

Mục lục Athena XIII

VX13. TUYỆT MẬT	Tên chương trình: TOP_SCT.CPP	3
VX14. ĐÁNH LUỒNG	Tên chương trình: DRILL.CPP	6
VX15. QUE NHỰA	Tên chương trình: STICKS.CPP	9
VX16. XÓA SỐ	Tên chương trình: ER_NUM.CPP	15
VX17. ĐỊNH LUẬT MORGAN	Tên chương trình: MORGAN.CPP	19
VX18. CUỐN SÁCH CỔ	Tên chương trình: OLDBOOK.CPP	21
VX19. MÁY PHUN SƯƠNG	Tên chương trình: PUFF.CPP	24
VX20. THANH TOÁN	Tên chương trình: PAY.CPP	28
VX21. TỔNG NHỎ NHẤT	Tên chương trình: MINSUM.CPP	31
VX22. ĐƯỜNG CHẠY	Tên chương trình: TRACK.CPP	34
VX24. QUAN SÁT	Tên chương trình: VISIBLE.CPP	40
VX25. TÂM LÝ	Tên chương trình: MENTALITY.CPP	43
VX26. ĐẤU VẬT	Tên chương trình: WRESTLING.CPP	46
VX27. VÔ HẠN	Tên chương trình: INFINITY.CPP	49
VX28. DNA	Tên chương trình: DNA.CPP	55
VX29. GIAO HÀNG	Tên chương trình: SHIFT.CPP	58
VX30. THI HỌC KỲ	Tên chương trình: EXAMINES.CPP	61
VX31. SỐ HÀNG	Tên chương trình: ADDENDUM.CPP	64
VX32. SỐ 0	Tên chương trình: ZEROS.CPP	66
VX33. LŨY THỪA NHỎ NHẤT	Tên chương trình: MINPOWER.CPP	69
VX34. ĐIỂM NGUYÊN	Tên chương trình: DOTS.CPP	72
VX35. SỐ CÁCH BỎ SUNG	Tên chương trình: INSNUM.CPP	74
VX36. SỐ LỚN NHẤT	Tên chương trình: MAXNUM.CPP	77
VX37. HIỆU ỨNG ĐÔ MI NỘ	Tên chương trình: DOMINO.CPP	79
VX38. KHÔNG CÓ 9	Tên chương trình: WITHOUT_9.CPP	81
VX39. NGÀY THÁNG	Tên chương trình: DATES.CPP	83
VX40. THOÁT HIỂM	Tên chương trình: EMERGENCY.CPP	86
VX41. NHÀ HÀNG	Tên chương trình: RESTAURANT.CPP	89
VX42. TỐT NHẤT	Tên chương trình: BEST.CPP	92
VX43. TIỂU THUYẾT	Tên chương trình: NOVEL.CPP	96
VX45. TẠO TEST	Tên chương trình: GEN_T.CPP	98
VX46. CỌC NHỒI	Tên chương trình: SPILES.CPP	102
VX47. CHUỖI HẠT	Tên chương trình: NECKLACE.CPP	105

VX48. TRÒ CHƠI NHI PHÂN	<i>Tên chương trình: BINGAME.CPP</i>	108
VX49. TRẠI HÈ	<i>Tên chương trình: SUM_ASS.CPP</i>	114
VX50. ĐÀN CỪU	<i>Tên chương trình: SHEEPS.CPP</i>	118
VY01. CHUỖI HẠT ĐẸP	<i>Tên chương trình: BONNY.CPP</i>	121
VY02. ĐÉ TƯƠNG ĐÀI	<i>Tên chương trình: PEDESTAL.CPP</i>	124
BA03. CHỦ ĐỀ	<i>Tên chương trình: THEMES.CPP</i>	127
BA05. QUẢNG CÁO	<i>Tên chương trình: BLURB.CPP</i>	130
BA06. NHÌ BẰNG	<i>Tên chương trình: MAX2.CPP</i>	133
BA07. ĐỘ TĨNH LÂNG	<i>Tên chương trình: SILENCE.CPP</i>	136
BA08. TÀI CHƯƠNG TRÌNH	<i>Tên chương trình: PROGRAMS.CPP</i>	139
BA09. HOÁN VI	<i>Tên chương trình: PERM.CPP</i>	143
VY03. MIỀN LIÊN THÔNG	<i>Tên chương trình: CONNECTED.CPP</i>	146
VY04. NGHỆ THUẬT IKEBANA	<i>Tên chương trình: IKEBANA.CPP</i>	150
VY05. CHUỖI ĐIỀU HÒA	<i>Tên chương trình: HARMONIC.CPP</i>	153
VY06. SỐ K BÍT 1	<i>Tên chương trình: K_BIN.CPP</i>	159
VY07. LAZER	<i>Tên chương trình: LAZER.CPP</i>	162
VY08. CẦU VỒNG	<i>Tên chương trình: RAINBOW.CPP</i>	166
VY09. BÁN VÉ TÀU NHANH	<i>Tên chương trình: TICKETS.CPP</i>	171
VY10. TỔNG OR	<i>Tên chương trình: OR_SUM.CPP</i>	176
BA11. LŨY THỬA CỦA 2	<i>Tên chương trình: POWER2.CPP</i>	179
VY11. KHÔNG DỪNG	<i>Tên chương trình: NON_STOP.CPP</i>	182
BA12. TRUNG BÌNH	<i>Tên chương trình: AVERAGE.CPP</i>	188
VY12. CÔNG NGHỆ GEN	<i>Tên chương trình: GENETECH.CPP</i>	190

Có những hồ sơ tuyệt mật chỉ có thể được công bố ít nhất sau 50 năm nếu trước đó Quốc hội không gia hạn thêm thời gian bảo mật.

Kho lưu trữ có n két sắt chống cháy đánh số từ 1 đến n , đặt dọc thành một hàng dài ở các vị trí cách đều nhau. Cửa sổ tiếp tài liệu nằm ở giữa két k và két $k+1$. Nếu $k = 0$ thì có nghĩa là cửa sổ nằm ở vị trí trước két 1, nếu $k = n - 1$ cửa sổ nằm ở vị trí sau két n . Mỗi tài liệu mật được lưu trong một hoặc một vài kẹp hồ sơ chuẩn. Két thứ i chứa được a_i kẹp hồ sơ chuẩn, $i = 1 \dots n$. Nếu không chứa hồ sơ, két ở trạng thái mở. Sau bỏ hồ sơ vào lưu trữ và đóng cửa két sẽ chuyển sang trạng thái đóng vĩnh viễn, chỉ có thể phá két mới hồ sơ ra được và tất nhiên két đó sẽ bị hỏng, không sử dụng tiếp được.

Khi có một hồ sơ gồm b kẹp đưa tới, giá trị b được chuyển cho robot xử lý. Robot sẽ tìm két trống có khả năng lưu trữ đúng b kẹp. Nếu không tồn tại két như vậy thì robot sẽ tìm két trống có $a_i > b$. Trong mọi trường hợp, nếu có nhiều két cùng có thể chọn thì két ở gần cửa sổ nhất. Khoảng cách được tính theo số két nằm giữa cửa sổ và két được chọn. Nếu có thể chọn 2 két cùng khoảng cách thì robot chọn két với số nhỏ hơn. Nếu tìm được két có thể lưu trữ thì robot phát tín hiệu nhận hồ sơ, tài liệu sẽ được băng chuyền đưa xuống để robot cho vào két. Trong trường hợp không tìm được két phù hợp, robot phát tín hiệu từ chối và tài liệu sẽ được chuyển sang băng chuyền thuộc kho lưu trữ khác.

Có m hồ sơ được chuyển tới, hồ sơ thứ j có b_j kẹp. Thông tin về mỗi hồ sơ (lưu trữ ở két nào hay bị chuyển sang kho khác với thứ tự két là -1) được tự động ghi lại và cũng chỉ có thể truy nhập khi hết thời hạn bảo mật.

Tuy vậy, đã là tự động hóa thì không có gì là bí mật. Dựa trên biên bản tiếp nhận hồ sơ người ta có thể xác định được hồ sơ thứ j được lưu trữ ở két nào hay bị chuyển sang kho khác.

Hãy đưa ra thông tin lưu trữ các hồ sơ đã nhận.

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

- ✚ Dòng đầu tiên chứa 3 số nguyên n , m và k ($1 \leq n, m \leq 1000$, $0 \leq k \leq n$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 1000$, $i = 1 \dots n$),
- ✚ Dòng thứ 3 chứa m số nguyên b_1, b_2, \dots, b_m ($1 \leq b_j \leq 1000$, $j = 1 \dots m$).

Kết quả: Đưa ra file văn bản TOP_SCT.OUT trên một dòng m số nguyên, số thứ j xác định két lưu trữ hồ sơ thứ j , giá trị này bằng -1 nếu hồ sơ bị chuyển sang kho khác, $j = 1 \dots m$.

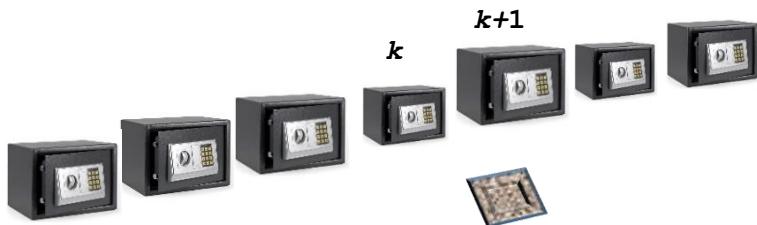
Ví dụ:

TOP_SCT.INP	TOP_SCT.OUT
<pre>4 5 2 1 3 2 2 1 1 3 2 1</pre>	<pre>1 2 -1 3 4</pre>



Giải thuật: Phân tích hoạt động ô tô mát.

Kích thước bài toán không lớn vì vậy không cần sử dụng các cấu trúc dữ liệu và giải thuật phức tạp.



Với mỗi giá trị b_j :

- ⊕ Tìm vị trí két thỏa mãn gần nhất bên trái cửa sổ chuyển tài liệu,
- ⊕ Tìm vị trí két thỏa mãn gần nhất bên phải cửa sổ chuyển tài liệu,
- ⊕ Chọn và đưa ra vị trí tối ưu trong 2 kết quả.

Trong quá trình tìm kiếm cũng như ghi nhận: Ưu tiên điều kiện bằng.

Nếu tìm thấy két lưu trữ là p : Cho $a_p=0 \rightarrow$ két p sẽ tự động bị loại trong quá trình tìm kiếm tiếp sau.

Tổ chức dữ liệu:

Mảng `vector<int> a(n)` – lưu dữ liệu vào a_i .

Xử lý:

Tìm vị trí thỏa mãn gần nhất bên trái (tương tự như vậy với phía bên phải):

```

int l = -1, r = -1; // Khởi tạo cho cả hai hướng tìm kiếm

for (int i = k; i >= 0; i--)
{
    if (a[i] == b && (l == -1 || a[l] != b))
        l = i;

    if (a[i] > b && l == -1)
        l = i;
}

```

Chưa gặp $a_i=b$

Ghi nhận kết quả khi có két lưu được:

```

res = l;
if (a[l] != b && a[r] == b) res = r;

if (((a[l]==b && a[r]==b) ||
      (a[l]!=b && a[r]!=b)) && k-l>r-(k+l))
    res = r;

```

*Cả hai bên cùng bằng
hoặc cùng khác b*

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

Chương trình

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

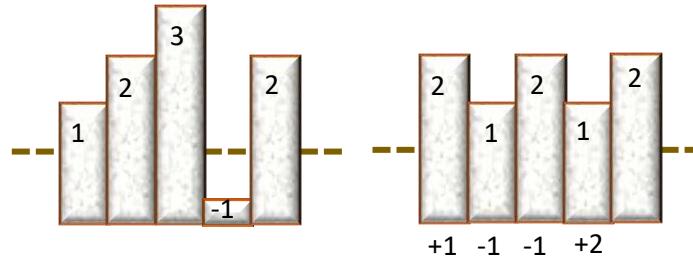
int main()
{
    int n, m, k;
    fi >> n >> m >> k;
    k--;
    vector<int> a(n);
    for (int i = 0; i < n; i++)
        fi >> a[i];
    for (int j = 0; j < m; j++)
    {
        int b;
        fi >> b;
        int l = -1, r = -1;
        for (int i = k; i >= 0; i--)
        {
            if (a[i] == b && (l == -1 || a[l] != b))
                l = i;
            if (a[i] > b && l == -1)
                l = i;
        }
        for (int i = k + 1; i < n; i++)
        {
            if (a[i] == b && (r == -1 || a[r] != b))
                r = i;
            if (a[i] > b && r == -1)
                r = i;
        }
        int res = -1;
        if (l == -1 && r != -1)
            res = r;
        else if (l != -1 && r == -1)
            res = l;
        else
        {
            res = l;
            if (a[l] != b && a[r] == b) res = r;
            if ((a[l]==b && a[r]==b) || (a[l]!=b && a[r]!=b)) && k-l>r-(k+1))
                res = r;
        }
        if (res != -1)
        {
            a[res] = 0;
            res++;
        }
        fo << res << ' ';
    }
    fo << "\nTime: " << clock() / (double)1000 << " sec";
    return 0;
}
```



VX14. ĐÁNH LUỐNG

Tên chương trình: DRILL.CPP

Mảnh vườn của Viện Nghiên cứu Giống và Cây trồng được đánh thành n luồng, luồng thứ i có độ cao a_i so với mốc tính, a_i có thể âm, $i = 1 \div n$, phù hợp cây trồng trên luồng. Nhiệm vụ sắp tới của Viện là cung cấp cây giống cho 2 loại cây có tác dụng hỗ trợ nhau khi trồng xen. Vì vậy người ta phải cải tạo lại cách đánh luồng để các luồng ở vị trí chẵn có cùng độ cao, các luồng ở vị trí lẻ có cùng độ cao và chênh lệch độ cao giữ 2 luồng liên tiếp là k .



Máy đánh luồng chạy dọc theo luồng và mỗi lần chạy có thể bóc đất bè mặt, giảm độ cao luồng 1 đơn vị hoặc đắp thêm đất để độ cao luồng tăng thêm 1.

Hãy xác định số lần vận hành máy ít nhất để có mảnh vườn với các luồng có độ cao thỏa mãn yêu cầu mới.

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

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

Kết quả: Đưa ra file văn bản DRILL.OUT một số nguyên – số lần vận hành máy ít nhất.

Ví dụ:

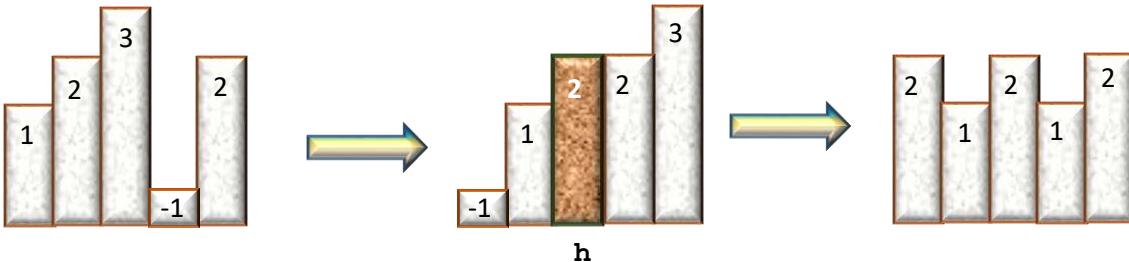
DRILL.INP	DRILL.OUT
5 1 1 2 3 -1 2	5



Giải thuật: Kỹ thuật lập trình.

Xét trường hợp $k = 0$ (Trường hợp san bằng).

Gọi độ cao của luồng sau khi san bằng là \mathbf{h} . Số lần san sẽ là $\sum_{i=1}^n |a_i - h|$. Tổng này nhận giá trị nhỏ nhất khi \mathbf{h} bằng giá trị a_i nằm ở điểm giữa trong dãy $\{\mathbf{a}_i\}$ đã sắp xếp tăng dần.



Xét trường hợp $k > 0$.

Có 2 khả năng:

- + Luồng 1 là luồng cao,
- + Luồng 1 là luồng thấp.

Xét trường hợp *luồng 1 là luồng cao*: Nếu ta lấy độ cao tất cả các luồng ở *vị trí lẻ* trừ đi \mathbf{k} thì bài toán cần giải tương đương với bài toán san bằng với các giá trị \mathbf{a}_i mới!

Xét trường hợp *luồng 1 là luồng thấp*: Nếu ta lấy độ cao tất cả các luồng ở *vị trí chẵn* trừ đi \mathbf{k} thì bài toán cần giải tương đương với bài toán san bằng với các giá trị \mathbf{a}_i mới! Có thể thay vì giảm độ cao các vị trí chẵn ta có thể *tăng k ở độ cao các luồng vị trí lẻ* (giữ nguyên độ cao các luồng ở vị trí chẵn), kết quả cần tìm không thay đổi!

Tổ chức dữ liệu:

- ─ Mảng **vector<int>** $\mathbf{a}(n)$ – lưu các độ cao \mathbf{a}_i ,
- ─ Các mảng **vector<int>** $\mathbf{tmp}, \mathbf{tmp2}$ – lưu \mathbf{a}_i mới phục vụ tìm kiếm \mathbf{h} .

Xử lý:

Không cần sắp xếp tường minh dãy các giá trị \mathbf{a}_i mới vì ta chỉ cần tìm phần tử đứng giữa trong dãy đã sắp xếp.

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

Chương trình

```
#include "bits/stdc++.h"
#define NAME "drill."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef int64_t ll;
const ll INFL = 1e18 + 123;

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

    ll best = INFL;
    for (int i = -1; i <= 1; i += 2)
    {
        vector<int> tmp = a;
        for (int j = 0; j < n; j += 2)
            tmp[j] += k * i;
        auto tmp2 = tmp;

        ll sum = 0;
        nth_element(tmp.begin(), tmp.begin() + tmp.size() / 2, tmp.end());
        int val = tmp[tmp.size() / 2];
        for (int num : tmp) sum += abs((ll) num - val);
        if (sum < best)
            best = sum;
    }

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



VX15. QUE NHỰA

Tên chương trình: STICKS.CPP

Jimmy tìm thấy trong đống đồ chơi của mình 4 que nhựa độ dài nguyên tương ứng là **a**, **b**, **c** và **d**. Trên mỗi que nhựa có các khía đánh dấu từng đoạn độ dài đơn vị, vì vậy có thể dễ dàng bẻ mỗi que thành nhiều que nhỏ độ dài nguyên.

Jimmy muốn có 4 que để làm khung ảnh hình chữ nhật, mỗi cạnh là một que và dĩ nhiên, hình chữ nhật phải có diện tích lớn nhất có thể.

Hãy xác định độ dài 2 cạnh (chiều dài và chiều rộng) của hình chữ nhật có diện tích lớn nhất.

Dữ liệu: Vào từ file văn bản STICKS.INP gồm một dòng chứa 4 số nguyên **a**, **b**, **c** và **d** ($1 \leq a, b, c, d \leq 10^{15}$).

Kết quả: Đưa ra file văn bản STICKS.OUT trên một dòng 2 số nguyên xác định kích thước của hình chữ nhật diện tích lớn nhất. Nếu có nhiều đáp án thì đưa lời giải tùy chọn.

Ví dụ:

STICKS.INP
1 8 6 19

STICKS.OUT
9 6



Giải thuật: Kỹ thuật bảng phương án.

Chuẩn hóa dữ liệu: đưa về cấu hình $\mathbf{a} \geq \mathbf{b} \geq \mathbf{c} \geq \mathbf{d}$.

Xét các tình huống tạo hình chữ nhật với diện tích là lớn nhất:

- ⊕ Mỗi cạnh hình chữ nhật từ một que,
- ⊕ Hai cạnh hình chữ nhật từ một que, hai cạnh kia – mỗi cạnh từ một que trong số còn lại,
- ⊕ Ba cạnh hình chữ nhật từ một que, cạnh thứ tư – từ một trong số các que còn lại,
- ⊕ Bốn cạnh hình chữ nhật – từ một que.

Que ngắn nhất sẽ không bị bẻ.

Sau một số lần bẻ, ta có các que với độ dài $\mathbf{s}\mathbf{t}_0, \mathbf{s}\mathbf{t}_1, \dots, \mathbf{s}\mathbf{t}_{k-1}$,

Sắp xếp $\mathbf{s}\mathbf{t}_i$ giảm dần, diện tích hình chữ nhật lớn nhất tạo được với bộ que này là $\mathbf{s}\mathbf{t}_1 \times \mathbf{s}\mathbf{t}_3$ (giá trị lớn thứ nhì và thứ tư),

Dễ dàng thấy rằng $k \leq 7$ (trường hợp bẻ que độ dài \mathbf{a} thành 4 phần). Trên thực tế, để tìm số lớn thứ nhì và thứ tư ta không cần giữ hết các giá trị giống nhau, vì vậy $k < 7$.

Số lượng trường hợp cần xét là không lớn, vì vậy mỗi lần bẻ que (hoặc các que), để tìm giá trị lớn thứ nhì và thứ tư đơn giản nhất là sắp xếp lại bộ giá trị nhận được.

Duyệt tất cả các trường hợp bẻ cần xét, so sánh diện tích nhận được và lưu kết quả tối ưu.

Cần thành lập bảng ghi nhận cách bẻ cần xét để tránh bỏ sót hoặc trùng lặp.

Lưu ý các trường hợp $\mathbf{a} = \mathbf{b}$ hoặc $\mathbf{a} = \mathbf{b} = \mathbf{c}$. Bảng ghi nhận các khả năng cần bẻ phải bao quát các trường hợp này nhưng *không cần kiểm tra* nhận dạng. Việc kiểm tra sẽ làm *độ phức tạp lập trình tăng* còn *độ phức tạp giải thuật giảm không đáng kể*.

Diện tích hình chữ nhật có thể rất lớn (bậc 10^{30}) vì vậy cần tổ chức thực hiện phép *nhân số lớn*.

Khi bẻ một que thành nhiều đoạn, các đoạn con phải bằng nhau hoặc gần bằng nhau (trường hợp độ dài không chia hết cho số cách bẻ), ví dụ, với $\mathbf{a} = 11$, khi bẻ thành 4 đoạn ta có các độ dài 2, 3, 3 và 3.

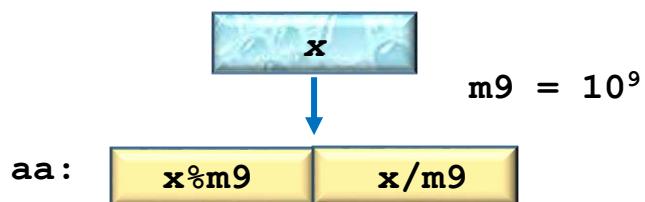
Các trường hợp cần xét:

$i \setminus st_j$	0	1	2	3	4	5
1	d	c	b	a		
2	d	c	$b/2$	$a/2$	$a-a/2$	$b-b/2$
3	d	$c/2$	b	$a/2$	$a-a/2$	$c-c/2$
4	d	c	b	$a-b$	b	
5	d	c	b	$a-c$	c	
6	d	c	b	$a/4$	$(a-2 \times (a/4))/2$	$a-st_3-st_4$
7	d	c	b	$(a-b)/2$	$a-st_3$	
8	d	c	b	$b-1$	$(a-b+1)/2$	$a-st_3-st_4$
9	d	c	b	b	$(a-b)/2$	$a-st_3-st_4$
10	d	c	$b/2$	a	$b-b/2$	
11	d	$c/2$	b	a	$c-c/2$	
12	d	c	b	$a/2$	$(a-a/2)/2$	$a-st_3-st_4$
13	d	c	b	$a/2+1$	$(a-a/2-1)/2$	$a-st_3-st_4$
14	d	c	b	$a/2$	$a-st_3$	

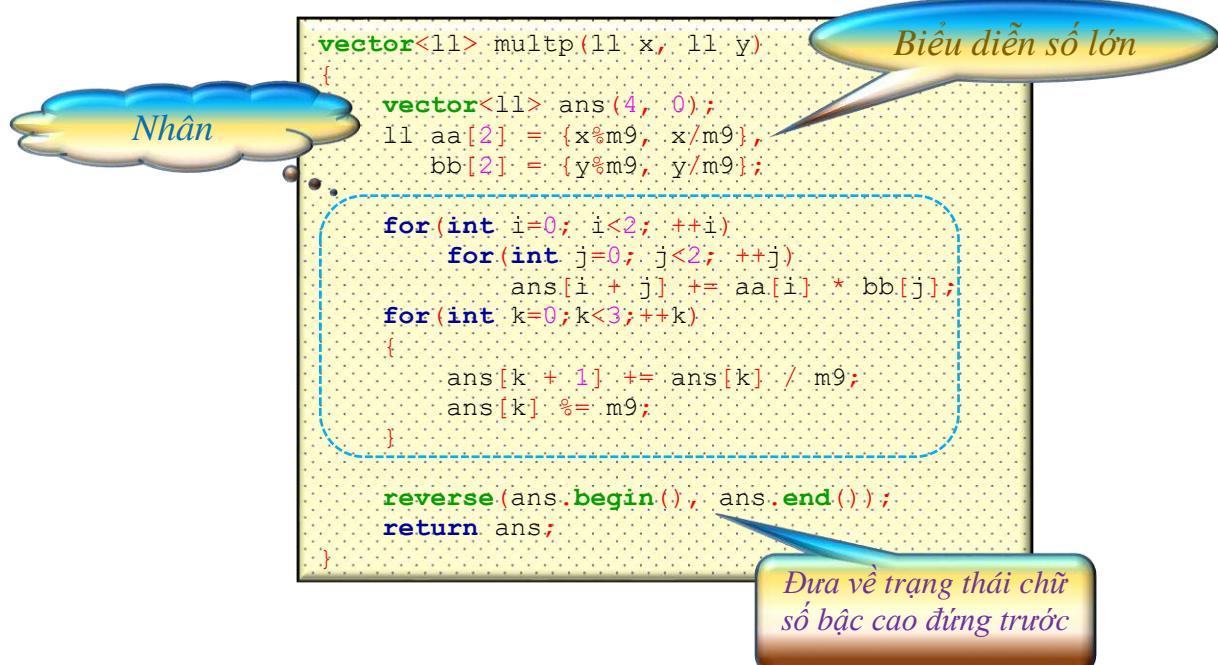
Tổ chức dữ liệu:

- ─ Mảng `int64_t st[8]` – lưu độ dài các đoạn cần xét,
- ─ Mảng `vector<int64_t> area(4, 0LL)` – lưu diện tích lớn nhất của hình chữ nhật có thể tạo.

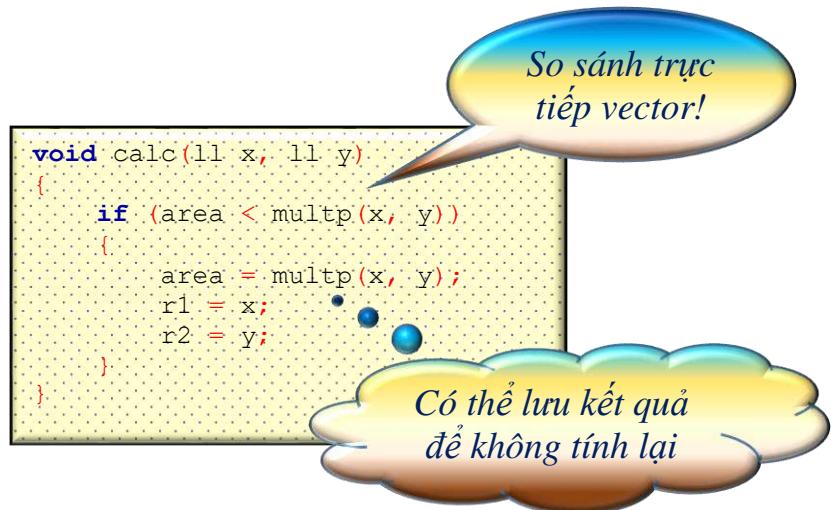
Xử lý:



Nhân số lớn:



Chọn kết quả tối ưu:



Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "sticks."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef int64_t ll;
const ll m9=(ll)1e9;
ll r1,r2,st[8],a,b,c,d,e;
vector<ll> area(4,0LL);

vector<ll> multp(ll x, ll y)
{
    vector<ll> ans(4, 0);
    ll aa[2] = {x%m9, x/m9},
        bb[2] = {y%m9, y/m9};
    for(int i=0; i<2; ++i)
        for(int j=0; j<2; ++j)
            ans[i + j] += aa[i] * bb[j];
    for(int k=0; k<3; ++k)
    {
        ans[k + 1] += ans[k] / m9;
        ans[k] %= m9;
    }
    reverse(ans.begin(), ans.end());
    return ans;
}

void calc(ll x, ll y)
{
    if (area < multp(x, y))
    {
        area = multp(x, y);
        r1 = x;
        r2 = y;
    }
}

void init()
{
    st[0]=d, st[1]=c; st[2]=b; st[3]=a;
}

int main()
{
    for(int i=0;i<4;++i)fi>>st[i];
    sort(st,st+4); a=st[3]; b=st[2]; c=st[1]; d=st[0]; calc(b, d);

    init();st[3]/=2; st[4]=a-st[3]; st[1]/=2; st[5]=c-st[1];
    sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
    init();st[3]/=2; st[4]=a-st[3]; st[2]/=2; st[5]=b-st[2];
    sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
    init(); st[3]=a-b; st[4]=b;
    sort(st,st+5,greater<ll>()); calc(st[1],st[3]);
    init(); st[3]=a-c; st[4]=c;
    sort(st,st+5,greater<ll>()); calc(st[1],st[3]);
    init(); st[3]=a/4; st[4]=(a - 2*(a/4))/2; st[5]=a-st[3]-st[4];
    sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
    init(); st[3]=(a-b)/2; st[4]=a-st[3];
    sort(st,st+5,greater<ll>()); calc(st[1],st[3]);
    init();st[3]=b-1; st[4]=(a-b+1)/2; st[5]=a-st[3]-st[4];
```

```

        sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
init(); st[3]=b; st[4]=(a-b)/2; st[5]=a-st[3]-st[4];
        sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
init(); st[2]/=2; st[4]=b-st[2];
        sort(st,st+5,greater<ll>()); calc(st[1],st[3]);
init(); st[1]/=2; st[4]=c-st[1];
        sort(st,st+5,greater<ll>()); calc(st[1],st[3]);
init(); st[3]/=2; st[4]=(a-a/2)/2; st[5]= a-st[3]-st[4];
        sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
init(); st[3]=a/2+1; st[4]=(a-a/2-1)/2; st[5]= a-st[3]-st[4];
        sort(st,st+6,greater<ll>()); calc(st[1],st[3]);
init(); st[3]=a/2; st[4]=a-st[3];
        sort(st,st+5,greater<ll>()); calc(st[1],st[3]);

fo << r1 << ' ' << r2 ;
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```



Bob và Alice cùng nhau làm bài tập Tin học. Bài tập được giải quyết rất nhanh. Còn thừa thời gian, hai bạn quyết định thư giãn một chút. Sẵn có dãy số nguyên dương a_1, a_2, \dots, a_n hai bạn thi nhau trong trò chơi xóa số. Đầu tiên 2 người lấy ra một tờ giấy, chép vào đó p số đầu tiên trong dãy, tức là các số a_1, a_2, \dots, a_p và xóa các số đó khỏi dãy số ban đầu. Hai người lần lượt đi với tổng điểm của mỗi người ban đầu là 0. Alice đi trước.

