bằng phương pháp loang theo chiều sâu xác định có thể đưa điểm đầu thanh tre tới những vị trí nào. Cần có 2 mảng đánh dấu những *ô đã tới* và *hướng* nằm của thanh tre khi tới vị trí đó.

Thanh tre có thể đưa qua khe núi nếu kết quả loang cho biết đưa được điểm đầu thanh tre về cột phải nhất với hướng dọc.

Tổ chức dữ liệu:

- ▀ Mảng **string s[301]** – lưu dữ liệu input,
- ▀ Mảng **int le[301][301]** – tích lũy số ô trống liên tiếp theo hàng,
- ▀ Mảng **int up[301][301]** – tích lũy số ô trống liên tiếp theo cột,
- ▀ Mảng **int used[301][301][2]** – đánh dấu phục vụ loang.

Xử lý:

Tích lũy ô trống liên tiếp theo hàng:

```
cin >> n >> m;
for (int i = 0; i < n; i++)
{
    cin >> s[i];
    if (s[i][0] == ',') le[i][0] = 1;
    for (int j = 1; j < m; j++)
        if (s[i][j] == '#') le[i][j] = 0;
        else le[i][j] = le[i][j - 1] + 1;
}
```

Tích lũy ô trống liên tiếp theo cột:

```

for (int j = 0; j < m; j++)
{
    if (s[0][j] == ',') up[0][j] = 1;
    for (int i = 1; i < n; i++)
        if (s[i][j] == '#') up[i][j] = 0;
        else up[i][j] = up[i - 1][j] + 1;
}

```

Tìm kiếm nhị phân:

```

int le = 0, ri = min(n, m) + 1;
while (ri - le > 1)
{
    mi = (le + ri) / 2;
    bool d = 0;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            used[i][j][0] = used[i][j][1] = 0;

    for (int i = 0; i + mi <= n; i++)
        if (up[i + mi - 1][0] >= mi && used[i][0][0] == 0)
            dfs(i, 0, 0);

    for (int i = 0; i + mi <= n; i++)
        if (used[i][m - 1][0]) { d = 1; break; }

    if (d) le = mi; else ri = mi;
}

```

Loang từ điểm đầu (\mathbf{x}, \mathbf{y}) theo hướng xuất phát ban đầu là đọc ($\mathbf{q} = 0$):

```

// Cách di chuyển 1
if (le[x][y] >= mi && used[x][y - mi + 1][1] == 0)
    dfs(x, y - mi + 1, 1);
// Cách di chuyển 2
if (y + mi <= m && le[x][y + mi - 1] >= mi && used[x][y][1] == 0)
    dfs(x, y, 1);
// Cách di chuyển 3
if (le[x + mi - 1][y] >= mi && used[x + mi - 1][y - mi + 1][1] == 0)
    dfs(x + mi - 1, y - mi + 1, 1);
// Cách di chuyển 4
if (y + mi <= m && le[x + mi - 1][y + mi - 1] >= mi && used[x + mi - 1][y][1] == 0)
    dfs(x + mi - 1, y, 1);
// Cách di chuyển 5
if (x + mi < n && up[x + mi][y] >= mi && used[x + 1][y][0] == 0)
    dfs(x + 1, y, 0);
// Cách di chuyển 6
if (x > 0 && up[x + mi - 2][y] >= mi && used[x - 1][y][0] == 0)
    dfs(x - 1, y, 0);

```

Loang từ điểm đầu (**x**, **y**) theo hướng xuất phát ban đầu là ngang (**q** = 1):

```
// Cách di chuyển 1
if (up[x][y] >= mi && used[x - mi + 1][y][0] == 0)
    dfs(x - mi + 1, y, 0);
// Cách di chuyển 2
if (x + mi <= n && up[x + mi - 1][y] >= mi && used[x][y][0] == 0)
    dfs(x, y, 0);
// Cách di chuyển 3
if (up[x][y + mi - 1] >= mi && used[x - mi + 1][y+mi-1][0] == 0)
    dfs(x - mi + 1, y + mi - 1, 0);
// Cách di chuyển 4
if (x+mi<=n && up[x+mi-1][y+mi-1]>=mi && used[x][y+mi-1][0] == 0)
    dfs(x, y + mi - 1, 0);
// Cách di chuyển 5
if (y + mi < m && le[x][y + mi] >= mi && used[x][y + 1][1] == 0)
    dfs(x, y + 1, 1);
// Cách di chuyển 6
if (y > 0 && le[x][y + mi - 2] >= mi && used[x][y - 1][1] == 0)
    dfs(x, y - 1, 1);
```

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

Chương trình

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

int mi;
int n, m;
int up[301][301], le[301][301], used[301][301][2];
string s[301];

void dfs(int x, int y, int q)
{
    used[x][y][q] = 1;
    if (q == 1)
    {
        if (up[x][y] >= mi && used[x - mi + 1][y][0] == 0)
            dfs(x - mi + 1, y, 0);
        if (x + mi <= n && up[x + mi - 1][y] >= mi && used[x][y][0] == 0)
            dfs(x, y, 0);
        if (up[x][y + mi - 1] >= mi && used[x - mi + 1][y+mi-1][0] == 0)
            dfs(x - mi + 1, y + mi - 1, 0);
        if (x+mi<=n && up[x+mi-1][y+mi-1]>=mi && used[x][y+mi-1][0] == 0)
            dfs(x, y + mi - 1, 0);
        if (y + mi < m && le[x][y + mi] >= mi && used[x][y + 1][1] == 0)
            dfs(x, y + 1, 1);
        if (y > 0 && le[x][y + mi - 2] >= mi && used[x][y - 1][1] == 0)
            dfs(x, y - 1, 1);
    }
    else
    {
        if (le[x][y] >= mi && used[x][y - mi + 1][1] == 0)
            dfs(x, y - mi + 1, 1);
        if (y + mi <= m && le[x][y + mi - 1] >= mi && used[x][y][1] == 0)
            dfs(x, y, 1);
        if (le[x + mi - 1][y] >= mi && used[x + mi - 1][y - mi + 1][1] == 0)
            dfs(x + mi - 1, y - mi + 1, 1);
        if (y+mi<=m && le[x+mi-1][y+mi-1]>=mi && used[x+mi-1][y][1] == 0)
            dfs(x + mi - 1, y, 1);
        if (x + mi < n && up[x + mi][y] >= mi && used[x + 1][y][0] == 0)
            dfs(x + 1, y, 0);
        if (x > 0 && up[x + mi - 2][y] >= mi && used[x - 1][y][0] == 0)
            dfs(x - 1, y, 0);
    }
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    freopen("input", "r", stdin);
    freopen("bamboo.out", "w", stdout);
    cin >> n >> m;
    for (int i = 0; i < n; i++)
    {
        cin >> s[i];
        if (s[i][0] == '.') le[i][0] = 1;
        for (int j = 1; j < m; j++)
            if (s[i][j] == '#') le[i][j] = 0;
            else le[i][j] = le[i][j - 1] + 1;
    }
    for (int j = 0; j < m; j++)
    {
```

```

if (s[0][j] == '.') up[0][j] = 1;
for (int i = 1; i < n; i++)
    if (s[i][j] == '#') up[i][j] = 0;
    else up[i][j] = up[i - 1][j] + 1;
}

int le = 0, ri = min(n, m) + 1;
while (ri - le > 1)
{
    mi = (le + ri) / 2;
    bool d = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            used[i][j][0] = used[i][j][1] = 0;

    for (int i = 0; i + mi <= n; i++)
        if (up[i + mi - 1][0] >= mi && used[i][0][0] == 0)
            dfs(i, 0, 0);

    for (int i = 0; i + mi <= n; i++)
        if (used[i][m - 1][0]) { d = 1; break; }
    if (d) le = mi; else ri = mi;
}
cout << le << endl;
cout<<"\nTime: "<<clock()/(double)1000<<" sec";
return 0;
}

```



Alice tham gia vào 3 mạng xã hội và muốn vào mỗi mạng với một mật khẩu riêng, khác với hai mật khẩu kia. Để khỏi quên Alice lấy xâu s ưa thích của mình, chặt thành 3 xâu con các ký tự liên tiếp nhau làm mật khẩu. Gọi 3 đoạn chặt được là a , b và c , ta có $a + b + c = s$, các mật khẩu sử dụng sẽ là $a+b$, $b+c$ và $a+c$. Hai cách chặt gọi là khác nhau nếu có ít nhất một xâu trong số a , b và c có ở trong cách chặt thứ nhất và không có trong cách chặt thứ hai.

Hãy xác định số cách chặt khác nhau để tạo được các mật khẩu đúng yêu cầu.

Dữ liệu: Vào từ file INPUT chuẩn của hệ thống gồm một dòng chứa xâu s độ dài không quá 5×10^5 chỉ gồm các ký tự la tinh thường.

Kết quả: Đưa ra file OUTPUT chuẩn của hệ thống một số nguyên – số cách chặt khác nhau có thể thực hiện.

Ví dụ:

INPUT	OUTPUT
ababcb	9



Giải thuật; Nguyên lý bù trừ, Ứng dụng hàm Z.

Gọi n là độ dài xâu s .

Số các chia s thành 3 xâu khác rỗng sẽ là $C_{n-1}^2 = \frac{(n-1) \times (n-2)}{2}$.

Cần loại bỏ các cách chia cho $a = b$ hoặc $b = c$.

Tính số cách chia cho $a = b$:

- Đếm số xâu con $s[0..2p-1]$ có $s[0..p-1] = s[p..2p-1]$,
- Việc so sánh: dùng hàm Z .

Tính số cách chia cho $b = c$: thực hiện tương tự khi đảo ngược s .

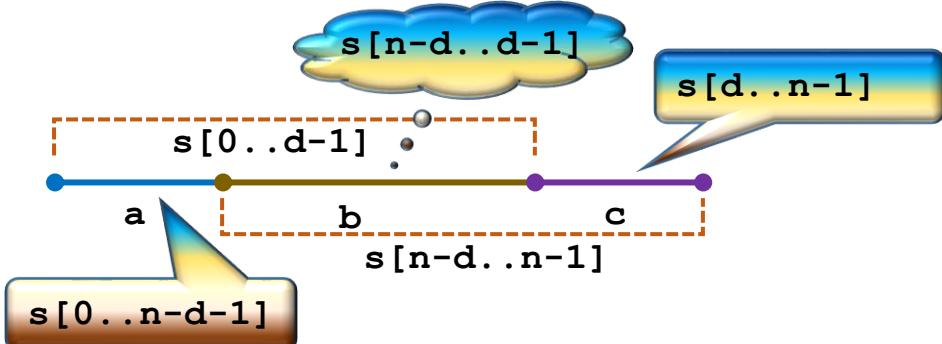
Cần chú ý: Trường hợp $a = b = c$, một cách phân chia bị loại bỏ 2 lần \rightarrow cần tăng kết quả số lượng cách phân chia còn lại lên 1.

Cần xét tiếp trường hợp $a \neq b, b \neq c$ nhưng $a + b = b + c$, đếm số lượng cách phân chia dẫn đến trường hợp này và loại bỏ khỏi kết quả hiện có.

Gọi d – độ dài xâu $a+b$.

Cần kiểm tra:

- ✳ $s[0..d-1] = s[n-d..n-1]$,
- ✳ $s[0..n-d-1] \neq s[n-d..d-1]$,
- ✳ $s[n-d..d-1] \neq s[d..n-1]$.



Tổ chức dữ liệu:

- ☰ Xâu s – lưu dữ liệu input,
- ☰ Mảng `vector<int>` z – lưu giá trị hàm Z với xâu s ,
- ☰ Mảng `vector<int>` zr – lưu giá trị hàm Z cho hậu tố với xâu s .

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

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

typedef long long ll;
int solve();

int main()
{
    ios_base::sync_with_stdio(0);
    freopen("input", "r", stdin);
    freopen("choice.out", "w", stdout);
    cin.tie(0); cout.tie(0);
    solve();
    cout<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}

ll ans = 0;
vector<int> Z(string s, bool need)
{
    int n = (int)s.size();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r) z[i] = min(r - i + 1, z[i - 1]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) z[i]++;
        if (i + z[i] - 1 > r)
            l = i,
            r = i + z[i] - 1;
        if (need && i + z[i] == n && i < z[i] && i * 2 != z[i]) ans--;
    }
    return z;
}

int solve()
{
    string s; cin >> s;
    int n = s.size();

    ans = (n - 111) * (n - 2) / 2;
    vector<int> z = Z(s, 1);

    reverse(s.begin(), s.end());
    vector<int> zr = Z(s, 0);

    for (int i = 2; i < n; i += 2)
    {
        if (i / 2 <= z[i / 2]) ans--;
        if (i / 2 <= zr[i / 2] && i / 2 * 3 != n) ans--;
    }

    cout << ans;
    return 0;
}
```



Giáo sư Brawn hướng dẫn n sinh viên làm các đề tài nghiên cứu khoa học. Với người thứ i ông yêu cầu cứ d_i ngày phải tới gặp ông báo cáo kết quả, tức là nếu a và b là 2 lần gặp liên tiếp thì $b-a = d_i$, $i = 1 \div n$.

Các ngày trong tuần được đánh số từ 1 đến 7. Lần cuối cùng mọi người đến gặp Giáo sư trong cùng một ngày là ngày thứ s .

Hãy xác định lần tiếp theo mọi người cùng gặp Giáo sư trong một ngày là ngày thứ mấy trong tuần.

Dữ liệu: Vào từ file INPUT chuẩn của hệ thống :

- ✚ Dòng đầu tiên chứa 2 số nguyên n và s ($1 \leq n \leq 10^5$, $1 \leq s \leq 7$),
- ✚ Dòng thứ 2 chứa n số nguyên d_1, d_2, \dots, d_n ($1 \leq d_i \leq 20$, $i = 1 \div n$).

Kết quả: Đưa ra file OUTPUT chuẩn của hệ thống một số nguyên – ngày xác định được.

Ví dụ:

INPUT	OUTPUT
<pre>3 1 2 5 10</pre>	<pre>4</pre>



Giải thuật; Cơ sở lập trình.

Chu kỳ tất cả mọi người cùng gặp nhau một lần là $d = \text{BSCNN}\{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n\}$.