Mỗi người, khi đến lượt mình, sẽ làm những việc sau:

- Chọn một số trên giấy cộng nó vào số điểm của mình và xóa số đó,
- Ghi số đầu tiên còn lại trong dãy ban đầu vào giấy và xóa số đó khỏi dãy ban đầu. Nếu dãy còn lại ban đầu là rỗng thì bỏ qua bước này.

Trò chơi kết thúc khi tất cả các số trên giấy bị xóa hết. Mỗi người đều cố gắng đạt điểm cao nhất có thể.

Các bạn chơi k ván, ván thứ j được thực hiện với p_j số đầu tiên được chép vào giấy.

Ở mỗi ván hãy xác định hiệu điểm số của Alice trừ cho điểm số của Bob.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và k ($1 \leq n \leq 10^5$, $1 \leq k \leq 2000$, $k \leq n$),
- ✚ 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 k số nguyên p_1, p_2, \dots, p_k ($1 \leq p_j \leq n$, $j = 1 \div k$).

Kết quả: Đưa ra file văn bản ER_NUM.OUT k số nguyên, mỗi số trên một dòng – hiệu tìm được. Số thứ j tương ứng với kết quả ván j , $j = 1 \div k$.

Ví dụ:

ER_NUM.INP	ER_NUM.OUT
<pre>5 2 2 4 2 3 5 4 3</pre>	<pre>2 6</pre>



Giải thuật: Phân tích hoạt động ô tô mát, Xử lý phòng đệm.

Tờ giấy ghi các số đóng vai trò **phòng đệm** xử lý.

Xét trò chơi với một **p** cố định (Xử lý với chế độ *Truy vấn online*):

Các số a_p, a_{p+1}, \dots, a_n lần lượt được đưa vào phòng đệm,

Có thể tổ chức phòng đệm dưới dạng một **priority_queue<int> q**,

Ở mỗi bước người đến lượt đi sẽ lấy số từ đầu phòng đệm, tích lũy vào tổng của mình, loại bỏ số được chọn khỏi phòng đệm, đưa số tiếp theo còn lại (nếu có) của dãy số ban đầu vào phòng đệm.

Trò chơi kết thúc khi phòng đệm rỗng, từ các tổng đã tích lũy – dẫn xuất kết quả.

```
#include <bits/stdc++.h>
#define NAME "er_num."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,m,p,t,flg=1;
int64_t s=0,sa=0;
priority_queue<int> q;

int main()
{
    fi>>n>>p;
    for(int i=0; i<p; ++i)
    {
        fi>>t; s+=t;
        q.push(t);
    }
    for(int i=p; i<n; ++i)
    {
        fi>>t; s+=t;
        if(flg)sa+=q.top();
        q.pop(); flg^=1; q.push(t);
    }
    while (!q.empty())
    {
        if(flg)sa+=q.top();
        q.pop(); flg^=1;
    }
    fo<<sa*2-s;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```

Dưa p số đầu tiên của dãy vào phòng đệm

Xử lý các số còn lại của dãy

Thay cho tổng của người thứ II

Xử lý kết thúc

Có thể áp dụng sơ đồ xử lý trên cho từng giá trị **p** cần xét (loại bỏ việc tính lặp giá trị $\sum_{i=1}^n a_i$).

Độ phức tạp của giải thuật: O(nlogn×k).

Xử lý ở chế độ các giá trị **pj** biết trước (*Chế độ offline*):

Xử lý mỗi giá trị a_i với đồng thời tất cả các trường hợp kích thước khác nhau của mỗi phòng đệm.

Sơ đồ xử lý mỗi giá trị a_i với mọi phòng đệm chỉ khác ở giá trị kích thước p_j của phòng đệm,

Để đơn giản, ký hiệu \mathbf{p} đại diện cho các p_j cần xét.

Các chỉ số được bắt đầu từ 0.

Số a_i ($i = p \div n$) được lần lượt nạp vào phòng đệm. Phân loại theo chẵn lẻ, nếu i và p cùng loại thì a_i được *Alice đưa vào* phòng đệm và *Bob có quyền ưu tiên chọn*, trong trường hợp ngược lại – a_i do *Bob đưa vào* phòng đệm và *Alice có quyền ưu tiên chọn*.

Sau khi một giá trị được chọn, trong phòng đệm luôn chứa các giá trị nhỏ nhất của dãy số trong phần đã duyệt,

Sắp xếp cặp giá trị (a_i, i) theo trình tự không tăng,

Ký hiệu **f1g** – dấu hiệu đến lượt ai được quyền ưu tiên chọn số trong phòng đệm (**f1g** = 1 – quyền ưu tiên của Alice, **f1g** = 0 – quyền ưu tiên của Bob),

Sau khi chọn phần tử trong phòng đệm – quyền ưu tiên được trao cho người khác.

Phân biệt 2 trường hợp:

- + a_i trong phòng đệm ($i < p$ hoặc $p-1$ phần tử cuối cùng của dãy đã sắp xếp) – tính tổng tích lũy theo quyền ưu tiên,
- + a_i được chọn ngay khi vừa mới đưa vào phòng đệm: dựa vào phân loại chẵn lẻ của chỉ số i để tích lũy tổng.

Tổ chức dữ liệu:

- ─ Mảng `vector<pii>` $a(n)$ – lưu cặp giá trị (a_i, i),
- ─ Mảng `vector<int>` $p(m)$ – lưu các kích thước phòng đệm,
- ─ Mảng `vector<int>` $f1g(m, 1)$ – lưu thứ tự ưu tiên chọn giá trị trong phòng đệm, giá trị khởi tạo 1 để đảm bảo Alice sẽ chọn đầu tiên,
- ─ Mảng `vector<int64_t>` $sa(m, 0)$ – lưu tổng tích lũy của Alice.

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

Nhận xét: Giải thuật trên cho phép dễ dàng xử lý các điều kiện bổ sung với các truy vấn hoặc luật chơi.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "er_num."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
int n,m,p,t,flg;
int64_t s=0,sa;
```

Nhập dữ liệu và chuẩn bị

```
int main()
{
    fi>>n>>m;
    vector<pii>a(n);
    for(int i=0; i<n; ++i) {fi>>t; a[i]={t,i}; s+=t;}
    vector<int>p(m), flg(m, 1);
    vector<int64_t> sa(m, 0);
    for(int j=0; j<m; ++j) fi>>p[j];
    sort(a.rbegin(), a.rend());
```

Sắp xếp giảm dần

```
for(int i=0; i<n; ++i)
{
    for(int j = 0; j <m; ++j)
    {
        if(n-i+1==p[j]) flg[j]=1; // Chuyển sang trường hợp chỉ còn số trong phòng
đêm
```

if(n-i+1<=p[j])
{ if(flg[j]) sa[j]+=a[i].first; flg[j]^=1;
continue;
}

```
if(a[i].second<p[j])
{
    if(flg[j]) sa[j]+=a[i].first; flg[j]^=1;
    continue;
}
```

//Xử lý phần còn lại của phòng đêm
//trường hợp vẫn
//còn số đưa vào

```
if(((a[i].second-p[j])&1)==1) sa[j]+=a[i].first;
```

```
}
```

```
for(int j=0; j<m; ++j) fo<<sa[j]*2-s<<' \n';
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
```

Số được chọn ngay khi mới đưa vào phòng đêm



Trong cuộc sống, bên cạnh những định luật nghiêm túc như các định luật I, II và III của Newton còn có những định luật vui nhung phản ánh khá chân thật bức tranh của cuộc sống. Một trong số đó là định luật Morgan: “*Một định ốc bị rơi bao giờ cũng lăn vào chỗ có thể gây tác hại lớn nhất!*”.

Hôm nay Alice được trải nghiệm hiệu ứng của định luật này. Cô phải gửi gấp báo cáo của mình lên Hội đồng khoa học trước khi ra sân bay đi dự hội nghị. Để làm được việc đó cần tải báo cáo lên Google.com/Drive. Google Drive của Alice đang ở trạng thái rỗng. Việc đầu tiên là tải báo cáo lên đó. Nhưng khi copy, phím chuột trái bị dính. Khi Alice nhận ra điều này và ấn được phím gõ trạng thái dính thì trong Drive đã có **n** bản sao! Vẫn đề bây giờ là phải xóa hết các bản sao thừa, chỉ để lại một.

Việc xóa một file đã tải lên mạng không đơn giản như xóa file trong máy, đặc biệt khi yêu cầu đã được tiếp nhận, nhưng file chưa được tải hoặc đang tải.

Nhưng vận may đã không bỏ rơi Alice. Chắc ai đó cũng gặp trường hợp dính phím như cô và đã xây dựng một chương trình xóa files, rút số files trong thư mục xuống **k** lần, có điều chương trình hoạt động khi và chỉ khi số files trong thư mục chia hết cho **k**.

Để trực tiếp xóa một file trên drive cần **a** giây, còn một lần chạy chương trình đã nêu – cần **b** giây.

Hãy xác định khoảng thời gian nhỏ nhất cần thiết để giữ lại trên drive đúng một file.

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

- ✚ Dòng đầu tiên chứa số nguyên **n** ($1 \leq n \leq 2 \times 10^9$),
- ✚ Dòng thứ 2 chứa số nguyên **k** ($1 \leq k \leq 2 \times 10^9$),
- ✚ Dòng thứ 3 chứa số nguyên **a** ($1 \leq a \leq 2 \times 10^9$),
- ✚ Dòng thứ 4 chứa số nguyên **b** ($1 \leq b \leq 2 \times 10^9$).

Kết quả: Đưa ra file văn bản MORGAN.OUT một số nguyên – khoảng thời gian nhỏ nhất tìm được.

Ví dụ:

MORGAN.INP	MORGAN.OUT
19	
3	
4	
2	12



VX15 Mos 8_9 20180224 A AXIII

Giải thuật: Phân tích mô hình toán học.

Với $k = 1$ việc chạy chương trình xóa là vô nghĩa, chỉ có thể lần lượt xóa từng file và thời gian cần thiết sẽ là $(n-1) \times a$.

Với $k > 1$: ở mỗi thời điểm có một trong 3 tình huống:

- $n < k \rightarrow$ xóa trực tiếp từng file với chi phí thời gian là $t = (n-1) \times a$,
- $n > k$ và n không chia hết cho k : Cần đưa về trường hợp n chia hết cho k với chi phí thời gian là $n \% k \times a$, n mới sẽ là $n - n \% k$,
- $n \geq k$ và chia hết cho k .

Xét chi phí thời gian đưa drive từ trạng thái chứa n files về trạng thái chứa n/k files:

- Có thể chạy chương trình một lần với chi phí thời gian b ,
- Xóa lần lượt từng file với chi phí $(n - n/k) \times a$.

Chi phí lựa chọn là \min của hai giá trị trên.

Các bước xử lý trên được thực hiện cho đến khi có $n = 1$.

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

Chương trình

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

int main()
{
    fi>>n>>k>>a>>b;
    if(k==1) k=n+1;
    while(n>1)
    {
        if(n<k) {ans+=(n-1)*a; break;}
        if(n%k) ans+=a*(n%k); n-=n%k;
        t=min(b, (n-n/k)*a); n/=k; ans+=t;
    }
    fo<<ans;
}
```



Ở các cửa hàng sách cũ người ta có thể tìm được những cuốn sách hết sức độc đáo. Bob đã mua được một cuốn Bách khoa toàn thư cổ, xuất bản từ đầu thế kỷ trước. Bob mua nó vì có rất nhiều hình ảnh minh họa chi tiết và đẹp. Ngôn ngữ trong cuốn sách, tuy dùng bảng chữ cái la tinh, nhưng hoàn toàn xa lạ với Bob. Tuy vậy, Bob hy vọng, với sự trợ giúp của Internet, có thể hiểu được nhiều thứ mà mình quan tâm.

Một bài viết thu hút sự chú ý của Bob, nhưng đáng tiếc đầu đề đã bị phai mờ nhiều chỗ, chỉ xác định được là có độ dài **k**. Biết sao được, chất lượng mực in thế kỷ trước không trụ được trước sự thách thức của thời gian!

Theo quy luật quan sát được, Bob biết đầu đề này có thứ tự từ điền lớn hơn đầu đề mục trước đó còn đọc được là xâu **s** độ dài **n** và có khả năng vẫn chỉ chứa các ký tự dùng trong tiêu đề trước.

Bob quyết định tra trên mạng các từ có thể, bắt đầu từ tiêu đề độ dài **k** có thứ tự từ điền nhỏ nhất lớn hơn so với **s** và chỉ chứa các ký tự xuất hiện trong **s**.

Hãy xác định xâu mà Bob bắt đầu tra cứu.

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

- ✚ Dòng đầu tiên chứa một số nguyên **n** ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa xâu **s** độ dài **n**, chỉ chứa các ký tự la tinh thường,
- ✚ Dòng thứ 3 chứa số nguyên **k** ($1 \leq k \leq 10^5$).

Dữ liệu đảm bảo bài toán có lời giải.

Kết quả: Đưa ra file văn bản OLDBOOK.OUT xâu tìm được.

Ví dụ:

OLDBOOK.INP	OLDBOOK.OUT
3 ayy 3	yaa



Giải thuật: Xử lý xâu..

Nếu $n < k$ thì chỉ cần kéo dài xâu s cho đến khi có độ dài n bằng ký tự có thứ tự từ điển nhỏ nhất trong số các ký tự đã xuất hiện.

Trường hợp $n \geq k$:

Xét xâu r tiền tố độ dài k của s ,

Duyệt các ký tự của r từ cuối về đầu, xóa các ký tự cuối xâu bằng ký tự có thứ tự từ điển lớn nhất trong s ,

Theo điều kiện có nghiệm xâu r sẽ không bị rỗng sau quá trình xóa,

Kéo dài r bằng ký tự có thứ tự từ điển tiếp ngay sau ký tự cuối của r ,

Xâu kết quả nhận được bằng cách bổ sung vào cuối r ký tự có thứ tự từ điển nhỏ nhất cho đến khi có xâu độ dài k .

Để xác định các ký tự có thứ tự từ điển nhỏ nhất, lớn nhất, có thứ tự từ điển ngay sau một ký tự khác có thể dùng bảng đánh dấu các ký tự xuất hiện trong s .

Tổ chức dữ liệu:

Mảng **bool** $b[26] = \{0\}$ – đánh dấu các ký tự xuất hiện trong s ,

Các biến **mn** – xác định ký tự có thứ tự từ điển nhỏ nhất, **mx** – xác định ký tự có thứ tự từ điển lớn nhất xuất hiện trong s .

Độ phức tạp của giải thuật: $O(n+k)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "oldbook."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,k,mn,mx,p,q;
string s,ans;
bool b[26]={0};
int main()
{
    fi>>n>>s>>k;
    for(int i=0;i<n;++i)b[s[i]-97]=true;
    for(int i=0;i<26;++i) if(b[i]) {mn=i; break;}
    for(int i=25;i>=0;--i) if(b[i]) {mx=i; break;}

    if(n<k)
    {
        ans=s;
        for(int i=0;i<k-n;++i) ans+=char(97+mn);
        fo<<ans; return 0;
    }

    p=k-1;
    while(p>=0 && s[p]-97==mx)--p;
    for(int i=s[p]-96;i<26;++i) if(b[i]) {q=i; break;}

    ans=s.substr(0,p)+char(q+97);
    for(int i=p+1;i<k;++i) ans+=char(97+mn);
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VX19. MÁY PHUN SƯƠNG

Tên chương trình: PUFF.CPP

Các bạn ở câu lạc bộ Tuổi trẻ thời đại 4.0 đã chế tạo thành công một máy phun sương duy trì độ ẩm thích hợp cho nhà kính trồng ươm cây giống của một loại chè mới lai tạo.

Máy có bộ cảm biến đo tự động độ ẩm trong nhà và xác định chênh lệch với độ ẩm tối ưu đã xác lập trước. Cứ đầu mỗi giờ bộ điều khiển tham chiếu tới chênh lệch độ ẩm đo được. Nếu k lần liên tiếp các giá trị nhận được đều nhỏ hơn h_{min} thì thiết bị phun sương sẽ được bật, hoạt động suốt cả giờ đó và các giờ tiếp theo cho đến khi nhận được tín hiệu tắt. Nếu k lần liên tiếp các giá trị nhận được đều lớn hơn h_{max} thì thiết bị phun sương sẽ được tắt trong suốt giờ đó và các giờ tiếp theo cho đến khi nhận được tín hiệu bật. Trạng thái từng giờ (bật/tắt) của máy được ghi lại dưới dạng chuỗi các số 1 và 0, trong ứng với trạng thái bật, 0 – trạng thái tắt. Độ lệch của độ ẩm đo ở mỗi giờ dưới dạng các số nguyên cũng được tự động ghi lại để kiểm tra Ban đầu, lúc đưa vào hoạt động thử nghiệm máy ở trạng thái tắt.

Máy được thử nghiệm và hoạt động hoàn hảo trong suốt n giờ liên tục. Bỗng nhiên toàn khu vực bị mất điện! Đến khi có điện lại mọi người tái khởi động máy, bắt đầu từ việc nạp các tham số k , h_{min} và h_{max} . Thật không may, bạn giữ tham số lại đang đi họp hội nghị Cán bộ Đoàn và không liên lạc được. Giá trị k thì mọi người đều nhớ vì đã thảo luận nhiều về nó. Với hai tham số còn lại thì chịu, vất óc cũng không ra! Lối thoát, có lẽ là duy nhất trong hoàn cảnh này: tìm lại h_{min} , h_{max} từ dãy chênh lệch độ ẩm a_1, a_2, \dots, a_n và biên bản trạng thái s của máy.

Hãy xác định các giá trị h_{min} , h_{max} phù hợp với hoạt động của máy trong n giờ vừa qua và $h_{max} - h_{min}$ là nhỏ nhất, biết rằng các số này đều nguyên và có giá trị tuyệt đối không vượt quá 10^9 .

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và k , ghi cách nhau một dấu cách ($1 \leq n \leq 10^5$, $1 \leq k \leq 10$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($|a_i| \leq 10^9$, $i = 1 \div n$), các số ghi cách nhau một dấu cách,
- ✚ Dòng thứ 3 chứa xâu s độ dài n , mỗi ký tự thuộc tập {0, 1}.

Dữ liệu đảm bảo bài toán có lời giải.

Kết quả: Đưa ra file văn bản PUFF.OUT trên một dòng 2 số nguyên cách nhau bởi dấu cách h_{min} và h_{max} ($h_{min} \leq h_{max}$). Nếu có nhiều cặp giá trị thích hợp thì đưa ra cặp giá trị tương ứng với h_{min} nhỏ nhất

Ví dụ:

PUFF.INP	PUFF.OUT
11 5 2 1 -1 -1 1 1 6 7 8 8 6 00000111111	2 6



Giải thuật: Nguyên lý cực trị, Kỹ thuật lập trình.

Các khoảng thời gian liên tục đủ dài, máy không thay đổi trạng thái hoặc chỉ thay đổi trạng thái ở thời điểm cuối của khoảng, sẽ cung cấp thông tin giá trị điều khiển trạng thái,

Do việc bật/tắt máy chỉ phụ thuộc vào chênh lệch độ âm hiện tại và **k-1** chênh lệch độ âm trước đó nên *khoảng thời gian đủ dài cần xét là k*.

Đoạn thời gian liên tục từ **u** đến **u+k-1** máy *không bật* cho thấy có ít nhất một giá trị **a_i** trong đoạn này không nhỏ hơn **h_{min}**, ta có

$$h_{\min} \leq \max_{u \leq i \leq u+k-1} a_i \quad (1)$$

Nếu máy đang ở trạng thái tắt và ở thời điểm cuối của khoảng này – chuyển sang trạng thái bật cho thấy trong khoảng phải các giá trị **a_i** nhỏ hơn **h_{min}**:

$$h_{\min} \geq \max_{u \leq i \leq u+k-1} a_i + 1 \quad (2)$$

Đoạn thời gian liên tục từ **u** đến **u+k-1** máy *không tắt* cho thấy có ít nhất một giá trị **a_i** trong đoạn này không lớn hơn **h_{max}**:

$$h_{\max} \geq \min_{u \leq i \leq u+k-1} a_i \quad (3)$$

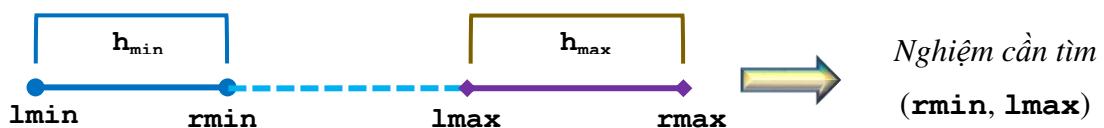
Nếu máy đang ở trạng thái bật và ở thời điểm cuối của khoảng này – chuyển sang trạng thái tắt cho thấy trong khoảng phải các giá trị **a_i** nhỏ hơn **h_{max}**:

$$h_{\min} \geq \min_{u \leq i \leq u+k-1} a_i - 1 \quad (4)$$

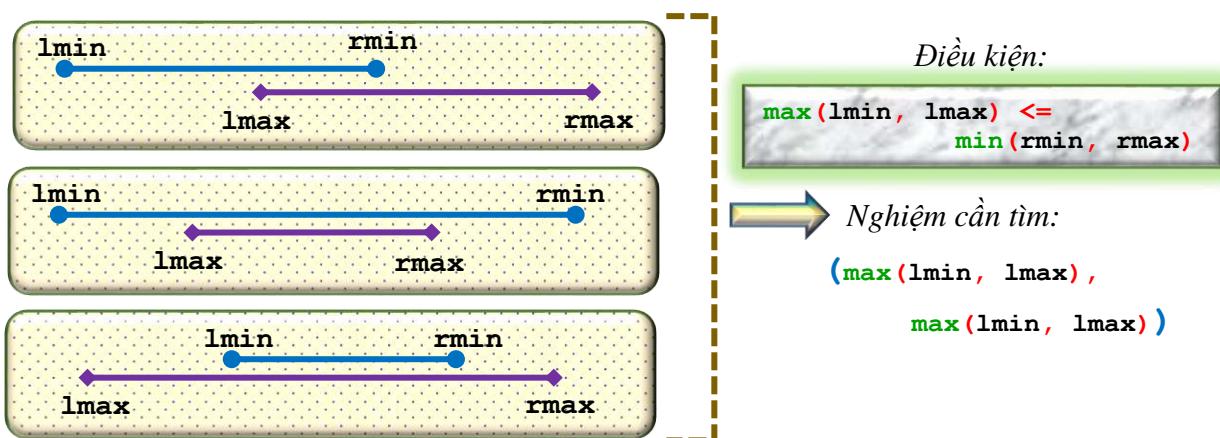
Từ (1) và (2) xác định được khoảng **[lmin, rmin]** chứa **h_{min}**.

Từ (3) và (4) xác định được khoảng **[lmax, rmax]** chứa **h_{max}**.

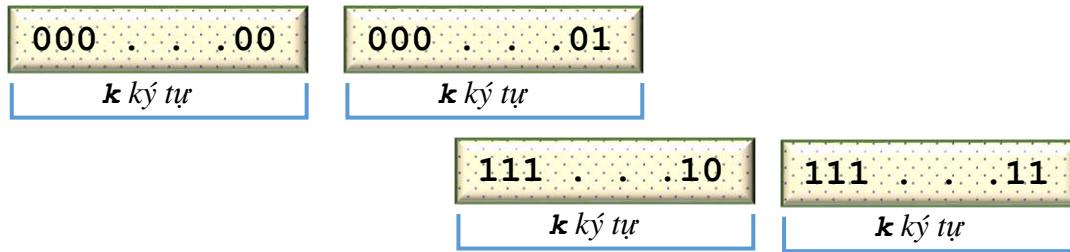
Hai khoảng này có thể không giao nhau:



Hoặc có thể giao nhau theo một trong các情形 sau:



Để nhận dạng nhanh tình huống xử lý khoảng, cần các mẫu nhận dạng:



Tổ chức dữ liệu:

Mảng **vector** `<int>` `a(n)` – lưu các chênh lệch độ âm cho trong input,

Xâu **string** `s` – trạng thái của máy,

Cặp giá trị **int** `lmin=-inf, rmin=inf` – lưu miền xác định của `h_min`,

Cặp giá trị **int** `lmax=-inf, rmax=inf` – lưu miền xác định của `h_max`.

Xử lý:

Lần lượt tách các đoạn độ dài **k** ký tự liên tiếp nhau của `s` bắt đầu từ vị trí `i` (`i = 0 ÷ n-k+1`, so sánh với mẫu nhận dạng và tiến hành cập nhật các miền xác định `h_min, h_max` tương ứng với kết quả nhận dạng:

```
st = string(s.begin() + i, s.begin() + i + k);
if(st==s01) lmin=max(lmin, *max_element(a.begin() + i, a.begin() + i + k) + 1);
if(st==s00) rmin=min(rmin, *max_element(a.begin() + i, a.begin() + i + k));
if(st==s10) rmax=min(rmax, *min_element(a.begin() + i, a.begin() + i + k) - 1);
if(st==s11) lmax=max(lmax, *min_element(a.begin() + i, a.begin() + i + k));
```

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

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

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

    string s,s01,s10,st,s00(k, '0'),s11(k, '1');
    s01=s00; s01[k-1]='1'; s10=s11; s10[k-1]='0';
    fi >> s;

    int lmin=-inf, rmin=inf, lmax=-inf, rmax=inf;
    for (int i=0; i<n-k+1; i++)
    {
        st = string(s.begin() + i, s.begin() + i+k);
        if(st==s01) lmin=max(lmin,*max_element(a.begin() + i, a.begin() + i+k) + 1);
        if(st==s00) rmin=min(rmin,*max_element(a.begin() + i, a.begin() + i+k));

        if(st==s10) rmax=min(rmax,*min_element(a.begin() + i, a.begin() + i+k) - 1);
        if (st==s11) lmax=max(lmax, *min_element(a.begin() + i, a.begin() + i+k));
    }

    if (max(lmin, lmax) <= min(rmin, rmax))
        fo<<max(lmin, lmax)<< ' ' <<max(lmin, lmax)<< '\n';
    else
        fo<<rmin<< ' '<<lmax<< '\n';
}

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



Siêu thị bán hàng qua mạng Pacific có rất nhiều chi nhánh phục vụ và có nhiều phương tiện giao hàng tiên tiến nhanh chóng bằng thiết bị bay tự động (drones), ngoài ra Siêu thị thường có các chương trình khuyến mãi hấp dẫn vì vậy thu hút được một số lượng lớn khách hàng.

Alice vào trang WEB của cửa hàng, chọn được n món hàng đưa vào danh sách các thứ sẽ mua, món hàng thứ i có giá a_i , $i = 1 \div n$. Khi Alice chuẩn bị chuyển danh sách hàng đã chọn vào giỏ mua thì xuất hiện thông báo về một chương trình khuyến mãi mới. Nếu giỏ hàng mua của khách có từ 10 đến 19 mặt hàng thì món hàng giá thấp nhất trong số đó sẽ được nhận miễn phí, nếu giỏ hàng mua có từ 20 đến 29 mặt hàng thì hai món hàng giá thấp nhất trong số đó sẽ được nhận miễn phí, nếu giỏ hàng mua có từ 30 đến 39 mặt hàng thì ba món hàng giá thấp nhất trong số đó sẽ được nhận miễn phí . . . Tóm lại, nếu số lượng hàng mua tăng thêm 10 thì lại được miễn phí thêm một mặt hàng trong số các hàng có giá trị thấp nhất.

Alice không thể thay đổi trình tự hàng trong danh sách đã đăng ký mua nhưng có thể cắt danh sách thành các phần, mỗi phần gồm một dãy liên tiếp các hàng trong danh sách và bỏ vào một giỏ hàng riêng để nhận được chính sách ưu đãi đã nêu với giỏ hàng.

Là một người giỏi tính toán, Alice nhanh chóng hoàn thành các giỏ đặt hàng để tổng chi phí phải trả cho n mặt hàng muốn mua là nhỏ nhất.

Hãy xác định số tiền mà Alice sẽ phải thanh toán.

Dữ liệu: Vào từ file văn bản PAY.INP:

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

Kết quả: Đưa ra file văn bản PAY.OUT một số nguyên – số tiền ít nhất cần thanh toán.

Ví dụ:

PAY.INP	PAY.OUT
12	
1 1 10 10 10 10 10 10 10 9 10 10 10	92



Giải thuật: Quy hoạch động, tổng tiền tố.

Nhận xét:

- Nếu số lượng hàng mua không quá 9 thì việc để chúng trong một giỏ hàng hay chia thành các giỏ hàng mỗi giỏ một mặt hàng – tổng giá trị thanh toán không thay đổi,
- Khi mua n mặt hàng, việc chia thành $n/10$ giỏ, mỗi giỏ 10 mặt hàng và thêm một giỏ cho phần còn lại sẽ mang lại kết quả không xấu hơn sự phân chia trong đó có các giỏ hàng lớn hơn hay bằng 20 mặt hàng (khi chia nhỏ tập, min trong mỗi tập không giảm),
- Như vậy việc phân chia mặt hàng tiếp theo trong danh sách chỉ phụ thuộc vào 9 mặt hàng trước đó.

Sơ đồ tính toán: Quy hoạch động.

Gọi \mathbf{dp}_i – chi phí nhỏ nhất mua các mặt hàng từ 1 đến i trong danh sách, $\mathbf{dp}_0 = 0$

Có 2 lựa chọn:

$$\mathbf{dp}_i = \begin{cases} \mathbf{dp}_{i-1} + a_i & \text{Để riêng mặt hàng thứ } i \text{ thành một giỏ,} \\ \mathbf{dp}_{i-10} + \sum_{j=i+1}^i a_j - \min_{i-10 \leq j \leq i} a_j & \text{Tạo giỏ từ 10 mặt hàng cuối cùng.} \end{cases}$$

Để tính tổng giá phải trả cho 10 mặt hàng cuối cùng: Sử dụng tổng tiền tố,

Để tìm min nhanh: Lưu thông tin dưới dạng cấu trúc multiset.

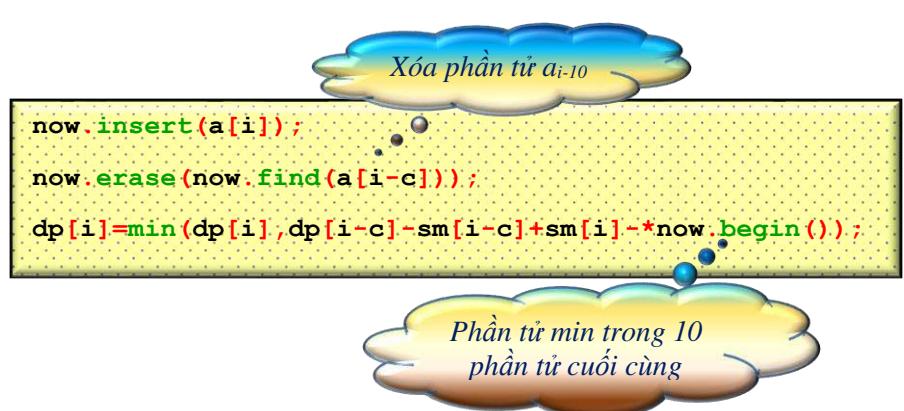
Tổ chức dữ liệu:

- ▀ Mảng **a** [**MAX_N**] – Lưu các giá trị mặt hàng,
- ▀ Mảng **dp** [**MAX_N**] – Phục vụ sơ đồ quy hoạch động,
- ▀ Mảng **sm** [**MAX_N**] – Lưu tổng tiền tố.
- ▀ Tập **multiset<int>** **now** – Lưu giá mặt hàng để tìm min.

Xử lý:

Cập nhật \mathbf{dp}_i với $i \geq$

10: $c = 10$



Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "pay."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef int64_t ll;

const int MAX_N = 100001;
int n, c = 10;
ll a[MAX_N];
ll dp[MAX_N], sm[MAX_N];
multiset<int> now;

int main()
{
    fi>>n;
    now.insert(0); dp[0]=0; sm[0]=0;
    for (int i = 1; i <= n; ++i)
    {
        fi>>a[i];

        dp[i] = dp[i-1] + a[i];
        sm[i] = sm[i-1] + a[i];
        now.insert(a[i]);

        if (i >= c)
        {
            now.erase(now.find(a[i-c]));
            dp[i]=min(dp[i], dp[i-c]-sm[i-c]+sm[i]-*now.begin());
        }
    }

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



Trong thời đại Cách mạng khoa học kỹ thuật 4.0 mọi học sinh đều rất ngại tính toán. Một phép tính đơn giản như $11 \times 12 = 132$ cũng phải dùng Casio. Ở buổi học cuối cùng trong năm học cô giáo cho bài tập về nhà làm trong hè. Cô viết trên bảng 6 số nguyên dương $x_1, x_2, x_3, x_4, x_5, x_6$ và yêu cầu từ 6 số này chọn một số làm số a , xóa số đó khỏi dãy, trong các số còn lại – chọn một số làm số b , xóa số được chọn khỏi dãy, trong các số còn lại – chọn một số làm số c , xóa số được chọn khỏi dãy, cuối cùng chọn một số trong số còn lại làm số d .

Rõ ràng ta có 6 cách chọn số a . Với mỗi số a đã chọn ta có 5 cách chọn số b , với mỗi cặp số a, b đã chọn ta có 4 cách chọn số c , với nhóm 3 số a, b, c đã chọn ta có 3 cách chọn số d . Như vậy tổng cộng ta có $6 \times 5 \times 4 \times 3 = 360$ cách chọn. Trong số các cách này hãy chọn a, b, c, d sao cho $\frac{a}{b} \leq \frac{c}{d}$ và $\frac{a}{b} + \frac{c}{d}$ là *nhỏ nhất*. Lưu ý, $\frac{a}{b}, \frac{c}{d}$ là phân số và có thể cho kết quả thực, ví dụ $\frac{3}{4} = 0.75$.

Cả lớp tái mét, mặt cắt ra không còn hột máu! Cô giáo nói thêm một câu, “đặt dấu chấm lên chữ i ” (tức là thực hiện nốt thao tác cuối cùng để hoàn thành một công việc): “Cô đã in và đưa cho Lớp trưởng n bộ 6 số như vậy, với mỗi bộ 6 số hãy chỉ ra các số a, b, c, d được chọn. Các em nhận đầu bài từ bạn Lớp trưởng. Chúc các em một kỳ nghỉ hè vui vẻ!” Các bạn đứng lên chào Cô với vẻ mặt như chính mình vừa làm mất số đầu bài. Ra đến cửa, cô giáo ngoái lại cười một cách bí hiểm: “Đừng bi quan như vậy. Không nhất thiết phải duyệt cả 360 cách đâu!”

Với mỗi bộ 6 số đã cho hãy đưa ra các số a, b, c và d được chọn.

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

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Mỗi dòng sau n dòng sau chứa 6 số nguyên dương, mỗi số có giá trị không vượt quá 10^9 . Các số ghi cách nhau một dấu cách.

Kết quả: Đưa ra file văn bản MINSUM.OUT với mỗi bộ số đã cho đưa ra trên một dòng các số a, b, c và d chọn, các đưa ra cách nhau một dấu cách.

Ví dụ:

MINSUM.INP
2
9 2 7 6 8 3
6 6 6 6 6 6

MINSUM.OUT
2 8 3 9
6 6 6 6



Giải thuật: Cơ sở lập trình và phân tích mô hình toán học.

Với tử số và mẫu số đều dương, giá trị phân số tỷ lệ thuận với tử số và tỷ lệ nghịch với mẫu số,

Cần chọn 2 số nhỏ nhất trong các số đã cho làm tử số, hai số lớn trong các số còn lại làm mẫu số.

Như vậy, sắp xếp các số đã cho theo thứ tự tăng dần ta có:

$$x_1 \leq x_2 \leq x_3 \leq x_4 \leq x_5 \leq x_6$$

Các tử số Các mẫu số

Ta có 3 lựa chọn:

$$\frac{x_1}{x_2} + \frac{x_5}{x_6}$$

$$\frac{x_1}{x_5} + \frac{x_2}{x_6}$$

$$\frac{x_1}{x_6} + \frac{x_2}{x_5}$$

Dễ dàng chứng minh được rằng cách lựa chọn thứ II cho kết quả nhỏ nhất (việc chứng minh có thể không cần thiết, thay vào đó – tính và so sánh giá trị của 3 biểu thức đã nêu với kết quả kiểu **long double**).

Để lựa chọn vai trò của **a, b, c, d** ta cần kiểm tra $\frac{x_1}{x_5} - \frac{x_2}{x_6}$.

$$\frac{x_1}{x_5} - \frac{x_2}{x_6} = \frac{x_1 \times x_6 - x_2 \times x_5}{x_5 \times x_6}$$

Dấu của hiệu cần tìm được xác định bởi biểu thức nguyên ở tử số. Lưu ý là giá trị của biểu thức có thể có bậc 10^{18} .

Độ phức tạp của giải thuật: Với mỗi bộ 6 số - độ phức tạp là O(1). Với bộ dữ liệu n bộ 6 – độ phức tạp sẽ là O(n).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "minsum."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;
int64_t a[6];

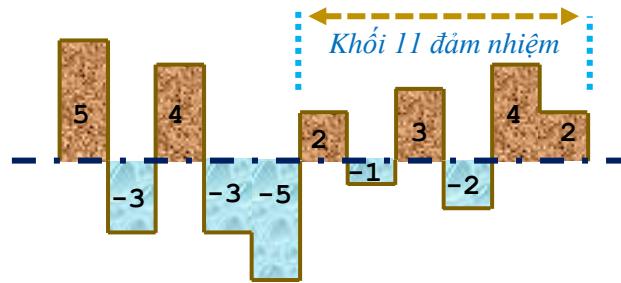
int main()
{
    fi>>n;
    for(int j=0; j<n; ++j)
    {
        for(int i=0;i<6;++i) fi>>a[i];
        sort(a,a+6);
        if(a[0]*a[5]- a[1]*a[4] <= 0)
            fo<<a[0]<<' '<<a[4]<<' '<<a[1]<<' '<<a[5]<<'\n';
        else fo<<a[1]<<' '<<a[5]<<' '<<a[0]<<' '<<a[4]<< '\n';
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Nhà thi đấu đa năng của Trường đang được khởi công xây dựng. Kinh phí từ ngân sách cấp có hạn nên hạng mục đường chạy ở trước nhà thi đấu không được duyệt. Nhà trường quyết định kêu gọi phụ huynh và học sinh cùng các thầy cô đóng góp công sức để có một đường chạy hoàn hảo cho học sinh rèn luyện thể lực.

Đường chạy phải hoàn toàn bằng phẳng, mọi điểm trên đường phải có cùng độ cao (gọi là *cùng một mặt thủy chuẩn*). Địa hình đường chạy tương lai được chia thành n khúc đánh số từ 1 đến n , từ trái sang phải. Độ cao khúc thứ i so với độ cao mặt đường thiết kế là h_i , $i = 1 \div n$. $h_i > 0$ nghĩa là phải hót đi khối lượng đất cao h_i , ngược lại, nếu $h_i < 0$ – chở trũng, cần lấp đất thêm vào để đạt độ bằng phẳng thiết kế. Đất hót ở chỗ cao có thể lấp vào các chỗ trũng, việc này không có gì khó khăn. Việc cào thêm đất ở đâu đó lấp vào chỗ trũng cũng nhẹ nhàng. Phần đất hót còn thừa sẽ chuyển vào tôn nền cho nhà thi đấu. Đây là một công việc hết sức vất vả. Ai nhận thi công đoạn nào thì chỉ lấp các khúc trũng trong đoạn đảm nhiệm.

Với tinh thần “*Đâu cần thanh niên có, đâu khó có thanh niên*” các chi đoàn khối 11 xung phong đảm nhiệm thi công đoạn các khúc liên tục có phần đất dư thừa chuyển đi tôn nền là lớn nhất. Nếu có nhiều đoạn có cùng số đất dư thừa như nhau thì bạn sẽ chọn đoạn có độ dài lớn nhất, tức là nhiều khúc nhất. Nếu có nhiều đoạn cùng độ dài và cùng số đất dư thừa, đoạn trái nhất (có số thứ tự khúc đầu tiên của đoạn nhỏ nhất) sẽ được chọn. Đoàn trường đánh giá rất cao tinh thần của các bạn khối 11 vì vậy bổ sung thêm một quy tắc là nếu *mỗi khúc đều phải lấp thêm* thì các bạn của chúng ta được chọn đoạn dài nhất và trái nhất đòi hỏi tổng số đất lấp thêm là ít nhất.



Ví dụ, đường chạy được chia thành 11 khúc với độ cao tương ứng của các khúc là $(5, -3, 4, -3, -5, 2, -1, 3, -2, 4, 2)$. Đoàn viên thanh niên khối 11 sẽ đảm nhiệm đoạn từ khúc 6 đến hết khúc 11 với số lượng đất chuyển đi tôn nền là 8.

Hãy xác định số đất các bạn khối 11 phải chuyển đi tôn nền, số thứ tự của khúc đầu và khúc cuối của đoạn được chọn.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^6$),
- ✚ Dòng thứ 2 chứa n số nguyên h_1, h_2, \dots, h_n ($|h_i| \leq 10^9$, $i = 1 \div n$), các số ghi cách nhau một dấu cách.

Kết quả: Đưa ra file văn bản TRACK.OUT, dòng đầu tiên chứa một số nguyên ans xác định số đất cần xử lý trong đoạn được chọn. $ans \geq 0$ chỉ số đất dư thừa cần chuyển đi, $ans < 0$ – số đất cần lấp thêm, dòng thứ hai chứa 2 số nguyên ghi cách nhau một dấu cách xác định khúc đầu và khúc cuối của đoạn được chọn.

Ví dụ:

TRACK.INP	TRACK.OUT
11 5 -3 4 -3 -5 2 -1 3 -2 4 2	8 6 11



Giải thuật: Phương pháp hai con trỏ, Tổng tiền tố.

Mô hình toán học của bài toán: Cho dãy số $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n$. Hãy tìm \mathbf{p} và \mathbf{q} ($1 \leq \mathbf{p} \leq \mathbf{q} \leq n$) sao cho $\mathbf{s} = \sum_{i=p}^q h_i$ là lớn nhất. Nếu có nhiều cặp (\mathbf{p}, \mathbf{q}) cùng cho \mathbf{s} lớn nhất thì chọn \mathbf{p}, \mathbf{q} sao cho $\mathbf{q}-\mathbf{p}$ lớn nhất và \mathbf{p} nhỏ nhất có thể.

Phân loại trường hợp để xây dựng sơ đồ xử lý một cách có hiệu quả:

- ⊕ $\mathbf{h}_i < 0 \forall i \rightarrow \mathbf{s} = \max_{1 \leq i \leq n} h_i$, đoạn (\mathbf{p}, \mathbf{q}) chỉ chứa *một phần tử*,
- ⊕ $\mathbf{h}_i \leq 0 \forall i \rightarrow \mathbf{s} = 0$, đoạn cần tìm: *đoạn liên tục chứa nhiều phần tử 0 nhất*,
- ⊕ Tồn tại $\mathbf{h}_i > 0$.

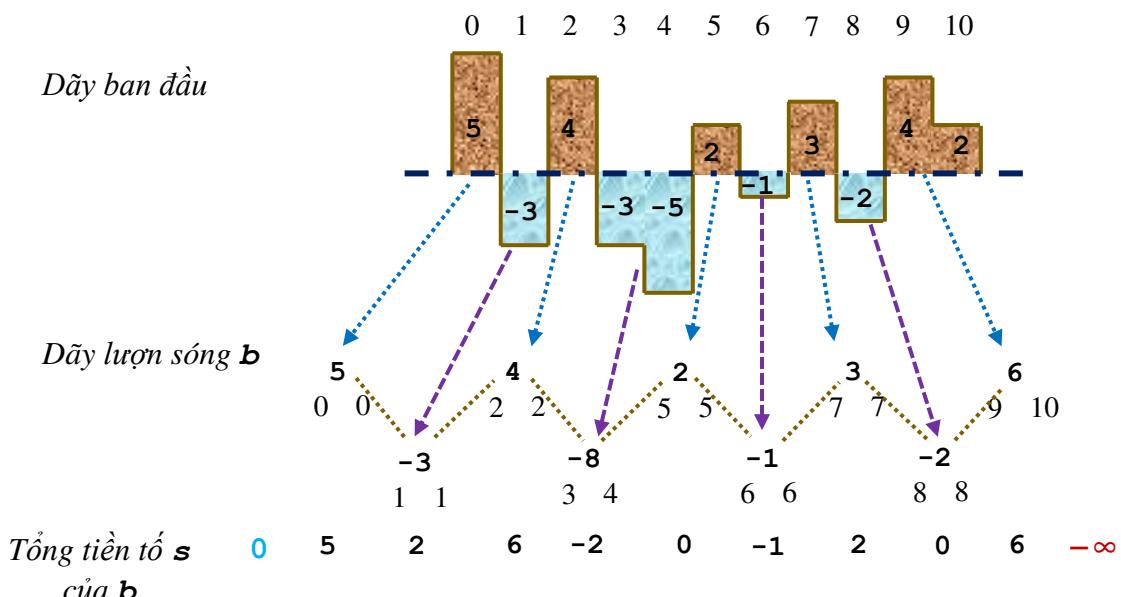
Quá trình nhập dữ liệu kết hợp với tìm $\max\{\mathbf{h}_i\}$ sẽ cung cấp thông tin phân loại trường hợp cần xử lý.

Xét trường hợp $\exists \mathbf{h}_i > 0$.

Để dàng nhận thấy:

- ⊕ \mathbf{p} phải là điểm đầu của một đoạn các phần tử \mathbf{h}_i liên tiếp không âm,
- ⊕ \mathbf{q} phải là điểm cuối của một đoạn các phần tử \mathbf{h}_i liên tiếp không âm.

Từ nhận xét trên có thể rút gọn kích thước bài toán và đưa về trường hợp dãy số đan xen các phần tử dương /âm: *dãy lượn sóng \mathbf{b}* .



Bài toán ban đầu được đưa về bài toán áp dụng trên dãy lượn sóng \mathbf{b} , mỗi phần tử của \mathbf{b} gồm 3 trường **(hb, ibegin, iend)**.

Sử dụng 2 con trỏ \mathbf{p} và \mathbf{q} quản lý đoạn có *tổng không âm* trên \mathbf{b} , \mathbf{p} chỉ tới điểm đầu, \mathbf{q} – điểm cuối của đoạn.

Khởi tạo \mathbf{p} : Chỉ tới phần tử đầu tiên của \mathbf{b} có $\mathbf{hb} \geq 0$ (\mathbf{p} bằng 1 hoặc 2).

\mathbf{p} thay đổi với bước nhảy bằng 2, \mathbf{q} không giảm khi \mathbf{p} tăng!

Điều kiện cập nhật kết quả: khi đoạn có tổng lớn hơn hoặc bằng nhung độ dài lớn hơn.

Dộ phức tạp của giải thuật: O(n).

Tổ chức dữ liệu:

- Mảng `vector<int64_t> h` – Lưu độ cao ban đầu,
- Mảng `vector<tlii> b` – Lưu dãy lượn sóng, trong đó `tlii` là `tuple<int64_t, int, int>`,
- Mảng `vector<int64_t> s` – Lưu tổng tiền tố của dãy lượn sóng,
- Các đại lượng `int64_t ans=-MXLL, int il=0, ir=0` – Lưu kết quả cần tìm.

Xử lý:

Nhập dữ liệu và xử lý trường hợp dãy âm:

```
fi>>n;
h.resize(n+2);
for(int i=1;i<=n;++i)
{
    fi>>h[i];
    if(ans<h[i]) ans=h[i], il=i, ir=i;
}

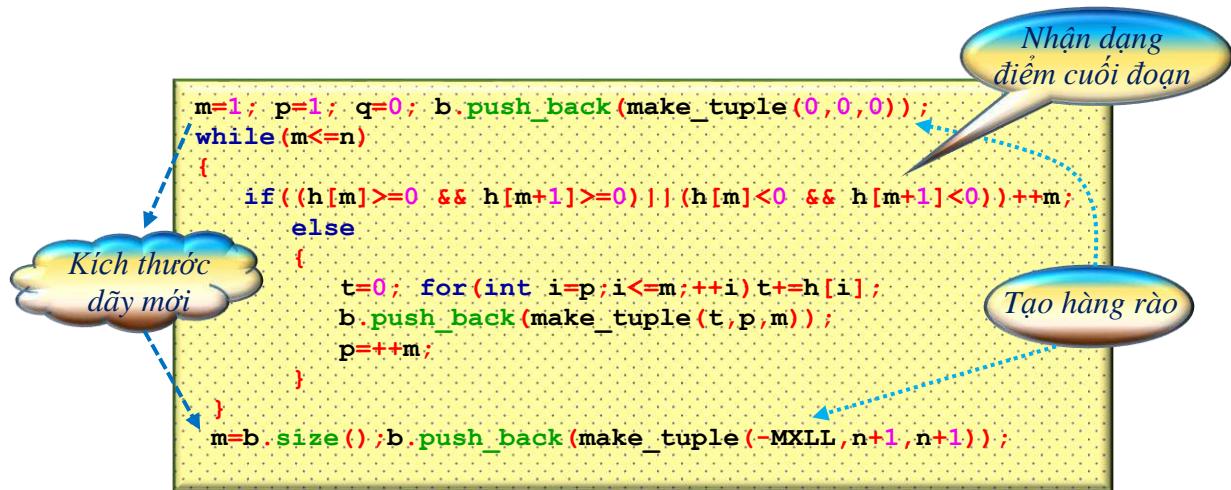
if(ans<0){fo<<ans<<'\\n'<<il<<' ' <<ir; return 0;}
```

Trường hợp dãy không dương:

```
h[n+1]=h[n]>=0? -1:1;
if(ans==0)
{
    p=il;
    while(p>0)
    {
        get_q();
        if(q-p>ir-il) il=p, ir=q;
        get_p(q+1);
    }
    fo<<ans<<'\\n'<<il<<' ' <<ir; return 0;
}
```

Trường hợp tồn tại $h_i > 0$:

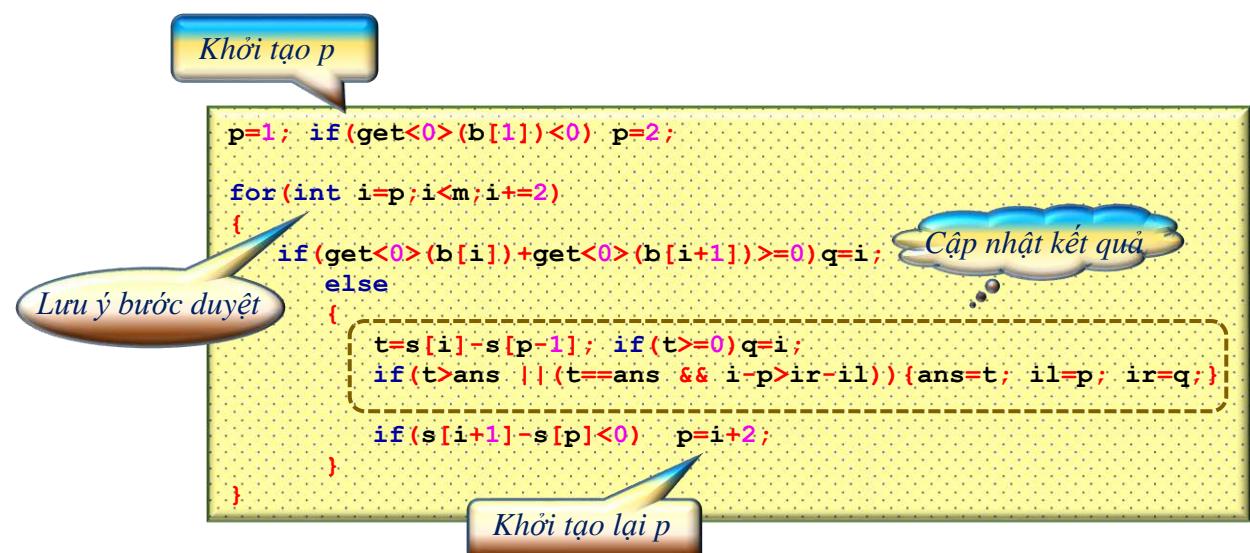
Xây dựng dãy lượn sóng:



Việc tính tổng tiền tố s được thực hiện với các giá trị $t = \text{get}(b[i])$,

Để thuận tiện xử lý cần xác lập giá trị hàng rào $s[m] = -MXLL$.

Tìm kết quả cuối cùng:



Nhận xét:

- + Có thể tránh việc sử dụng mảng **b**, tuy vậy khi đó việc lập trình trở nên phức tạp hơn,
- + Việc chia quá trình xử lý thành *các công đoạn*, mỗi công đoạn ứng với một *đoạn hoặc mô đun chương trình độc lập*, các công đoạn xử lý nối với nhau thông qua *input/output* của *công đoạn* sẽ làm *giảm đáng kể độ phức tạp lập trình*.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "track."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef tuple<int64_t,int,int> tlii;
const int64_t MXLL=(int64_t)1e18;
int n,m,t,il=0,ir=0,p,q;
int64_t ans=-MXLL;
vector<int64_t>s,h;
vector<tlii>b;
bool flag=false;

void get_q()
{
    for(int i=p;i<=n+1;++i)
        if(h[i]!=0){q=i-1; return;}
}

void get_p(int x)
{
    p=-1;
    for(int i=x+1;i<=n;++i)
        if(h[i]==0){p=i; break;}
}
int main()
{
    fi>>n;
    h.resize(n+2); ans=-MXLL;
    for(int i=1;i<=n;++i){fi>>h[i]; if(ans<h[i]) ans=h[i], il=i, ir=i;}
    if(ans<0){fo<<ans<<'\\n'<<il<<' ' <<ir; return 0;}
    h[n+1]=h[n]>=0? -1:1;
    if(ans==0)
    {
        p=il;
        while(p>0)
        {
            get_q();
            if(q-p>ir-il) il=p, ir=q;
            get_p(q+1);
        }
        fo<<ans<<'\\n'<<il<<' ' <<ir; return 0;
    }

    m=1; p=1; q=0; b.push_back(make_tuple(0,0,0));
    while(m<=n)
    {
        if((h[m]>=0 && h[m+1]>=0) || (h[m]<0 && h[m+1]<0)) ++m;
        else
        {
            t=0; for(int i=p;i<=m;++i)t+=h[i];
            b.push_back(make_tuple(t,p,m));
            p=++m;
        }
    }
    m=b.size(); b.push_back(make_tuple(-MXLL,n+1,n+1));
    s.resize(m+2);
    s[0]=0;
    for(int i=1;i<m;++i)
```

```

{
    t=get<0>(b[i]); if(ans<t) ans=t, il=ir=i;
    s[i]=s[i-1]+t;
}
s[m]=-MXLL;
p=1; if(get<0>(b[1])<0) p=2; ans=0; q=p;
for(int i=p; i<m; ++i)
    {t=get<0>(b[i]); if(t>ans || (t==ans && i-p>ir-il)) ans=t,
     il=i, ir=i, flag=true; }
for(int i=p; i<m; i+=2)
{
    if(get<0>(b[i])+get<0>(b[i+1])>=0) q=i;
    else
    {
        t=s[i]-s[p-1]; if(t>=0) q=i;
        if(t>ans || (t==ans && i-p>ir-il))
            {ans=t; il=p; ir=q; flag=true; }
        if(s[i+1]-s[p]<0) p=i+2;
    }
}

if(flag) fo<<ans<<'\\n'<<get<1>(b[il])<<' ' <<get<2>(b[ir]);
else fo<<ans<<'\\n'<<il<<' '<<ir;

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

```



Dãy núi có n đỉnh nằm khá thẳng hàng. Đỉnh núi thứ i tính từ trái sang phải có độ cao h_i , $i = 1 \div n$. Vào mùa khô, vì nhiều lý do khác nhau ở đây thường xảy ra cháy rừng. Để có thể nhận được sớm thông tin các vụ cháy có thể xảy ra người ta lắp trên mỗi đỉnh một camera quan sát cảnh báo cháy. Trên đỉnh núi gió thổi rất mạnh vì vậy camera phải gắn cố định, quan sát sang trái (L) hoặc sang phải (R).



Camera ở đỉnh núi i có thể quan sát được đỉnh núi j nếu nó hướng về phía đỉnh j đồng thời giữa các đỉnh i và đỉnh j không có đỉnh nào cao hơn đỉnh j . Ví dụ, với $n = 5$, độ cao các đỉnh tính từ trái sang phải là 168, 170, 165, 160, 180, camera lắp ở đỉnh 4 và hướng sang trái sẽ quan sát thấy đỉnh 2 và đỉnh 3. Nếu có cháy xảy ra ở các núi camera quan sát được thông tin báo về sẽ cụ thể và chuẩn xác.

Số lượng đỉnh núi mà một camera quan sát được theo hướng lắp đặt được gọi là phạm vi bao quát của camera đó.

Cho biết n , độ cao các núi và hướng quan sát của mỗi camera dưới dạng xâu s chỉ chứa các ký tự từ tập $\{‘L’, ‘R’\}$, ký tự thứ i xác định hướng của camera thứ i , $i = 1 \div n$. Hãy xác định phạm vi bao quát của từng camera.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên h_1, h_2, \dots, h_n ($1 \leq h_i \leq 10^9$, $i = 1 \div n$),
- ✚ Dòng thứ 3 chứa xâu s độ dài n , có các ký tự thuộc tập $\{‘L’, ‘R’\}$.

Kết quả: Đưa ra file văn bản VISIBLE.OUT trên một dòng xác định phạm vi bao quát của các camera, các số ghi cách nhau một dấu cách, số thứ i xác định phạm vi bao quát của camera lắp trên đỉnh núi i ($i = 1 \div n$).

Ví dụ:

VISIBLE.INP	VISIBLE.OUT
5 168 170 165 160 180 LLRLL	0 1 2 2 3



Giải thuật: *Ứng dụng stack.*

Việc xử lý hướng quan sát sang phải tương tự như xử lý hướng quan sát sang trái,
Xét hướng quan sát sang trái.

Nếu từ đỉnh i không quan sát thấy đỉnh j ($j < i$) thì từ mọi đỉnh k ($k > i$) cũng
không quan sát thấy đỉnh j .

Như vậy, bắt đầu từ một thời điểm nào đó ta không nhìn thấy đỉnh j thì đỉnh này
sẽ bị loại khỏi danh sách những đỉnh thấy được,

Độ cao của các đỉnh thấy được tạo thành một *dãy không tăng*.

Như vậy có thể tổ chức một **stack** lưu trữ độ cao của các đỉnh quan sát được.

Duyệt các độ cao h_i từ trái sang phải, nếu $s_i = 'L'$: *số lượng phần tử trong stack*
là kết quả cần tìm với vị trí i ,

Trong mọi trường hợp của s_i , nạp h_i vào stack, duy trì trạng thái dãy không tăng
của **stack**.

Với hướng quan sát sang phải: xử lý tương tự, nhưng duyệt các độ cao h_i từ phải
sang trái.

Tổ chức dữ liệu:

- ▀ Xâu **string** s – Lưu các hướng quan sát.
- ▀ Mảng **vector<int>** $h(n)$ – Lưu độ cao các đỉnh,
- ▀ Mảng **vector<int>** $res(n)$ – Lưu kết quả,
- ▀ Các **stack<int>stl, str** – Phục vụ duyệt độ cao tìm kết quả.

Độ phức tạp của giải thuật: $O(n)$.

Lưu ý: Tự tổ chức stack sẽ giảm thời gian thực hiện chương trình so với việc
dùng stack của hệ thống.

Chương trình

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

int main()
{
    fi>>n;
    vector<int>h(n),res(n);
    stack<int>stl;
    for(int i=0;i<n;++i)fi>>h[i];
    fi>>s;
    for(int i=0;i<n;++i)
    {
        if(s[i]=='L') res[i]=stl.size();
        while(!stl.empty() && stl.top()<h[i])stl.pop();
        stl.push(h[i]);
    }

    stack<int>str;
    for(int i=n-1;i>=0;--i)
    {
        if(s[i]=='R') res[i]=str.size();
        while(!str.empty() && str.top()<h[i])str.pop();
        str.push(h[i]);
    }
    for(int i=0;i<n;++i)fo<<res[i]<<' ';
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



Thầy cô giáo giỏi đồng thời cũng là một nhà tâm lý học. Cô tổ trưởng Tổ Tin của Trường là một giáo viên giỏi. Cô nhận thấy các em học sinh lớp 10 chuyển từ PTCS lên, được học trong trường mới với các bạn mới và các thầy cô mới nên rất hào hứng với những cái gì mới và đa dạng, ngay cả các con số cũng vậy. Nếu 2 chữ số đứng cạnh nhau mà giống nhau thì chẳng có gì là mới, là đa dạng cả! Vì vậy các số được gọi là đa dạng, tức là không có 2 chữ số nào cạnh nhau và giống nhau sẽ hấp dẫn, lôi cuốn các bạn trẻ hơn nhiều.