Ngày gặp mặt chung tiếp theo sẽ là $(s-1+d) \% 7 + 1$.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

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

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int n, s;
    cin >> n >> s;
    int64_t ans = 1;
    for (int i = 0; i < n; i++)
    {
        int64_t x;
        cin >> x;
        ans = ans * x / __gcd(ans, x);
    }
    cout << (s - 1 + ans) % 7 + 1 << endl;
    cerr << "\nTime: " << clock() / (double)1000 << " sec";
}
```



Nhóm các nhà khoa học đi khảo sát thực địa một vùng rừng núi, họ thu được rất nhiều mẫu vật quý hiếm. Khi trở về họ gặp một trận mưa lớn làm ngập hẻm núi trên đường ra.

Hẻm núi có hình chữ nhật độ rộng n và độ dài m khúc, khúc thứ i có đá chia ra giữa chẵn a_i khúc lòng hẻm từ phải và chẵn b_i khúc hẻm từ phía trái, $i = 1 \dots m$.

Mọi người quyết định đóng một chiếc bè hình chữ nhật để chuyên chở thiết bị cùng mẫu vật vượt qua hẻm. Để an toàn, bè phải được đầy sao cho các cạnh luôn song song với cạnh của hẻm núi và không được đè lên các móm đá. Ngoài ra bè phải có diện tích lớn nhất để chở được nhiều đồ vật.

Hãy xác định diện tích lớn nhất của bè đóng được.

Dữ liệu: Vào từ file INPUT chuẩn của hệ thống:

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n \leq 10^9$, $1 \leq m \leq 5\,000$),
- ✚ Dòng thứ 2 chứa m số nguyên a_1, a_2, \dots, a_m ,
- ✚ Dòng thứ 3 chứa m số nguyên b_1, b_2, \dots, b_m .

$$0 \leq a_i, b_i, a_i + b_i \leq n, i = 1 \dots m.$$

Kết quả: Đưa ra file OUTPUT chuẩn của hệ thống một số nguyên – diện tích lớn nhất tìm được.

Ví dụ:

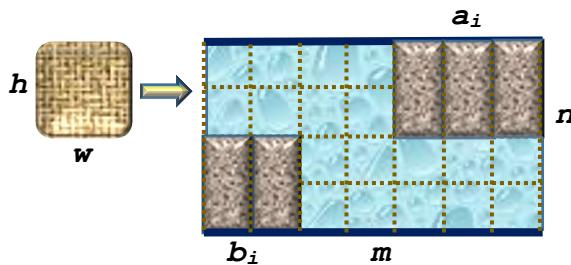
INPUT	OUTPUT
<pre>4 7 0 0 0 0 2 2 2 2 2 0 0 0 0 0</pre>	<pre>4</pre>



Giải thuật; Vết cạn .

Xét chiếc bè kích thước $\mathbf{h} \times \mathbf{w}$. Nếu chiếc bè ở các cột $i \div i + w - 1$ thì

$$\mathbf{h} \leq \mathbf{n} - \max\{\mathbf{a}_j, j = i \div i + w - 1\} - \max\{\mathbf{b}_j, j = i \div i + w - 1\}$$



Để đẩy được bè sang phải rồi sau đó – đẩy lên trên hoặc xuống dưới bất đẳng thức trên phải thỏa mãn với cả $j = i \div i + w$.

Giá trị m cần xét là không lớn ($m \leq 5000$) vì vậy có thể duyệt vết cạn để tìm các w thỏa mãn, từ đó tính \mathbf{h} và diện tích của bè. Trong số các w và \mathbf{h} thỏa mãn – tìm $\max\{\mathbf{h} \times \mathbf{w}\}$.

Các giá trị lớn nhất của a_i và b_i được sử dụng nhiều lần, vì vậy để có giải thuật hiệu quả cao cần lưu trữ $\max a_i$ và $\max b_i$, trong đó $\max a_i = \max\{a_j, j = 0 \div i\}$, $\max b_i = \max\{b_j, j = 0 \div i\}$.

Tổ chức dữ liệu:

- Các mảng `vector<int> a(m)` , `b(m)` – lưu dữ liệu vào,
- Các mảng `vector<int> maxa()` , `maxb(m)` – lưu các giá trị max đã nêu ở trên.

Độ phức tạp của giải thuật: $O(m^2)$.

Chương trình

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

int main()
{
    freopen("input", "r", stdin);
    freopen("output", "w", stdout);
    ios::sync_with_stdio(false);
    int n, m;
    cin >> n >> m;
    vector<int> a(m);
    for (int &x : a) cin >> x;
    vector<int> b(m);
    for (int &x : b) cin >> x;

    vector<int> maxa=a;
    vector<int> maxb=b;
    int64_t answer = 0;
    for (int w = 1; w <= m; ++w)
    {
        for (int i = 0; i < m - w; ++i)
        {
            maxa[i] = max(maxa[i], maxa[i + 1]);
            maxb[i] = max(maxb[i], maxb[i + 1]);
        }
        int window = min(w + 1, m);
        int h = n;
        for (int i = 0; i + window <= m; ++i)
            h = min(h, n - maxa[i] - maxb[i]);
        answer = max(answer, (int64_t) h * w);
    }

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



Tồn tại trào lưu nghệ thuật mang hội họa xuống đường phố. Người ta vẽ tranh lên phía ngoài các tường nhà để chúng không còn là các bê tông đơn điệu, buồn tẻ.

Một khu nhà được vây quanh bằng các bức tường song song với trục tọa độ, tạo thành đường gấp khúc khép kín cạnh không tự cắt n đỉnh có tọa độ nguyên, đỉnh thứ i có tọa độ (x_i, y_i) , $i = 1 \dots n$. Phía ngoài bức tường rào được trang trí bằng các hình vẽ theo những trường phái nghệ thuật khác nhau và thu hút sự hiếu kỳ của những ai đi ngang qua.

Nhưng khu nhà lại nằm giữa 4 đường cao tốc không cho phép dừng xe: hai đường theo hướng bắc – nam ở bên phải và bên trái, hai đường theo hướng đông – tây ở trên và ở dưới. Như vậy người ta chỉ nhìn thấy các phần của bức tường khi quan sát vuông góc theo hướng từ bắc xuống, từ nam lên, từ đông hoặc tây sang.

Không ít người đã bỏ nhiều thời gian đi vòng quanh khu nhà để ngắm nhìn các tác phẩm hội họa và ai cũng tiếc là không được chiêm ngưỡng toàn bộ tác phẩm trên tường.

Hãy xác định tổng độ dài phần tranh bị khuất.

Dữ liệu: Vào từ file văn bản ART.INP:

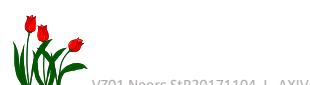
- ✚ Dòng đầu tiên chứa số nguyên n ($4 \leq n \leq 1000$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên x_i và y_i ($-10^6 \leq x_i, y_i \leq 10^6$). Các đỉnh được liệt kê theo một chiều nào đó.

Kết quả: Đưa ra file văn bản ART.OUT một số nguyên – tổng độ dài phần tranh bị khuất.

Ví dụ:

ART.INP
10
1 1
6 1
6 4
3 4
3 3
5 3
5 2
2 2
2 3
1 3

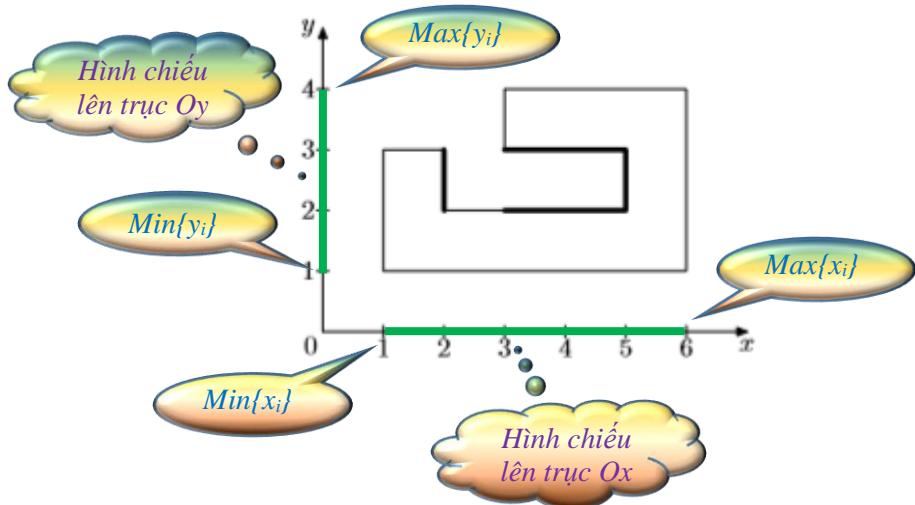
ART.OUT
6



Giải thuật; Kỹ thuật lập trình .

Tổng độ dài các bức tường khi nhìn từ trên xuống hay dưới lên bằng độ dài hình chiếu của khu nhà lên trực Ox,

Tổng độ dài các bức tường khi nhìn từ trái sang hay phải vào bằng độ dài hình chiếu của khu nhà lên trực Oy.



Độ dài hình chiếu lên trực Ox bằng $\max_{i=1 \dots n} x_i - \min_{i=1 \dots n} x_i$,

Độ dài hình chiếu lên trực Oy bằng $\max_{i=1 \dots n} y_i - \min_{i=1 \dots n} y_i$.

Phần không nhìn thấy bằng chu vi khu nhà trừ đi phần nhìn thấy.

Tính chu vi khu nhà: tổ chức dữ liệu vòng tròn, thêm $x_{n+1} = x_1$, $y_{n+1} = y_1$.

Tổ chức dữ liệu:

- Mảng `vector<int>` `x(n)` – lưu các tọa độ x_i ,
- Mảng `vector<int>` `y(n)` – lưu các tọa độ y_i .

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

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

int main()
{
    int n;
    fi>>n;
    vector<int> x(n);
    vector<int> y(n);
    for (int i = 0; i < n; i++) fi>>x[i]>>y[i];
    x.push_back(x[0]);
    y.push_back(y[0]);

    int64_t sum = 0;
    for (int i = 0; i < n; i++)
    {
        sum += abs(x[i + 1] - x[i]);
        sum += abs(y[i + 1] - y[i]);
    }
    sum -= 2 * (*max_element(x.begin(), x.end()) -
                *min_element(x.begin(), x.end()));
    sum -= 2 * (*max_element(y.begin(), y.end()) -
                *min_element(y.begin(), y.end()));
    fo<<sum;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Trong thiên văn phần lớn các số liệu quan sát được xử lý gần đúng. Với toán học hay vật lý giá trị 10^{-6} có thể vẫn còn rất lớn, nhưng với thiên văn 10^6 vẫn còn được coi là nhỏ hoặc thậm chí – rất nhỏ!

Các anten vô tuyến của trạm quan sát bầu trời ở sa mạc Chi lê phát hiện một nguồn sóng vô tuyến bí ẩn trong vũ trụ. Khoảng cách tính bằng km tới nguồn phát sóng được xác định bởi hai bộ tham số $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$, $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_m)$ và được tính bằng công thức

$$d = \frac{a_1 \times a_2 \times \dots \times a_n}{b_1 \times b_2 \times \dots \times b_m}$$

Người ta biết chắc chắn là d không vượt quá 10^{18} và muốn xác định nó dưới dạng số nguyên không âm có chênh lệch với giá trị đúng của biểu thức không quá 10^6 .

Hãy tính và đưa ra một giá trị thỏa mãn yêu cầu đã nêu.

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

- + Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n, m \leq 10^5$),
- + Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^9$, $i = 1 \div n$),
- + Dòng thứ 3 chứa m số nguyên b_1, b_2, \dots, b_m ($1 \leq b_j \leq 10^9$, $j = 1 \div m$).

Kết quả: Đưa ra file văn bản APPROXIMATION.OUT một số nguyên không âm – khoảng cách tính được thỏa mãn yêu cầu đã nêu.

Ví dụ:

APPROXIMATION.INP	APPROXIMATION.OUT
<pre>3 2 5 8 13 3 4</pre>	<pre>43</pre>



Giải thuật; Cơ sở lập trình .

Vì việc phân tích ra thừa số nguyên tố, giản ước và tính biểu thức nhận được sẽ cho kết quả chính xác của d nhưng đòi hỏi nhiều thời gian cũng như công sức lập trình.

Có thể bỏ qua khâu phân tích ra thừa số nguyên tố nhưng phải lưu ý chọn trình tự các phép tính cần thực hiện để không phải làm việc với số lớn và chống tích lũy sai số làm tròn.

Kết quả không vượt quá 10^{18} và sai số cho phép là 10^6 nên có thể tính d dựa vào hàm $\ln(d)$.

$$\ln(d) = \sum_{i=1}^n \ln(a_i) - \sum_{j=1}^m \ln(b_j)$$
$$d = e^{\ln(d)}$$

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("approximation.inp");
ofstream fo ("approximation.out");
int main()
{
    int n, m;
    fi >> n >> m;
    long double sum = 0;
    for (int i = 0; i < n; i++)
    {
        long double x;
        fi >> x;
        sum += log(x);
    }
    for (int i = 0; i < m; i++)
    {
        long double x;
        fi >> x;
        sum -= log(x);
    }
    fo<< (int64_t) (exp(sum)) << '\n';
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



B12. BỨC TRANH

Tên chương trình: PICTURE.CPP

Huấn luyện viên bóng đá thường không ở lâu một nơi vì hợp đồng có thể không được gia hạn hay thậm chí – bị sa thải sớm, vì vậy họ thường không mua nhà mà chỉ thuê để ở.

Tường căn phòng mà huấn luyện viên đội bóng thành phố thuê được lát bằng gạch men nghệ thuật (gạch có in hình), viên gạch có hình vuông cạnh m cm, trong rất đẹp nhưng khá đơn điệu. Ông mua một bức tranh hình chữ nhật về treo lên tường để phủ được các nhiều viên gạch càng tốt. Các cạnh của bức tranh phải song song với các cạnh của viên gạch. Bức tranh có kích thước $w \times h$. Một viên gạch gọi là được phủ nếu nó có diện tích chung khác 0 với bức tranh.



Hãy xác định với cách treo hợp lý số viên gạch nhiều nhất được phủ là bao nhiêu.

Dữ liệu: Vào từ file văn bản PICTURE.INP gồm một dòng chứa 3 số nguyên m , w và h ($1 \leq m, w, h \leq 2 \times 10^9$).

Kết quả: Đưa ra file văn bản PICTURE.OUT một số nguyên – số viên gạch nhiều nhất được phủ.

Ví dụ:

PICTURE.INP
10 30 20

PICTURE.OUT
12



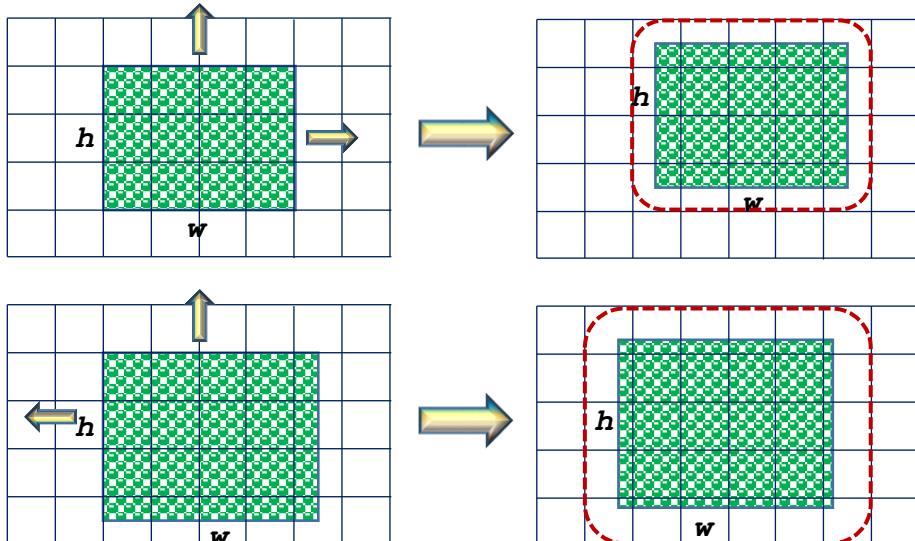
B12.Mos7_820181216 A

A XIV

Giải thuật; Cơ sở lập trình, Khả năng phân tích giải thuật.

Khi treo bức tranh để đinh trên trái của nó trùng với một đỉnh của viên gạch, nếu w chia hết cho m thì ở mỗi hàng, số viên gạch bị phủ là w/m . **Tịnh tiến bức tranh sang phải một chút** để đinh trên trái của nó không trùng với đỉnh của viên gạch, ở mỗi hàng số viên gạch bị phủ sẽ là $w/m+1$.

Trường hợp w không chia hết cho m , khi treo bức tranh để đinh trên trái trùng với đỉnh một viên gạch thì ở mỗi hàng, số viên gạch bị phủ là $w/m+1$. **Tịnh tiến bức tranh sang trái một chút** để đinh trên trái của nó không trùng với đỉnh của viên gạch, ở mỗi hàng số viên gạch bị phủ sẽ là $w/m+2 = w/m+1+w\%m$.



Lập luận tương tự như vậy với h để tính số lượng hàng bị phủ, ta có số lượng hàng bị phủ là $h/m+1+h\%m$.

Kết quả cần tìm là tích 2 đại lượng đã tính được nói trên.

Chú ý: Khai báo kiểu dữ liệu thích hợp.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("picture.inp");
ofstream fo ("picture.out");
int64_t m,w,h,ans, t1,t2;

int main()
{
    fi>>m>>w>>h;
    t1=w/m+1+(w%m>0);
    t2=h/m+1+(h%m>0);
    ans=t1*t2;
    fo<<ans;
}
```



B13. THANG MÁY

Tên chương trình: ELEVATOR.CPP

Tòa nhà của trung tâm thương mại có n tầng, đánh số từ 1 đến n từ dưới lên trên. Hàng hóa được đưa vào tầng hầm dưới tầng 1 và dùng thang máy chở hàng đưa đến tầng bất kỳ.

Để chuẩn bị đón năm mới Ban Giám đốc cho dựng ở mỗi tầng một cây thông lớn. Thang máy có khả năng chở cùng một lúc cả n cây thông nhưng toàn bộ trung tâm chỉ có một máy có thể nhanh chóng tải cây ra khỏi thang máy, vì vậy người ta quyết định thiết bị này lên tầng thứ k , chở hết thông lên đó và đưa mọi cây ra lối đi. Từ đó nhân viên các tầng sẽ dùng tay chuyển đến tầng của mình. Thời gian để bê cây lên một tầng là a , bê xuống một tầng là b . Thời gian thang máy đi hết một tầng là c . Bê xuống dễ hơn bê lên nên ta có $a > b$, ngoài ra còn có $a > c$. Thời gian đưa cây ra khỏi thang máy là không đáng kể.