Thông thường trong bài toán tin học thường có ràng buộc giá trị một đại lượng nào đó phải nhỏ hơn hoặc bằng n . Giá trị n thường là 1000 hay 100 000 hoặc 1 000 000, . . . Khá là buồn tẻ! Nhận thấy yếu tố tâm lý trên của học sinh Cô tổ trưởng đề nghị các thầy cô trong tổ sửa lại một chút tham số ở các bài tập trong sách giáo khoa để các em hào hứng làm việc hơn bằng cách thay số n bằng số đa dạng nhỏ nhất lớn hơn n (ngay cả khi bản thân n đã là số đa dạng vì giữ nguyên thì không có yếu tố mới trong bài tập). Ví dụ nếu n bằng 255 thì thay bằng 256, n bằng 98 thay bằng 101.

Có m số cần thay. Với mỗi số hãy xác định số đa dạng nhỏ nhất lớn hơn số cần thay.

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

- ✚ Dòng đầu tiên chứa một số nguyên m ($1 \leq m \leq 101$),
- ✚ Mỗi dòng trong m dòng tiếp theo chứa một số nguyên n ($1 \leq n \leq 10^{18}$).

Kết quả: Đưa ra file văn bản MENTALITY.OUT m số nguyên mới nhận được, mỗi số trên một dòng. Các số đưa ra theo trình tự các số cần thay trong file input.

Ví dụ:

MENTALITY.INP	MENTALITY.OUT
3	256
255	101
98	5010
4992	

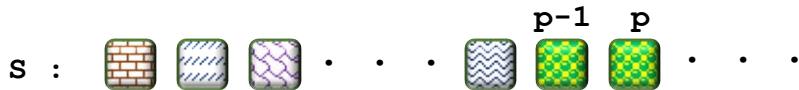


Giải thuật: Xử lý xâu.

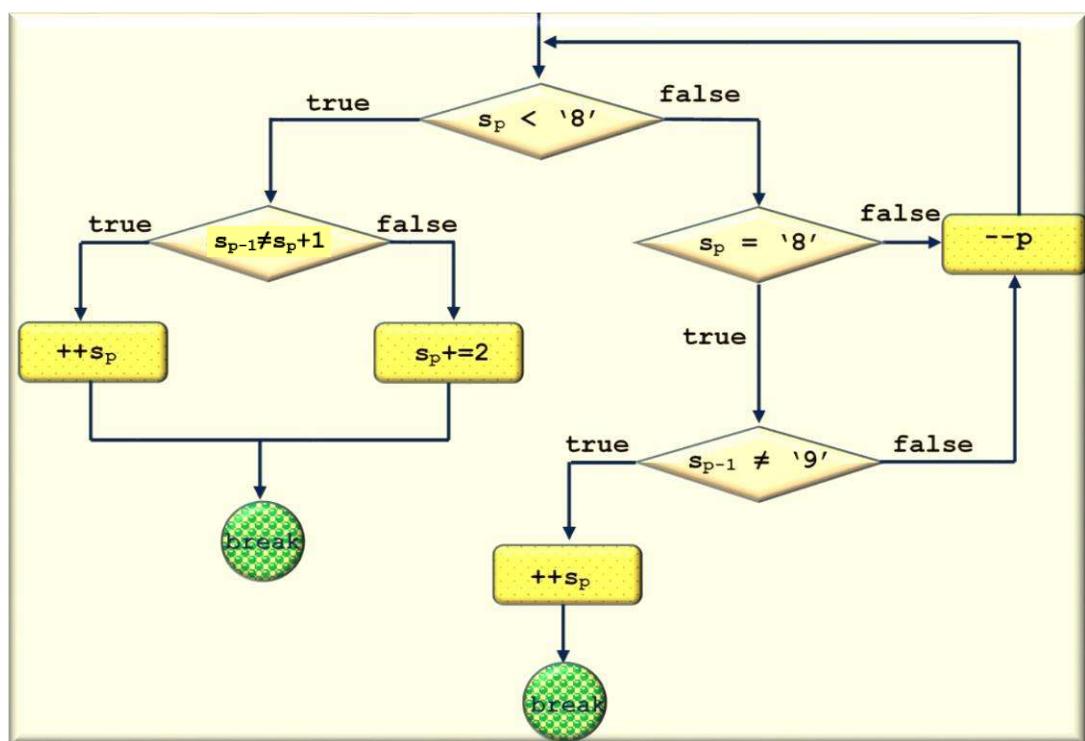
Lưu trữ số cần xử lý dưới dạng xâu s ,

Gọi n – độ dài của xâu s ($n = s.size()$).

Tìm p – vị trí đầu tiên từ trái sang phải có $s_p = s_{p-1}$. $p = n-1$ nếu trong s không có cặp ký tự liên tiếp giống nhau.



Sơ đồ xử lý:



Việc lùi p có thể dẫn đến $p < 0 \rightarrow$ cần thêm ký tự ‘0’ ở đầu xâu s : $s = '0' + s$;

Phần còn lại từ $p+1$ về cuối xâu: bổ sung đan xen các ký tự ‘0’ và ‘1’.

Sau khi xử lý: xóa s_0 nếu $s_0 = '0'$.

Độ phức tạp của giải thuật:

- ⊕ Cho một xâu: có thể coi là $O(1)$ vì $n \leq 18$,
- ⊕ Cho toàn bộ bài toán: $O(m)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "mentality."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,k,m,p;
string s;

int main()
{
    fi>>m;
    for(int j=0; j<m; ++j)
    {
        fi>>s;
        s='0'+s; n=s.size();
        p=n-1;
        for(int i=1;i<n;++i)
            if(s[i]==s[i-1]) {p=i; break;}
        while(1)
        {
            if(s[p]<56) {if(s[p-1]!=s[p]+1)++s[p]; else s[p]+=2;break;}
            if(s[p]==56 && s[p-1]!=57) {++s[p]; break;}
            --p;
        }
        k=0;
        for(int i=p+1;i<n;++i) s[i]=48+k, k^=1;
        if(s[0]==48)s.erase(0,1);
        fo<<s<<'\n';
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```

Kỹ thuật con lắc

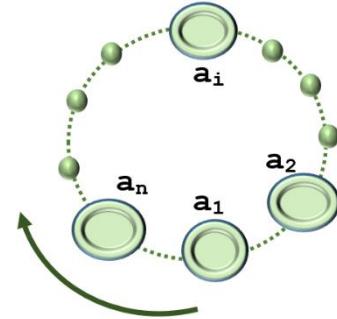


VX26. ĐÁU VẬT

Tên chương trình: WRESTLING.CPP

Bên lề của Hội thi Phù Đổng năm nay Ban Tổ chức còn đưa vào môn thi đấu vật đồng đội. Mỗi trường cử ra một đội gồm n đô vật đấu với đội của trường khác. Cứ 2 người, mỗi người ở một trường đấu với nhau. Để tiết kiệm thời gian cả n cặp cùng đấu một lúc trên n sàn. Các sàn đấu được bố trí thành một vòng tròn trên sân vận động.

Vào chung kết là các đội của hai trường **A** và trường chủ nhà **B**. Hai trường này đã gặp nhau ở một hội thi khác, khi đó người thứ i của trường **A** đã gặp người thứ i của trường **B**. Danh sách xếp hàng được máy tính đưa ra một cách ngẫu nhiên, với trường **A** là a_1, a_2, \dots, a_n , trong đó $1 \leq a_i \leq n$, các a_i nguyên và khác nhau từng đôi một, $i = 1 \div n$, với trường **B** là danh sách b_1, b_2, \dots, b_n , các số b_i cũng có tính chất như a_i với mọi i . Các bạn ở đội B có quyết tâm rất cao, xem kỹ băng hình lần gặp trước với đội A, phân tích kỹ chiến lược và chiến thuật của phía bạn. Ai cũng khao khát được gặp lại chính đối thủ cũ của mình, nhưng trình tự xếp hàng đã cố định.



Theo truyền thống, mỗi đội sẽ chạy quanh sân theo chiều kim đồng hồ chào khán giả rồi mới lên sàn. Đội **A** ra sân trước. Các sàn được đánh số từ 1 đến n theo chiều ngược kim đồng hồ. Ở sàn thứ i là bạn a_i của trường **A**. Sau khi các đội thủ đã lên sàn, đội **B** ra sân. Đội chủ nhà bao giờ cũng được khán giả đón tiếp nồng nhiệt. Bạn dãy đầu đưa ra một ý tưởng được mọi người rất hoan nghênh: Thay vì chạy đúng một vòng bạn đó có thể dẫn đội chạy quá thêm vài sàn đấu rồi mới dừng lại để mỗi người lên sàn đấu cạnh mình đang đứng, như vậy vẫn không phạm luật!

Ví dụ, $n = 5$, danh sách xếp hàng đội **A** là (1, 3, 5, 2, 4) và của đội **B** là (1, 3, 2, 5, 4).

<i>Trường hợp đội B</i>	A: 1 3 5 2 4	<i>Trường hợp đội B chạy</i>	A: 1 3 5 2 4
<i>chạy đúng một vòng</i>	B: 2 5 4 1 3	<i>quá thêm 3 sàn đấu</i>	B: 1 3 2 5 4

*Không có bạn nào
gặp đối thủ cũ*

*Có 3 bạn gặp
đối thủ cũ*

Hãy xác định bạn dãy đầu sau một vòng cần chạy thêm mấy sàn đấu để có nhiều bạn gặp được đối thủ cũ nhất. Nếu có nhiều cách cùng đạt được kết quả nhiều nhất thì chọn cách phải chạy thêm ít nhất.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 5 \times 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ,
- ✚ Dòng thứ 3 chứa n số nguyên b_1, b_2, \dots, b_n .

Các số a_i, b_i với mọi i thỏa mãn điều kiện đã nêu. Các số trên một dòng ghi cách nhau một dấu cách.

Kết quả: Đưa ra file văn bản WRESTLING.OUT trên một dòng hai số nguyên xác định số sàn đấu cần chạy thêm và số người nhiều nhất gặp lại được đối thủ cũ. Các số ghi cách nhau một dấu cách.

Ví dụ:

WRESTLING.INP	WRESTLING.OUT
5 1 3 5 2 4 2 5 4 1 3	3 3



VX25 AXIII

Giải thuật: Hoán vị vòng tròn.

Xây dựng mảng $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n)$, trong đó \mathbf{p}_i – vị trí người thứ i của đội **A** trong danh sách xếp hàng: $p_{a_i} = i$.

Người thứ i trong danh sách xếp hàng của **B** là \mathbf{b}_i ,

Để gấp được đối thủ cũ của mình người \mathbf{b}_i cần đứng cạnh sàn đấu p_{b_i} .

Như vậy, để từ vị trí i sang vị trí p_{b_i} số sàn đấu cần chạy thêm sau một vòng theo chiều kim đồng hồ sẽ là $\mathbf{k} = (i - p[\mathbf{b}[i]] + n) \% n$;

Giá trị \mathbf{k} nằm trong phạm vi từ 0 đến $n-1$.

Với mỗi người của B tính giá trị k tương ứng, tích lũy số người có cùng giá trị k và tìm max trong các giá trị tích lũy được.

Dộ phức tạp của giải thuật: O(n).

Tổ chức dữ liệu:

- Mảng `vector<int> p(n+1)` – Lưu vị trí của người thứ i của **A**, $i = 1 \div n$,
- Mảng `vector<int> r(n, 0)` – Lưu số người gấp đôi thủ cũ khi chạy thêm 0,1, 2, ..., $n-1$ sàn đấu.

Chương trình

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

int main()
{
    fi>>n;
    vector<int> p(n+1), r(n, 0);
    for(int i=1; i<=n; ++i) {fi>>k; p[k]=i;}
    for(int i=1; i<=n; ++i)
    {
        fi>>k;
        k = (i-p[k]+n)%n;
        ++r[k];
    }
    for(int i=0; i<n; ++i) if(ans<r[i]) ans=r[i], k=i;
    fo<<ans<<' ' <<k;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Trong Tin học không có cái gì là vô hạn, mọi thứ đều hữu hạn: bộ nhớ hữu hạn, tốc độ xử lý hữu hạn, kích thước bài toán hữu hạn, . . . Trong cuộc sống quanh ta, như A. Einstein đã nhận xét, chỉ có 2 thứ là không có giới hạn.

Còn trong Toán học thì có vô số thứ vô hạn, ví dụ dãy số nguyên dương 1, 2, 3, 4, . . . là một dãy vô hạn, nhưng một bạn trong lớp đã giơ tay xin có ý kiến: "Tuy dãy này là vô hạn nhưng mỗi số nguyên dương (số tự nhiên) gặp trong dãy một và chỉ một lần!" Thầy giáo đồng ý với nhận xét đó, viết tiếp lên bảng một dãy số khác và cho biết đây là một dãy có quy luật, độ dài vô hạn, trong đó mỗi số tự nhiên gặp vô hạn lần:

$$\boxed{1, \quad 1, \quad 2, \quad 1, \quad 1, \quad 2, \quad 3, \quad 2, \quad 1, \quad 1, \quad 2, \quad 3, \quad 4, \quad 3, \quad 2, \quad 1, \quad \dots}$$

Các phần tử của dãy được đánh số từ 1 trở đi. Như vậy ở vị trí 3 là số 2, ở vị trí 12 là số 3, . . .

Vì đây là một dãy có quy luật nên ta hoàn toàn dễ dàng xác định được số ở vị trí thứ n của dãy.

Cho m vị trí. Với mỗi vị trí đã cho hãy xác định số ở vị trí đó trong dãy.

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

- ✚ Dòng đầu tiên chứa một số nguyên m ($1 \leq m \leq 10^5$),
- ✚ Mỗi dòng sau chứa số nguyên n – vị trí trong dãy ($1 \leq n \leq 10^{32}$).

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

Ví dụ:

INFINITY.INP	INFINITY.OUT
6	1
5	2
6	3
7	2
8	1
9	151
500000	



Giải thuật: Xử lý số lớn, tìm kiếm nhị phân.

Mỗi số tự nhiên mới trong dãy sẽ tương ứng với một nhóm các phần tử của dãy:

```

1 → 1
2 → 1 2 1
3 → 1 2 3 2 1
4 → 1 2 3 4 3 2 1
    . . . .

```

Số m sẽ tương ứng với nhóm $2 \times m - 1$ phần tử.

Độ dài phần đầu của dãy, khi có m nhóm là $1 + 3 + 5 + \dots + (2 \times m - 1) = m^2$.

Để xác định số ở vị trí n ta cần biết vị trí cần tìm thuộc nhóm nào.

Gọi k là nhóm chứa vị trí n .

k sẽ là số tự nhiên nhỏ nhất thỏa mãn điều kiện $k^2 \geq n$.

Với $n \leq 10^{10} \rightarrow$ bằng cách duyệt trực tiếp có thể tìm k với độ phức tạp $O(\sqrt{n})$ (trong phạm vi hạn chế thời gian đủ nhỏ):

```

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

int main()
{
    fi>>n;
    k=1;
    while(k*k<n)++k;
    t=k*k;
    if(t==n)ans=1;
    else
    {
        t=n-(k-1)*(k-1);
        if(t<=k)ans=t; else ans=t+1-k;
    }
    fo<<ans<<'\\n';
}
fo<<"\\nTime: "<<clock()/(double)1000<<" sec";
}

```

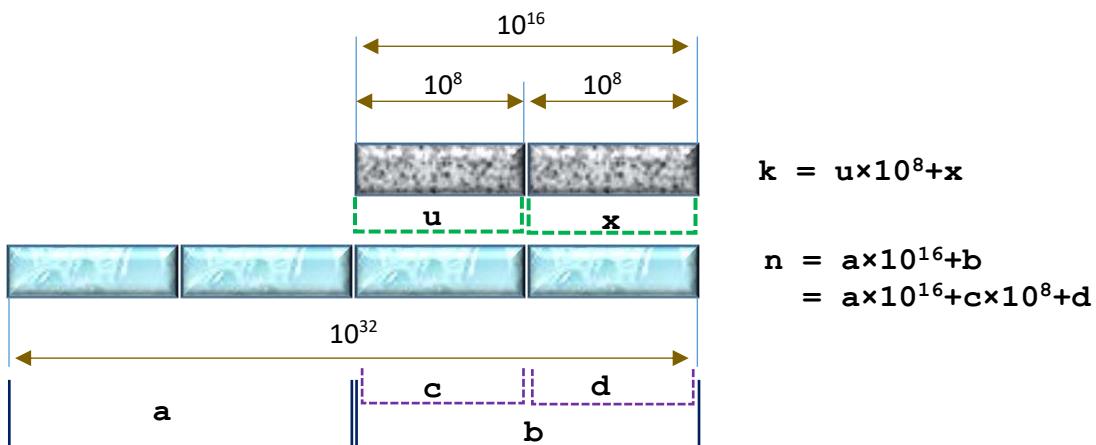
Với $n > 10^{10}$: Cần tổ chức tìm kiếm nhị phân.

Theo điều kiện đầu bài, $n \leq 10^{32} \Rightarrow k \leq 10^{16}$.

Có thể trực tiếp xác định k bằng phương pháp tìm kiếm nhị phân. Nhưng việc này đòi hỏi phải tổ chức nhân và so sánh số lớn – một điều khá phức tạp và hiệu quả thực hiện chương trình giảm một cách đáng kể.

Có một cách tiếp cận hiệu quả hơn.

Để tính k^2 ta cần tách chức biểu diễn k dưới dạng số lớn, trong trường hợp này – hiệu quả nhất là theo cơ số 10^8 :



Để dàng tính được u : $u = \lfloor \sqrt{a} \rfloor$.

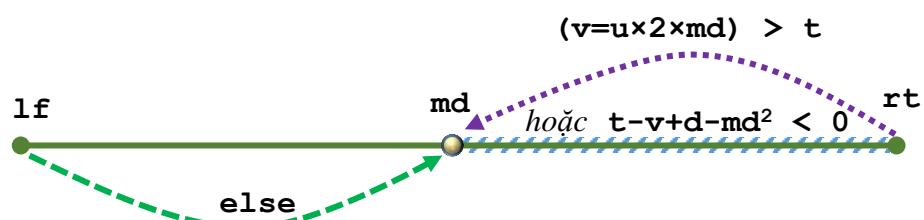
Xác định v : $n = (a, b)$.

$$n - u^2 = (a - u^2, b) \leq 2 \times u \times x + x^2$$

Lưu ý về phải là một số trong phạm vi biểu diễn của kiểu dữ liệu `int64_t`.

Đặt $t = (a - u^2) \times 10^8 + c$.

Tiêu chuẩn tìm kiếm nhị phân:



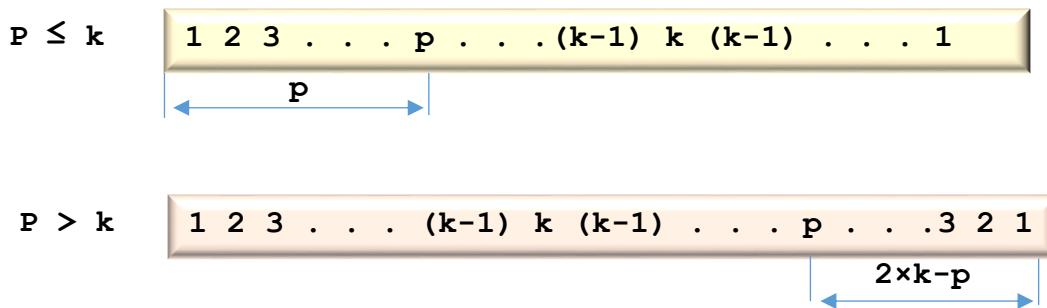
Kết quả nhóm tìm được sẽ là $k = u \times 10^8 + lf$.

Cần tăng kết quả từ tìm kiếm nhị phân lên 1 nếu n không phải là số chính phương.

Vị trí n trong dãy sẽ tương ứng với vị trí p trong nhóm k :

$$p = t - v + d - (lf - 1)^2$$

Có 2 trường hợp:



Tổ chức dữ liệu:

- ☒ Lưu trữ n dưới dạng xâu,
- ☒ Hệ thống các biến đơn **a**, **b**, **c**, **d** – biểu diễn lại n phục vụ các phép tính số lớn.

Xử lý:

Với $n \leq 10^{18}$: trực tiếp tìm kiếm nhị phân xác định nhóm chứa vị trí n :

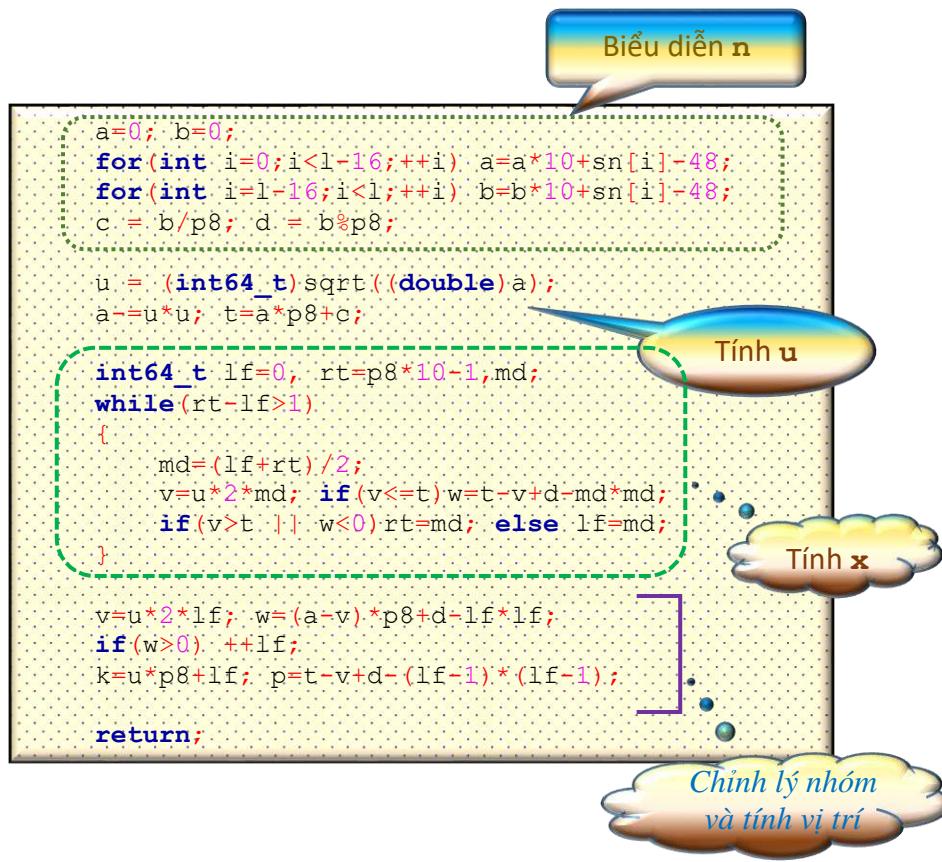
```
int l = sn.size(); t = 0;
int64_t u, v, w;

if(l<=18)
{
    for(int i = 0; i<l; ++i) t=t*10+sn[i]-48;
    k = (int64_t)sqrt((double)t);

    if(k*k<t) ++k;
    p=t-(k-1)*(k-1);
}

return;
}
```

Trường hợp cần làm việc với số lớn:



Nhận xét: Việc triển khai hai sơ đồ xử lý chỉ để giảm thời gian xử lý trong trường hợp cụ thể.

Dộ phức tạp của giải thuật: $\approx O(1)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "infinity."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int p8 = (int)1e8;
const int64_t p16 = (int64_t)1e16;
int n,m;
int64_t ans,t,a,b,c,d,k,p;
string sn;

void calc_n()
{
    int l = sn.size(); t = 0;
    int64_t u,v,w;

    if(l<=18)
    {
        for(int i = 0; i<l; ++i) t=t*10+sn[i]-48;
        k = (int64_t)sqrt((double)t);
        if(k*k<t) ++k;
        p=t-(k-1)*(k-1);
        return;
    }

    a=0; b=0;
    for(int i=0;i<l-16;++i) a=a*10+sn[i]-48;
    for(int i=l-16;i<l;++i) b=b*10+sn[i]-48;
    c = b/p8; d = b%p8;
    u = (int64_t)sqrt((double)a);
    a-=u*u; t=a*p8+c;
    int64_t lf=0, rt=p8*10-1, md;
    while(rt-lf>1)
    {
        md=(lf+rt)/2;
        v=u*2*md; if(v<=t) w=t-v+d-md*md;
        if(v>t || w<0) rt=md; else lf=md;
    }
    v=u*2*lf; w=(a-v)*p8+d-lf*lf;
    if(w>0) ++lf;
    k=u*p8+lf; p=t-v+d-(lf-1)*(lf-1);
    return;
}

int main()
{
    fi>>m;
    for(int i=0; i<m; ++i)
    {
        fi>>sn;
        calc_n();
        if(p>k) p=k*2-p;
        fo<<p<<'\n';
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



DNA là phân tử mang thông tin di truyền mã hóa cho hoạt động sinh trưởng, phát triển, chuyên hóa chức năng và sinh sản của các sinh vật. DNA của các sinh vật trên trái đất có thể được mô tả bằng xâu ký tự từ bảng chữ cái {A, C, G, T}, mỗi chữ cái tương ứng với một nucleobase (gọi tắt là base): Adenine, Cytosine, Guanine và Thymine.

Chương trình tìm kiếm sự sống ngoài trái đất phát hiện trên Sao Kim (Venus) có một dạng sống đặc biệt. Việc phân tích các mẫu vật thu được cho thấy DNA chứa tới k base và có độ dài n . Để thuận tiện nghiên cứu người ta đánh số các base này từ 0 đến $k - 1$. Như vậy mỗi DNA có thể biểu diễn như một dãy số nguyên không âm, mỗi số nằm trong khoảng $[0, k-1]$.

Một nhóm nhà khoa học muốn nghiên cứu và triển khai cấu trúc DNA này vào việc xây dựng trí tuệ nhân tạo. Để làm việc đó họ cần một đoạn DNA trong đó có r base khác nhau, loại thứ j có số lượng không ít hơn q_j , $j = 1 \div r$.

Hãy xác định độ dài ngắn nhất của đoạn DNA có thể cắt và chuyển giao cho nhóm nghiên cứu.

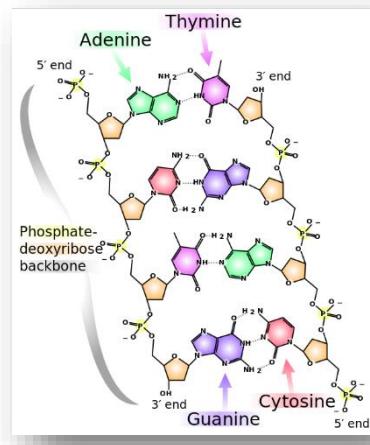
Dữ liệu: Vào từ file văn bản DNA.INP:

- ✚ Dòng đầu tiên chứa ba số nguyên n , k và r ($1 \leq n \leq 2 \times 10^5$, $1 \leq r \leq k \leq n$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n mô tả DNA thu được ($0 \leq a_i \leq k-1$, $i = 1 \div n$),
- ✚ Dòng thứ j trong r dòng sau chứa 2 số nguyên b_j và q_j xác định loại base và số lượng cần có ($0 \leq b_j < k$, $1 \leq q_j \leq n$), $b_j \neq b_u$ với $j \neq u$.

Kết quả: Đưa ra file văn bản DNA.OUT một số nguyên – độ dài ngắn nhất tìm được. Nếu không có đoạn nào thỏa mãn yêu cầu đã nêu thì đưa ra thông báo **impossible**.

Ví dụ:

DNA.INP	DNA.OUT
<pre> 13 4 3 1 1 3 2 0 1 2 0 0 0 0 3 1 0 2 2 1 1 2 </pre>	<pre> 7 </pre>

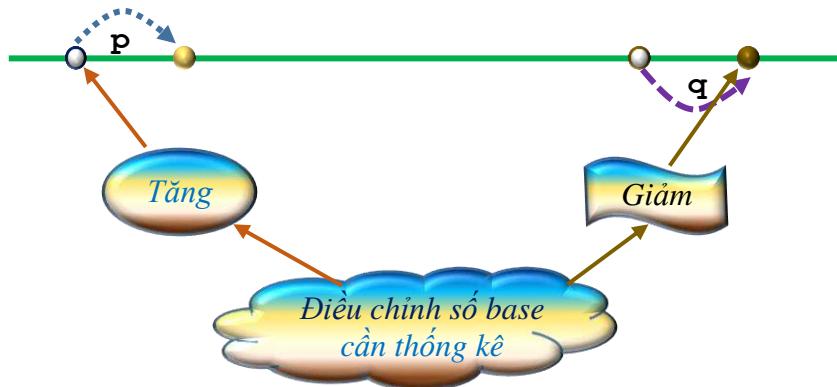


Giải thuật: Phương pháp 2 con trỏ.

Ghi nhận tần số xuất hiện của các nucleobase cần có,

Khoảng đang xét: $[p, q]$,

Ban đầu số lượng nucleobase còn thiếu là r , $p = 0$, $q = 0$.



Đoạn $[p, q]$ có thể chọn nếu số lượng base còn thiếu bằng 0.

Nếu số lượng base còn thiếu lớn hơn 0 – cần tăng q .

Nếu số lượng base còn thiếu lớn bằng 0 – cần tăng p để thu hẹp đoạn có thể chọn.

Tổ chức dữ liệu:

- Mảng `vector<int>` $a(n)$ – lưu dữ liệu mô tả DNA thu được,
- Mảng `vector<int>` $b(k+1, 0)$ – lưu số lượng base theo yêu cầu cần có.

Dộ phức tạp của giải thuật: $O(n)$.

Chương trình

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

int n, k, r, bad, p=0, q=0, ans, u, v;

int main()
{
    fi>>n>>k>>r;
    vector<int> a(n), b(k+1, 0);
    bad=r;
    for(int i=0; i<n; ++i) fi>>a[i];
    for(int i=0; i<r; ++i)
    {
        fi>>u>>v;
        b[u]=v;
    }
    ans=n+1;
    for(p=0; p<n; ++p)
    {
        while(q<n && bad>0)
        {
            --b[a[q]];
            if(b[a[q]]==0) --bad;
            ++q;
        }
        if(bad==0) ans=min(ans, q-p);
        ++b[a[p]];
        bad+=b[a[p]]==1;
    }
    if(ans==n+1) fo<<"impossible"; else fo<<ans;
}
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Việc mua bán hàng qua mạng ngày càng trở nên phổ biến. Một cửa hàng nhận được đơn chuyển hàng tới căn hộ **k1** của một chung cư lớn. Chung cư này là một tòa nhà **m** tầng và có nhiều đơn nguyên, mỗi đơn nguyên có một sảnh vào riêng. Các căn hộ được đánh số từ 1 ở tầng 1, bắt đầu từ đơn nguyên 1 lên dần các tầng trên cho đến hết đơn nguyên, sau đó đánh số tiếp sang đơn nguyên tiếp, theo quy tắc tương tự. Số lượng căn hộ ở mỗi tầng trong đơn nguyên là như nhau và các đơn nguyên đều giống nhau. Các đơn nguyên được đánh số bắt đầu từ 1.

Khi chuẩn bị đi giao hàng người chuyển hàng mới phát hiện ra là không ghi lại đơn nguyên của căn hộ. Tra cứu lại các đơn đặt hàng cũ, người ta tìm thấy một đơn đặt hàng cũng tới chung cư này, nhưng tới căn hộ **k2** với lối vào thuộc đơn nguyên **p2**, tầng **n2**. Thông tin này cũng đủ để xác định đơn nguyên cần vào để tới căn hộ **k1** và tầng của nó trong phần lớn các trường hợp.

Hãy xác định đơn nguyên và tầng của căn hộ **k1**. Nếu đơn nguyên không thể xác định một cách đơn trị thì đưa ra giá trị 0 đối với đơn nguyên.

Dữ liệu: Vào từ file văn bản SHIFT.INP gồm một dòng chứa 5 số nguyên dương **k1**, **m**, **k2**, **p2**, **n2**. Các số có giá trị không vượt quá 10^3 .

Kết quả: Đưa ra file văn bản SHIFT.OUT trên một dòng 2 số nguyên xác định đơn nguyên và tầng của căn hộ **k1**. Trong trường hợp thông tin mâu thuẫn – đưa ra hai số -1.

Ví dụ:

SHIFT.INP	SHIFT.OUT
89 20 41 1 11	2 3



Giải thuật: Địa chỉ tuyệt đối và địa chỉ vật lý.

Số k của căn hộ – ***địa chỉ tuyệt đối***,

Nhóm 3 (*Đơn nguyên, Tầng, Số căn hộ trên tầng*) – ***địa chỉ vật lý***.

Chỉ cần xác định 2 tham số đầu của địa chỉ vật lý.

Gọi **x** – số lượng căn hộ trên mỗi tầng trong một đơn nguyên, ta có:

$$(p_2 - 1) \times m \times x + (n_2 - 1) \times x < k_2$$

x phải là số nguyên lớn nhất thỏa mãn bất đẳng thức trên.

Đặt **t** = $(p_2 - 1) \times m + n_2 - 1$, ta có $t \times x < k_2$.

Nếu **t** = 0 \rightarrow có vô số nghiệm, có thể coi $x \geq \max(k_1, k_2)$, tức là căn hộ cần tìm ở ngay tầng 1. Theo yêu cầu đầu bài – cần đưa ra **p1** = 0.

Nếu **t** $\neq 0$ \rightarrow có $x = (k_2 - 1)/t$. Nếu $x < 0 \rightarrow$ bài toán vô nghiệm.

Trong trường hợp ngược lại: dễ dàng dẫn xuất ra đơn nguyên và từ đó xác định được tầng của căn hộ cần tới.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "shift."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int k1,m,k2,p2,n2,p1,n1,t;

int main()
{
    fi>>k1>>m>>k2>>p2>>n2;
    t=(p2-1)*m+n2-1;
    if(t==0) {p1=0; n1=1; }
    else
    {
        t=(k2-1)/t;
        if(t<=0) {n1=-1; p1=-1; }
        else
        {
            p1 = (k1+t*m-1) / (t*m);
            n1=(k1-(p1-1)*t*m +t-1)/t;
        }
    }
    fo<<p1<<' ' <<n1;
    fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
}
```



Để học sinh làm quen với kỷ luật và tinh thần thi cử các buổi thi cuối học kỳ cũng được tổ chức một cách nghiêm túc và chặt chẽ như các kỳ thi Quốc gia. Lớp học có n dãy bàn đánh số từ 1 đến n từ đầu lớp đến cuối lớp. Mỗi dãy bàn có m chỗ đánh số từ trái sang phải từ 1 trở đi. Cả lớp có $n \times m$ bạn vì vậy không còn chỗ trống.

Ở buổi thi môn Toán cô giáo gọi tên theo danh sách lớp và các bạn vào ngồi lần lượt theo từng hàng từ trái sang phải, hết một hàng – sang hàng tiếp theo. Ở buổi kiểm tra Lý thầy giáo gọi tên theo danh sách lớp và yêu cầu vào ngồi theo cột: vào vị trí 1 của hàng đầu tiên, sau đó – vị trí 1 hàng thứ 2, . . . Sau khi các vị trí 1 ở các hàng đã ngồi hết thì bắt đầu từ vị trí 2 của hàng 1, . . . cứ như thế cho đến khi hết cả lớp.

Ví dụ, với $n = 3$ và $m = 3$, vị trí các bạn ngồi ở hai buổi thi là như sau:

1 2 3

4 5 6

7 8 9

1 4 7

2 5 8

3 6 9

Buổi thi Toán

Buổi thi Lý

Như vậy, trong trường hợp này có 3 bạn vẫn ngồi nguyên vị trí của mình trong cả 2 buổi thi, đó là các bạn số 1, số 5 và số 9.

Trong trường hợp tổng quát, hãy xác định số bạn vẫn được ngồi vị trí cũ trong buổi thi thứ 2.

Dữ liệu: Vào từ file văn bản EXAMINES.INP, gồm một dòng chứa 2 số nguyên n và m ($1 \leq n, m \leq 10^9$).

Kết quả: Đưa ra file văn bản EXAMINES.OUT một số nguyên – số người vẫn ngồi nguyên vị trí của mình trong cả 2 buổi thi.

Ví dụ:

EXAMINES.INP
3 3

EXAMINES.OUT
3



Giải thuật: Phương trình Diophantine.

Phương trình Diophantine tuyến tính bậc 2 có dạng:

$$ax + by = c$$

Phương trình có nghiệm khi c chia hết cho $\text{gcd}(a, b)$.

Nếu phương trình có một nghiệm thì nó sẽ có vô số nghiệm.

Gọi (x_0, y_0) là một nghiệm của phương trình, g là USCLN của a và b .

Các nghiệm còn lại sẽ có công thức:

$$\begin{aligned} & \triangleright x = x_0 + k \times b/g, \\ & \triangleright y = y_0 - k \times a/g, \end{aligned}$$

trong đó k – số nguyên bất kỳ.

Xét vị trí (i, j) .

Ở môn Toán học sinh ngoài ở vị trí (i, j) có số thứ tự trong danh sách là

$$p = (i-1) \times m + j$$

Ở môn Toán học sinh ngoài ở vị trí (i, j) có số thứ tự trong danh sách là

$$q = (j-1) \times n + i$$

Một học sinh ngoài nguyên tại chỗ ở cả 2 môn thi khi $p = q$.

Ta có phương trình $(i-1) \times m + j = (j-1) \times n + i$

$$(m-1) \times i - (n-1) \times j = m - n$$

trong đó i và j là ẩn số, $a = m-1$, $b = 1-n$.

Phương trình này chắc chắn có 2 nghiệm là $(1, 1)$ và (n, m) , vậy nó có vô số nghiệm. Ta chỉ cần tìm số lượng nghiệm với $1 \leq i \leq n$.

Dễ dàng thấy rằng kết quả cần tìm là $| (n-1) / (b/g) | + 1$.

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int main()
{
    fi>>n>>m;
    a=m-1; b=1-n; c=m-n;
    g=__gcd(a,b);
    k=(n-1)*g/b;
    fo<<abs(k)+1;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VX31. SỐ HẠNG

Tên chương trình: ADDENDUM.CPP

Cho số nguyên dương n .

Hãy tìm a và b thỏa mãn các điều kiện:

- ✚ $a > 0, b \geq a,$
- ✚ $a+b = n,$
- ✚ Uớc số chung lớn nhất của a và b là lớn nhất.

Nếu tồn tại nhiều cặp (a, b) thỏa mãn các điều kiện trên thì đưa ra cặp giá trị với a là nhỏ nhất.

Dữ liệu: Vào từ file văn bản ADDENDUM.INP gồm một dòng chứa số nguyên n ($2 \leq n \leq 10^9$).

Kết quả: Đưa ra file văn bản ADDENDUM.OUT trên một dòng 2 số nguyên a và b .

Ví dụ:

ADDENDUM.INP
75

ADDENDUM.OUT
25 50



Giải thuật: Tìm ước nguyên tố nhỏ nhất.

Với $n = a + b$, ước chung của a và b cũng là ước của n ,
 Nếu x là ước của n thì $y = n/x$ cũng là ước của n ,
 Với ước y bao giờ cũng phân tích n thành tổng của a và b , trong đó a và b đều chia hết cho y , ví dụ $a = y, b = n-y$.
 y sẽ là ước lớn nhất của n nếu x là ước nhỏ nhất của n .

Ước nhỏ nhất của n là số nguyên tố nhỏ nhất trong phép phân tích n ra thừa số nguyên tố nếu n là hợp số và bằng n nếu n là nguyên tố.

Để giảm độ phức tạp của giải thuật cần phân biệt 2 trường hợp:

- + n là số chẵn, khi đó $x = 2$,
- + n là số lẻ: cần kiểm tra với $x = 3, 5, 7, \dots, \sqrt{n}$.

Trường hợp n là số nguyên tố: $a = 1, b = n-1$.

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

Chương trình

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

int main()
{
    fi>>n; p=1;
    if((n&1)==0) a=n/2, b=a;
    else
    {
        for(k=3;k*k<=n;k+=2)
            if(n%k==0) {p=k;break;}
        if(p==1) a=1;else a=n/p; b=n-a;
        if(a>b) swap(a,b);
    }
    fo<<a<<' '<<b;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Sao Hỏa đã được chinh phục. Số lượng người lên đó định cư đã khá đông và cuộc sống đã di dời vào luồng phát triển bình thường của nó. Người ta bắt đầu quan tâm đến khảo cổ.

Trong một hang động sâu có rất nhiều dấu vết của một nền văn minh nào đó, hoặc của cư dân cổ xưa của sao Hỏa hoặc của một nền văn minh nào đó đã đến sao Hỏa trước chúng ta.

Có một dòng ký tự dài khác trên đá, trong đó dãy các ký tự cuối cùng đều giống nhau. Có thể đây là chìa khóa giải mã để đọc các thông báo khác. Dòng ký tự này có vẻ như là giá trị $n!$ ($n! = 1 \times 2 \times 3 \times \dots \times (n-1) \times n$) được tính theo một cơ số nào đó.

Để kiểm tra giả thuyết này, người ta cần xác định với một số nguyên dương n giá trị $n!$ sẽ có bao nhiêu số 0 ở cuối nếu nó được tính theo cơ số b . Ví dụ, nếu $n = 100$ thì với $b = 10$, $n!$ sẽ có 24 số 0 ở cuối, còn với $b = 6$ – sẽ có 48 số 0 ở cuối.

Cho n và b . Hãy xác định số lượng số 0 ở cuối của $n!$ trong dạng biểu diễn theo cơ số b .

Dữ liệu: Vào từ file văn bản ZEROS.INP gồm một dòng chứa 2 số nguyên n và b ($1 \leq n \leq 10^9$, $2 \leq b \leq 10^3$).

Kết quả: Đưa ra file văn bản ZEROS.OUT một số nguyên – số lượng số 0 tìm được.

Ví dụ:

ZEROS.INP
100 6

ZEROS.OUT
48



Giải thuật: Phân tích ra thừa số nguyên tố.

Để xác định $n!$ có bao nhiêu số 0 ở cuối trong dạng biểu diễn cơ số b cần phân tích $n! = m \times b^u$, trong đó m không chia hết cho b .

Cần phân tích b ra thừa số nguyên tố:

$$b = p_1^{q_1} \times p_2^{q_2} \times \dots \times p_v^{q_v}$$

trong đó p_1, p_2, \dots, p_v – các số nguyên tố.

Giá trị b cần xét không lớn vì vậy có thể duyệt các số nguyên từ 2 đến \sqrt{b} để xác định các p_i và q_i tương ứng.

Với mỗi số nguyên tố p_i cần xác định c_i – số lần p_i tham gia vào $n!$.

Số lượng số 0 cần tìm sẽ là $\min_{1 \leq i \leq v} \{c_i / q_i\}$.

Ví dụ: Với $n = 100$, $b = 6$ ta có:

$$b = 6 = 2 \times 3$$

Trong 100 số tự nhiên đầu tiên có $100/3 = 33$ số chia hết cho 3,

Trong đó có $33/3 = 11$ số chia hết cho 3^2 ,

Trong đó có $11/3 = 3$ số chia hết cho 3^3 ,

Trong đó có $3/3 = 1$ số chia hết cho 3^4 .

Như vậy nếu phân tích $100!$ ra thừa số nguyên tố sẽ có thừa số 3 với số mũ bằng $33 + 11 + 3 + 1 = 48$.

Tương tự, thừa số 2 sẽ tham gia vào $100!$ với số mũ là $50 + 25 + 12 + 6 + 3 + 1 = 97$.

Số lượng số 0 ở cuối của $100!$ Trong cơ số 6 sẽ là $\min\{97/1, 48/1\} = 48$.

Độ phức tạp của giải thuật: $O(\sqrt{b})$.

Chương trình

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

int calc(int x)
{
    int t=a[x],u=0,v=n;
    while(v>1) v/=t, u+=v;
    return u;
}

int main()
{
    fi>>n>>k;
    p=2;
    while(p*p<= k)
    {
        if(k%p==0)
        {
            q=0;
            a.push_back(p);
            while(k%p==0) k/=p,++q;
            b.push_back(q);
        }
        ++p;
    }
    if(k>1){a.push_back(k); b.push_back(1);}

    for(int i=0; i<a.size(); ++i)
        c.push_back(calc(i));
    ans=n;
    for(int i=0; i<a.size(); ++i)
        ans=min(ans,c[i]/b[i]);
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VX33. LÝ THUYẾT NHỎ NHẤT

Tên chương trình: MINPOWER.CPP

Việc kiểm tra đánh giá kỹ năng của học sinh không đòi hỏi phải dùng những bài tập mới lạ, ngược lại, chính những dạng bài quen thuộc lại cho kết quả đánh giá sát thực hơn. Chính vì vậy, trong buổi học đầu tiên của học sinh mới vào trường cô giáo luôn ra bài cùng một dạng: “*Cho số nguyên dương k . Hãy xác định số nguyên dương n nhỏ nhất sao cho n^n chia hết cho k .*” Để hạn chế quay còp, mỗi học sinh trong lớp nhận được một số k khác nhau.

Để chấm bài được nhanh và chính xác cần có bảng kết quả tương ứng với các số đã cho trên lớp.

Hãy xác định kết quả cần có với k cho trước.

Dữ liệu: Vào từ file văn bản MINPOWER.INP gồm một dòng chứa số nguyên k ($1 \leq k \leq 10^9$).

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

Ví dụ:

MINPOWER.INP
24

MINPOWER.OUT
6



Giải thuật: Phân tích ra thừa số nguyên tố.

Điều kiện cần để n^n chia hết cho k là n phải là tích các ước nguyên tố của k .

Điều kiện đủ để n^n chia hết cho k là mỗi ước nguyên tố tham gia vào n^n với số mũ lớn hơn hoặc bằng số mũ của nó trong k .

Xây dựng n thỏa mãn điều kiện cần:

- ⊕ Phân tích k ra thừa số nguyên tố và lưu bậc của mỗi thừa số tham gia vào k ,
- ⊕ Tính n : tích của các thừa số nguyên tố khác nhau nhận được ở bước trước.

Kiểm tra kết quả và chỉnh lý n theo điều kiện đủ:

Theo cách xây dựng trên mỗi thừa số nguyên tố sẽ tham gia vào n^n với bậc là n ,

Gọi (a_i, b_i) tương ứng là thừa số nguyên tố và bậc của trong phân tích số k ,

Nếu n nhỏ hơn b_i thì lặp lại việc thay n bằng $n \times a_i$ cho đến khi nhận được $n \geq b_i$,

Thực hiện việc kiểm tra và tăng n nói trên với mọi i .

Giá trị cần xét của k không quá lớn nên không cần thiết phải áp dụng các thuật toán phức tạp phân tích k ra thừa số nguyên tố.

Tổ chức dữ liệu:

- ─ Mảng `vector<int>` a – lưu các thừa số nguyên tố khi phân tích k ,
- ─ Mảng `vector<int>` b – lưu bậc của thừa số nguyên tố tương ứng trong phân tích k .

Độ phức tạp của giải thuật: O(\sqrt{k}).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "minpower."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int k,p,q,ans=1;
vector<int>a,b;

int main ()
{
    fi>>k;
    p=2;
    while (p*p<= k)
    {
        if (k%p==0)
        {
            q=0;
            a.push_back (p);
            while (k%p==0) k/=p, ++q;
            b.push_back (q);
        }
        ++p;
    }
    if (k>1) {a.push_back (k); b.push_back (1);}

    p=1;
    for (int i:a) ans*=i;
    for (int i=0; i<b.size(); ++i)
    {
        if (p<b[i]) p=b[i], q=i;
        while (ans<p) ans*=a[q];
    }

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



VX34. ĐIỂM NGUYÊN

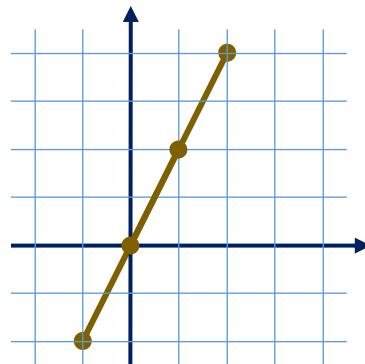
Tên chương trình: DOTS.CPP

Cho đoạn thẳng trên mặt phẳng có tọa độ các điểm đầu, cuối là nguyên và là (x_1, y_1) và (x_2, y_2) .

Hãy xác định số điểm có tọa độ nguyên nằm trên đoạn thẳng.

Dữ liệu: Vào từ file văn bản DOTS.INP gồm một dòng chứa 4 số nguyên x_1, y_1, x_2 và y_2 , các số có giá trị tuyệt đối không quá 10^8 .

Kết quả: Đưa ra file văn bản DOTS.OUT một số nguyên – số điểm có tọa độ nguyên nằm trên đoạn thẳng đã cho.



Ví dụ:

DOTS.INP
-1 -2 2 4

DOTS.OUT
4



VX34Theory AXIII

Giải thuật: Phương trình Diophantine.

Phương trình Diophantine tuyến tính bậc 2 có dạng:

$$ax + by = c$$

Phương trình có nghiệm khi c chia hết cho $\gcd(a, b)$.

Nếu phương trình có một nghiệm thì nó sẽ có vô số nghiệm.

Gọi (x_0, y_0) là một nghiệm của phương trình, g là USCLN của a và b .

Các nghiệm còn lại sẽ có công thức:

$$\begin{aligned} & \triangleright x = x_0 + k \times b/g, \\ & \triangleright y = y_0 - k \times a/g, \end{aligned}$$

trong đó k – số nguyên bất kỳ.

Phương trình đường thẳng đi qua 2 điểm $(x_1, y_1), (x_2, y_2)$ có dạng:

$$\begin{aligned} \frac{x - x_1}{x_2 - x_1} &= \frac{y - y_1}{y_2 - y_1} \\ (x - x_1) \times (y_2 - y_1) &= (y - y_1) \times (x_2 - x_1) \\ (y_2 - y_1) \times x - (x_2 - x_1) \times y &= x_1 \times (y_2 - y_1) - y_1 \times (x_2 - x_1) \end{aligned}$$

Ta có phương trình Diophantine với $a = y_2 - y_1$, $b = -(x_2 - x_1)$.

Ngoài ra, ta có 2 nghiệm phương trình là (x_1, y_1) và (x_2, y_2) .

Ký hiệu $g = \gcd(a, b)$ ta có số điểm tọa độ nguyên trên đoạn thẳng là

$$| (x_2 - x_1) \times g / b | + 1$$

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "dots."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t x1, x2, y1, y2, g, a, b, c, k;

int main()
{
    fi>>x1>>y1>>x2>>y2;
    a=y2-y1; b=x1-x2;
    if(a==0 || b==0) k=abs(a)+abs(b);
    else k=abs(__gcd(a,b));
    fo<<k+1;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



VX35. SỐ CÁCH BỎ SUNG

Tên chương trình: INSNUM.CPP

Khác với biểu thức toán học bình thường, trong các ngôn ngữ lập trình biểu thức toán học phải viết tuyển tính trên dòng và chỉ dùng loại cặp ngoặc tròn để xác định trình tự thực hiện các phép tính trong biểu thức.

Alice nǎm rất nhanh nguyên tắc viết này. Trong lúc thầy giáo giảng giải chi tiết và minh họa cách viết biểu thức Alice thấy nhầm chán và bắt đầu tưởng tượng ra đủ mọi thứ. Từ biểu thức viết trên bảng $(2+2) / (3-(5-2)+4)$ Alice bỏ hết các toán hạng cũng như dấu phép tính và nhận được xâu ký tự $()()$.

Đi ngang qua chỗ Alice, thầy giáo liếc nhìn thấy xâu ngoặc và biết rằng với những học sinh năng khiếu như Alice, nội dung này quá đơn giản. Thầy giáo giải thích cho Alice xâu này được gọi là một biểu thức ngoặc đúng và đề xuất Alice tính số lượng cách bỏ sung khác nhau một ngoặc mở và một ngoặc đóng để vẫn nhận được biểu thức ngoặc đúng. Lưu ý một biểu thức ngoặc là đúng khi có số lượng ngoặc mở bằng số lượng ngoặc đóng và số lượng ngoặc mở gấp khi duyệt từ trái sang phải bao giờ cũng lớn hơn hoặc bằng số lượng ngoặc đóng đã gặp.

Hai cách bỏ sung gọi là khác nhau nếu vị trí của ngoặc bỏ sung khác nhau hoặc có cùng vị trí nhưng ngoặc viết vào đó là khác nhau. Ví dụ, với biểu thức ngoặc ban đầu là $()$ ta có 7 cách bỏ sung một cặp ngoặc: $(())$, $((())$, $((()()$, $((()()$, $((()()$, $(()())$, $(()()$.

Cho xâu s là một biểu thức ngoặc đúng. Hãy xác định số cách khác nhau bỏ sung một cặp ngoặc để vẫn nhận được biểu thức ngoặc đúng,

Dữ liệu: Vào từ file văn bản INSNUM.INP gồm một dòng chứa xâu s độ dài không quá 50000.

Kết quả: Đưa ra file văn bản INSNUM.OUT một số nguyên – số cách bỏ sung khác nhau tính được.

Ví dụ:

INSNUM.INP	INSNUM.OUT
$()()$	17



Giải thuật: Thống kê đơn giản.

Gọi n là độ dài biểu thức ngoặc đúng s đã cho, các ký tự đánh số bắt đầu từ 0.

Có 2 cách bổ sung một cặp ngoặc vào biểu thức ngoặc đúng để vẫn nhận được biểu thức ngoặc đúng:

- + Vị trí bổ sung ngoặc ')' ở *sau* vị trí bổ sung ngoặc '(',
- + Vị trí bổ sung ngoặc ')' ở *trước* vị trí bổ sung ngoặc '('.

Trường hợp 1:

- Gọi vị trí bổ sung ngoặc ')' là i (tức là *bổ sung ngay trước ký tự s_i*), ký tự ngoặc đóng ')' có thể bổ sung vào bất kỳ vị trí nào trong số các vị trí *từ i tới n* , như vậy ta có $n-i+1$ cách bổ sung ngoặc đóng,
- Có $n+1$ vị trí bổ sung ngoặc mở (từ 0 đến $n-1$ và bổ sung sau s),
- Như vậy, số lượng $s_{1:n}$ cách bổ sung ngoặc đóng sau ngoặc mở sẽ là:

$$s_{1:n} = \sum_{i=0}^n (n - i + 1) = \frac{(n+1) \times (n+2)}{2}$$

Trường hợp 2:

Xét tổng tích lũy ngoặc mở f_j :

$$f_0 = 1,$$

$$f_j = \begin{cases} f_{j-1} + 1, & \text{nếu } s_j = ')' \\ f_{j-1} - 1, & \text{nếu } s_j = '(' \end{cases} \quad j > 0$$

Các biến a_1, a_2, \dots lưu chỉ số j với $f_j = 0$, theo thứ tự tăng dần của j .

Xét khoảng đầu tiên, từ 0 đến a_1 . Ký tự ')' *không thể bổ sung ở đầu khoảng* mà chỉ có thể từ vị trí 1 cho đến a_1 . Nếu ký tự đóng ngoặc được bổ sung ở vị trí m ($1 \leq m \leq a_1$) thì ký tự mở ngoặc có thể bổ sung ở một trong số các vị trí $m, m+1, \dots, a_1$ và *không thể muộn hơn!*

Bức tranh bổ sung với các khoảng còn lại cũng tương tự.

Như vậy, từ a_{i-1} đến a_i có $a_i - a_{i-1} - 1$ vị trí cho phép bổ sung ngoặc đóng và số lượng cách bổ sung sẽ là $k \times (k-1) / 2$ với $k = a_i - a_{i-1}$. *Công thức này vẫn đúng với $i = 1$ nếu đặt $a_0 = -1$.*

Số lượng cách bổ sung hợp lệ cho trường hợp 2 là tổng số cách ở mỗi khoảng xác định bởi các a_i .

Tổ chức dữ liệu: vector<int> a với vai trò và ý nghĩa đã nêu.

Độ phức tạp của giải thuật: O(n).

Chương trình

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

string s;
int n, k=0;
vector<int> a;
int64_t ans;

int main()
{
    fi>>s; n=s.size();
    a.push_back(-1);
    for(int i=0; i<n; ++i)
        if(s[i]==')'+k;
        else
        {
            --k;
            if(k==0) a.push_back(i);
        }
    ans=(int64_t)(n+1)*(n+2)/2;
    for(int i=1; i<a.size(); ++i)
    {
        k=a[i]-a[i-1];
        ans+=(int64_t)(k)*(k-1)/2;
    }
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VX36. SỐ LỚN NHẤT

Tên chương trình: MAXNUM.CPP

Cho số nguyên dương n có k bit có nghĩa trong dạng biểu diễn ở cơ số 2.

Ví dụ $n = 19_{10} = 10011_2$, số 19_{10} có 5 bit có nghĩa.

Các bit trong một số được đánh số từ 0 bắt đầu từ phải sang trái:

Bít 4 3 2 1 0
 $19_{10} = \underline{1} \text{ } \underline{0} \text{ } \underline{0} \text{ } \underline{1} \text{ } \underline{1}$

Xét các số nhận được bằng cách đẩy vòng sang phải k bit của n , tức là các bit từ vị trí $k-1$ đến 1 dịch sang phải một vị trí, bit vị trí 0 – về vị trí $k-1$.

Rõ ràng kết quả đẩy vòng cho tối đa k số khác nhau.

Kết quả đẩy vòng số 19_{10} cho các số:

1 0 0 1 1
0 0 1 1 1
0 1 1 1 0
1 1 1 0 0
1 1 0 0 1

Hãy xác định số lớn nhất nhận được theo cách đẩy vòng nêu trên đối số n .

Dữ liệu: Vào từ file văn bản MAXNUM.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 10^9$).

Kết quả: Đưa ra file văn bản MAXNUM.OUT số nguyên lớn nhất nhận được.

Ví dụ:

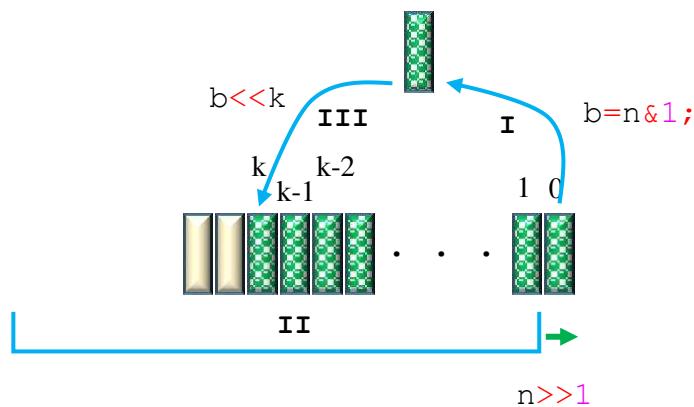
MAXNUM.INP	MAXNUM.OUT
19	28



Giải thuật: Xử lý bit, vét cạn.

Chỉ các bít có nghĩa của n tham gia vào các phép đầy vòng bít,

Gọi k là vị trí bít có nghĩa trái nhất của n , sẽ có không quá $k+1$ giá trị khác nhau do kết quả đầy vòng sinh ra,



k không vượt quá 31 → có thể trực tiếp so sánh các giá trị nhận được để tìm \max .

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int main ()
{
    fi>>n;
    ans=n;
    for(int i=31; i>=0; --i)
        if((n>>i) & 1) {k=i; break;}
    for(int i=0; i<k; ++i)
    {
        b=n&1;
        n=(b<<k) | (n>>1);
        ans=max(ans, n);
    }

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



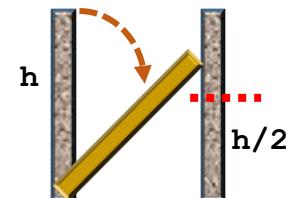
VX37. HIỆU ỨNG ĐÔ MI NÔ

Tên chương trình: DOMINO.CPP

Xét bộ quân đô mi nô, mỗi quân có dạng hình hộp chữ nhật độ cao h , bè dày không đáng kể (chiều rộng không đóng vai trò quan trọng). Các quân đô mi nô được đặt dựng đứng giống như xếp sách trên giá.

Khi người ta đẩy một quân đô mi nô đổ sang phải, nếu đầu trên của quân đô mi nô này chạm vào quân bên phải ở vị trí cao hơn hoặc bằng $h/2$, quân bên phải cũng sẽ bị đổ theo.

Việc làm đổ một quân đô mi nô gây ra hiệu ứng đổ dây chuyền các quân đô mi nô khác được gọi là hiệu ứng đô mi nô.



Ở vị trí 0 và r có đặt quân đô mi nô.

Hãy xác định số lượng ít nhất các quân đô mi nô cần đặt thêm để khi làm đổ sang phải quân đô mi nô ở vị trí 0, quân đô mi nô ở vị trí r cũng sẽ bị đổ.

Dữ liệu: Vào từ file văn bản DOMINO.INP gồm một dòng chứa 2 số nguyên h và r ($1 \leq h, r \leq 10^6$).

Kết quả: Đưa ra file văn bản DOMINO.OUT một số nguyên – số quân đô mi nô cần đặt thêm.

Ví dụ:

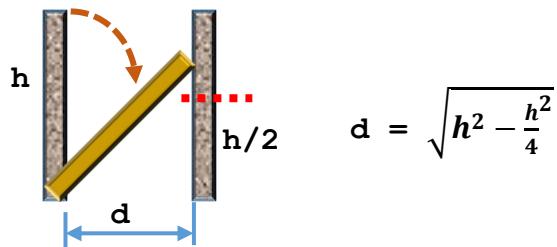
DOMINO.INP	DOMINO.OUT
2 5	4



VX37 Io20180513 AE AXIII

Giải thuật: Ép kiểu dữ liệu.

Gọi d – khoảng cách lớn nhất giữa 2 quân đô mi nô để đảm bảo hiệu ứng đỗ dây chuyền.



Độ phức tạp của giải thuật: O(1).

Chương trình

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

int r,h,d,ans;

int main()
{
    fi>>h>>r;
    d=(int)(h*sqrt((double)0.75));
    ans=r/d;
    if(ans>0)--ans;
    fo<<ans;

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



VX38. KHÔNG CÓ 9

Tên chương trình: WITHOUT_9.CPP

Bob rất bực vì phải đi dự một cuộc họp chả liên quan gì tới công việc của mình mà chỉ vì cần có đủ thành phần. Cuộc họp bắt đầu từ 9 giờ. Để giết thời gian và đồng thời giải tỏa úc ché tâm lý Bob ngồi ghi vào trong sổ của mình các số tự nhiên liên tiếp nhau, bắt đầu từ **a**, trong đó không có số nào chứa chữ số 9. Cuộc họp kết thúc khi Bob ghi đến số **b** trong sổ.

Hãy xác định có bao nhiêu số Bob đã ghi lại trong suốt buổi họp.

Dữ liệu: Vào từ file văn bản WITHOUT_9.INP gồm một dòng chứa 2 số nguyên **a** và **b**. Dĩ nhiên cả 2 số này đều không chứa chữ số 9 ($1 \leq a \leq b \leq 10^{15}$).

Kết quả: Đưa ra file văn bản WITHOUT_9.OUT một số nguyên – số lượng số đã được viết.

Ví dụ:

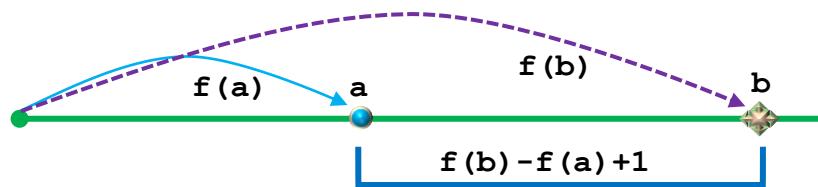
WITHOUT_9.INP	WITHOUT_9.OUT
80 100	10



Giải thuật: Tổng tiền tố.

Các số được viết là các số nguyên dương trong cơ số 9.

Gọi $f(x)$ là số lượng các số không chứa chữ số 9 trong khoảng từ 0 đến x .



Lưu ý: không tính theo công thức $f(b) - f(a-1)$ vì $a-1$ có thể chứa chữ số 9.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "without_9."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;
int64_t fa=0, fb=0, ans ;
string a,b;

int main ()
{
    fi>>a>>b;
    n=a.size();
    for(int i=0; i<n; ++i) fa=fa*9+a[i]-48;
    n=b.size();
    for(int i=0; i<n; ++i) fb=fb*9+b[i]-48;
    ans=fb-fa+1;
    fo<<ans;

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



Bộ phận chông buôn lậu và làm hàng giả chặn được một thông báo đã mã hóa của nhóm nghi can đang bị theo dõi. Chìa khóa giải mã thông báo nằm ở đoạn thông báo ngày tháng năm và có dạng **YYYY MM DD**, trong đó **Y, M, D** là các ký tự số. Khóa để giải mã là một ngày, tháng năm đúng nhận được bằng cách đổi các ký tự số trong đoạn nói trên.

Một ngày tháng năm là đúng nếu 3 số nhận được đều lớn hơn không và tương ứng với một ngày trong lịch đang dùng. Với năm nhuận tháng 2 có 29 ngày. Năm nhuận là năm chia hết cho 400 hoặc không chia hết cho 100 nhưng chia hết cho 4. *Riêng những năm lớn hơn 0 và là bội của 3328 là những năm nhuận đặc biệt, tháng 2 sẽ có 30 ngày.*

Cho đoạn thông báo tách được. Hãy xác định số lượng ngày tháng hợp lý có thể rút ra từ thông báo và đưa ra các ngày tháng đó theo thứ tự tăng dần trên lịch. Ngày tháng năm đưa ra dưới dạng **YYYY MM DD** (xem ví dụ).

Dữ liệu: Vào từ file văn bản DATES.INP gồm một dòng chứa xâu 10 ký tự dạng

YYYY MM DD.

Kết quả: Dưa ra file văn bản DATES.OUT:

- ✚ Dòng đầu tiên chứa một số nguyên – số lượng ngày tháng hợp lệ,
- ✚ Mỗi dòng trong các dòng sau – một ngày tháng hợp lệ tìm được. Thông tin đưa ra theo thứ tự tăng dần trên lịch.

Ví dụ:

DATES.INP	DATES.OUT
0001 01 01	16
	0001 01 01
	0001 01 10
	0001 10 01
	0001 10 10
	0010 01 01
	0010 01 10
	0010 10 01
	0010 10 10
	0100 01 01
	0100 01 10
	0100 10 01
	0100 10 10
	1000 01 01
	1000 01 10
	1000 10 01
	1000 10 10



Giải thuật: Vết cạn, kỹ năng khai thác hệ thống lập trình.

Mỗi ngày tháng có thể nhận được bằng cách đổi chỗ các ký tự số,
Có tất cả 8 ký tự số → tối đa có $8! = 40320$ khả năng khác nhau.

Tạo thông tin ngày tháng có thể:

- ✚ Lưu các chữ số vào mảng `int a[8]`,
- ✚ Sắp xếp a theo thứ tự tăng dần,
- ✚ Kiểm tra tính hợp lệ của ngày tháng nhận được và ghi nhận nếu hợp lệ,
- ✚ Chuyển sang hoán vị tiếp theo bằng các gọi hàm `next_permutation(a, a+8)`

Kiểm tra tính hợp lệ của ngày tháng:

- ✚ Ngày, tháng, năm phải lớn hơn 0,
- ✚ Kiểm tra năm nhuận và loại năm nhuận,
- ✚ Kiểm tra tính hợp lệ của ngày trong tháng.

Để thuận tiện kiểm tra tính hợp lệ của ngày trong tháng nên tổ chức mảng lưu trữ số ngày trong mỗi tháng.

Lưu ý:

- ✓ Theo lịch Grigory đang dùng, cứ 3328 năm cần bù thêm một ngày và việc bù thêm – bổ sung vào tháng 2, như vậy tháng 2 sẽ có 30 ngày. Tuy nhiên, quy tắc bù này *chưa được chính thức thông qua*.
- ✓ Cách đưa ra dữ liệu với các số 0 không có nghĩa và với độ rộng trường xác định: các định hướng đưa ra `setw()` và `setfill()`,
- ✓ Mỗi đơn vị dữ liệu đưa ra là (*Năm, Tháng, Ngày*) – *có thể* lưu trữ theo kiểu `tuple<int, int, int>`,
- ✓ Vì cần đưa số lượng trước khi chỉ ra các ngày tháng cụ thể nên cần lưu lại ngày tháng hợp lệ vào vector hoặc tập hợp, việc dùng tập hợp sẽ an toàn hơn, dữ liệu đảm bảo được sắp xếp khi đưa ra,
- ✓ Các truy cập dữ liệu từ kiểu `tuple`: có thể dùng `get<p>()` hoặc thông qua chỉ thị `tie()`.

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int dd[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int y,m,d,a[8];
string sa,sb,sc;
set<tuple<int,int,int>>s;

bool check()
{
    int t;
    if(y==0 || m==0 || m>12 || d==0 || d>31) return 0;
    if(y>0 && y%3328==0) t=2; else t = (y%400==0) || (y%100!=0 && y%4==0);
    if(m!=2 && d>dd[m-1]) return 0;
    if(m==2 && d>28+t) return 0;
    return 1;
}

int main()
{
    fi>>sa>>sb>>sc;
    sa+=sb+sc;
    for(int i=0; i<8; ++i) a[i]=sa[i]-48;
    sort(a,a+8);
    do
    {
        y= ( (a[0]*10+a[1])*10+a[2])*10+a[3];
        m=a[4]*10+a[5];
        d=a[6]*10+a[7];
        if(check()) s.insert(make_tuple(y,m,d));
    }
    while(next_permutation(a,a+8));

    fo<<s.size()<<'\n';
    for(auto i:s) fo<<setw(4)<<setfill('0')<<get<0>(i)<<' '
    <<setw(2)<<setfill('0')<<get<1>(i)<<' '
    <<setw(2)<<setfill('0')<<get<2>(i)<<'\n';

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



Trên trạm không gian việc thoát hiểm phức tạp hơn nhiều so với việc thoát hiểm khi có cháy ở các nhà cao tầng trên mặt đất. Các nhà nghiên cứu trên trạm phải chạy tới cửa thoát hiểm, mặc áo giáp bảo hộ sau đó di chuyển tới tàu cứu hộ.

Hành lang thoát hiểm có thể coi như một đường thẳng. Dọc hành lang có lắp n thùng chứa áo giáp bảo hộ, thùng thứ i ở tọa độ x_i và chứa a_i bộ áo, $i = 1 \dots n$. Khi có tín hiệu báo động các thùng áo sẽ tự động tách ra khỏi chỗ được lắp và di chuyển về vị trí cửa thoát hiểm.

Năng lượng di chuyển các thùng áo giáp tỷ lệ với đường đi và số bộ áo trong thùng. Như vậy, để giảm thiểu năng lượng tập kết áo bảo hộ cần chọn vị trí đặt cửa thoát hiểm ở nơi sao cho tất cả áo bảo hộ được tập trung đầy đủ với chi phí năng lượng nhỏ nhất.

Hãy xác định vị trí đặt cửa thoát hiểm. Nếu tồn tại nhiều cách chọn – đưa ra cách chọn với tọa độ nhỏ nhất.

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

- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 100$, $i = 1 \dots n$),
- ✚ Dòng thứ 3 chứa n số nguyên x_1, x_2, \dots, x_n ($1 \leq x_i \leq 100$, $x_i < x_{i+1}$, $i = 1 \dots n$).

Kết quả: Đưa ra file văn bản EMERGENCY.OUT một số nguyên – tọa độ cửa thoát hiểm.

Ví dụ:

EMERGENCY.INP
4
1 4 1 1
1 2 3 3

EMERGENCY.OUT
2



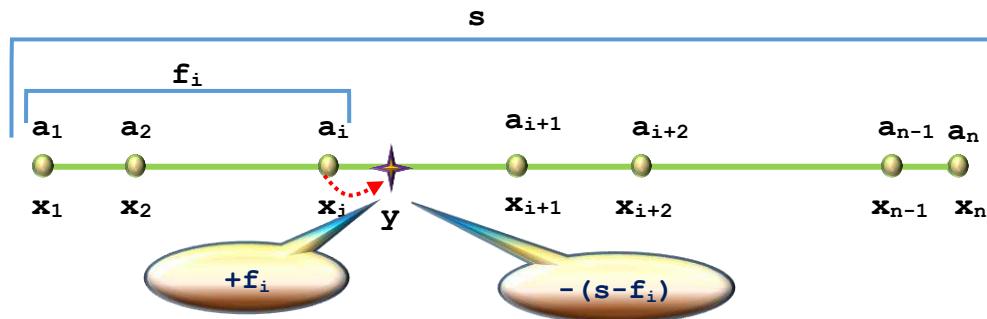
VX40 Io20180513 BC AXIII

Giải thuật: Tổng tiền tố.

Ký hiệu:

$$\mathbf{s} = \sum_{i=1}^n a_i, \mathbf{f}_i = \sum_{j=1}^i a_j.$$

Giả thiết cửa thoát hiểm được chọn ở điểm có tọa độ \mathbf{y} , $\mathbf{x}_i \leq \mathbf{y} < \mathbf{x}_{i+1}$ và chi phí năng lượng tập kết là \mathbf{d} .



Nếu chuyển điểm chọn sang phải một đơn vị, giá trị \mathbf{d} sẽ thay đổi: tăng thêm một lượng là \mathbf{f}_i và giảm đi $(\mathbf{s} - \mathbf{f}_i)$.

Dễ dàng thấy rằng \mathbf{y} nên chọn trùng với một trong số các \mathbf{x}_i .

Như vậy chỉ cần duyệt với mọi \mathbf{x}_i , xử lý nó như một điểm được chọn, so sánh các kết quả để xác định min. Ban đầu $\mathbf{d} = \sum_{i=1}^n (x_i - x_1) \times a_i$. Khi chuyển sang điểm mới – cập nhật lại \mathbf{d} theo công thức nêu trên với bước chuyển không phải 1 mà là $\mathbf{x}_i - \mathbf{x}_{i-1}$.

Kết hợp quá trình nhập dữ liệu với việc tính s và d ban đầu. Giá trị f_i có thể cập nhật khi chuyển sang điểm mới và như vậy – không cần lưu trữ dưới dạng mảng.

Độ phức tạp của giải thuật: O(n).

Chương trình

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

int main()
{
    int n;
    fi >> n;
    std::vector<int> x(n), a(n);
    for (int i = 0; i < n; i++) fi >> a[i];
    int64_t d = 0, s = 0;
    for (int i = 0; i < n; i++)
    {
        fi >> x[i];
        d += (x[i] - x[0]) * a[i];
        s += a[i];
    }
    int64_t mind = d, f = a[0], ans = x[0];
    for (int i = 1; i < n; i++)
    {
        d += (x[i] - x[i - 1]) * f;
        d -= (x[i] - x[i - 1]) * (s - f);
        if (d < mind)
        {
            mind = d;
            ans = x[i];
        }
        f += a[i];
    }
    fo << ans << '\n';
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VX41. NHÀ HÀNG

Tên chương trình: RESTAURANT.CPP

Ba bạn Alice, Bob và Catherine cùng đi du lịch mùa hè theo chương trình giá rẻ giành cho học sinh – sinh viên. Đã thấm mệt sau khi thăm các thắng cảnh của thành phố ba bạn quyết định chọn một cửa hàng để ăn trưa. Bản đồ du lịch giới thiệu n nhà hàng, nhưng mỗi bạn có một sở thích khác nhau. Alice thích nhà hàng có nhiều loại kem, Bob thích ăn pizza còn Catherine thích nơi có nhiều đồ uống.

Các bạn quyết định sẽ chọn nhà hàng phù hợp nhất với cả ba, tức là nếu chọn bất kỳ nhà hàng nào khác thì cũng có ít nhất 2 người không thích bằng nhà hàng ban đầu.

Hãy xác định nhà hàng được chọn hay cho biết không tồn tại một nhà hàng như vậy.

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

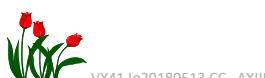
- ✚ Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa hoán vị các số nguyên từ 1 đến n xác định ưu tiên nhà hàng theo sở thích của Alice,
- ✚ Dòng thứ 3 chứa hoán vị các số nguyên từ 1 đến n xác định ưu tiên nhà hàng theo sở thích của Bob,
- ✚ Dòng thứ 4 chứa hoán vị các số nguyên từ 1 đến n xác định ưu tiên nhà hàng theo sở thích của Catherine.

Kết quả: Đưa ra file văn bản RESTAURANT.OUT một số nguyên – nhà hàng được chọn hoặc đưa ra số -1 nếu không thể chọn nhà hàng theo tiêu chuẩn đã nêu.

Ví dụ:

RESTAURANT.INP
4
2 1 3 4
3 1 4 2
4 1 2 3

RESTAURANT.OUT
1



Giải thuật: Phân tích tình huống lô gic.

Xây dựng mảng $a[n][3]$, trong đó $a_{i,j}$ là thứ tự ưu tiên lựa chọn nhà hàng i của người thứ j .

Từ mảng này ta có thể tìm được nhà hàng có 2 người thích hơn các nhà hàng khác (*Điều kiện cần*).

Duyệt lại theo mảng ưu tiên lựa chọn, kiểm tra xem nhà hàng tìm được có đáp ứng yêu cầu đã nêu trong đầu bài hay không (*Điều kiện đủ*).

Nếu điều kiện đủ không thỏa mãn – bài toán vô nghiệm.

Tổ chức dữ liệu: mảng $a[n][3]$, với vai trò, ý nghĩa đã nêu.

Xử lý:

Hàm kiểm tra nhà hàng y có được ít hơn 2 người yêu thích hơn nhà hàng x :