Hãy xác định k – nơi đặt thiết bị kéo cây ra khỏi thang máy để khoảng thời gian tính từ khi thang máy bắt đầu chạy từ tầng hầm đến lúc mọi cây thông được đưa đến hết các tầng là nhỏ nhất.

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

- ✚ Dòng đầu tiên chứa số nguyên dương n ,
- ✚ Dòng thứ 2 chứa số nguyên dương a ,
- ✚ Dòng thứ 3 chứa số nguyên dương b ,
- ✚ Dòng thứ 4 chứa số nguyên dương c .

Các số đã cho không vượt quá 2×10^9 .

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

Ví dụ:

ELEVATOR.INP
6
20
10
5

ELEVATOR.OUT
4

I



Giải thuật; Cơ sở lập trình, Khả năng phân tích giải thuật.

Khi mang được cây thông lên tầng **n** thì ở các tầng từ **k+1** đến **n-1** cũng đã có cây.

Khi mang được cây thông xuống tầng 1 thì ở các tầng từ 2 đến **k-1** cũng đã có cây.

Như vậy phải chọn **k** sao cho chênh lệch thời gian đưa cây từ tầng **k** lên tầng **n** và thời gian đưa cây từ tầng **k** xuống tầng 1 là nhỏ nhất.

Trường hợp lý tưởng là chênh lệch bằng 0:

$$(k-1) \times b = (n-1) \times a$$

$$k = \frac{n \times a + b}{a + b}$$

Nhưng **k** cần có giá trị nguyên, vì vậy ta cần tính thời gian với các trường hợp **k=(n×a+b) / (a+b)** và **k=(n×a+b) / (a+b)+1**, so sánh để chọn kết quả tối ưu.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("elevator.inp");
ofstream fo ("elevator.out");
int64_t n,a,b,c,k,t1,t2;

int main()
{
    fi>>n>>a>>b>>c;
    k=(n*a+b) / (a+b);
    t1=max((k-1)*b, (n-k)*a)+k*c;
    t2=max(k*b, (n-k-1)*a)+(k+1)*c;
    if(t1>t2) ++k;
    fo<<k;
}
```



B14. GIẢI ĐẤU CỜ NHANH

Tên chương trình: CHESS.CPP

Giải đấu cờ nhanh (hạn chế thời gian cho mỗi ván) có n đấu thủ ($n = 2^k$), đánh số từ 1 đến n , tham gia. Giải được tổ chức theo nguyên tắc đấu loại trực tiếp. Vòng 1 là các cặp đấu (1, 2), (3, 4), ..., ($n-1$, n). Ở vòng 2, người thắng ở cặp đấu thứ nhất sẽ đấu với người thắng ở cặp đấu thứ hai, người thắng ở cặp đấu thứ ba sẽ đấu với người thắng ở cặp đấu thứ tư, ... Cứ như vậy cho đến khi chọn được nhà vô địch – người chiến thắng cuối cùng. Các quy tắc phụ đảm bảo không có ván hòa.

Trong biên bản kết quả mỗi cặp đấu được nhận bằng một số nguyên: 1 là đấu thủ thứ nhất trong cặp thắng, 2 – đấu thủ thứ 2 thắng.

Cho biên bản tất cả các ván đấu ở các vòng theo trình tự liệt kê nêu trên.

Hãy xác định nhà vô địch.

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

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 2^{16}$),
- ✚ Các dòng sau cho biết biên bản các trận đấu theo trình tự đã ghi, mỗi dòng chứa biên bản một trận đấu.

Kết quả: Đưa ra file văn bản CHESS.OUT một số nguyên – số của nhà vô địch.

Ví dụ:

CHESS.INP
8
1
2
2
1
2
1
1

CHESS.OUT
4

I



B14.Mos7_8 20181216 7 A XIV

Giải thuật; Cơ sở lập trình, Khả năng phân tích giải thuật.

Tạo mảng ghi số thứ tự các cờ thủ,

Sau mỗi vòng: thu hẹp mảng một nửa, thay mỗi cặp số bằng số của người thắng.

Quá trình thu hẹp kết thúc khi chỉ còn một người.

Kết quả: Ở phần tử đầu tiên của mảng.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("chess.inp");
ofstream fo ("chess.out");
int64_t n,r;

int main()
{
    fi>>n;
    vector<int> ans(n);
    for(int i=0; i<n; ++i) ans[i]=i;
    while((n=n/2)>0)
        for(int i=0; i<n; ++i)
        {
            fi>>r;
            if(r==1) ans[i]=ans[2*i]; else ans[i]=ans[2*i+1];
        }
    fo<<ans[0]+1;
}
```



Giáo sư Braun có thói quen xuống xe bus trước một vài bến và đi bộ về nhà. Đó là một hình thức thể dục có lợi cho sức khỏe. Ở đường phố mà ông đi một phía các nhà được đánh số chẵn liên tiếp nhau 2, 4, 6, ... còn phía kia – các nhà được đánh số lẻ liên tiếp nhau 1, 3, 5, ... Nhà số 1 đối diện với nhà số 2, nhà số 3 đối diện với nhà số 4, ...

Nhà của giáo sư có số là **b**. Hôm nay ông xuống xe cùng với người bạn ở nhà số **a**. Xe dừng trước nhà có số là **a** (phía bên này hay bên kia phố). Ông đi bộ về nhà mà không phải qua đường (còn người bạn thì có thể phải qua đường ☺). Cứ một phút thì ông qua được một nhà, như vậy sau phút thứ nhất ông tới trước nhà có số **a+2** hoặc **a-2** phụ thuộc vào hướng cần đi về nhà.

Là một nhà khoa học, giáo sư không có thói quen để để trống thời gian, ông vừa đi vừa cộng các số nhà mà ông đi qua kể cả **a** và **b**. Đây cũng là một hình thức thể thao cho trí não.

Hãy xác định sau bao nhiêu phút thì ông đến trước cửa nhà mình và số cộng được trong đầu là bao nhiêu.

Dữ liệu: Vào từ file văn bản STREET.INP gồm một dòng chứa 2 số nguyên **a** và **b** ($1 \leq a, b \leq 2 \times 10^9$).

Kết quả: Đưa ra file văn bản STREET.OUT trên một dòng 2 số nguyên xác định thời gian đi và số cộng được.

Ví dụ:

STREET.INP	STREET.OUT
1 8	3 20



Giải thuật; Cơ sở lập trình, Cáp số cộng .

Chuẩn hóa dữ liệu vào:

- ❖ Dẫn xuất **a** về số nhà tương ứng cùng phía với nhà số **b**,
- ❖ Đưa về trường hợp **a ≤ b**.

Thời gian đi: **ans1 = b-a,**

Tổng các số nhà: **ans2 = a + (a+2) + (a+4) + . . . + b = $\frac{(a+b) \times (ans1+1)}{2}$**

Chú ý:

- Khai báo kiểu dữ liệu thích hợp,
- Trình tự thực hiện phép tính khi tính **ans2**.

Dộ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("street.inp");
ofstream fo ("street.out");
uint64_t a,b,ans1,ans2;

int main()
{
    fi>>a>>b;
    if((a&1) != (b&1))
        if(a&1) ++a; else --a;
    if(a>b) swap(a,b);
    ans1=(b-a)/2;
    ans2=(a+b)/2*(ans1+1);
    fo<<ans1<<'\n'<<ans2;
}
```



B16. BIỂN QUẢNG CÁO

Tên chương trình: SHIELD.CPP

Biển quảng cáo được gắn vào cột với 3 chốt bảo vệ. Chốt thứ nhất giữ được tầm biển khỏi đỗ khi tốc độ gió không vượt quá **a** m/gy, chốt thứ hai giữ được tầm biển khỏi đỗ khi tốc độ gió không vượt quá **b** m/gy và chốt thứ ba giữ được tầm biển khỏi đỗ khi tốc độ gió không vượt quá **c** m/gy.

Biển quảng cáo ở trạng thái an toàn tuyệt đối khi tốc độ gió ở mức mà ít nhất 2 trong số 3 chốt bảo vệ vẫn giữ được.

Hãy xác định tốc độ gió tối đa mà biển quảng cáo vẫn giữ được trạng thái an toàn tuyệt đối.

Dữ liệu: Vào từ file văn bản SHIELD.INP gồm một dòng chứa 3 số nguyên a, b và c ($1 \leq a, b, c \leq 2 \times 10^9$).

Kết quả: Đưa ra file văn bản SHIELD.OUT một số nguyên – tốc độ gió tối đa tính được.

Ví dụ:

SHIELD.INP	SHIELD.OUT
28 15 10	15



B16 Mos9_11 20181216 1 A XIV

Giải thuật; Cơ sở lập trình .

Sắp xếp **a, b, c** theo thứ tự tăng dần hoặc giảm dần,

Kết quả cần tìm: phần tử trung vị.

Lưu ý:

- + Có thể tính $u = \max\{a, b, c\}$, $v = \min\{a, b, c\}$,
- + Kết quả cần tìm: $a+b+c-u-v$,
- + Cần chú ý chọn kiểu dữ liệu thích hợp.

Độ phức tạp của giải thuật: O(1).

Chương trình I

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("shield.inp");
ofstream fo ("shield.out");
int64_t a,b,c,u,v;

int main()
{
    fi>>a>>b>>c;
    u=max(a,b); u=max(u,c);
    v=min(a,b); v=min(v,c);
    fo<<a+b+c-u-v;
}
```

Chương trình II

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

int main()
{
    fi>>a>>b>>c;
    if(a<b) swap(a,b);
    if(a<c) swap(a,c);
    if(b<c) swap(b,c);
    fo<<b;
}
```



B17. ĐỘI NGŨ

Tên chương trình: PARADE.CPP

Lễ khai mạc Đại hội Thể thao khu vực bắt đầu bằng màn biểu diễn thể dục nhịp điệu của n người. Ban Tổ chức muốn chia họ đứng thành một hoặc các khối vuông giống nhau và yêu cầu các khối vuông này phải có kích thước lớn nhất có thể.

Hãy xác định số người nhiều nhất của khối vuông.

Dữ liệu: Vào từ file văn bản PARADE.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 2 \times 10^9$).

Kết quả: Đưa ra file văn bản PARADE.OUT một số nguyên – số người nhiều nhất trong một khối vuông.

Ví dụ:

SHIELD.INP
180

SHIELD.OUT
36



B17 Mos9_11 20181216 3

A XIV

Giải thuật; Cơ sở lập trình, Duyệt vét cạn.

Kiểm tra n có chia hết cho k^2 hay không với $k = \sqrt{n} \div 2$.

Gặp giá trị k thỏa mãn – ghi nhận và thoát khỏi chu trình kiểm tra.

Nếu không tìm được k – kết quả cần tìm là 1.

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

Chương trình I

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

int main()
{
    fi>>n;
    k=(int)sqrt(n)+1;
    while(k>1)
    {
        if(n%(k*k)==0) {ans=k*k; break;}
        --k;
    }

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

Chương trình II

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

int main()
{
    fi>>n;
    k=2;
    while(k*k <= n)
    {
        if(n%(k*k)==0) ans=k*k;
        ++k;
    }

    fo<<ans;
    cerr<<"Time: "<<clock() / (double)1000<<" sec";
}
```



Phần lớn các bài lý thuyết số học đều bắt đầu với các ví dụ liên quan tới dãy số tự nhiên 1, 2, 3, ..., n . Trong Tin học cũng vậy, một số lượng lớn các bài tập về cơ sở lập trình và nhập môn giải thuật có dữ liệu vào là dãy số tự nhiên, còn yêu cầu đưa ra thì hết sức đa dạng.

Alice không ngạc nhiên khi bài tập tin học về nhà hôm nay lại dính dáng đến dãy số tự nhiên. Chỉ có điều bài toán phát biểu quá đơn giản, mà thông thường phát biểu càng đơn giản bao nhiêu thì càng khó giải bấy nhiêu!

Nội dung của bài toán là “Hãy đặt các dấu **+** (cộng) hoặc **-** (trừ) trước các số từ 1 đến n để nhận được một biểu thức có kết quả bằng 0 và đưa ra dãy dấu cần đặt. Nếu không có cách nhận được biểu thức theo yêu cầu thì đưa ra thông báo **IMPOSSIBLE**”.

Mặc dù không có mấy hứng thú với bài tập loại sơ cấp này, nhưng rồi Alice cũng phải mở máy tính ra lập trình và cuối cùng cũng làm xong!

Hãy cho biết nội dung file kết quả trong máy của Alice.

Dữ liệu: Vào từ file văn bản NUMERIC_SQ.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 10^5$).

Kết quả: Đưa ra file văn bản NUMERIC_SQ.OUT xâu các ký tự từ tập {**+**, **-**} xác định một cách đặt dấu trước các số hoặc thông báo **IMPOSSIBLE** nếu không có cách đặt dấu thỏa mãn yêu cầu đã nêu.

Ví dụ:

NUMERIC_SQ.INP	NUMERIC_SQ.OUT
3	++-



Giải thuật; Cơ sở lập trình, Kỹ năng đoán nhận giải thuật .

Dễ dàng thấy rằng với $n = 1$ hoặc $n = 2$ không tồn tại biểu thức cho kết quả 0.

Với $n = 3$:

Có 2 cách tạo biểu thức

$$\begin{array}{ccc} 1+2-3 & & -1-2+3 \\ \downarrow & & \downarrow \\ +-+ & & ---+ \end{array}$$

Với $n = 4$:

Có 2 cách tạo biểu thức

$$\begin{array}{ccc} 1-2-3+4 & & -1+2+3-4 \\ \downarrow & & \downarrow \\ +--+ & & -++- \end{array}$$

Các cách đặt dấu này có thể áp dụng với **4 số nguyên liên tiếp nhau** để nhận được biểu thức cho kết quả 0.

Xét trường hợp $n > 4$:

Gọi $k = n \% 4$,

$$k = \begin{cases} 0 & \rightarrow \text{Đưa ra } “+--+” \frac{n}{4} \text{ lần,} \\ 1 & \rightarrow \text{Đưa ra } “\text{IMPOSSIBLE}” , \\ 2 & \rightarrow \text{Đưa ra } “\text{IMPOSSIBLE}” , \\ 3 & \rightarrow \text{Đưa ra } “+-+” \text{ và } “+--+” \frac{(n-3)}{4} \text{ lần,} \end{cases}$$

Độ phức tạp của giải thuật: O(n).

Chương trình

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

int main()
{
    fi>>n;
    k=n%4;
    switch(k)
    {
        case 1: case 2:{fo<<"IMPOSSIBLE"; break;}
        case 3: {fo<<"++-"; for(int i=0; i<(n-3)/4; ++i)
                    fo<<"---"; break;}
        case 0: {for(int i=0; i<n/4; ++i) fo<<"---"; break;}
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Các nhà địa chất có nhiệm vụ khảo sát tìm khoáng sản ở một vùng đài nguyên mênh mông. Thiết bị và nhu yếu phẩm được chở tới trại Trung tâm tại điểm có tọa độ (0, 0) trên bản đồ. Nhóm khảo sát đi thăm dò từng điểm có tọa độ nguyên trên bản đồ. Nhóm di chuyển từ điểm có tọa độ nguyên này tới điểm có tọa độ nguyên khác và luôn đi theo một trong các hướng “Đông” (**E**), “Tây” (**W**), “Nam” (**S**) hoặc “Bắc” (**N**). Việc đổi hướng có thể được thực hiện tại các điểm có tọa độ nguyên. Khoảng cách bằng một đơn vị tọa độ được gọi là bước. Đường đi của họ được thiết bị tự động mang theo báo về dưới dạng xâu ghi nhận từng nhóm số bước di chuyển và hướng. Ví dụ, dòng thông báo “**7N5E2S3E**” cho biết nhóm đã đi 7 bước theo hướng nam, sau đó đi 5 bước về hướng đông, đi tiếp theo hướng bắc 2 bước rồi đi theo hướng đông 3 bước.

Có thể tại một nơi nào đó nhóm tìm thấy những mẫu vật đặc biệt, cần phân tích ngay để có hành động phù hợp tiếp theo và yêu cầu Trung tâm gửi thiết bị bay tới lấy về. Các thiết bị bay mang theo chỉ được lập trình cho đường bay theo các hướng E, W, S và N. Khi nạp thông báo về hành trình của nhóm khảo sát vào bộ nhớ, các microchips trong máy bay sẽ tính toán, tìm đường đi có xâu (biểu diễn dưới dạng của thiết bị ghi nhận đường đi đã nói ở trên) ngắn nhất.