```
bool f(int x, int y)
{
    int val = 0;
    for(int i=0; i<3; ++i) val += a[x][i] < a[y][i];
    return val >= 2;
}
```

Tích lũy số
người thích hơn

Xác định nhà hàng theo điều kiện cần và kiểm tra điều kiện đủ:

```
for (int i = 1; i < n; ++i) if (f(i, res)) res = i;

for(int i=0; i<n; ++i)
    if (i != res && !f(res, i))
    {
        res = -1;
        break;
    }

```

Tìm theo điều
kiện cần

Kiểm tra điều
kiện đủ

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "restaurant."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int a[100001][3];

bool f(int x, int y)
{
    int val = 0;
    for(int i=0; i<3; ++i) val += a[x][i] < a[y][i];
    return val >= 2;
}

int main()
{
    int n,x;
    fi>>n;
    for(int i=0; i<3; ++i)
        for(int j=0; j<n; ++j)
        {
            fi>>x;
            a[x - 1][i] = j;
        }
    int res = 0;
    for (int i = 1; i < n; ++i) if (f(i, res)) res = i;
    for(int i=0; i<n; ++i)
        if (i != res && !f(res, i))
        {
            res = -1;
            break;
        }
    if (res >= 0) res++;
    fo<<res;

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



Thị trường bán lẻ đang là một lĩnh vực có tiềm năng lớn thu hút sự chú ý của nhiều nhà đầu tư. Tuy vậy việc đảm bảo cho một siêu thị hay cửa hàng bán lẻ hoạt động có hiệu quả không phải là chuyện đơn giản. Lợi nhuận mỗi ngày của cửa hàng là tổng số tiền bán hàng thu được trong ngày trừ tiền vốn mua hàng, tiền thuê mặt bằng, chi phí nhân công và điện nước, . . .

Một siêu thị trong chi nhánh bán lẻ mở ở một địa điểm mới và đã hoạt động được n ngày, ngày thứ i có lợi nhuận là a_i , a_i có thể là dương, âm hoặc bằng 0, $i = 1 \div n$.

Giám đốc điều hành hệ thống bán lẻ muốn thu thập số liệu trong khoảng thời gian từ ngày i đến hết ngày j ($i \leq j$) có tổng lợi nhuận là lớn nhất và lớn hơn 0 để phân tích và rút kinh nghiệm chỉ đạo. Thông tin đầu tiên ông muốn biết là tổng lợi nhuận trong khoảng thời gian đó.

Hãy đưa ra tổng tìm được hoặc một số 0 nếu không có ngày nào có lợi nhuận dương.

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

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

Kết quả: Đưa ra file văn bản BEST.OUT tổng tìm được hoặc một số 0 nếu không có ngày nào có lợi nhuận dương.

Ví dụ:

BEST.INP
5
12 -4 -10 4 9

BEST.OUT
13



Giải thuật: Tổng tiền tố, Phương pháp 2 con trỏ.

Nhận xét:

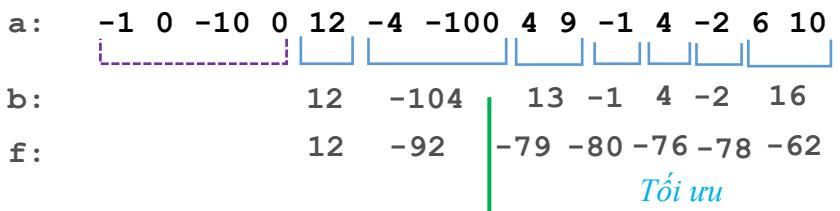
- ⊕ Những ngày đầu tiên không có lợi nhuận dương sẽ không tham gia vào việc tìm kiếm kết quả,
- ⊕ Vì không phải dẫn xuất khoảng thời gian nên:
 - ✓ Các ngày *liên tiếp có lợi nhuận không âm* có thể gộp thành một ngày với tổng lợi nhuận các ngày được gộp,
 - ✓ Các ngày *liên tiếp có lợi nhuận không dương* có thể gộp thành một ngày với tổng lợi nhuận các ngày được gộp,
- ⊕ Dãy **a** ban đầu được thu gọn về dãy **b** đan xen liên tiếp các giá trị dương, âm.

Với dãy **b**: Xây dựng dãy tổng tiền tố **f** để tính lợi nhuận khoảng các ngày liên tiếp.

Dễ dàng thấy rằng:

- ⊕ Đoạn có tổng lớn nhất cần tìm phải bắt đầu từ phần tử vị trí chẵn của dãy **b** (phần tử có giá trị dương),
- ⊕ Nếu $b_u > 0$ và $\sum_{i=u}^v b_i < 0$ thì đoạn $[u, v]$ sẽ không là phần đầu (*prefix*) của đoạn cần tìm vì bỏ đoạn $[u, v]$ giá trị hàm mục tiêu sẽ tăng.

Ví dụ:



30

Sơ đồ tìm kiếm:

- ♣ Kết cần tìm **ans** không nhỏ hơn b_{2i} , $i = 0, 1, 2, \dots$
- ♣ Xuất từ $p = 0$,
- ♣ Tìm **q** lớn nhất ($q \geq p$) thỏa mãn điều kiện $\sum_{i=p}^j b_i > 0$ với mọi $j \leq q$, cập nhật **ans** với mỗi **j**,
- ♣ Gán **p** bằng chỉ số của phần tử **b** dương tiếp theo sau vị trí **q**, lặp lại quá trình tìm **q** đã nêu ở bước trên,
- ♣ Giải thuật kết thúc khi **q** đạt tới vị trí cuối cùng của dãy **b**.

Tổ chức dữ liệu:

- ▀ Mảng `vector<int64_t>` **a(n)** – lưu dữ liệu ban đầu (*có thể vòng tránh sử dụng mảng này*),
- ▀ Mảng `vector<int64_t>` **b** – lưu dãy giá trị đan xen dãy,

■ Mảng `vector<int64_t>` `f` – lưu tổng tiền tố của dãy đan xen dấu.

Xử lý:

Nhập dữ liệu và phân lập tường hợp không có giá trị dương:

```
fi>>n;
vector<int64_t> a(n), b;
p=-1;
for(int i=0; i<n; ++i) fi>>a[i];
for(int i =0; i<n; ++i) if(a[i]>0) {p=i; break;}
if(p<0) {fo<<'0'; return 0;}
```

Có thể gắn với quá trình nhập dữ liệu

Xây dựng dãy đan xen dấu và tổng tiền tố:

```
t=0; q=1;
while(p<n)
{
    if(a[p]*q>=0) t+=a[p++];
    else {b.push_back(t); t=a[p++]; q=-q;}
    if(p==n) {b.push_back(t); break;}
}

m=b.size();
f.resize(m+2);

f[0]=0; f[m+1]=-1e16;
for(int i=0; i<m; ++i) f[i+1]=f[i]+b[i];
```

Con lắc đảo dấu

Hàng rào

Tìm kiếm **q** theo **p**:

```
void get_q()
{
    q=m+1;
    for(int i=p+1; i<=m+1; i+=2)
    {
        if(f[i]-f[p+1]>0) {ans=max(ans, f[i]-f[p+1]);}
        if(f[i]-f[p+1]<0) {q=i-1; break;}
    }
    return;
}
```

Cập nhật kết quả bên trong khoảng

Tiêu chuẩn
kết thúc

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "best."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
int n,m,p=1,q;
int64_t ans=0,t;
vector<int64_t> f;

void get_q()
{
    q=m+1;
    for(int i=p+1;i<=m+1;i+=2)
    {
        if(f[i]-f[p-1]>0) {ans=max(ans,f[i]-f[p-1]);}
        if(f[i]-f[p-1]<0) {q=i-1; break;}
    }
    return;
}

int main()
{
    fi>>n;
    vector<int64_t> a(n),b;
    p=-1; t=0; q=1;
    for(int i=0; i<n; ++i) fi>>a[i];
    for(int i = 0; i < n; ++i) if(a[i] > 0) {p=i; break;}
    if(p<0){fo<<'0'; return 0;}
    while(p<n)
    {
        if(a[p]*q>=0) t+=a[p++];
        else {b.push_back(t); t=a[p++]; q=-q;}
        if(p==n) {b.push_back(t); break;}
    }

    m=b.size();
    f.resize(m+2);
    f[0]=0; f[m+1]=-1e16;
    ans=b[0]; p=1;
    for(int i=0; i<b.size(); i+=2) ans=max(ans,b[i]);
    for(int i=0; i<m; ++i) f[i+1]=f[i]+b[i];
    while(q<=m)
    {
        get_q(); ans=max(ans,f[q]-f[p-1]);
        if((q&1)==0) p=q+1; else p=q+2;
    }
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
}
```



VX43. TIỂU THUYẾT

Tên chương trình: NOVEL.CPP

Nhà xuất bản nhận được bản thảo một cuốn tiểu thuyết rất hay về đề tài khoa học viễn tưởng và nhận phát hành.

Cuốn tiểu thuyết có n chương, chương thứ i có a_i trang, $i = 1 \div n$, nếu in thành một cuốn sách thì quá dày, vì vậy người ta quyết định in thành k tập, mỗi chương phải ngắn gọn trong một tập, tập 1 bao gồm một số chương đầu tiên, mỗi tập tiếp theo bao gồm một số chương tiếp, theo đúng trình tự như in tất cả các chương liên tiếp thành một cuốn.

Ban biên tập phải có nhiệm vụ phân chia sao cho số trang của tập dày nhất là ít nhất.

Ví dụ, với $n = 5$, số trang trong mỗi chương tương ứng lần lượt là 3, 7, 12, 8, 5 và dự kiến in thành 3 tập thì tập 1 sẽ chứa chương 1 và 2 với tổng số trang là 10, tập 2 chứa chương 3 với tổng số trang là 12, tập 3 chứa hai chương cuối với tổng số trang là 13. Như vậy, tập dày nhất có số trang là 13 và đây cũng là cách phân chia phù hợp với yêu cầu đã nêu.

Hãy xác định số trang của tập dày nhất nhận được sau kết quả làm việc của ban biên tập.

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

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

Kết quả: Đưa ra file văn bản NOVEL.OUT một số nguyên – số trang của tập dày nhất.

Ví dụ:

NOVEL.INP
5
3 7 12 8 5

NOVEL.OUT
13



VX43 Mcd20180221FNSM B AXIII

Giải thuật: Tìm kiếm nhị phân.

Số trang trong mỗi tập phải không nhỏ hơn số trang của chương dày nhất,

Số trang của tập dày nhất có thể xác định bằng phương pháp tìm kiếm nhị phân, bắt đầu từ số trang của chương dày nhất.

Với số trang dự kiến cho tập dày nhất là **mid** ta có thể xác định số lượng **p** tập cần phát hành,

Nếu $p \leq k \rightarrow$ giá trị **mid** cho phép in thành **k** tập.

Lưu ý: Việc điều khiển quá trình tìm kiếm bằng điều kiện **right-left>1** và sau đó kiểm tra lại việc phân tập với giá trị **left** là sơ đồ điều khiển vạn năng cho nhiều bài toán ứng dụng tìm kiếm nhị phân, độ phức tạp của giải thuật không thay đổi!

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "novel."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,k,m,maxa;
int64_t t,ans,sm,lf,rt,mid,p;

int main()
{
    fi>>n>>k;
    vector<int> a(n+1); sm=0; maxa=0;
    for(int i=0; i<n; ++i) {fi>>a[i]; sm+=a[i]; maxa=max(maxa,a[i]);}
    a[n]=sm;
    lf=maxa; rt=sm;
    while(rt-lf>1)
    {
        mid=lf+(rt-lf)/2;
        m=0; p=0;
        for(int i=0;i<=n;++i)
            if(p+a[i]>mid) { p=a[i]; ++m;} else p+=a[i];
        if(m>k) lf=mid; else rt=mid;
    }

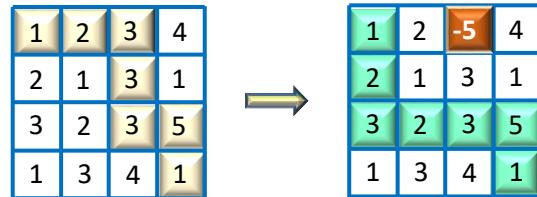
    m=0; p=0;
    for(int i=0;i<=n;++i)
        if(p+a[i]>lf) {p=a[i]; ++m;} else p+=a[i];
    ans=(m<=k)? lf:rt;

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



Bài giảng về quy hoạch động được minh họa bằng bài toán tìm đường đi trên lưới ô vuông: “Cho lưới ô vuông hình chữ nhật có n dòng và m cột. Mỗi ô (i, j) của lưới có ghi một số nguyên $a_{i,j}$, $i = 1 \div n$, $j = 1 \div m$. Từ mỗi ô chỉ có thể di chuyển sang ô kề cạnh bên phải hoặc xuống dưới (nếu tồn tại ô đó). Giá trị đường đi là tổng các số ghi trên những ô nằm trên đường đi. Hãy tìm đường đi có giá trị lớn nhất, xuất phát từ ô trên trái và kết thúc ở ô dưới phải. Đường đi này được gọi là đường đi tối ưu.” Một loạt các tests được tạo ra để kiểm tra chương trình của học sinh.

Chuyên đề trên được mở rộng và nâng cao ở buổi ngoại khóa cho học sinh giỏi. Xây dựng tests mới là một việc khó và buồn tẻ. Vì vậy thầy giáo thông báo là sử dụng các files tests cũ đã có, nhưng ở mỗi test giá trị một ô bị thay đổi thành số cực nhỏ (có thể bằng 0 hoặc âm) để đường đi tối ưu không còn được như cũ và giá trị đường đi tối ưu mới sẽ là nhỏ nhất trong số các khả năng giá trị một ô bị thay đổi. Ô trên trái và ô dưới phải vẫn giữ nguyên giá trị cũ.



Cho test ban đầu. Hãy xác định giá trị của đường đi tối ưu sau khi thay đổi giá trị ở một ô theo điều kiện đã nêu.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($2 \leq n, m \leq 1\ 500$),
- ✚ Dòng thứ i trong n dòng sau chứa m số nguyên $a_{i,1}, a_{i,2}, \dots, a_{i,m}$ ($1 \leq a_{i,j} \leq 10^9$, $j = 1 \div m$).

Kết quả: Đưa ra file văn bản GEN_T.OUT một số nguyên – giá trị đường đi tối ưu mới.

Ví dụ:

GEN_T.INP
4 4
1 2 3 4
2 1 3 1
3 2 3 5
1 3 4 1

GEN_T.OUT
17

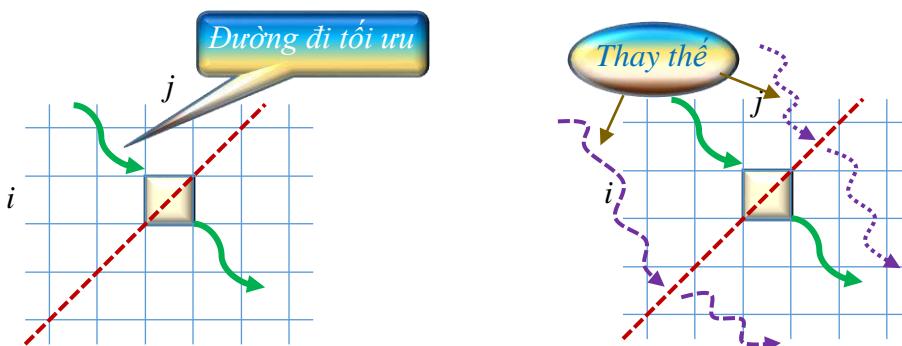


Giải thuật: Quy hoạch động, Bảng phương án.

Việc gán giá trị $-\infty$ cho một ô (i, j) nào đó tương ứng với việc cấm đi qua ô đó. Để có đường đi mới với giá trị đường đi tối ưu cũ thì ô cấm phải nằm trên đường đi tối ưu của lối ban đầu (*Điều kiện cấm*).

Theo điều kiện đầu bài, ô ở góc trái và ô góc dưới phải không bị cấm.

Xét đường chéo phụ đi qua ô (i, j) . Mọi đường đi hợp lệ đều *cắt đường chéo phụ này một và chỉ một lần*.



Để nhanh chóng tìm đường đi tốt nhất qua ô cho trước cần thành lập *2 bảng giá trị d₁_{i,j} và d₂_{i,j}*, trong đó $d_{1,i,j}$ – giá trị đường đi tốt nhất từ góc trái tới ô $(i-1, j)$, $d_{2,i,j}$ – giá trị đường đi tốt nhất từ góc dưới phải tới ô (i, j) . Các bảng này được xây dựng với chi phí $O(n \times m)$ theo sơ đồ quy hoạch động.

Với mỗi đường chéo phụ cần tìm 2 ô cho giá trị các đường đi là lớn nhất. Một trong 2 ô đó thuộc đường đi tối ưu của lối ban đầu. Đường đi thứ 2 cho giá trị lớn nhất (gọi là *giá trị tối ưu thứ 2*) khi ô trên đường đi tối ưu ban đầu bị cấm.

Trong số các giá trị tối ưu thứ 2 – tìm giá trị nhỏ nhất.

Lưu ý:

- ⊕ Sự khéo léo trong sơ đồ xử lý các ô ở biên sẽ giảm đáng kể việc kiểm tra rẽ nhánh!
- ⊕ Trên thực tế giải thuật đòi hỏi vét cạn mọi khả năng cấm có thể, nhưng với mỗi ô cấm – chi phí xử lý là $O(1)$ với 2 bảng giá trị hỗ trợ đã nêu.

Tổ chức dữ liệu:

- ▀ Mảng `int a[MAXN][MAXN]` – lưu bảng giá trị của lối,
- ▀ Các mảng `ll d1[MAXN][MAXN], d2[MAXN][MAXN]` – lưu 2 bảng giá trị hỗ trợ,
- ▀ Mảng `vector<ll> suff = {-1}` – lưu giá trị các đường đi tối ưu qua mỗi ô trên một cột của lối.

Xử lý:

Nhập dữ liệu và chuẩn bị:

Tiết kiệm thời gian hơn gán giá
trị đầu trong khai báo!

```
fi >> n >> m;
for (int i = 0; i < n; ++i)
    for (int j = 0; j < m; ++j) fi >> a[i][j];
for (int i = 0; i < MAXN; ++i)
    for (int j = 0; j < MAXN; ++j) d1[i][j] = d2[i][j] = -1;
```

Tính 2 bảng giá trị hỗ trợ:

```
d1[0][0] = 0;
for (int i = 0; i < n; ++i)
    for (int j = 0; j < m; ++j)
    {
        d1[i + 1][j] = d1[i][j];
        if (j > 0) d1[i + 1][j] = max(d1[i + 1][j], d1[i + 1][j - 1]);
        d1[i + 1][j] += a[i][j];
    }

d2[n][m - 1] = 0;
for (int i = n - 1; i >= 0; --i)
    for (int j = m - 1; j >= 0; --j)
    {
        d2[i][j] = d2[i + 1][j];
        if (j < m - 1) d2[i][j] = max(d2[i][j], d2[i][j + 1]);
        d2[i][j] += a[i][j];
    }
```

Chương trình

```
#include <bits/stdc++.h>
#define NAME "Gen_t."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef long long ll;
const ll INFL = 1e18 + 123;
const int MAXN = 2007;
int a[MAXN][MAXN];
ll d1[MAXN][MAXN], d2[MAXN][MAXN];
int n, m;
int main()
{
    fi >> n >> m;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j) fi >> a[i][j];

    for (int i = 0; i < MAXN; ++i)
        for (int j = 0; j < MAXN; ++j) d1[i][j] = d2[i][j] = -1;
    d1[0][0] = 0;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
    {
        d1[i + 1][j] = d1[i][j];
        if (j > 0) d1[i + 1][j] = max(d1[i + 1][j], d1[i + 1][j - 1]);
        d1[i + 1][j] += a[i][j];
    }
    d2[n][m - 1] = 0;
    for (int i = n - 1; i >= 0; --i)
        for (int j = m - 1; j >= 0; --j)
    {
        d2[i][j] = d2[i + 1][j];
        if (j < m - 1) d2[i][j] = max(d2[i][j], d2[i][j + 1]);
        d2[i][j] += a[i][j];
    }
    ll ans = INFL;
    for (int i = 0; i < n; ++i)
    {
        vector<ll> suff = {-1};
        for (int j = m - 1; j >= 0; --j)
            suff.push_back(max(suff.back(), d1[i][j] + d2[i][j]));
        reverse(suff.begin(), suff.end());
        ll pref = -1;
        for (int j = 0; j < m; ++j)
        {
            if (i == 0 && j == 0) continue;
            if (i == n - 1 && j == m - 1) continue;
            ll cur = -1;
            if (j < m - 1 && i > 0) cur = max(cur, suff[j + 1]);
            if (j && i < n - 1)
            {
                pref = max(pref, d1[i + 1][j - 1] + d2[i + 1][j - 1]);
                cur = max(pref, cur);
            }
            ans = min(ans, cur);
        }
    }
    fo << ans << '\n';
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VX46. CỌC NHỒI

Tên chương trình: SPILES.CPP

Một biệt thự có móng là một đa giác cạnh không tự cắt n đỉnh, đỉnh thứ i có tọa độ nguyên (x_i, y_i), $i = 1 \dots n$. Các đỉnh được liệt kê theo một chiều nào đó. Khu vực xây dựng có nền đất yếu vì vậy người ta phải tiến hành đóng cọc nhồi để gia cố móng. Cọc được đóng ở các điểm có tọa độ nguyên thuộc móng, tức là trên biên hoặc bên trong đa giác xác định móng.

Hãy xác định số cọc cần đóng.

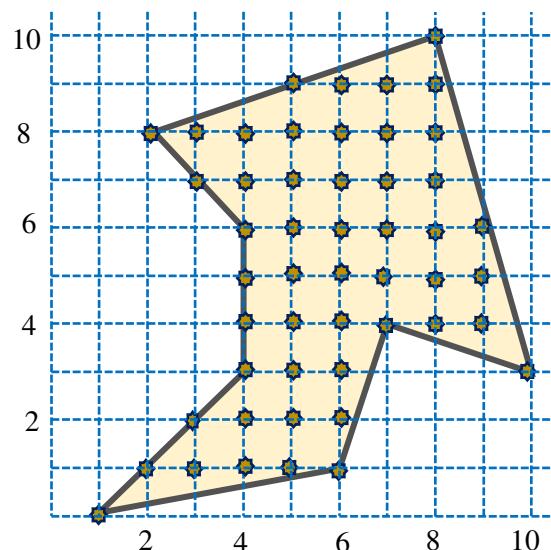
Dữ liệu: Vào từ file văn bản SPILES.INP

- ✚ Dòng đầu tiên chứa một số nguyên n ($3 \leq n \leq 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên x_i, y_i ($|x_i|, |y_i| \leq 10^6$).

Kết quả: Dưa ra file văn bản SPILES.OUT một số nguyên – số cọc cần đóng.

Ví dụ:

SPILES.INP
8
2 8
4 6
4 3
1 0
6 1
7 4
10 3
8 10



SPILES.OUT
50



VX46_AXIII

Giải thuật: Định lý Pick, Phương trình Diophantine.

Xét đa giác trên mặt phẳng có cạnh không tự cắt và đỉnh có tọa nguyên.

$$\text{Định lý Pick: } S = I + \frac{B}{2.0} - 1$$

Trong đó:

- ✚ S – diện tích đa giác,
- ✚ I – số lượng điểm có tọa độ nguyên nằm trong đa giác,
- ✚ B – số lượng điểm có tọa độ nguyên nằm trên cạnh của đa giác.

Mặt khác, ta có $2 \times S = |\sum_{i=0}^{n-1} (x_i \times y_{i+1} - x_{i+1} \times y_i)|$ với $x_n = x_0, y_n = y_0$.

Suy ra $2 \times S = 2 \times I + B - 2$.

Số điểm nguyên trên cạnh nối 2 điểm (x_i, y_i) và (x_{i+1}, y_{i+1}) là $|x_i - x_{i+1}| + |y_i - y_{i+1}|$ nếu $x_i = x_{i+1}$ hoặc $y_i = y_{i+1}$ hoặc bằng $|\gcd(x_i - x_{i+1}, y_i - y_{i+1})|$ (Xem bài VX34). Ở đây số điểm tìm được không cộng thêm 1 vì điểm cuối của đoạn sẽ được tính ở đoạn sau.

Biết $2 \times S$ và B , ta có thể tính được I và dẫn xuất kết quả cần tìm.

Chương trình

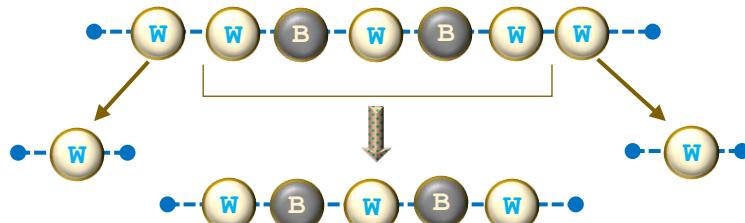
```
#include <bits/stdc++.h>
#define NAME "spiles."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;
int64_t s,s1,s2,ans,u1,u2,v1,v2;
int64_t dots(int x1, int y1, int x2,int y2)
{
    int64_t a,b, k;
    a=y2-y1; b=x1-x2;
    if(a==0 || b==0) {k=abs (a)+abs (b); return (k);}
    k=__gcd(a,b);
    return (abs (k));
}

int main()
{
    fi>>n;
    vector<int64_t> x(n+1), y(n+1);
    for(int i=0; i<n; ++i) fi>>x[i]>>y[i];
    x[n]=x[0]; y[n]=y[0];
    s1=0; s=0;
    for(int i=0; i<n; ++i)
    {
        u1=x[i]; v1=y[i]; u2=x[i+1]; v2=y[i+1];
        s1+=dots(u1,v1,u2,v2);
        s+=x[i]*y[i+1]-x[i+1]*y[i];
    }
    s=abs (s);
    s2=(s+2-s1)/2;
    ans=s1+s2;
    fo<<ans;
    fo<<"\nTime: "<<clock () / (double) 1000<<" sec";
}
```



Trong chuyến du lịch mùa hè Alice đến tham quan một trại nuôi ngọc trai và mua n hạt, mỗi hạt có màu trắng hoặc đen. Cửa hàng đã cẩn thận xâu tất cả các hạt thành một chuỗi theo giá trị tăng dần của mỗi hạt từ trái sang phải.

Về nhà Alice tách chuỗi hạt mua được thành nhiều chuỗi để tặng cho bạn bè. Để các chuỗi hạt tách được vẫn có vẻ đẹp hài hòa, Alice muốn mỗi chuỗi con vẫn đảm bảo tính chất giá trị tăng dần từ trái sang phải, ngoài ra mỗi chuỗi con chỉ chứa một hạt trắng hoặc đan xen các hạt trắng/đen, bắt đầu từ hạt trắng và kết thúc cũng bằng hạt trắng. Dĩ nhiên Alice không muốn còn thừa hạt nào không được xâu thành chuỗi con. Các hạt cạnh nhau trong chuỗi con không nhất thiết là phải cạnh nhau trong chuỗi ban đầu.



Hãy xác định Alice có thể chia được chuỗi ban đầu thành các chuỗi con theo yêu cầu trên hay không. Nếu được thì chỉ ra một cách chia thích hợp tùy chọn. Không nhất thiết phải tối ưu hóa số lượng chuỗi con.

Dữ liệu: Vào từ file văn bản NECKLACE.INP gồm một dòng chứa xâu độ dài không quá 2×10^5 , mỗi ký tự trong xâu là 0 (tương ứng với hạt trắng) hoặc 1 (tương ứng với hạt đen). Các hạt được đánh số bắt đầu từ 1.

Kết quả: Đưa ra file văn bản NECKLACE.OUT. Nếu không có cách chia thì đưa ra một số nguyên -1. Trong trường hợp có cách chia:

- ✚ Dòng đầu tiên chứa số nguyên k – số chuỗi con,
- ✚ Mỗi dòng trong k dòng tiếp theo chứa số nguyên m – số hạt trong chuỗi con, tiếp theo – m số nguyên theo thứ tự tăng dần xác định vị trí trong dãy ban đầu của hạt thuộc chuỗi con.

Ví dụ:

NECKLACE.INP	NECKLACE.OUT
0010100	3 1 1 5 2 3 4 5 6 1 7



Giải thuật: Bảng phương án và kỹ thuật tổ chức dữ liệu.

Ghi nhận điểm cuối của các chuỗi con hợp lệ đang có vào vector **white** (dòng xếp hàng điểm cuối các chuỗi hợp lệ),

Gọi chuỗi con đan xen các hạt trắng – đen và kết thúc bằng hạt đen là *chuỗi đen*,

Ghi nhận điểm cuối của các chuỗi đen đang có vào vector **black** (dòng xếp hàng điểm cuối các chuỗi đen).

Duyệt xâu **st** của input từ trái sang phải:

⊕ Gặp ký tự 0:

- ✓ Nếu có chuỗi đen (**black** ≠ \emptyset) → bổ sung vào một chuỗi đen và *giảm số lượng chuỗi đen*, ta được *một chuỗi hợp lệ* và ghi nhận nó,
- ✓ Nếu không có chuỗi đen → ghi nhận vào **white** và nạp nó như một *chuỗi hợp lệ độ dài 1* vào vector kết quả.

⊕ Gặp ký tự 1:

- ✓ Nếu không còn chuỗi hợp lệ (**white** = \emptyset): bài toán vô nghiệm,
- ✓ Nạp 1 vào một chuỗi hợp lệ đã có, biến nó thành chuỗi đen, loại bỏ chuỗi tương ứng khỏi dòng xếp hàng điểm cuối của chuỗi hợp lệ.

Sau khi duyệt hết các ký tự của **st**, dòng xếp hàng điểm cuối các chuỗi đen khác rỗng – bài toán vô nghiệm.

Tổ chức dữ liệu:

- ☰ Mảng **vector<int>** **white** – Dòng xếp hàng điểm cuối các chuỗi trắng,
- ☰ Mảng **vector<int>** **black** – Dòng xếp hàng điểm cuối các chuỗi đen,
- ☰ Mảng **vector<vector<int>>** **ans** – Ghi nhận các chuỗi kết quả.

Xử lý:

st_i	0		1	
black=∅	Yes	No	-	-
white=∅	-	-	Yes	No
Xử lý	white.push_back(ans.size()); ans.push_back(vector<int>(1, i));	ans[black.back()].push_back(i); white.push_back(black.back()); black.pop_back();	Vô nghiệm	ans[white.back()].push_back(i); black.push_back(white.back()); white.pop_back();

Dộ phức tạp của giải thuật: O(n).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "necklace."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;
string st;
vector<vector<int>> ans;
vector<int> white;
vector<int> black;

void no()
{
    fo<<"-1\n";
    exit(0);
}

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

    for (int i = 0; i < n; ++i)
    {
        if (st[i] == '0')
        {
            if (!black.empty())
            {
                ans[black.back()].push_back(i);
                white.push_back(black.back());
                black.pop_back();
            }
            else
            {
                white.push_back(ans.size());
                ans.push_back(vector<int>(1, i));
            }
        }
        else
        {
            if (white.empty()) no();
            ans[white.back()].push_back(i);
            black.push_back(white.back());
            white.pop_back();
        }
    }

    if (!black.empty()) no();

    fo<<ans.size()<<'\n';
    for (int i = 0; i < (int)ans.size(); ++i)
    {
        fo<<ans[i].size()<< ' ';
        for (int j:ans[i]) fo<<j+1<< ' ';
        fo<<'\n';
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}
```



VX48. TRÒ CHƠI NHÌ PHÂN

Tên chương trình: BINGAME.CPP

Trò chơi trên TV đòi hỏi những người chơi phải có khả năng quan sát, trí nhớ tốt và tính toán giỏi. Trên bàn có các quân bài, mỗi quân bài ghi một số nguyên bằng 2^k hoặc -2^k , $k = 0, 1, 2, \dots, 30$, số quân bài cùng giá trị là đủ nhiều. Người chơi được xem trên màn hình trong một thời gian ngắn các số nguyên a_1, a_2, \dots, a_n , sau đó tự chọn cho mình các quân bài tùy ý với số lượng tùy ý các quân bài cùng giá trị.

Trò chơi bao gồm n bước. Ở bước thứ i số a_i xuất hiện trên màn hình. Người chơi phải trích từ những quân bài của mình một nhóm nào đó các lá bài cho tổng bằng a_i và đặt chúng lên bàn để mọi người thấy. Số lượng quân bài đặt lên bàn có thể là 0 nếu $a_i = 0$. Người nào không chọn được các quân bài cho tổng bằng a_i sẽ bị loại ra khỏi cuộc chơi. Những người còn lại thu hồi các quân bài đã đặt trở về tập bài của mình và trò chơi chuyển sang bước tiếp theo. Sau n bước, ai hoặc những ai trong số còn lại có tập bài với ít quân nhất sẽ chiến thắng.

Hãy xác định các quân bài cần chọn để đảm bảo bạn là người chiến thắng.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($|a_i| \leq 10^6$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản BINGAME.OUT trên một dòng số nguyên k – số quân bài cần chọn và ở dòng thứ 2 k số nguyên – giá trị các quân bài được chọn.

Ví dụ:

BINGAME.INP	BINGAME.OUT
4	3
2 -2 14 18	2 -2 16



VX48 OOM20180308 D AXIII

Giải thuật: Quy hoạch động.

Nhận xét:

Việc giữ 2 hay nhiều quân bài cùng giá trị là không tối ưu: thay vì việc giữ 2 quân bài giá trị **x** thì việc giữ quân bài **x** và quân bài **2x** sẽ mang lại nhiều khả năng đáp ứng với dãy **a** cho trước hơn,

Dễ dàng chứng minh việc giữ quân bài có giá trị tuyệt đối lớn hơn $4 \times \max\{|\mathbf{a}_i|\}$ sẽ không giúp ích cho việc cực tiểu hóa số quân bài cần có,

Việc giữ đồng thời 2 quân bài **x** và **-x** là không tối ưu,

Nếu tất cả các số \mathbf{a}_i đều chẵn thì việc giữ các quân bài **1** và **-1** là không tối ưu,

Trong quá trình xử lý: chỉ giữ một đại diện cho các giá trị giống nhau.

Nếu trong dãy số có số lẻ - cần có quân bài **1** hoặc **-1**.

Trừ tất cả các số lẻ trong dãy đang xét cho 2, ta có bài toán tương tự với các giá trị cần xét nhỏ hơn!

Ở mỗi bước: Có không quá 2 khả năng cần lựa chọn.

Số lượng bước: không vượt quá $\ln(\max\{|\mathbf{a}_i|\})$.

Tổ chức dữ liệu:

- ─ Mảng `vector<int>` cards – Lưu dãy số ban đầu,
- ─ Mảng `vector<int>` ans – Lưu kết quả.

Gọi **c** – số lượng bít có nghĩa của $\max\{|\mathbf{a}_i|\}$.

Độ phức tạp của giải thuật: O(clogc).

Chương trình 1

```
#include<bits/stdc++.h>
#define NAME "bingame."
#define all(x) (x).begin(), (x).end()
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
struct ans
{
    int cnt;
    int p, m;
    ans(int cnt = 0, int p = 0, int m = 0): cnt(cnt), p(p), m(m) {}
};

int md(int a, int b)
{
    a %= b;
    if (a < 0)a += b;
    return a;
}

int fnd(int a, int k)
{
    int x = md(a, (1 << (k + 1)));
    if (x > (1 << k))x -= (1 << (k + 1));
    return x;
}

ans dp[20];
int n;
const int INF = 1e6;

int main()
{
    fi >> n;
    vector<int> a(n);
    for (int i = 0; i < n; ++i) fi >> a[i];
    sort(a.begin(), a.end());
    ans res(INF, 0, 0);
    for (int i = 0; i < 20; ++i)
    {
        vector<int> vv;
        int fl = 0;
        int fl2 = 0;
        for (int j = 0; j < n; ++j)
        {
            int x = fnd(a[j], i);
            if (x == (1 << i)) {fl = 1; break;}
            if (x != a[j]) fl2 = 1;
            vv.push_back(x);
        }
        dp[i] = ans(INF, 0, 0);
        if (fl) continue;
        sort(vv.begin(), vv.end());
        vv.resize(unique(vv.begin(), vv.end()) - vv.begin());
        if (vv.back() - vv.front() < (1 << i))
        {
            int x = max(vv.back(), 0);
            dp[i] = ans(i, 0, 0);
            for (int k = 0; k < i; ++k)
                if ((x >> k) & 1) dp[i].p |= (1 << k);
        }
    }
}
```

```

        else dp[i].m |= (1 << k);
    }
    for (int j = 0; j < i; ++j)
    {
        if (dp[j].cnt == INF) continue;
        int mn = 0, mx = 0;
        for (int x: vv)
        {
            x -= fnd(x, j);
            mn = min(mn, x >> (j + 1));
            mx = max(mx, x >> (j + 1));
        }
        if (dp[j].cnt+i-j-1<dp[i].cnt && mx - mn < (1 << (i - j - 1)))
        {
            dp[i] = dp[j];
            dp[i].cnt += i - j - 1;
            for (int k = 0; k < i - j - 1; ++k)
                if ((mx >> k) & 1) dp[i].p |= (1 << (k + j + 1));
                else dp[i].m |= (1 << (k + j + 1));
        }
    }
    if (!fl2)
        if (dp[i].cnt < res.cnt) res = dp[i];
}
fo << res.cnt << '\n';
int cur = res.p;
int cc = 0;
while (cur)
{
    if (cur & 1) fo << (1 << cc) << ' ';
    ++cc; cur >>= 1;
}
cur = res.m;
cc = 0;
while (cur)
{
    if (cur & 1) fo << -(1 << cc) << " ";
    ++cc;
    cur >>= 1;
}
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
return 0;
}

```

Chương trình 2

```
#include <bits/stdc++.h>
#define NAME "bingame."
#define all(x) (x).begin(), (x).end()
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

vector<int> ans;
int k = 1000001;
int q = 1;
vector<int> cards;

void run(vector<int>&& a, int x, int cur = 0)
{
    sort(all(a)), a.resize(unique(all(a)) - a.begin());
    if (x >= q)
        if (a.size() == 1 && a.front() == 0)
        {
            if (k > cur)
            {
                k = cur;
                cards = ans;
            }
            return ;
        }

    bool need = false;
    for (int i : a)
        if (abs(i) & (1 << x))
        {
            need = true;
            break;
        }

    if (!need)
        run(move(a), x + 1, cur);
    else
    {
        vector<int> na;
        for (int& i : a)
            if (abs(i) & (1 << x)) na.push_back(i + (1 << x));
            else na.push_back(i);
        ans.push_back(-(1 << x));
        run(move(na), x + 1, cur + 1);
        ans.pop_back();
        na.clear();
        for (int& i : a)
            if (abs(i) & (1 << x)) na.push_back(i - (1 << x));
            else na.push_back(i);
        ans.push_back(1 << x);
        run(move(na), x + 1, cur + 1);
        ans.pop_back();
    }
}

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

```
int mx = -1e8;
for (int i : a)
    mx = max(mx, abs(i));

while ((1 << q) < mx * 2) q++;
run(move(a), 0);
fo << k << endl;
for(int i:cards) fo<<i<<' ';
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
return 0;
}
```



Rất nhiều nước và khu vực trên thế giới tổ chức các trại hè chuyên đề để các bạn trẻ kết hợp giao lưu với nhau và bồi dưỡng nâng cao trình độ kiến thức chuyên ngành.

Các bạn học sinh được bố trí nghỉ trong n phòng, mỗi phòng b bạn, vừa khít cho $n \times b$ học sinh tham gia trại hè. Các phòng đều nằm trên một tầng thành một hàng dài, đánh số từ 1 đến n từ trái qua phải.

Sau giờ sinh hoạt buổi tối mọi người về phòng chuẩn bị đi ngủ. Thay vì đánh răng, rửa mặt, dọn dẹp giường, các bạn lại đi qua các phòng khác tán gẫu và khi chuông reo báo giờ ngủ, ở phòng thứ i đang có a_i bạn, $0 \leq a_i, i = 1 \div n$ và $\sum_{i=1}^n a_i = n \times b$. Có thể có một thầy đi lên từ cầu thang bên trái, duyệt các phòng từ 1 trở đi hoặc có hai thầy, thầy thứ 2 đi lên từ cầu thang bên phải và duyệt các phòng $n, n-1, \dots$. Nếu có 2 thầy và n là số lẻ thì thầy thứ nhất sẽ duyệt nốt phòng ở giữa. Trường hợp có 2 thầy, các thầy cùng đồng thời bước vào phòng mình duyệt và đồng thời ra khỏi phòng sau khi duyệt. Trong lúc thầy giáo (hoặc các thầy) đang duyệt một phòng các bạn ở những phòng chưa được duyệt có thể chạy sang nhau, nhưng do thời gian hạn chế nên chỉ có thể di chuyển không quá d phòng, tức là từ phòng i chỉ có thể tới một trong các phòng từ $i-d$ đến $i+d$. Khi vào một phòng thầy giáo sẽ đếm số người trong phòng. Nếu phòng đang có nhiều hơn b bạn thì những người thừa có thể trốn dưới các gầm giường với số lượng không hạn chế và thầy giáo không nhìn thấy được các bạn đang trốn. Không muốn là quá ngặt, các thầy chỉ đếm số người trong phòng và ghi lại số phòng nếu thiếu người rồi tắt đèn, đi ra và đóng cửa lại. Sẽ không có ai ra/vào các phòng đã bị đóng. Danh sách các phòng thiếu người của mỗi thầy sẽ được chuyển cho thầy phụ trách ký túc xá và thầy phụ trách cũng chỉ chọn một danh sách (nếu có 2 thầy đi duyệt) để hôm sau đi phê bình, nhắc nhở các bạn.

Các bạn học sinh biết rõ quy cách làm việc này qua các trại hè những năm trước và tìm cách di chuyển sao cho số phòng bị xử lý hôm sau là ít nhất.

Hãy xác định số phòng ít nhất có thể bị xử lý nếu các bạn học sinh biết cách di chuyển tối ưu.

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

- ✚ Dòng đầu tiên chứa 4 số nguyên p, n, d và b , trong đó p – số lượng thầy giáo đi duyệt các phòng ($1 \leq p \leq 2, 2 \leq n \leq 10^5, 1 \leq d \leq n-1, 1 \leq b \leq 10^4$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($0 \leq a_i \leq 10^9, i = 1 \div n, \sum_{i=1}^n a_i = n \times b$).

Kết quả: Đưa ra file văn bản SUM_ASS.OUT một số nguyên – số phòng ít nhất có thể bị xử lý.

Ví dụ:

SUM_ASS.INP
2 6 1 2
3 8 0 1 0 0

SUM_ASS.OUT
2



Giải thuật: Tìm kiếm nhị phân .

Gọi **f (m)** – số phòng còn thiếu học sinh khi thầy giáo thứ nhất duyệt đến phòng **m**, **g (m)** – số phòng còn thiếu học sinh khi thầy giáo thứ hai duyệt đến phòng **m**.

Dễ dàng thấy rằng **f (m)** – không giảm, còn **g (m)** – không tăng.

Xét riêng 2 trường hợp.

Trường hợp 1 giáo viên đi kiểm tra:

Nếu một phòng *không đủ* học sinh: cố gắng bổ sung bằng các học sinh *từ các phòng bên phải*,

Nếu *không thể có cách bổ sung đầy đủ* từ bên phải – *chuyển tất cả học sinh* ban đầu ở phòng này *sang các phòng tiếp sau*,

Nếu một phòng có *quá nhiều* học sinh – chuyển phần còn thừa *sang các phòng bên phải*.

Trong trường hợp còn thừa người: Trốn bót!

Lưu ý: Nếu ban đầu học sinh **a** ở vị trí trái hơn học sinh **b** thì ở trạng thái cuối **a** không thể có vị trí phải hơn **b** trong giải thuật di chuyển tối ưu.

Với những phòng thừa người: Chuyển bớt sang trái để có các phòng mới đủ người nếu có thể, chuyển hết phần thừa cò lại sang các phòng bên phải.

Việc xác định số phòng thiếu ít nhất – bằng giải thuật tìm kiếm nhị phân.

Trường hợp 2 giáo viên đi kiểm tra:

Chia số phòng thiếu thành 2 phần bằng nhau hoặc chênh lệch 1,

Thực hiện tìm kiếm nhị phân 2 lần với hai nửa hành lang.

Tổ chức dữ liệu:

- ▀ Mảng **int64_t a [120000]** – lưu dữ liệu ban đầu,
- ▀ Tập **set<pair<int64_t, int64_t>> ss** – Lưu kết quả phân phối lại học sinh, mỗi phần tử của tập là cặp dữ liệu (*Phòng, Số học sinh*).

Độ phức tạp của giải thuật: O(nlognb).

Chương trình

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

int p, n;
int64_t d, b, a[120000];

set<pair<int64_t, int64_t> > ss;

int main()
{
    fi >> p >> n >> d >> b;
    for (int i = 0; i < n; ++i) fi >> a[i];
    if (p == 1)
    {
        int lb = -1, rb = n;
        while (rb - lb > 1)
        {
            int mid = (lb + rb) >> 1;
            ss.clear();
            for (int i = 0; i < n; ++i)
                if (a[i]) ss.insert({i, a[i]});
            int fl = 0;
            for (int i = mid; !fl && i < n; ++i)
            {
                int x = i;
                int64_t t = i + 1, nd = b;
                while (nd)
                {
                    auto it = ss.lower_bound({x - d * t, 0});
                    if (it == ss.end() || it->first > x + d * t)
                    {
                        fl = 1; break;
                    }
                    int64_t go = min(nd, it->second);
                    nd -= go;
                    pair<int64_t,int64_t> nw={it->first,it->second-go};
                    ss.erase(it);
                    if (nw.second) ss.insert(nw);
                }
            }
            if (fl) lb = mid;
            else rb = mid;
        }
        fo << rb << "\n";
    }
    else {
        int lb = -1;
        int rb = (n + 1) / 2;
        while (rb - lb > 1) {
            int mid = (lb + rb) >> 1;
            ss.clear();
            for (int i = 0; i < n; ++i)
                if (a[i]) ss.insert({i, a[i]});
            int fl = 0;
            for (int i = mid; !fl && i * 2 < n; ++i)
            {
```

```

int x = i, y = n - 1 - i;
int64_t t = i + 1, nd = b;
while (nd)
{
    auto it = ss.lower_bound({x - d * t, 0});
    if (it==ss.end() || it->first>x+d*t)
        {fl = 1; break; }
    int64_t go = min(nd, it->second);
    nd -= go;
    pair<int64_t, int64_t> nw ={it->first, it->second- go};
    ss.erase(it);
    if (nw.second) ss.insert(nw);
}
if (!fl && x != y)
{
    int64_t nd = b;
    while (nd)
    {
        auto it=ss.lower_bound({y+d*t+1,0});
        if (it==ss.begin() || prev(it)->first<y-d*t)
            { fl = 1; break; }
        --it;
        int64_t go = min(nd, it->second);
        nd -= go;
        pair<int64_t, int64_t> nw={it->first, it->second-go};
        ss.erase(it);
        if (nw.second) ss.insert(nw);
    }
}
if (fl) lb = mid;
else rb = mid;
}
fo << rb << "\n";
}
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
return 0;
}