Có nhiều cách ngắn nhất để đi tới đích. Hãy xác định một trong số các xâu biểu diễn đường đi do thiết bị bay tạo ra. Đảm bảo đích cần tới không trùng với gốc tọa độ.

Dữ liệu: Vào từ file văn bản ROUTE.INP gồm một dòng chứa xâu độ dài không quá 250 xác định hành trình của nhóm khảo sát. Các hệ số có giá trị nguyên dương và không vượt quá 10^7 .

Kết quả: Đưa ra file văn bản ROUTE.OUT xâu do thiết bị bay tạo ra.

Ví dụ:

ROUTE.INP	ROUTE.OUT
7N5E2S3E	8E5N



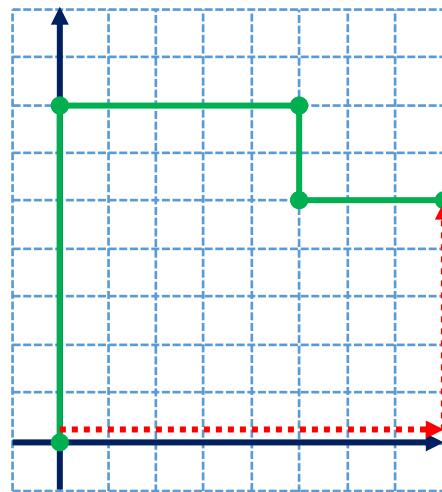
Giải thuật: Xử lý xâu.

Từ hành trình của nhóm khảo sát có thể xác định tọa độ (x_end, y_end) hành trình của nhóm,

Để từ $(0, 0)$ tới (x_end, y_end) chỉ cần đổi hướng không quá một lần và đường đi đó sẽ có dạng biểu diễn ngắn nhất.

Gọi \mathbf{lf} là khoảng cách của nhóm tới trục Oy, \mathbf{rt} là khoảng cách của nhóm tới trục Ox, duyệt xâu hành trình, tách ra từng nhóm *Bước di chuyển* (\mathbf{st}) và *Hướng* (\mathbf{c}).

$$\mathbf{c} = \begin{cases} \mathbf{E} \rightarrow \mathbf{lf}+=\mathbf{st} \\ \mathbf{W} \rightarrow \mathbf{lf}-=\mathbf{st} \\ \mathbf{N} \rightarrow \mathbf{rt}+=\mathbf{st} \\ \mathbf{S} \rightarrow \mathbf{rt}-=\mathbf{st} \end{cases}$$



Khi duyệt hết xâu hành trình, ta có $x_end = \mathbf{lf}$, $y_end = \mathbf{rt}$.

Kết quả đưa ra có dạng \mathbf{AuBv} , trong đó:

$\mathbf{A} = |\mathbf{lf}|$, $\mathbf{B} = |\mathbf{rt}|$, $\mathbf{u} = 'E'$ nếu $\mathbf{lf} > 0$ hoặc $\mathbf{u} = 'W'$ nếu $\mathbf{lf} < 0$, $\mathbf{v} = 'N'$ nếu $\mathbf{rt} > 0$ hoặc $\mathbf{v} = 'S'$ nếu $\mathbf{rt} < 0$. Nếu có \mathbf{A} hoặc \mathbf{B} bằng 0 thì không đưa ra nhóm ký tự tương ứng.

Dộ phức tạp của giải thuật: $O(n)$, trong đó n là độ dài của \mathbf{s} .

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("route.inp");
ofstream fo ("route.out");
int n, lf=0, rt=0, k=0, st;
string s;
char c1='E', c2='N';

int get_num(int &x)
{
    int t, tr=0;
    while( (t=s[x]-48)<10)
    {
        tr=tr*10+t; ++x;
    }
    return tr;
}

int main()
{
    fi>>s; n=s.size();
    while(k<n)
    {
        st=get_num(k);
        switch(s[k++])
        {
            case 'E': {lf+=st; break; }
            case 'W': {lf-=st; break; }
            case 'N': {rt+=st; break; }
            case 'S': {rt-=st; break; }
        }
    }
    if(lf<0) c1='W', lf=-lf;
    if(rt<0) c2='S', rt=-rt;
    if(lf) fo<<lf<<c1;
    if(rt) fo<<rt<<c2;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



B20. SỐ CÁCH THAY GIÁ TRỊ

Tên chương trình: CHANGE.CPP

Xét biểu thức lô gic đúng chỉ chứa phép tính **OR** (phép **|** trong C/C++) và phép **NOT** (phép **~** trong C/C++). Tên các biến là một chữ cái la tinh hoa hoặc thường và phân biệt chữ hoa với chữ thường. Là một biểu thức lô gic đúng nên dấu phép tính chỉ xuất hiện trước tên biến. Các biến chỉ nhận giá trị lô gic.

Thay mỗi biến bằng bằng giá trị **true** hoặc **false** người ta có thể tính được giá trị toàn biểu thức.

Hãy xác định có bao nhiêu cách thay giá trị để biểu thức cho kết quả là **true**. Hai cách thay gọi là khác nhau nếu tồn tại ít nhất một biến ở cách thứ nhất nhận giá trị **true** và ở cách thứ hai – nhận giá trị **false**.

Dữ liệu: Vào từ file văn bản CHANGE.INP gồm một dòng chứa xâu **s** độ dài không quá 1000 xác định một biểu thức lô gic đúng chỉ chứa tên biến, dấu 2 phép tính đã nêu (theo cách viết của C/C++) và không chứa ký tự nào khác kể cả dấu cách.

Kết quả: Đưa ra file văn bản CHANGE.OUT một số nguyên – số cách thay giá trị để biểu thức cho kết quả là true.

Ví dụ:

CHANGE.INP	CHANGE.OUT
i c p c	7



B20 Neerc SPt 20171104 B A XIV

Giải thuật; Xử lý xâu .

Tạo các mảng đánh dấu:

- Mảng **a** – Sự xuất hiện của biến,
- Mảng **b** – Biến tham gia vào biểu thức với giá trị trực tiếp của mình,
- Mảng **c** – Biến tham gia vào biểu thức với giá trị phủ định.

Sử dụng biến **k** làm cờ đánh dấu việc gấp phép phủ định.

Xử lý ký tự của xâu **s**:

$$s_i = \begin{cases} | \rightarrow \text{bỏ qua}, \\ \sim \rightarrow k=1, \\ \text{Các trường hợp còn lại: } \end{cases}$$

$a_{s_i} = 1$
 $\begin{matrix} k=0 & \xrightarrow{\quad} & b_{s_i} = 1 \\ k=1 & \xrightarrow{\quad} & c_{s_i} = 1 \end{matrix}$
 $k=0$

Từ mảng **a** dễ dàng tính **m** – số lượng các biến khác nhau trong biểu thức.

Từ các mảng **b** và **c** có thể nhận biết có tồn tại biến tham gia vào biểu thức với giá trị phủ định của mình hay không.

Phép **OR** trong biểu thức lô gic có tính giao hoán, vì vậy nếu biểu thức chứa biến **x** cùng với $\sim x$ thì $x | \sim x$ luôn cho giá trị **true** và từ đó – biểu thức luôn nhận giá trị **true** với mọi khả năng khác nhau của tất cả các biến.

Nếu biểu thức không chứa đồng thời **x** và $\sim x$ với mọi biến thì nó chỉ nhận giá trị **false** khi và chỉ khi mọi biến đều nhận giá trị **false**.

Từ đây suy ra kết quả cần tìm sẽ là 2^m nếu tồn tại $x | \sim x$ và bằng $2^m - 1$ trong trường hợp ngược lại.

Tổ chức dữ liệu:

Các mảng **int a[255]={0}, b[255]={0}, c[255]={0}** với chức năng đã nêu ở trên.

Lưu ý: Kiểu dữ liệu kết quả và cách tính 2^m .

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

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("change.inp");
ofstream fo ("change.out");
int n,k=0,m=0,a[255]={0},b[255]={0},c[255]={0};
string s;
int64_t ans;
char cs;

int main()
{
    fi>>s; n=s.size();
    for(int i=0; i<n; ++i)
    {
        if(s[i]=='~') {k=1; continue;}
        if(s[i]=='|') continue;
        a[s[i]]=1; if(k) c[s[i]]=1; else b[s[i]]=1; k=0;
    }
    for(int i=65; i<='z'; ++i) k+=a[i];
    for(int i=65; i<='z'; ++i) m+=b[i]*c[i];
    ans=((int64_t)1<<k)-(m==0);
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



Alice vừa hoàn thành một hợp đồng lắp bảng hiển thị số bằng đèn LED. Mỗi chữ số được hiển thị trong một khung chữ nhật với 7 ống đèn LED. Bằng cách bật các ống đèn LED thích hợp ta có thể hiển thị chữ số bất kỳ. Số đèn sáng càng nhiều thì việc hiển thị chữ số đó càng tốn năng lượng. Ví dụ, hiển thị chữ số 9 sẽ tốn năng lượng hơn hiển thị chữ số 7.



Sau khi bàn giao sản phẩm trong tay Alice còn thừa lại một số khá nhiều các khung hiển thị số và một cục pin nguồn. Dung lượng pin cho phép bật sáng n ống đèn LED. Alice muốn dùng pin bật đúng n ống đèn để hiển thị một số và số hiển thị được phải có tổng chữ số là lớn nhất. Hãy xác định tổng lớn nhất của các chữ số của số có thể bật sáng.

Dữ liệu: Vào từ file văn bản LED.INP gồm một dòng chứa số nguyên n ($2 \leq n \leq 10^6$).

Kết quả: Đưa ra file văn bản LED.OUT một số nguyên – tổng lớn nhất đạt được.

Ví dụ:

LED.INP	LED.OUT
7	11



VZ01 Neerc StP20171104 A AXIV

Giải thuật: Quy hoạch động, Bảng phương án.

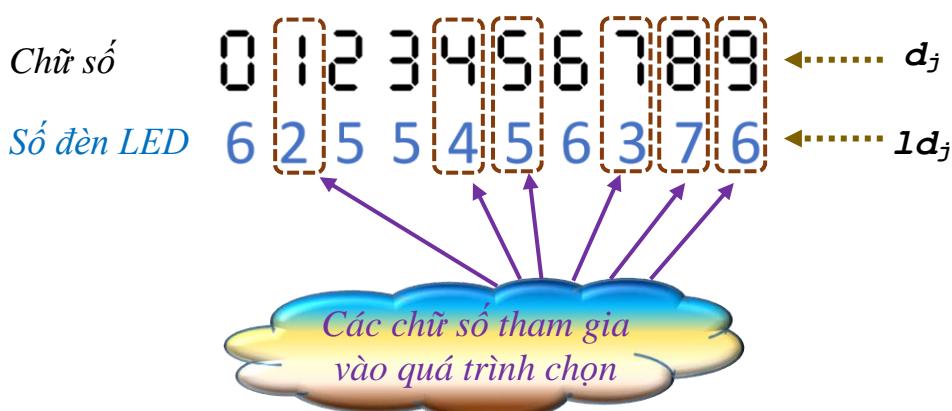
Giải thuật I

Dễ dàng thấy rằng có thể áp dụng sơ đồ quy hoạch động để tìm lời giải.

Gọi s_i – tổng lớn nhất của các chữ số khi dùng đúng i đèn LED.

Ta có $s_i = \max_{k=0 \div 9} (k + s_{i-led_k})$, trong đó led_k là số đèn để hiển thị chữ số k .

Kết quả cần tìm: s_n .



Nhận xét: Có một số chữ số cần dùng số lượng đèn như nhau để hiển thị, trong số đó rõ ràng chỉ có chữ số lớn nhất có khả năng thay đổi giá trị s_i , vì vậy ta không cần duyệt với $k = 0 \div 9$ mà chỉ cần làm việc với $k \in \{1, 4, 5, 7, 8, 9\}$ và led_k tương ứng.

Chuẩn bị giá trị đầu cho giải thuật quy hoạch động là rất quan trọng.

Trong công thức lặp đòi hỏi tính chỉ số $i - \text{led}_k$, $\max\{\text{led}_k\} = 7$. Vì vậy cần chuẩn bị s_i với $i = 0 \div 7$.

Có thể lập một *đoạn chương trình riêng để tính các giá trị đầu*, nhưng sẽ tốt hơn nếu tổ chức gán trực tiếp giá trị tính được cho các biến tương ứng ở phiên bản chương trình cuối cùng. Các giá trị đầu cũng có thể *tính tay trực tiếp*.

Các giá trị đầu sẽ là:

$s_0 = s_1 = 0$, $s_2 = 1$, $s_3 = 7$, $s_4 = 4$, $s_5 = 8 (=1+7)$, $s_6 = 14 (=7+7)$, $s_7 = 11 (=4+7)$.

Việc tính s_i được thực hiện với $i = 8$ cho tới n .

Độ phức tạp của giải thuật: $O(n)$.

Sơ đồ quy hoạch động với mô đun khởi tạo giá trị đầu

Bước quay lui để tìm giá trị lớn nhất là 6 vì vậy phải chuẩn bị các giá trị s_i với $i = 0 \div 6$.

Chương trình I

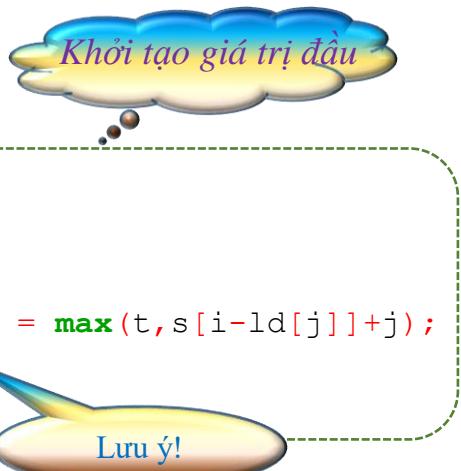
```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("Led.inp");
ofstream fo ("Led.out");
int ld[10]={6,2,5,5,4,5,6,3,7,6};

int main()
{
    int n,t,k;
    fi>>n;
    vector<int>s(n+10);

    for(int i=0; i<=6; ++i)
    {
        t=0;
        for(int j=0; j<=9; ++j)
            if(i>=ld[j] && i>=j) t = max(t,s[i-ld[j]]+j);
        s[i]=t;
    }

    for(int i=7; i<=n; ++i)
    {
        t=0;
        for(int j=0; j<=9; ++j) t = max(t,s[i-ld[j]]+j);
        s[i]=t;
    }

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



Sơ đồ quy hoạch động với các giá trị đầu của s_i được gán trực tiếp

Các giá trị này có thể được xác định bằng chạy chương trình chỉ chứa mô đun khởi tạo giá trị đầu, số lượng phép kiểm tra để tính mỗi giá trị s_i mới giảm đáng kể (*và còn có thể cải tiến để giảm thêm!*).

Cần tóm tắt chức mảng lưu trữ các chữ số tham gia xử lý.

Chương trình II

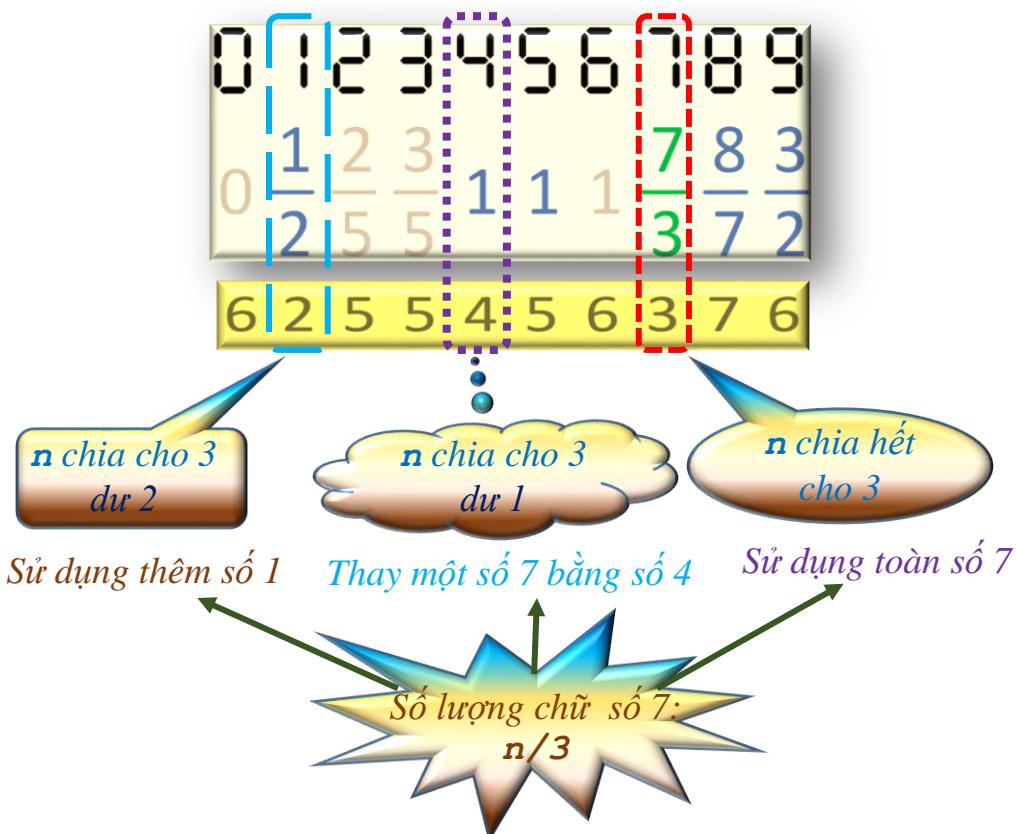
```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("Led.inp");
ofstream fo ("Led.out");
int d[6]={1,7,4,5,9,8},
     ld[6]={2,3,4,5,6,7};

int main()
{
    int n,t;
    fi>>n;
    vector<int>s(n+10);
    s[0]=s[1]=0; s[2]=1; s[3]=7; s[4]=4; s[5]=8; s[6]=14;
    s[7]=11;
    for(int i=8; i<=n; ++i)
    {
        t=0;
        for(int j=0; j<6; ++j) t = max(t,s[i-ld[j]]+d[j]);
        s[i]=t;
    }

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

Giải thuật II Kỹ thuật bảng phương án

Bảng số lượng đèn LED ứng với các chữ số cho thấy hiệu quả một đèn ở chữ số 7 là cao nhất.



Độ phức tạp của giải thuật: $O(1)$.

Chương trình II

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

int main()
{
    int n;
    fi>>n;
    if (n % 3 == 2) fo<< n / 3 * 7 + 1;
    else if (n % 3 == 1) fo<< n / 3 * 7 - 3;
    else fo<<n / 3 * 7;
    return 0;
}
```



Có hai loại âm thanh trong ngôn ngữ tự nhiên: nguyên âm và phụ âm. Nguyên âm là một âm thanh, được tạo ra với một giọng nói mỏ; và phụ âm được phát âm theo cách mà hơi thở bị tắc nghẽn ít nhất một phần. Ví dụ: chữ **a** và **o** được sử dụng để thể hiện âm nguyên âm, trong khi chữ **b** và **p** là phụ âm (ví dụ: **bad**, **pot**).

Một số chữ cái có thể được sử dụng để thể hiện cả âm nguyên âm và phụ âm: ví dụ: **y** có thể được sử dụng làm nguyên âm (ví dụ: **silly**) hoặc làm phụ âm (ví dụ: **yellow**). Chữ **w**, thường được sử dụng làm phụ âm (ví dụ: **wet**) có thể tạo ra nguyên âm sau một nguyên âm khác (ví dụ: **growth**) trong tiếng Anh và trong một số ngôn ngữ (ví dụ tiếng Wales), nó thậm chí có thể là nguyên âm duy nhất trong một từ.

Trong bài toán này, ta coi **y** và **w** là nguyên âm, vì vậy có bảy nguyên âm trong bảng chữ cái tiếng Anh: **a, e, i, o, u, w** và **y**, tất cả các chữ cái khác là phụ âm.

Gọi đối tính phụ âm của một xâu là số lượng các cặp chữ cái liên tiếp trong xâu mà cả hai đều là phụ âm và và không cùng là chữ hoa hay thường (chữ thường và chữ hoa hoặc ngược lại). Ví dụ: đối tính phụ âm của chuỗi **CoNsOnAnts** là 2, đối tính phụ âm của chuỗi **dEsTrUcTiOn** là 3 và đối tính phụ âm của chuỗi **StRenGtH** là 5.

Cho xâu **s** chỉ chứa các chữ cái la tinh thường. Hãy đổi một số chữ cái thành chữ hoa tương ứng để đảm bảo không có chữ cái nào xuất hiện trong xâu dưới vừa là chữ hoa vừa là chữ thường và đổi tính phụ âm trong xâu nhận được là lớn nhất.

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

Kết quả: Đưa ra file văn bản CONSONANT.OUT một dòng chứa xâu đã biến đổi có đổi tính phụ âm trong xâu nhận được là lớn nhất.

Ví dụ:

CONSONANT.INP
strength

CONSONANT.OUT
StRenGtH



VZ04 Neerc StP20171104 C AXIV

Giải thuật: Tổ chức vét cạn.

Theo quy ước đã nêu trong bảng chữ cái la tinh còn lại 19 phụ âm.

Như vậy số lượng cách khác nhau thay chữ thường bằng chữ hoa là 2^{19} .

Số lượng cách khác nhau là không lớn, vì vậy có thể tổ chức duyệt vét cạn để tìm cách thay thế tối ưu.

Cần tổ chức mảng 2 chiều **f[19][19]**, trong đó **f_{i,j}** là số lượng cặp phụ âm thứ **i** đứng cạnh phụ âm thứ **j**.

Gọi **c_i** là kiểu viết phụ âm thứ **i**.

Ta cần tính $\sum_{i=0}^{18} \sum_{j=0}^{18} [c_i \neq c_j] f_{i,j}$, tìm tổng lớn nhất, ghi nhận trạng thái tương ứng của các phụ âm và biến đổi xâu ban đầu.

Chương trình chạy đủ nhanh, vì vậy *không cần thiết tối ưu hóa* bằng cách *rút gọn danh sách các phụ âm* cần xét.

Tổ chức dữ liệu:

- Mảng **vector<int>** **flg(n)** – đánh dấu các phụ âm trong **s** và ghi nhận số thứ tự của phụ âm,
- Mảng **int f[19][19]={0}** – tính số lượng cặp phụ âm gặp trong **s**,
- Mảng **int u[19]** – đánh dấu các phụ âm cần chuyển thành chữ hoa,
- Xâu **string ct="bcdgfghjklmnpqrstvzxz"** – ghi nhận danh sách phụ âm.

Xử lý:

Đánh dấu và ghi nhận thứ tự củ phụ âm trong **s**:

```

vector<int> flg(n);
for(int i=0; i<n; ++i)
{
    m=ct.find(s[i]);
    if(m!=string::npos) flg[i]=m; else flg[i]=-1;
}


```

*Dấu hiệu
không tìm thấy*

Tính tần số các cặp phụ âm liên tiếp trong xâu:

```

for(int i=1; i<n; ++i)
    if(flg[i-1]>=0 && flg[i]>=0)
        ++f[flg[i-1]][flg[i]],
        f[flg[i]][flg[i-1]]=f[flg[i-1]][flg[i]];


```

*Dảm bảo tính
đối xứng*

Tìm mặt nạ biến đổi cho kết quả tối ưu:

```
m = (1<<19)-1;
for(int i=1; i<m; ++i)
{
    t=0;
    for(int j=0; j<19; ++j)
        for(int k=0; k<19; ++k)
        {
            b1=(i>>j)&1; b2=(i>>k)&1;
            if(b1+b2==1) t+=f[j][k];
        }
    if(ans<t) ans=t, imx=i;
}
```

Dẫn xuất kết quả:

```
for(int i=0; i<19; ++i) u[i]=(imx>>i)&1;
for(int i=0; i<n; ++i) if(u[flg[i]]) s[i]-=32;
```

Độ phức tạp của giải thuật: $\approx O(1)$.

Chương trình

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

string s, ct="bcdfghjklmnpqrstvxz";
int n,m,ans=0,imx,t,b1,b2,f[19][19]={0},u[19];
char c1,c2;

int main()
{
    fi>>s; n=s.size();
    vector<int> flg(n);
    for(int i=0; i<n; ++i)
    {
        m=ct.find(s[i]);
        if(m!=string::npos) flg[i]=m; else flg[i]=-1;
    }
    for(int i=1;i<n; ++i)
        if(flg[i-1]>=0 && flg[i]>=0)
            ++f[flg[i-1]][flg[i]],
            f[flg[i]][flg[i-1]]=f[flg[i-1]][flg[i]];
    m = (1<<19)-1;
    for(int i=1; i<m; ++i)
    {
        t=0;
        for(int j=0; j<19; ++j)
            for(int k=0; k<19; ++k)
            {
                b1=(i>>j)&1; b2=(i>>k)&1;
                if(b1+b2==1) t+=f[j][k];
            }
        if(ans<t) ans=t, imx=i;
    }

    for(int i=0; i<19; ++i) u[i]=(imx>>i)&1;
    for(int i=0; i<n; ++i) if(u[flg[i]]) s[i]-=32;
    fo<<s;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Trạm thăm dò tự động được đưa vào quỹ đạo bay quanh hành tinh P 2019. Có một đối tượng trên hành tinh thu hút sự chú ý đặc biệt của các nhà khoa học. Người ta quyết định đo đạc khảo sát nó hai lần để đảm bảo tính chính xác của dữ liệu nhận được.

Thời gian thực hiện một lần đo đạc là một giờ. Trạm thăm dò bay một vòng quanh hành tinh là **a** giờ, vì vậy sau khi thực hiện đo đạc khảo sát lần thứ nhất thì sau **a**, $2*a$, $3*a$, ... giờ trạm thăm dò mới quay lại trên địa điểm cần khảo sát. Mỗi lần đo phải bắt đầu từ thời điểm là số nguyên của một giờ.

Trái đất có thể nhận tín hiệu của trạm thăm dò tốt nhất bắt đầu từ giờ **1f** đến hết giờ **rt**. Số liệu được truyền về trực tiếp ngay trong lúc tiến hành đo.

Hãy xác định có bao nhiêu cách thực hiện 2 lần đo đạc khảo sát. Hai cách gọi là khác nhau nếu khác nhau ít nhất ở một thời điểm bắt đầu đo.

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

Kết quả: Đưa ra file văn bản MEASURES.OUT: một số nguyên – số cách thực hiện 2 lần đo đạc khảo sát.

Ví dụ:

MEASURES.INP	MEASURES.OUT
1 5 2	4



Giải thuật; Cơ sở lập trình.

Với mỗi thời điểm bắt đầu i của lần đo thứ nhất – xác định số lượng thời điểm bắt đầu j đo lần 2:

- + $i < j$,
- + $j \leq rt$,
- + $(j-i) \% a = 0$.

Ký hiệu

- $k = (rt-lf+1)/a$,
- $m = (rt-lf+1)\%a = (rt-lf+1-k\times a)$.

Với mỗi điểm đầu i trong đoạn $[lf, lf+m-1]$ ta có k cách chọn j thỏa mãn các điều kiện nêu trên.

Từ $lf+m$ trở đi, mỗi điểm trong a điểm đầu tiếp theo sẽ lần lượt có $k-1, k-2, k-3, \dots$ điểm đầu thứ 2 tương ứng.

Như vậy, tổng số các cách chọn khác nhau sẽ là

$$(rt-lf+1-k\times a) \times k + a \times k \times (k-1) / 2$$

Dộ phức tạp của giải thuật: O(1).

Lưu ý: Kết quả có thể lớn hơn 10^9 .

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("measures.inp");
ofstream fo ("measures.out");
int64_t l,r,a,k,ans;

int main()
{
    fi>>l>>r>>a;
    r-=l-1;
    k = r/a;
    fo<<(r-k*a)*k + a*(k*(k-1)/2);
}
```



Để kiểm nghiệm một giải thuật mới người ta đã xây dựng dãy các xâu độ dài $0, 0+1, 0+1+3, \dots, 0+1+3+\dots+(2n-1), \dots$. Như vậy độ dài xâu thứ i sẽ là i^2 , $i = 0, 1, 2, \dots, n, \dots$

Người có nhiệm vụ xâu đã xây dựng một chương trình tổng quát, xây dựng các xâu với độ dài $k, k+1, k+1+3, \dots, k+1+3+\dots+(2n-1), \dots$. Sự tổng quát hóa này mang lại nhiều rắc rối hơn là thuận tiện vì với $k < 0$, nhiều số sẽ có giá trị âm và đương nhiên không thể xây dựng xâu với độ dài như vậy, với những xâu xây dựng được không phải xâu nào cũng có độ dài là số chính phương.

Với k cho trước hãy xác định số nguyên không âm nhỏ nhất mà bình phương của nó là độ dài của một trong số các xâu được chương trình khởi tạo.

Dữ liệu: Vào từ file văn bản SQUARES.INP: gồm một dòng chứa số nguyên k ($-10^{12} \leq k \leq 10^{12}$).

Kết quả: Đưa ra file văn bản SQUARES.OUT số nguyên tìm được hoặc thông báo **none** không tồn tại xâu nào có độ dài là số chính phương.

Ví dụ:

SQUARES.INP
-5

SQUARES.OUT
2



Giải thuật; Lý thuyết số.

Với \mathbf{k} cho trước cần tìm $\mathbf{b} \geq 0$ nhỏ nhất thỏa mãn điều kiện $\mathbf{k} + \mathbf{a}^2 = \mathbf{b}^2$.

Từ đây có $\mathbf{k} = \mathbf{b}^2 - \mathbf{a}^2 = (\mathbf{b} + \mathbf{a}) \times (\mathbf{b} - \mathbf{a})$.

Đặt $\mathbf{u} = \mathbf{b} - \mathbf{a}$, $\mathbf{v} = \mathbf{b} + \mathbf{a}$, ta có:

- $\mathbf{b} = (\mathbf{v} + \mathbf{u}) / 2$, $\mathbf{a} = (\mathbf{v} - \mathbf{u}) / 2$,
- $\mathbf{u} \leq \mathbf{v}$,
- \mathbf{u}, \mathbf{v} – ước của \mathbf{k} .

Nếu $\mathbf{k} < 0$ thì $\mathbf{u} < 0$.

Với $\mathbf{k} = 0$ dễ dàng thấy ngay $\mathbf{b} = 0$ là số cần tìm.

Với $\mathbf{k} \neq 0$ bài toán có nghiệm khi tồn tại \mathbf{u} và \mathbf{v} cùng tính chẵn lẻ. Như vậy, nếu $\mathbf{k} = 4 \times m + 2$ bài toán sẽ vô nghiệm.

Trường hợp $\mathbf{k} > 0$: $\mathbf{b} = \min\{(\mathbf{v} + \mathbf{u}) / 2, \mathbf{u}, \mathbf{v}\}$ – ước của \mathbf{k}

Trường hợp $\mathbf{k} < 0$: $\mathbf{b} = \min\{(\mathbf{v} - \mathbf{u}) / 2, \mathbf{u}, \mathbf{v}\}$ – ước của \mathbf{k} , $\mathbf{u} < 0$.

Độ phức tạp của giải thuật: $O(k^{1/2})$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("squares.inp");
ofstream fo ("squares.out");
int64_t ans= LLONG_MAX;

int main()
{
    int64_t k;
    fi >> k;
    if (k == 0)
    {
        fo << 0 << '\n';
        return 0;
    }
    if (abs(k) % 4 == 2)
    {
        fo<<"none\n";
        return 0;
    }
    if (k < 0)
    {
        k = -k;
        for (int64_t i = 1; i * i <= k; i++)
            if (k % i == 0 && (k / i - i) % 2 == 0)
                ans = min(ans, (k / i - i) / 2);
    } else
        for (int64_t i = 1; i * i <= k; i++)
            if (k % i == 0 && (i + k / i) % 2 == 0)
                ans = min(ans, (i + k / i) / 2);
    fo << ans << '\n';

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



Trải qua một đêm dài trên sao Hỏa một chiết áp của trạm thăm dò tự động bị tụt xuống mức **a**, một mức nguy hiểm. Từ trạm điều khiển ở Trái đất người ta phát các lệnh điều khiển nâng mức điện áp lên **b**. Có 2 loại lệnh: lệnh **X** nâng điện áp lên 1 và lệnh **Y** – nâng mức điện áp lên 2. Trong quá trình nâng phải tránh đưa chiết áp vào trạng thái có mức là bội của **c** – trạng thái không có lợi cho nhiều linh kiện khác.

Hãy xác định số lệnh tối thiểu cần truyền đi.

Dữ liệu: Vào từ file văn bản CORRECTION.INP gồm một dòng chứa 3 số nguyên **a**, **b** và **c** ($1 \leq a < b \leq 10^9$, $2 \leq c \leq 10^9$, **a** và **b** không phải là bội của **c**).

Kết quả: Đưa ra file văn bản CORRECTION.OUT một số nguyên – số lệnh ít nhất cần truyền đi.

Ví dụ:

CORRECTION.INP
4 10 3

CORRECTION.OUT
4

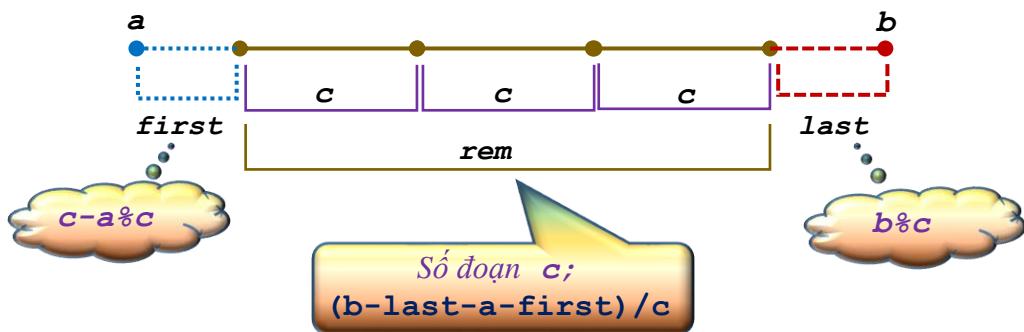


Giải thuật; Số học.

Đoạn $[a, b]$ có thể chia thành 3 phần:

- Phần đầu (**first**): Từ a đến số nguyên gần nhất lớn hơn a và chia hết cho c ,
- Phần cuối (**last**): Từ số nguyên gần nhất nhỏ hơn b và chia hết cho c tới b ,
- Phần còn lại (**rem**): Từ a đến số nguyên gần nhất lớn hơn a và chia hết cho c tới số nguyên gần nhất nhỏ hơn b và chia hết cho c .

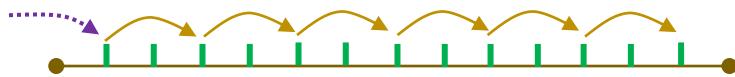
Phần còn lại có thể không có.



Xét đoạn độ dài c từ $m*c$ đến $(m+1)*c$:

Các giá trị cần đi qua khi ra lệnh tăng, phụ thuộc vào c , sẽ là:

$$m*c+1, m*c+3, \dots, m*c+2*p$$



hoặc là

$$m*c+1, m*c+3, \dots, m*c+2*p-2, m*c+2*p-1$$



Trong mọi trường hợp, số câu lệnh ít nhất cần thực hiện cho khoảng nói trên là $(c+1)/2$.

Lưu ý: Cần một lệnh để chuyển từ đoạn **rem** sang đoạn **last**!

Trường hợp riêng: tồn tại m thỏa mãn điều kiện $c*m < a < b < c*(m+1)$. Số lệnh ít nhất cần gửi sẽ là $(b-a+1)/2$.

Độ phức tạp của giải thuật: $O(1)$.

Chương trình

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

int main()
{
    fi>>a>>b>>c;
    first=c-a%c;
    last=b%c;
    if(a/c*c+c >b) ans=(b-a+1)/2;
    else
    {
        ans=first/2+last/2+1;
        k=(b-last-a-first)/c;
        ans+=k*( (c+1)/2 );
    }
    fo<<ans;
}
```



Trong quan sát và thực nghiệm các dữ liệu có giá trị ít khác biệt và xuất hiện gần nhau đóng vai trò rất quan trọng. Đài quan sát vũ trụ Paranal ở sa mạc Atacama (Chi lê) ghi nhận được chuỗi n tín hiệu lạ từ vũ trụ, tín hiệu thứ i có cường độ a_i , $i = 1 \div n$.

Để xác định bản chất các tín hiệu này việc đầu tiên người ta muốn xác định có cặp tín hiệu nào có khoảng cách xuất hiện không quá k và độ lệch cường độ nằm trong khoảng từ lf đến rt , tức là có tồn tại i và j thỏa mãn các điều kiện:

- $|i - j| \leq k$,
- $lf \leq |a_i - a_j| \leq rt$.

Hãy tìm và đưa ra một cặp chỉ số i , j thỏa mãn các điều kiện đã nêu. Nếu không tồn tại cặp dữ liệu thỏa mãn thì đưa ra 2 số -1 và -1 .

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

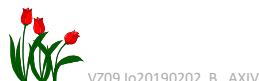
- ✚ Dòng đầu tiên chứa 4 số nguyên n , k , lf và rt ($2 \leq n \leq 10^5$, $1 \leq k \leq n$, $0 \leq lf \leq rt \leq 10^9$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^9$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản PAIR.OUT trên một dòng cặp số xác định được.

Ví dụ:

PAIR.INP
5 2 2 3
2 1 6 5 4

PAIR.OUT
3 5



Giải thuật: Tìm kiếm nhị phân .

Duy trì tập hợp **numbers** chứa **k-1** phần tử liên tiếp nhau của dãy **a** cùng với chỉ số của nó, tức là cặp **{a_j, j}**.

Với mỗi phần tử **a_i** tiếp theo, xét 2 trường hợp:

■ Tìm **a_j ≤ a_i**:

- Bằng phương pháp tìm kiếm nhị phân, xác định giá trị **a_j** nhỏ nhất trong **numbers** không nhỏ hơn **a_i-rt** (theo tham số thứ nhất của cặp dữ liệu),
- Nếu tồn tại phần tử cần tìm và giá trị đầu trong cặp không vượt quá **a_i-lf** → ta có cặp giá trị cần tìm:

$$\left. \begin{array}{l} a_j \geq a_i - rt \rightarrow a_i - a_j \leq rt \\ a_j \leq a_i - lf \rightarrow a_i - a_j \geq lf \end{array} \right\} \begin{array}{l} lf \leq a_i - a_j \leq rt \\ i > j \\ i - j \leq k \end{array}$$

■ Tìm **a_j ≥ a_i**:

- Bằng phương pháp tìm kiếm nhị phân, xác định phần tử **a_j** nhỏ nhất trong **numbers** không vượt quá **a_i+lf** theo tham số thứ nhất của cặp dữ liệu,
- Nếu tồn tại phần tử cần tìm và giá trị đầu trong cặp không vượt quá **a_i+rt** → ta có cặp giá trị cần

$$\left. \begin{array}{l} a_j \geq a_i + lf \rightarrow a_j - a_i \geq lf \\ a_j \leq a_i + rt \rightarrow a_j - a_i \geq rt \end{array} \right\} \begin{array}{l} lf \leq a_j - a_i \leq rt \\ i > j \\ i - j \leq k \end{array}$$

Trong mọi trường hợp: Kết thúc xử lý khi tìm được cặp số thích hợp.

Quản lý tập **numbers**:

Khi chưa đủ **k-1** phần tử: tiếp tục bổ sung phần tử mới,

Khi lực lượng bằng **k** – cần loại bỏ phần tử cũ nhất (phần tử **{a_{i-k}, i-k}**)

Tổ chức dữ liệu:

- Mảng **vector<int> a(n)** – lưu dữ liệu input,
- Tập **set< pair<int, int>> numbers** – phòng đệm xử lý.

Độ phức tạp của giải thuật: O(nlogn).

Chương trình

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

pair<int,int> get_p(const set<pair<int, int>> &s, int l, int r)
{
    auto it = s.lower_bound({l, -1});
    if (it != s.end() && it->first <= r) return *it;
    return {-1, -1};
}

int main()
{
    int n, k, l, r;
    fi >> n >> k >> l >> r;
    vector<int> a(n);
    set< pair<int, int>> numbers;
    for (int i = 0; i < n; i++)
    {
        fi >> a[i];
        auto xl = get_p(numbers, a[i] - r, a[i] - l);
        auto xr = get_p(numbers, a[i] + l, a[i] + r);
        if (xl.first != -1)
        {
            fo << xl.second + 1 << ' ' << i + 1 << '\n';
            Times; return 0;
        }
        if (xr.first != -1)
        {
            fo << xr.second + 1 << ' ' << i + 1 << '\n';
            Times; return 0;
        }
        numbers.insert({a[i], i});
        if (i >= k) numbers.erase({a[i - k], i - k});
    }
    fo << "-1 -1\n";
    Times; return 0;
}
```



Khu vực bị nhiễm Dioxin có hình chữ nhật, được chia thành n hàng và m cột được quy hoạch làm khu chế xuất công nghiệp. Ô giao giữa hàng x và cột y có tọa độ (x, y). Do kinh phí có hạn người ta chỉ có thể làm sạch chất độc ở một số ô để đưa vào khai thác mỗi khi được cấp vốn.

Khi một nhà đầu tư muốn thuê ô (x, y), tùy theo cách sử dụng người ta quan tâm đến ô gần nhất chưa được làm sạch theo một trong số 4 hướng lên trên (' u '), xuống dưới (' d '), sang trái (' l ') hoặc sang phải (' r ') ô gần nhất chưa được làm sạch là ô nào hay mọi ô theo hướng đó đã được làm sạch.

Cho q truy vấn dạng “ $c \ x \ y$ ”, trong đó c là ký tự thuộc tập {‘ c ’, ‘ u ’, ‘ d ’, ‘ l ’, ‘ r ’}, $c = 'c'$ – tẩy sạch ô (x, y), $c \neq 'c'$ – truy vấn tìm ô chưa được làm sạch gần nhất theo hướng c .

Với mỗi truy vấn tìm ô chưa được làm sạch gần nhất hãy đưa ra tọa độ ô tìm được hoặc giá trị -1 nếu mọi ô theo hướng đó đều đã được làm sạch.

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

- ✚ Dòng đầu tiên chứa 3 số nguyên n, m và q , ($1 \leq n, m \leq 2\,000, 1 \leq q \leq 10^6$),
- ✚ Mỗi dòng trong q dòng sau chứa thông tin về một truy vấn theo dạng đã nêu.

Kết quả: Đưa ra file văn bản DIOXIN.OUT các giá trị tìm được với mỗi truy vấn có $c \neq 'c'$, mỗi kết quả đưa ra trên một dòng.

Ví dụ:

DIOXIN.INP	DIOXIN.OUT
3 4 6	1 3
u 2 3	-1
c 2 4	2 2
r 2 4	3 3
c 2 3	
l 2 4	
d 1 3	



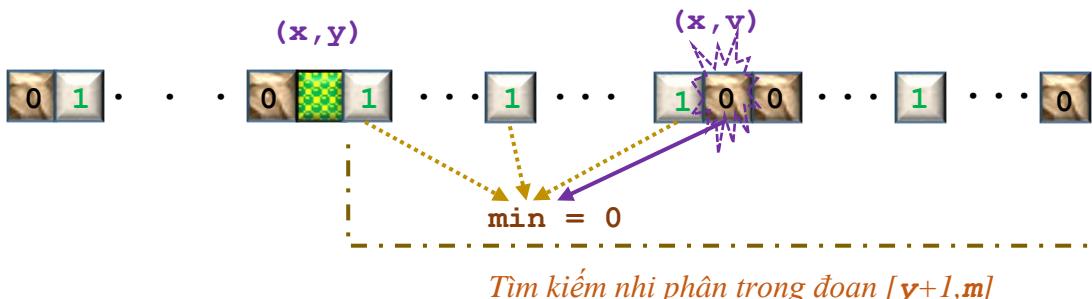
Giải thuật I: Cây phân đoạn 2 chiều có cập nhật giá trị đơn, tìm kiếm nhị phân

Ban đầu tất cả các ô được gán giá trị 0 – chưa có ô nào được làm sạch, Nếu ô (x, y) được làm sạch – giá trị tương ứng được gán 1.

Với các truy vấn tìm ô chưa được làm sạch gần nhất theo hướng đã cho với ô (x, y) cần tìm ô (x, v) hoặc (u, y) gần nhất có giá trị 0 (tùy theo hướng).

Việc tìm theo *mỗi hướng* trong số 4 hướng là *tương tự nhau nhau*, vì vậy ta chỉ cần xét chi tiết về một hướng.

Xét hướng ‘r’ (tìm theo hàng, từ ô (x, y) về cuối):



Dùng cây phân đoạn *quản lý min* các phần tử,

Cây phân đoạn phải *có khả năng cập nhật* khi *thay đổi giá trị phần tử* trên đoạn,

Thực hiện tìm kiếm nhị phân trong đoạn $[y+1, m]$ để xác định vị trí số 0 gần nhất, nếu không tìm được → kết quả là -1.

Với *mỗi hàng* và với *mỗi cột* – cần *một cây phân đoạn*.

Tổ chức dữ liệu:

█ Mảng vector<vector<int>> vector<int>(bpv_hor * 2)) – tổ chức cây quản lý hàng,	rmq_hor(n, bpv_ver * 2)) – tổ chức cây quản lý cột.
█ Mảng vector<vector<int>> vector<int>(bpv_ver * 2)) – tổ chức cây quản lý cột.	rmq_ver(m, bpv_hor * 2))

Xử lý:

Xác định kích thước và khai báo cây phân đoạn 2 chiều:

```

int n, m, q;
fi >> n >> m >> q;
int bpv_hor = 1, bpv_ver = 1;
while (bpv_hor < m) bpv_hor <<= 1;
while (bpv_ver < n) bpv_ver <<= 1;
vector<vector<int>> rmq_hor(n, vector<int>(bpv_hor * 2));
vector<vector<int>> rmq_ver(m, vector<int>(bpv_ver * 2));

```

Xác định lũy thừa của 2
nhỏ nhất lớn hơn hoặc
bằng kích thước quản lý

Cách xin cấp phát bộ nhớ

Cập nhật cây khi thay đổi giá trị một nút:



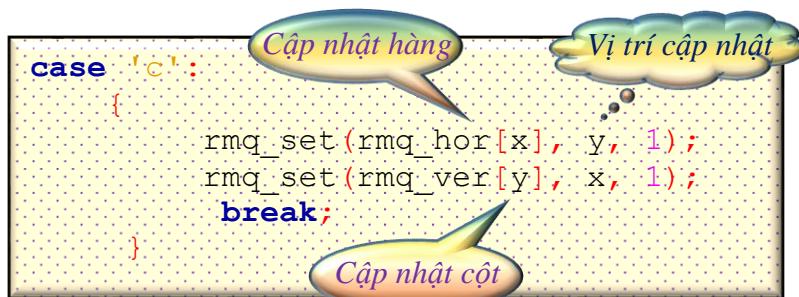
```
function<void(vector<int>&, int, int)>
    rmq_set = [&] (vector<int>& rmq, int pos, int val)
{
    pos += rmq.size() / 2;
    rmq[pos] = val;
    pos /= 2;
    while (pos)
    {
        rmq[pos] = min(rmq[pos * 2], rmq[pos * 2 + 1]);
        pos >>= 1;
    }
};
```

Hàm tìm *min* trong khoảng [l, r]:

```
function<int(vector<int>&, int, int, int, int, int)>
    rmq_get = [&]
        (vector<int>& rmq, int v, int vl, int vr, int l, int r)
{
    if (vr <= l || r <= vl) return INF;
    if (l <= vl && vr <= r) return rmq[v];
    int vm = (vl + vr) / 2;
    return min(rmq_get(rmq, v * 2, vl, vm, l, r),
               rmq_get(rmq, v * 2 + 1, vm, vr, l, r));
};
```

Cách xử lý: Theo đúng sơ đồ lý thuyết về cách tìm min ở cây có cập nhật. Sự thay đổi chỉ xảy ra ở *một giá trị*, vì vậy *không cần tổ chức truyền cập nhật trễ* (*Lazy Propagation*).

Xử lý truy vấn **c** = 'c':



```
case 'C':
{
    rmq_set(rmq_hor[x], y, 1);
    rmq_set(rmq_ver[y], x, 1);
    break;
}
```

Xử lý truy vấn **c = 'r'**: Xét khoảng $[y+1, m]$.



```
case 'r':
{
    int l = y, r = m;
    while (r - l > 1)
    {
        int mid = (l + r) / 2;
        if (rmq_get(rmq_hor[x], 1, 0,
                     bpv_hor, y + 1, mid + 1) == 0) r = mid;
        else l = mid;
    }
    if (r == m) fo << "-1\n";
    else fo << x + 1 << " " << r + 1 << "\n";
    break;
}
```

Các truy vấn khác: Xử lý tương tự.

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

Nhận xét: Việc sử dụng đệ quy sẽ làm chậm đáng để tiến độ thực hiện chương trình.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("dioxin.inp");
ofstream fo ("dioxin.out");
int const INF = (int)1e9 + 1e3;

int main()
{
    int n, m, q;
    fi >> n >> m >> q;
    int bpv_hor = 1, bpv_ver = 1;
    while (bpv_hor < m) bpv_hor <= 1;
    while (bpv_ver < n) bpv_ver <= 1;

    vector<vector<int>> rmq_hor(n, vector<int>(bpv_hor * 2));
    vector<vector<int>> rmq_ver(m, vector<int>(bpv_ver * 2));

    function<void(vector<int>&, int, int)>
        rmq_set = [&] (vector<int>& rmq, int pos, int val)
    {
        pos += rmq.size() / 2;
        rmq[pos] = val;
        pos /= 2;
        while (pos)
        {
            rmq[pos] = min(rmq[pos * 2], rmq[pos * 2 + 1]);
            pos >= 1;
        }
    };

    function<int(vector<int>&, int, int, int, int, int)>
        rmq_get = [&] (vector<int>& rmq, int v, int vl, int vr, int l, int r)
    {
        if (vr <= l || r <= vl) return INF;
        if (l <= vl && vr <= r) return rmq[v];
        int vm = (vl + vr) / 2;
        return min(rmq_get(rmq, v * 2, vl, vm, l, r),
                   rmq_get(rmq, v * 2 + 1, vm, vr, l, r));
    };

    for (int i = 0; i < q; ++i)
    {
        char c;
        int x, y;
        fi >> c >> x >> y;
        --x; --y;
        switch(c)
        {
            case 'c':
            {
                rmq_set(rmq_hor[x], y, 1);
                rmq_set(rmq_ver[y], x, 1);
                break;
            }
            case 'd':
            {
                int l = x, r = n;
                while (r - l > 1)
                {
                    int mid = (l + r) / 2;
                    if (rmq_get(rmq_ver[y], 1, 0,
```

```

                bpv_ver, x + 1, mid + 1) == 0) r = mid;
            else l = mid;
        }
        if (r == n) fo << "-1\n";
        else fo << r + 1 << " " << y + 1 << "\n";
        break;
    }
case 'u':
{
    int l = -1, r = x;
    while (r - l > 1)
    {
        int mid = (l + r) / 2;
        if (rmq_get(rmq_ver[y], 1, 0,
                     bpv_ver, mid, x) == 0) l = mid;
        else r = mid;
    }
    if (l == -1) fo << "-1\n";
    else fo << l + 1 << " " << y + 1 << "\n";
    break;
}
case 'r':
{
    int l = y, r = m;
    while (r - l > 1)
    {
        int mid = (l + r) / 2;
        if (rmq_get(rmq_hor[x], 1, 0,
                     bpv_hor, y + 1, mid + 1) == 0) r = mid;
        else l = mid;
    }
    if (r == m) fo << "-1\n";
    else fo << x + 1 << " " << r + 1 << "\n";
    break;
}
case 'l':
{
    int l = -1, r = y;
    while (r - l > 1)
    {
        int mid = (l + r) / 2;
        if (rmq_get(rmq_hor[x], 1, 0,
                     bpv_hor, mid, y) == 0) l = mid;
        else r = mid;
    }
    if (l == -1) fo << "-1\n";
    else fo << x + 1 << " " << l + 1 << "\n";
    break;
}
}
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```

Giải thuật II; Duyệt trực tiếp .

Kích thước mảng xử lý không quá lớn ($1 \leq n, m \leq 2000$) → có thể lưu *bảng trạng thái* của mảng và tổ chức duyệt trực tiếp để tìm vị trí 0 gần nhất theo hướng yêu cầu.

Để tăng tốc độ tìm kiếm cần tạo mảng 3 chiều xác định *vị trí ô kè cạnh* với ô đang xét *theo hướng xử lý*.

Mảng tra cứu được cập nhật bộ phận khi xử lý các truy vấn có $c \neq 'c'$.

Trong trường hợp xấu nhất (n, m – lớn) *thời gian thực hiện vẫn giảm từ 3 đến 4 lần* so với phương pháp tổ chức cây phân đoạn có cập nhật!

Chương trình sẽ *ngắn gọn* hơn nếu tổ chức *bảng tra cứu hướng* xử lý.

Tổ chức dữ liệu:

- Mảng `vector<vector<int>> field(n, vector<int>(m))` – lưu trạng thái các ô của mảng xử lý,
- Mảng ba chiều
`vector<vector<vector<pii>>`
`go(4, vector<vector<pii>>(n, vector<pii>(m)))`
lưu bảng xác định ô kè cạnh,
- Các dữ liệu bổ trợ:
`int dx[4] = {1, 0, -1, 0};`
`int dy[4] = {0, 1, 0, -1};`
`string p = "drul";`

Xử lý:

Chuẩn bị:

```

int n, m, q, x, y, dir;
fi >> n >> m >> q;
vector<vector<vector<pii>>>
    go(4, vector<vector<pii>>(n, vector<pii>(m)));
for (int i = 0; i < 4; ++i)
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < m; ++k)
            go[i][j][k] = {j + dx[i], k + dy[i]};
vector<vector<int>> field(n, vector<int>(m));

```

Khai báo và xin cấp phát bộ nhớ cho mảng 3 chiều

Khai báo và xin cấp phát bộ nhớ cho mảng 2 chiều

Xử lý truy vấn **c** = ‘**c**’: **field[x][y] = 1;**

Xử lý các truy vấn khác:

```
dir = p.find(c);
x += dx[dir];
y += dy[dir];
int memx = x, memy = y;

while (0 <= x && x < n && 0 <= y && y < m && field[x][y])
    tie(x, y) = go[dir][x][y];

int resx = x, resy = y;
x = memx, y = memy;
while (0 <= x && x < n && 0 <= y && y < m && field[x][y])
{
    auto tmp = go[dir][x][y];
    go[dir][x][y] = {resx, resy};
    tie(x, y) = tmp;
}

if (0 <= resx && resx < n && 0 <= resy && resy < m)
    fo << resx + 1 << " " << resy + 1 << "\n";
else fo << "-1\n";
```

Độ phức tạp của giải thuật: Độ phức tạp trung bình không vượt quá độ phức tạp theo phương pháp dùng cây phân đoạn, thông thường làm việc nhanh hơn do không dùng đệ quy và có cập nhật bảng tra cứu.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("dioxin.inp");
ofstream fo ("dioxin.out");
typedef pair<int,int> pii;
int dx[4] = {1, 0, -1, 0};
int dy[4] = {0, 1, 0, -1};
string p = "drul";

int main()
{
    int n, m, q, x, y, dir;
    fi >> n >> m >> q;
    vector<vector<vector<pii>>>
        go(4, vector<vector<pii>>(n, vector<pii>(m)));
    for (int i = 0; i < 4; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < m; ++k)
                go[i][j][k] = {j + dx[i], k + dy[i]};

    vector<vector<int>> field(n, vector<int>(m));
    for (int i = 0; i < q; ++i)
    {
        char c;
        fi >> c >> x >> y;
        --x; --y;
        if(c=='c') field[x][y] = 1;
        else dir = p.find(c);
        x += dx[dir];
        y += dy[dir];
        int memx = x, memy = y;
        while (0 <= x && x < n && 0 <= y && y < m && field[x][y])
            tie(x, y) = go[dir][x][y];
        int resx = x, resy = y;
        x = memx, y = memy;
        while (0 <= x && x < n && 0 <= y && y < m && field[x][y])
        {
            auto tmp = go[dir][x][y];
            go[dir][x][y] = {resx, resy};
            tie(x, y) = tmp;
        }
        if (0 <= resx && resx < n && 0 <= resy && resy < m)
            fo << resx + 1 << " " << resy + 1 << "\n";
        else fo << "-1\n";
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Sắp xếp dữ liệu đóng vai trò quan trọng trong tin học. Người ta luôn luôn phải tìm cách sắp xếp hiệu quả cho từng lớp bài toán cần giải quyết.

Bản chất của sắp xếp là tìm nghịch thế và đổi chỗ dữ liệu để khắc phục nghịch thế.

Xét một cách sắp xếp tăng dần dãy số nguyên a_1, a_2, \dots, a_n , trong đó $1 \leq a_i \leq n$, $a_i \neq a_j$ với $i \neq j$, $i, j = 1 \div n$. Hai số nguyên a_i và a_j tạo thành nghịch thế nếu $a_i > a_j$ và $i < j$. Bước đầu tiên người ta chọn số nguyên x làm phần tử chốt. Các phần tử của dãy có giá trị nhỏ hơn x sẽ chuyển lên đứng trước x , các phần tử lớn – chuyển về sau x , ở mỗi phần (trước và sau x) các phần tử của dãy vẫn giữ nguyên trình tự xuất hiện như trong dãy ban đầu: nếu a_i đứng trước a_j trong dãy ban đầu và $a_i < x, a_j < x$ hoặc $a_i > x, a_j > x$ thì sau bước đầu tiên a_i vẫn đứng trước a_j . Ví dụ, với dãy số 3, 6, 1, 4, 2, 5 và $x = 4$, sau bước sắp xếp đầu tiên ta có dãy 3, 1, 2, 4, 6, 5. Với $x = 2$ ta có kết quả 1, 2, 3, 6, 4, 5.

Các cách chọn phần tử chốt khác nhau sẽ cho dãy kết quả với số lượng cặp nghịch thế khác nhau ở bước đầu tiên.

Hãy xác định số lượng cặp nghịch ít nhất có thể đạt được ở bước đầu tiên.

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

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 3 \times 10^6$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n .

Kết quả: Đưa ra file văn bản PART_ST.OUT một số nguyên – số lượng cặp nghịch ít nhất có thể đạt được ở bước đầu tiên.

Ví dụ:

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



Giải thuật; Cây Fenwick.

Xét cặp nghịch thέ a_i và a_j : $a_i > a_j$ và $i < j$.

Nếu $x > a_i$ hoặc $x < a_j$ các phần tử a_i và a_j vẫn ở nguyên phần trái hoặc phải so với x với trình tự như cũ và do đó không làm thay đổi số lượng nghịch thέ.

Nếu $a_i > x > a_j$, $i < j$ thì 2 số này sẽ thay đổi vị trí và làm số lượng nghịch thέ giảm 1.

Như vậy *trong dãy kết quả* nghịch thέ chỉ *xuất hiện cục bộ* ở phần bên phải và ở phần bên trái của x .

Trong dãy kết quả phần tử giá trị x sẽ đứng ở vị trí x .

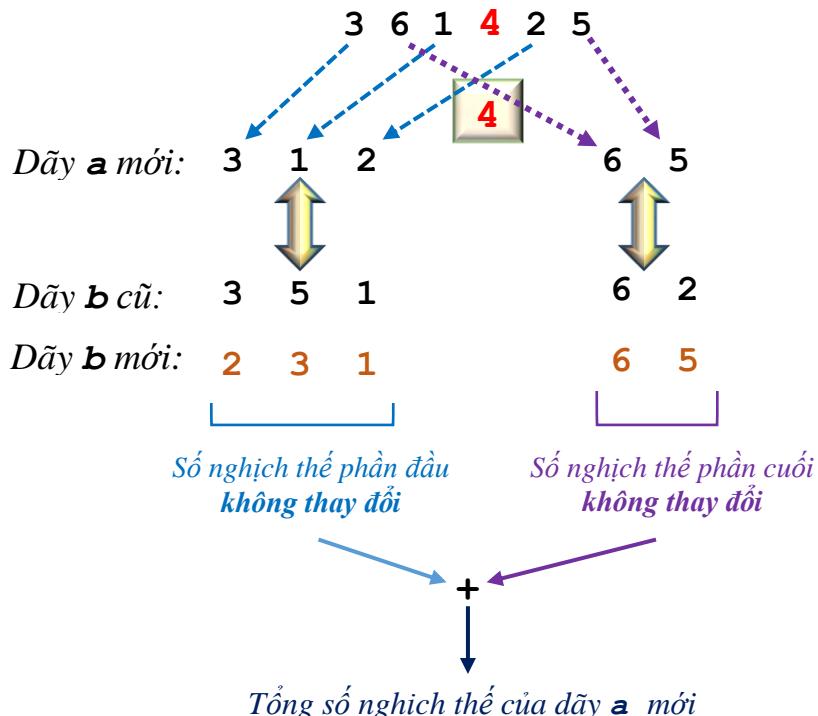
Gọi b_i là vị trí của phần tử giá trị i trong dãy ban đầu: $b_{a[i]} = i$.

Dãy b_1, b_2, \dots, b_n được gọi là dãy *đổi ngẫu* hay dãy *nghịch đảo* của dãy a_1, a_2, \dots, a_n .

$$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ \text{Dãy } a: & 3 & 6 & 1 & 4 & 2 & 5 \\ \text{Dãy } b: & 3 & 5 & 1 & 4 & 6 & 2 \end{array}$$

Số lượng nghịch thέ của 2 dãy luôn luôn bằng nhau.

Khi chọn phần tử chốt dãy bị tách thành 2 phần, ví dụ với $x = 4$:



Như vậy, khi cố định x , ta có thể tính số nghịch thέ phần đầu của dãy kết quả a thông qua *phần đầu* của dãy b ban đầu.

Số lượng nghịch thế của phần tử \mathbf{b}_i tạo ra với các phần tử đứng sau nó trong dãy gọi là số nghịch thế của \mathbf{b}_i ,

Số nghịch thế ở mỗi phần là tổng số nghịch thế của từng phần tử trong đó.

Việc tính số nghịch thế có thể dễ dàng thực hiện bằng cây Fenwick.

Sơ đồ xử lý chung:

- Chọn i làm phần tử chốt, $i = 0 \div n-1$,
- Với mỗi i : tính tổng số lượng nghịch thế ở 2 phần tương ứng,
- Cập nhật \min của tổng nhận được.

Để tránh tính lại số nghịch thế của từng phần tử (ở cả 2 phần) cần tổ chức mảng tích lũy số nghịch thế các phần tử đầu dãy (mảng *tổng tiền tố*) và mảng tích lũy số nghịch thế các phần tử cuối dãy (mảng *tổng hậu tố*).

Tính *số nghịch thế ở phần cuối*:

Sử dụng sơ đồ tích lũy offline với cây Fenwick: Lần lượt nạp \mathbf{b}_i vào cây với $i=n-1 \div 0$ và tính tổng bằng cách duyệt các phần tử của cây từ i về đầu.

Tính *số nghịch thế ở phần đầu*:

Sử dụng sơ đồ tích lũy online với cây Fenwick: Lần lượt nạp \mathbf{b}_i vào cây với $i=0 \div n-1$. Gọi hàm **sum_fw(b_i)** ta sẽ có số lượng thuận thế, số lượng nghịch thế sẽ là phần bù với \mathbf{b}_i .

Lưu ý: Việc sử dụng chỉ số bắt đầu từ 0 sẽ dẫn đến một số thay đổi nhỏ trong cách tính chỉ số ở các hàm xử lý cây Fenwick.

Tổ chức dữ liệu

- ─ Mảng **vector<int>** $a(n)$ – lưu trữ dãy \mathbf{a} ,
- ─ Mảng **vector<int>** $b(n)$ – lưu trữ dãy \mathbf{b} ,
- ─ Mảng **vector<int>** $str(n)$ – phục vụ tổ chức cây Fenwick chế độ offline,
- ─ Mảng **vector<int>** $stl(n)$ – phục vụ tổ chức cây Fenwick chế độ online,
- ─ Mảng **vector<ll>** $cntl(n)$ – lưu tổng tiền tố số nghịch thế của dãy \mathbf{b} ,
- ─ Mảng **vector<ll>** $cntr(n)$ – lưu tổng hậu tố số nghịch thế của dãy \mathbf{b} .

Xử lý

Với việc sử dụng phổ biến chỉ số bắt đầu từ 0 các hàm làm việc với cây Fenwick có dạng:

Hàm nạp dữ liệu vào cây:

```
void add(vector<int> &fw, int i, int v)
{
    while (i < fw.size())
    {
        fw[i] += v;
        i = i | (i + 1); Cho trường hợp tổng quát
    }
}
```

Nếu chỉ số bắt đầu từ 1:
 $i+=i \& (-i)$

Hàm tính tổng:

```
int pref(vector<int> &fw, int i)
{
    int s = 0;
    while (i >= 0)
    {
        s += fw[i];
        i = (i & (i + 1)) - 1;
    }
    return s;
}
```

Nếu chỉ số bắt đầu từ 1:
 $x \&= (x - 1)$

Hàm tính phần bù:Tính số nghịch thế trong đoạn $[0, l]$

```
int get(vector<int> &fw, int l, int r)
{
    return pref(fw, r) - pref(fw, l - 1);
}
```

Tính tổng tiền tố các nghịch thế (số lượng nghịch thế phần đầu):

```
Nap  $b_i$  vào cây
for (int i = 0; i < n; i++)
{
    cntl[i] = get(stl, b[i] + 1, n - 1);
    add(stl, b[i], 1);
    if (i > 0) cntl[i] += cntl[i - 1];
}
```

Tính số phần tử
lớn hơn b_i

Tích lũy tổng
tiền tố

Tính tổng hậu tố các nghịch thế:

Chế độ offline tích lũy số nghịch thế

```
for (int i = n - 1; i >= 0; i--)  
{  
    cntr[i] = pref(str, b[i]);  
    add(str, b[i], 1);  
    if (i < n - 1) cntr[i] += cntr[i + 1];  
}
```

Dẫn xuất kết quả:

```
ll best = INF;  
  
for (int i = 0; i < n; i++)  
    best=min(best, (i>0 ? cntr[i-1]:0)+(i<n-1 ? cntr[i+1]:0));  
  
fo << best << endl;
```

Trường hợp biên

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

Chương trình I

```
#include <bits/stdc++.h>
using namespace std;
typedef int64_t ll;
ifstream fi ("Part_st.inp");
ofstream fo ("Part_st.out");
const ll INF = (ll)1e18;

void add(vector<int> &fw, int i, int v)
{
    while (i < fw.size())
    {
        fw[i] += v;
        i = i | (i + 1);
    }
}

int pref(vector<int> &fw, int i)
{
    int s = 0;
    while (i >= 0)
    {
        s += fw[i];
        i = (i & (i + 1)) - 1;
    }
    return s;
}

int get(vector<int> &fw, int l, int r)
{
    return pref(fw, r) - pref(fw, l - 1);
}

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

    vector<long long> cntl(n), cntr(n);
    vector<int> stl(n), str(n);
    for (int i = 0; i < n; i++)
    {
        cntl[i] = get(stl,b[i] + 1, n - 1);
        add(stl,b[i], 1);
        if (i > 0) cntl[i] += cntl[i - 1];
    }
    for (int i = n - 1; i >= 0; i--)
    {
        cntr[i] = pref(str,b[i]);
        add(str,b[i], 1);
        if (i < n - 1) cntr[i] += cntr[i + 1];
    }

    ll best = INF;
```

```

for (int i = 0; i < n; i++)
    best = min(best, (i>0 ? cntl[i-1] : 0) + (i<n-1 ? cntr[i+1] : 0));
    fo << best << endl;
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
return 0;
}

```

Nhận xét: Với **n** khá lớn ($n \leq 3 \times 10^6$), *giảm các tham số cần truyền* cho các hàm sẽ giảm đáng kể thời gian thực hiện chương trình.

Để giảm độ phức tạp giải thuật cần giảm bớt tính tổng quát hóa của chương trình:

- Khai báo biến ở *vị trí thích hợp*,
- Giảm số mảng động, thay thế bằng khai báo tĩnh nếu có thể (sẽ tốn bộ nhớ hơn),
- Xây dựng riêng các hàm cho mỗi bộ dữ liệu.

Chương trình II áp dụng cơ chế xây dựng riêng các hàm cho mỗi bộ dữ liệu và thời gian thực hiện chương trình với $n \approx 10^6$ đã giảm một cách đáng kể.

Chương trình II

```

#include <bits/stdc++.h>
using namespace std;
typedef int64_t ll;
ifstream fi ("Part_st.inp");
ofstream fo ("Part_st.out");
const ll INF = (ll)1e18;
vector<int> stl, str;
int n;

void add_l(int i)
{
    while (i < n)
    {
        stl[i] += 1;
        i = i | (i + 1);
    }
}

int pref_l(int i)
{
    int s = 0;
    while (i >= 0)
    {
        s += stl[i];
        i = (i & (i + 1)) - 1;
    }
    return s;
}

int get_l(int l, int r)
{
    return pref_l(r) - pref_l(l - 1);
}

void add_r(int i)
{
}

```

```

        while (i < n)
        {
            str[i] += 1;
            i = i | (i + 1);
        }

    int pref_r(int i)
    {
        int s = 0;
        while (i >= 0)
        {
            s += str[i];
            i = (i & (i + 1)) - 1;
        }
        return s;
    }

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

    vector<ll> cntl(n), cntr(n);
    stl.resize(n); str.resize(n);
    for (int i = 0; i < n; i++)
    {
        cntl[i] = get_l(b[i] + 1, n - 1);
        add_l(b[i]);
        if (i > 0) cntl[i] += cntl[i - 1];
    }
    for (int i = n - 1; i >= 0; i--)
    {
        cntr[i] = pref_r(b[i]);
        add_r(b[i]);
        if (i < n - 1) cntr[i] += cntr[i + 1];
    }
}

ll best = INF;

for (int i = 0; i < n; i++)
    best = min(best, (i > 0 ? cntl[i - 1] : 0) + (i < n - 1 ? cntr[i + 1] : 0));

fo << best << endl;
fo << "\nTime: " << clock() / (double) 1000 << " sec";
return 0;
}

```

Giải pháp dung hòa:

Để không mất tính tổng quát nhưng vẫn giảm thời gian truyền tham số – áp dụng kỹ thuật OOP, gắn các hàm xử lý với dữ liệu. Thời gian thực hiện chương trình nằm giữa phương án I và II. Chương trình đủ gọn và không bị phụ thuộc vào thời điểm khai báo biến!

Chương trình III

```
#include <bits/stdc++.h>
using namespace std;
typedef int64_t ll;
ifstream fi ("Part_st.inp");
ofstream fo ("Part_st.out");
const ll INF = (ll)1e18;

struct fwTree {
    int sz;
    vector<int> fw;

    fwTree(int sz) : sz(sz) { fw.resize(sz); }

    void add(int i, int v)
    {
        while (i < sz) {
            fw[i] += v;
            i = i | (i + 1);
        }
    }

    int pref(int i)
    {
        int s = 0;
        while (i >= 0)
        {
            s += fw[i];
            i = (i & (i + 1)) - 1;
        }
        return s;
    }

    int get(int l, int r)
    {
        return pref(r) - pref(l - 1);
    }
};

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

    vector<ll> cntl(n), cntr(n);
    fwTree stl(n);
    for (int i = 0; i < n; i++)
    {
        cntl[i] = stl.get(b[i] + 1, n - 1);
        stl.add(b[i], 1);
        if (i > 0) {
            cntl[i] += cntl[i - 1];
        }
    }
}
```

```

}

fwTreee str(n);
for (int i = n - 1; i >= 0; i--)
{
    cntr[i] = str.pref(b[i]);
    str.add(b[i], 1);
    if (i < n - 1) cntr[i] += cntr[i + 1];
}

ll best = INF;
for (int i = 0; i < n; i++)
    best = min(best, (i>0 ? cntl[i-1] : 0) + (i<n-1 ? cntr[i+1] : 0));

fo << best << endl;
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
return 0;
}

```



VZ12. HAI NHIỆM VỤ

Tên chương trình: TWOTASKS.CPP

Mẹ đang dở tay trong bếp và vì vậy để nghị Alice chạy ra siêu thị mua thêm một mặt hàng, đồng thời qua bưu điện nhận bưu phẩm.

Alice không bao giờ từ chối giúp mẹ, nhưng muôn đi thật nhanh để về lập trình tiếp bài đang làm.

Đường đi từ nhà tới siêu thị có độ dài **a**, từ nhà tới bưu điện – độ dài **b** và từ bưu điện tới siêu thị - độ dài **c**. Với tay không tốc độ đi của Alice là **v0**, khi mang hàng mua được hoặc bưu phẩm, Alice đi với tốc độ **v1**, còn khi mang trên tay cả 2 thứ thì đi với tốc độ **v2** (**v2** ≤ **v1** ≤ **v0**).

Để hoàn thành nhiệm vụ được giao Alice có thể đi và mang về nhà cùng lúc cả 2 thứ hoặc lần lượt mang từng thứ một về nhà.

Với thói quen của người lập trình Alice nhầm tính cách thực hiện nhanh nhất nhiệm vụ được giao.

Hãy xác định thời gian ngắn nhất thực hiện nhiệm vụ.

Dữ liệu: Vào từ file văn bản TWOTASKS.INP gồm một dòng chứa 6 số nguyên **a**, **b**, **c**, **v0**, **v1**, **v2** ($1 \leq a, b, c \leq 100$, $1 \leq v2 \leq v1 \leq v0 \leq 100$).

Kết quả: Đưa ra file văn bản TWOTASKS.OUT: một số thực không ít hơn 4 chữ số sau dấu chấm thập phân xác định thời gian ngắn nhất thực hiện nhiệm vụ.

Ví dụ:

TWOTASKS.INP
2 3 4 7 6 5

TWOTASKS.OUT
1.495238095238095255

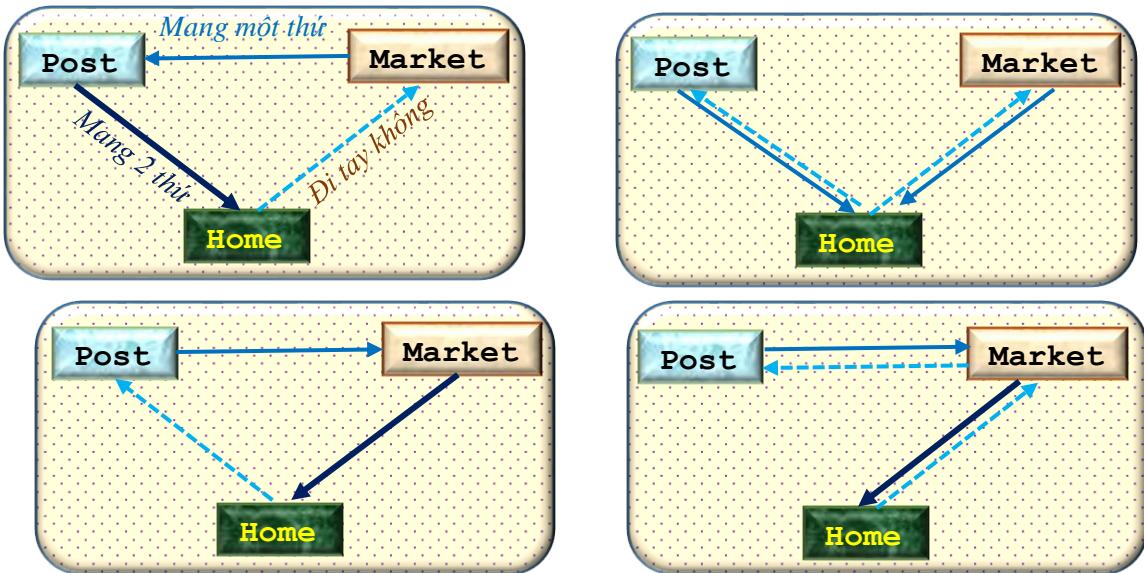


Giải thuật; Duyệt vét cạn .

Trước hết cần xác định độ dài đường đi ngắn nhất từ một địa điểm tới địa điểm khác:

- $a = \min(a, b+c)$,
- $b = \min(b, a+c)$,
- $c = \min(c, a+b)$.

Các cách khác nhau thực hiện nhiệm vụ:



Tính thời gian cho từng trường hợp và chọn kết quả nhỏ nhất.

Dộ phức tạp của giải thuật: O(1).

Chương trình

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

int main()
{
    double a, b, c, v0, v1, v2;
    fi >> a >> b >> c >> v0 >> v1 >> v2;
    a = min(a, b + c);
    b = min(b, a + c);
    c = min(c, a + b);
    double r1 = a / v0 + c / v1 + b / v2;
    double r2 = a / v0 + a / v1 + b / v0 + b / v1;
    double r3 = b / v0 + c / v1 + a / v2;
    double r4 = b / v0 + b / v1 + a / v0 + a / v2;
    fo.precision(20);
    fo << min(min(r1, r2), r3), r4) << '\n';
    return 0;
}
```



Để giảm thiểu tai nạn giao thông ô tô không người lái đã được chế tạo và đưa vào thử nghiệm. Đi xa hơn nữa, người ta đã chế tạo thiết bị bay tự động không người lái và đưa vào khai thác dưới dạng taxi bay tự động.

Các địa điểm trong địa bàn hoạt động của taxi bay được mã hóa bằng một số nguyên không âm. Mỗi taxi có một bộ các điểm chót a_1, a_2, \dots, a_n . Khi hành khách lên xe ở điểm A và đưa yêu cầu tới điểm B , hệ thống máy tính trên xe sẽ tính toán và đưa ra trên màn hình dãy k số nguyên $a_{i1}, a_{i2}, \dots, a_{ik}$. Hành khách phải lần lượt bấm vào các số này để tới đích. Nếu xe đang ở vị trí x thì khi bấm nút a_i , xe sẽ di chuyển tới địa điểm x or a_i .

Để hạn chế người đi chiếm dụng xe quá lâu, hệ thống chỉ cho phép bấm nút chọn không quá 100 lần. Ngoài ra, có thể hành trình đã thiết kế của xe không cho phép đi được từ A tới B , khi đó trên màn hình sẽ hiển thị số **-1**.

Hãy cho biết số k và dãy số hiển thị trên màn hình nếu $k \neq -1$.

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

- ✚ Dòng đầu tiên chứa 3 số nguyên n, A và B ($1 \leq n \leq 10^5, 0 \leq A, B \leq 10^9$),
- ✚ 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 TAXI.OUT:

- Dòng đầu tiên chứa số nguyên k ,
- Nếu $k \neq -1$ – dòng thứ 2 chứa k số nguyên $a_{i1}, a_{i2}, \dots, a_{ik}$.

Ví dụ:

TAXI.INP	TAXI.OUT
4 2 11 3 2 6 10	2 1 4



Giải thuật; Xử lý bit .

Nhận xét:

- Với \mathbf{a} và \mathbf{x} là các số nguyên, phép $\mathbf{a} \mid= \mathbf{x}$ sẽ không làm giảm số bít 1 trong \mathbf{a} , hơn thế nữa các bít của \mathbf{a} ban đầu là 1 vẫn giữ nguyên giá trị 1 sau phép **or**.
- Gọi \mathbf{u}_i bít thứ i của \mathbf{a} , \mathbf{v}_i – bít thứ i của \mathbf{b} . Nếu tồn tại i sao cho $\mathbf{u}_i = 1$, $\mathbf{v}_i = 0$ – bài toán vô nghiệm (không có cách chuyển từ **A** tới **B**). Trường hợp này xảy ra khi $\mathbf{a} \& (\sim \mathbf{b}) \neq 0$.

Gọi vị trí hiện tại là \mathbf{x} (ban đầu $\mathbf{x} = \mathbf{A}$).

Giá trị \mathbf{a}_i có thể chọn để bấm khi thỏa mãn 2 điều kiện:

- $\mathbf{a}_i \& \mathbf{b} == 0$ – không làm xuất hiện bít 1 thừa, không thể xóa được,
- $\mathbf{a}_i \& (\sim \mathbf{x}) != 0$ – thêm bít 1 mới để tiến gần tới kết quả.

Như vậy, sau mỗi lần chọn được số để bấm ta làm số bít 1 trong \mathbf{x} tăng ít nhất một.

Nếu có cách chọn để đi từ **A** tới **B** – số lần bấm không vượt quá 30.

Tổ chức dữ liệu:

Mảng **vector<int>** **ans** – lưu các số được chọn.

Xử lý:

■ VỚI $i = 1 \div n$:

- Kiểm tra điều kiện cho phép chọn \mathbf{a}_i ,
- Nếu chọn được:
 - ❖ Ghi nhận \mathbf{a}_i và cập nhật vị trí,
 - ❖ Nếu đạt tới **B** – thoát khỏi chu trình

■ Kiểm tra đã tới được **B** hay không và đưa ra thông tin tương ứng.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

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

int main()
{
    int n, a, b;
    fi >> n >> a >> b;
    if(a& ~b)
    {
        fo << "-1\n";
        Times; return 0;
    }
    vector<int> ans;
    for (int i = 0; i < n; ++i)
    {
        int num;
        fi>> num;
        if ((num & ~b) == 0 && (num & ~a) != 0)
        {
            a |= num;
            ans.push_back(i);
            if(a==b) break;
        }
    }
    if (a != b)
    {
        fo << "-1\n";
        Times; return 0;
    }
    fo << ans.size() << "\n";
    for (int ind : ans) fo << ind + 1 << " ";
    Times; fo << "\n";
}
```