```



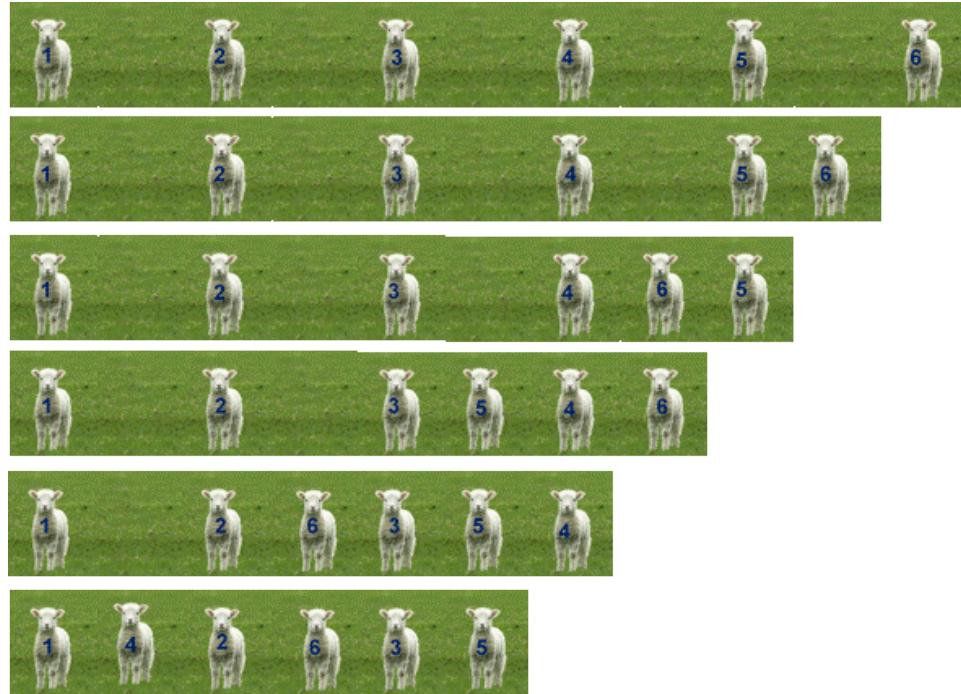
VX50. ĐÀN CỪU

Tên chương trình: SHEEPS.CPP

Đàn cừu có n con đánh số từ 1 đến n . Buổi sáng, khi thả ra đồng chúng dàn hàng ngang ăn cỏ và giữ khoảng cách để giữa 2 con liên tiếp có một vị trí trống. Như vậy con số 1 đứng ở vị trí 1, con số 2 ở vị trí 3, con số 3 ở vị trí 5, . . . Đến giữa buổi có xe chở cỏ voi lêmen rải cho cừu ăn bồ sung.

Đây là thứ mà các chú cừu rất thích. Xe sẽ rải thức ăn theo các vị trí có cừu từ trái sang phải.

Cừu cũng không ngốc như người ta hay nói. Chú cừu *đứng phải nhất trong hàng* sẽ nhận được thức ăn muộn nhất. Để được nhận thức ăn sớm hơn, chú chuyển sang ô trống gần nhất



bên trái của mình. Quá trình di chuyển nói trên sẽ diễn ra cho tới khi tất cả các chú cừu chiếm các vị trí liên tiếp từ 1 đến n và hoàn tất trước khi xe tới trải thức ăn bồ sung.

Cán bộ nghiên cứu đi theo xe quan tâm đến chú cừu đang ăn bồ sung ở vị trí x và ghi nhận vào sổ theo dõi các thông số cần thiết về chú cừu này. Anh định xuống xe xem số của cừu đang đeo biển gắn ở tai nhưng người lái xe nói là không cần thiết và đọc ngay số của chú cừu đó. Có q chú cừu được cán bộ nghiên cứu quan tâm, chú thứ i đang ăn ở vị trí x_i , $i = 1 \div q$.

Hãy xác định số của các chú cừu được quan tâm.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và q ($1 \leq n \leq 10^{18}$, $1 \leq q \leq 2 \times 10^5$),
- ✚ Dòng thứ i trong q dòng tiếp theo chứa số nguyên x_i ($1 \leq x_i \leq n$).

Kết quả: Đưa ra file văn bản SHEEPS.OUT q số nguyên, mỗi số trên một dòng, xác định số của cừu được lưu ý.

Ví dụ:

SHEEPS.INP
6 3
2
3
4

SHEEPS.OUT
4
2
6



Giải thuật: Công thức lắp.

Ở trạng thái ban đầu mọi chú cừu đều đứng ở vị trí lẻ,
Khi một con cừu di chuyển, nó luôn tới vị trí chẵn.

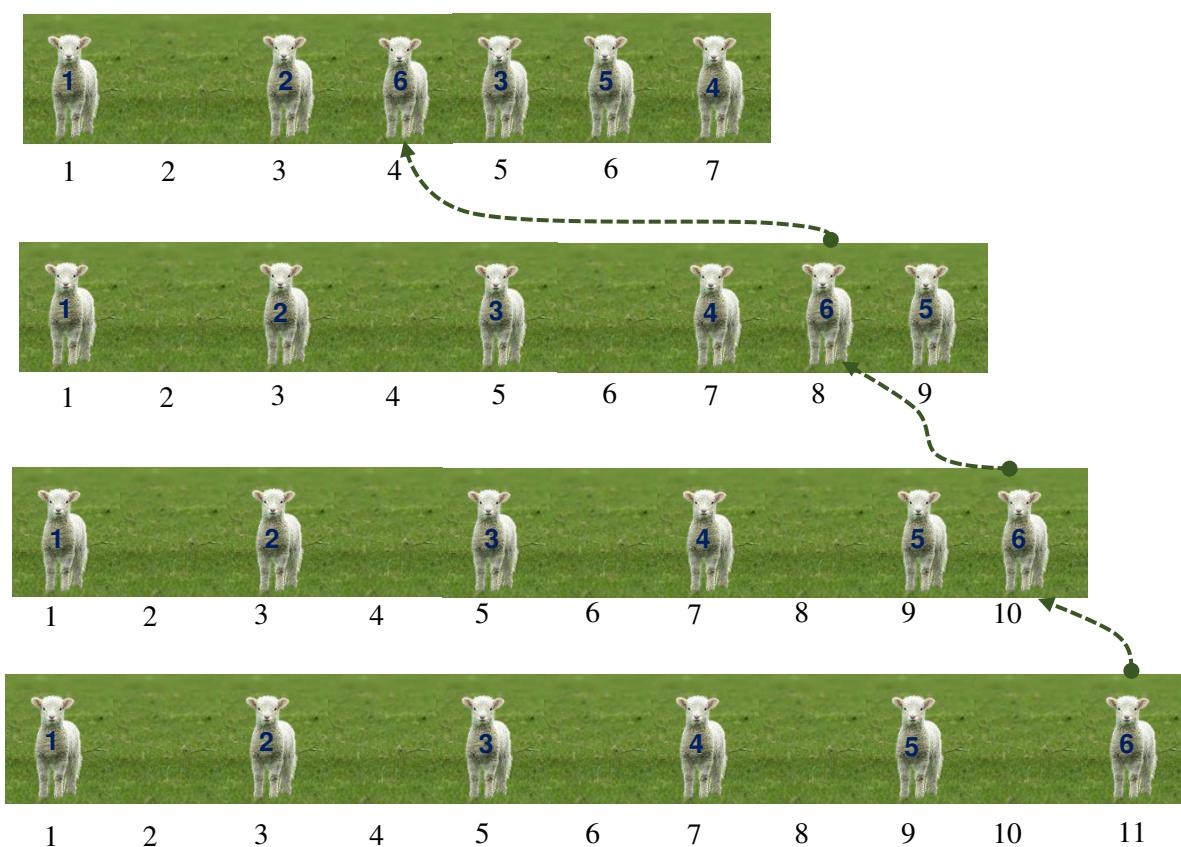
Xét vị trí x ở *trong quá trình cừu chuyển chỗ*:

- ⊕ Nếu x lẻ - cừu ở vị trí này không tham gia vào quá trình di chuyển và số của nó là $(x+1)/2 = x/2 + 1$,
- ⊕ Nếu x chẵn: trước khi có các cừu khác vượt qua nó tới các vị trí trống bên trái (nếu có):
 - ✓ Bên trái nó có $x/2$ con,
 - ✓ Bên phải có $n-x/2$ con đứng liền nhau, không có vị trí trống,
 - ✓ Như vậy vị trí x_{prev} trước đó của nó (trước khi nhảy tới vị trí x) là

$$x+n-x/2$$

Từ vị trí hiện tại x ta có thể tính vị trí x_{prev} trước đó: $x \leftarrow x_{prev}$.

Quá trình cập nhật vị trí được thực hiện cho đến khi nhận được x lẻ và từ đó – tính số của nó.



Với mỗi x : quá trình cập nhật không vượt quá $\log_2 n$.

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

Chương trình

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

int main()
{
    int64_t n, x;
    int q;
    fi>>n>>q;
    for (int i = 0; i < q; ++i)
    {
        fi>>x;
        while ((x&1)==0) x+= n- (x>>1);
        fo<<(x>>1)+1<<' \n';
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}
```



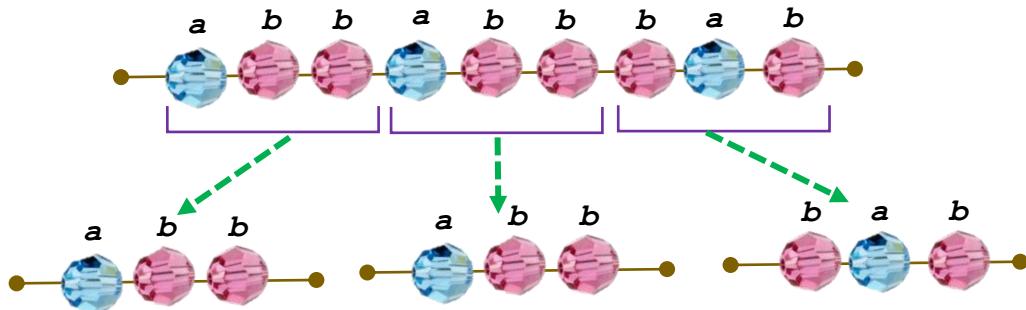
VY01. CHUỖI HẠT ĐẸP

Tên chương trình: BONNY.CPP

Pha lê Swarovski được làm đồ trang sức rất đẹp nhưng không quá đắt tiền. Các hạt pha lê được phân thành 26 loại khác nhau, mỗi loại ký hiệu bằng một chữ cái la tinh thường.

Để tiếp cận một bộ lạc trong vùng rừng rậm Amazon nhằm giúp họ bảo toàn tốt hơn cuộc sống trong môi trường tự nhiên các nhân chủng học chuẩn bị quà cho từng gia đình. Trong hành trang của đoàn thám hiểm có chuỗi dài các hạt pha lê Swarovski.

Các hạt trên chuỗi tương ứng với xâu **s** chỉ chứa ký tự la tinh thường. Người ta quyết định cắt chuỗi hạt ban đầu thành các chuỗi con các hạt liên tiếp có độ dài bằng nhau (tức là có cùng số hạt) và số hạt mỗi loại trong từng chuỗi con cũng phải giống nhau. Mỗi hạt trong chuỗi ban đầu phải thuộc một chuỗi con nào đó.



Hãy xác định số chuỗi con nhiều nhất có thể tạo ra.

Dữ liệu: Vào từ file văn bản BONNY.INP gồm một dòng chứa xâu **s** độ dài không quá 2×10^6 .

Kết quả: Đưa ra file văn bản BONNY.OUT một số nguyên – số chuỗi con nhiều nhất có thể tạo ra.

Ví dụ:

BONNY.INP	BONNY.OUT
abbabbbbab	3



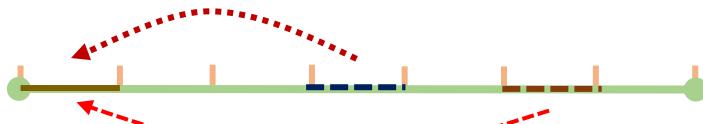
VY01 OOM20180309 D A XIII

Giải thuật: Hàm băm, Tổng tiền tố.

Gọi \mathbf{k} là độ dài xâu con tìm được, \mathbf{n} – độ dài xâu \mathbf{s} .

\mathbf{k} phải là ước của \mathbf{n} .

Để xác định việc chia xâu ban đầu thành các xâu con độ dài \mathbf{k} có thỏa mãn yêu cầu đã nêu hay không cần so sánh tàn số xuất hiện các chữ cái ở mỗi đoạn với tàn số xuất hiện các chữ cái ở đoạn đầu tiên.



Như vậy cần xây dựng mảng tổng tiền tố \mathbf{psum}_i , trong đó \mathbf{psum}_i là véc tơ 26 phần tử xác định tàn số xuất hiện các ký tự $\mathbf{a}, \mathbf{b}, \mathbf{c}, \dots, \mathbf{z}$ từ đầu xâu đến vị trí i của \mathbf{s} .

Từ \mathbf{psum}_i dễ dàng tính được tàn số xuất hiện các ký tự trong đoạn đang xét.

Sơ đồ xử lý trên, tuy đủ nhanh, nhưng đòi hỏi phải lưu 26 giá trị tàn số khác nhau, thực hiện 26 phép tính tàn số và phép so sánh với mỗi đoạn xử lý.

Thay cho véc tơ 26 phần tử xác định tàn số xuất hiện các ký tự ta có thể lưu trữ một giá trị hàm băm tương ứng.

Chọn một số nguyên dương \mathbf{h} đủ lớn và đặt tương ứng ký tự thứ i trong bảng chữ cái với \mathbf{h}^i , $i = 1 \div 26$. Thay vì tích lũy tàn số từng chữ cái ta sẽ xây dựng hàm f tích lũy các giá trị \mathbf{h}^i tương ứng ký tự gấp trên \mathbf{s} trong quá trình xử lý. Như vậy, để so sánh 2 đoạn chỉ cần thực hiện *một phép tính tàn số* và *một phép so sánh*!

Độ phức tạp:

Với $\mathbf{n} \leq 5 \times 10^6 < 2^{23}$ số ước của \mathbf{n} là không quá lớn (bậc 10^2),

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

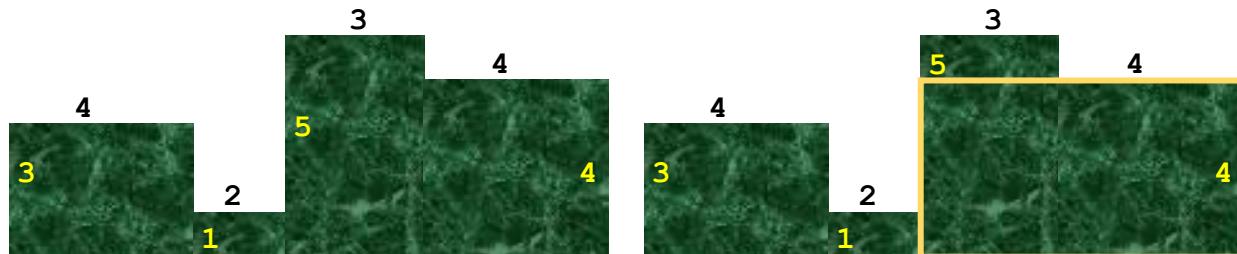
Chương trình

```
#include <bits/stdc++.h>
#define NAME "bonny."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int64_t h=30007;
int64_t p[27],q;
string s;
int n,k;
bool good;

int main()
{
    p[0]=1;
    for(int i=1; i<27; ++i) p[i]=p[i-1]*h;
    fi>>s;
    n=s.size();
    vector<int64_t> f(n+1);
    f[0]=0; k=n;
    for(int i=0; i<n; ++i) f[i+1]= f[i]+p[s[i]-96];
    for(int i=1; i<= n/2; ++i)
        if(n%i==0)
    {
        q=f[i]; good=true;
        for(int j=i; j<=n; j+=i)
            if(f[j]-f[j-i]!=q) {good=false; break;}
        if(good) {k=i; break;}
    }
    fo<<n/k;
    fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
    return 0;
}
```



Một tảng đá cẩm thạch nguyên khối lớn được phát hiện ở mỏ khai thác đá. Việc tồn tại những tảng đá nguyên khối lớn là rất hiếm nên người ta phải thảo luận nhiều về kế hoạch cắt khối đá đó thành các khối con sao cho việc sử dụng là hiệu quả nhất.



Đầu tiên người ta định xé khối đá thành từng phiến có thiết diện hình chữ nhật, mỗi phiến có diện tích lớn nhất có thể, cạnh đáy các phiến nằm trên một đường thẳng. Theo kế hoạch này khối đá có thể cắt thành n phiến, tính từ trái sang phải thiết diện phiến thứ i có độ cao h_i và độ rộng w_i , $i = 1 \div n$. Nhưng chính quyền thành phố muốn trước hết phải cắt một khối có thiết diện chữ nhật với diện tích lớn nhất để làm đế tượng đài đặt ở quảng trường trung tâm của thành phố.

Dĩ nhiên người ta không thể xoay cả khối đá lớn ban đầu ném cạnh của đế tượng đài sẽ song song với cạnh tương ứng của các phiến theo phương án đầu.

Hãy xác định diện tích lớn nhất của thiết diện đế tượng đài có thể tạo được.

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

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

Tổng tất cả w_i không vượt quá 10^9 .

Kết quả: Đưa ra file văn bản PEDESTAL.OUT một số nguyên – diện tích lớn nhất tìm được.

Ví dụ:

PEDESTAL.INP
4
3 4
1 2
5 3
4 4

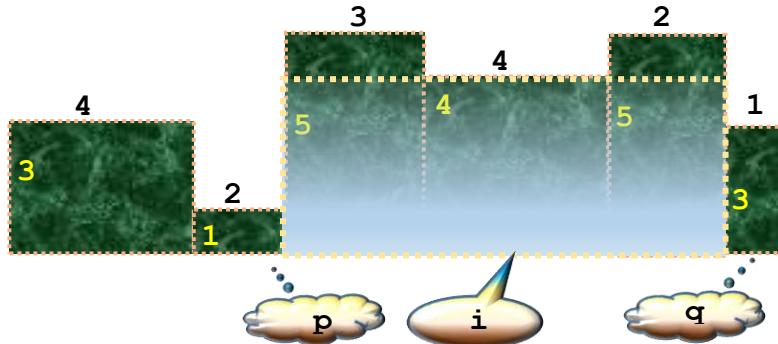
PEDESTAL.OUT
28



Giải thuật: Kỹ thuật 2 con trỏ, Cấu trúc dữ liệu stack, Tổng tiền tố.

Nhận xét:

Để tượng đài phải chứa nguyên một phiên nào đó trong phương án xé ban đầu, Nếu để chứa toàn bộ phiên i thì giới hạn chọn là phiên p gần nhất bên trái và phiên q gần nhất bên phải thấp hơn phiên i .



Như vậy, với mỗi i cần tìm p_i và q_i tương ứng.

Việc tìm p_i có thể thực hiện bằng cách tạo stack lưu các cặp dữ liệu (w_j, j) với $w_j < w_i$, $i = 1 \div n$,

Việc tìm q_i được thực hiện tương tự, nhưng dữ liệu được nạp vào stack theo trình tự từ n đến 1.

Để xác định độ rộng của để cần xây dựng tổng tiền tố f_i , trong đó f_i – tổng độ rộng các phiên từ 1 đến i , $i = 1 \div n$.

Nếu để chứa phiên i thì diện tích thiết diện của nó là $(f_{q-1} - f_p) \times h_i$.

Vấn đề còn lại chỉ là tìm \max trong số các diện tích tính được.

Tổ chức dữ liệu:

- Mảng `vector<pii>` $a(n+2)$ – lưu các cặp giá trị (h_i, i) ,
- Mảng `vector<int64_t>` $f(n+1)$ – lưu tổng tiền tố độ rộng các phiên,
- Mảng `vector<int>` $lf(n+1)$ – lưu các p_i ,
- Mảng `vector<int>` $rt(n+1)$ – lưu các q_i ,
- Mảng `vector<pii>` $s(n+1)$ – phục vụ tổ chức stack.

Xử lý:

Cần tổ chức các phần tử hàng rào $a_0=\{0, 0\}$ và $a_{n+1}=\{0, n+1\}$ để tiện tính toán,

Dùng vector thay cho công cụ stack của hệ thống để giảm thời gian xử lý và thuận tiện cho việc xóa rỗng stack.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "pedestal."
#define times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define ff first
#define ss second
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
typedef pair<int64_t,int> pli;

int main()
{
    int n,x,y;
    fi>>n;
    vector<pii> a(n+2);
    vector<int64_t> f(n+1);
    vector<int> lf(n+1),rt(n+1);
    f[0]=0;
    for(int i=1; i<=n; ++i)
    {
        fi>>x>>y;
        a[i]={x,i};
        f[i]=f[i-1]+y;
    }
    a[0]={0,0}; a[n+1]={0,n+1};
    vector<pii>s(n+1);
    int p=0,q;
    s[p]=a[0];
    for(int i=1; i<=n; ++i)
    {
        while(s[p].ff>= a[i].ff)--p;
        lf[i]=s[p].ss;
        s[++p]=a[i];
    }
    p=0; s[p]=a[n+1];
    for(int i=n; i>0;--i)
    {
        while(s[p].ff>= a[i].ff)--p;
        rt[i]=s[p].ss;
        s[++p]=a[i];
    }

    int64_t ans=0,area;
    for(int i=1; i<= n; ++i)
    {
        p=lf[i]; q=rt[i];
        area=(f[q-1]-f[p])*a[i].ff;
        ans=max(ans,area);
    }
    fo<<ans;
    times;
}
```



BA03. CHỦ ĐỀ

Tên chương trình: THEMES.CPP

UNESCO tổ chức một cuộc thi vẽ tranh cho thiếu nhi toàn thế giới và nhận được sự tham gia nhiệt tình của đông đảo bạn trẻ. Có n bức tranh được gửi tới tham gia dự thi. Tranh dự thi được phân loại theo chủ đề đánh số từ 1 trở đi. Theo kết quả phân loại, bức tranh thứ i có chủ đề a_i , $i = 1 \div n$.

Trong báo cáo tổng kết Ban giám khảo đã nêu số lượng chủ đề khác nhau được thiếu nhi thế giới quan tâm. Ví dụ, với $n = 11$ và các chủ đề tương ứng là 1, 2, 3, 4, 5, 1, 2, 1, 2, 7, 5 thì số chủ đề khác nhau được quan tâm là 6.

Hãy xác định số chủ đề khác nhau được bạn trẻ trên thế giới quan tâm.

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

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

Kết quả: Đưa ra file văn bản THEMES.OUT một số nguyên – số lượng chủ đề khác nhau.

Ví dụ:

THEMES.INP	THEMES.OUT
11 1 2 3 4 5 1 2 1 2 7 5	6



Giải thuật: Cơ sở lập trình .

Tồn tại nhiều cách triển khai giải thuật khác nhau phụ thuộc vào việc lựa chọn công cụ mà hệ thống lập trình C++ cung cấp.

Giải thuật 1: Sử dụng cấu trúc tập hợp `set<int>` s , đây là phương pháp dễ lập trình nhất (độ phức tạp lập trình thấp nhất), tuy thời gian thực hiện có lớn hơn đôi chút so với các giải thuật không dùng tập hợp.

Dữ liệu được nạp vào tập **s**.

Đặc trưng của tập hợp là chỉ lưu các giá trị khác nhau, vì vậy kết quả cần tìm sẽ là kích thước tập **s** sau khi nạp dữ liệu.

Giải thuật 2: Chỉ sử dụng hàm `sort` để sắp xếp dãy dữ liệu a_0, a_1, \dots, a_{n-1} . Các giá trị bằng nhau sẽ đứng cạnh nhau và vì vậy chỉ cần đếm số lượng cặp (a_i, a_{i+1}) , $i = 0, 1, \dots, n-2$ thỏa mãn điều kiện $a_i \neq a_{i+1}$.

Giải thuật 3: Sử dụng hàm `sort` để sắp xếp dữ liệu, sau đó dùng hàm `unique` để lọc những dữ liệu giống nhau và đứng cạnh nhau. Kết quả cần tìm là số lượng phần tử còn lại trong mảng.

Độ phức tạp của giải thuật:

Các loại giải thuật nêu trên đều có độ phức tạp chung là $O(n \log n)$,

Các khảo sát chi tiết cho thấy các giải thuật 2 và 3 thực hiện nhanh hơn đôi chút so với giải thuật 1.

Chương trình 1

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

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i)
    {
        fi>>x;
        s.insert(x);
    }
    fo<<s.size();
    Times;
}
```

Chương trình 2

```
#include <bits/stdc++.h>
#define NAME "themes."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,a[100001],ans;

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i) fi>>a[i];
    sort(a,a+n);
    ans=1;
    for(int i=1;i<n;++i) ans+=a[i]!=a[i-1];
    fo<<ans;
    Times;
}
```

Chương trình 3

```
#include <bits/stdc++.h>
#define NAME "themes."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,a[100001],ans;

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i) fi>>a[i];
    sort(a,a+n);
    ans=unique(a,a+n)-a;
    fo<<ans;
    Times;
}
```



Chương trình TV được phát đan xen với quảng cáo. Mỗi quảng cáo có thể xuất hiện nhiều lần trong tháng.

Steve không có mấy cảm tình với thông tin quảng cáo đan xen. Theo Steve, giới thiệu sản phẩm mới là cần thiết và đã có những kênh TV riêng, còn việc phát quảng cáo đan xen với nội dung chính của một chương trình chỉ là một nỗ lực với mục đích “*bán cát cho người Ả rập và bán tủ lạnh cho người Eskimo*”. Tuy vậy, ta đang sống trong một thế giới thực, bộ máy truyền thông cũng cần phải có thêm thu nhập, ta phải chấp nhận sự tồn tại của quảng cáo đan xen, phải thích nghi với thực tế v

à Steve đã tìm ra một cách thích nghi tuyệt vời – sử dụng quảng cáo như một công cụ rèn luyện trí nhớ. Khi xuất hiện một quảng cáo, Steve dựa vào một số đặc trưng trên màn hình, tính ra một số nguyên dương tương ứng với quảng cáo đó (tương tự như ánh xạ hàm băm) và xác định đây là quảng cáo mới, lần đầu tiên mình thấy hay là quảng cáo cũ, đã thấy trước đó.

Ở một cuốn sổ, Steve ghi số a_i tính được tương ứng với quảng cáo thứ i . Ở một cuốn sổ khác Steve ghi số 0 nếu đây là lần đầu tiên quảng cáo xuất hiện và ghi số 1 trong trường hợp ngược lại.

Sau một thời gian rèn luyện Steve có một khả năng quan sát tuyệt vời và trí nhớ tốt. Tất cả n quảng cáo nêu trong tháng đều được xác định đúng lần xuất hiện đầu tiên!

Hãy xác định dãy số 0, 1 ghi trong cuốn sổ thứ 2 của Steve.

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

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

Kết quả: Đưa ra file văn bản BLURB.OUT trên một dòng n số ghi trong cuốn sổ thứ 2, các số cách nhau một dấu cách.

Ví dụ:

BLURB.INP	BLURB.OUT
8	0 1 0 1 1 0 1 1
1 1 3 1 3 8 3 1	



Giải thuật: Cơ sở lập trình.

Giải thuật I:

Dữ liệu được lưu trữ dưới dạng cặp số (a_i, i).

Sau khi sắp xếp theo tiêu chuẩn tăng dần, các phần tử có a_i giống nhau sẽ đứng cạnh nhau, nhưng những phần tử đứng trước trong dãy ban đầu cũng vẫn đứng trước trong dãy đã sắp xếp (*tương tự như sắp xếp súi bot!*).

Việc lưu trữ giá trị kèm chỉ số cho phép xác định vị trí của số đầu tiên trong dãy các số bằng nó.

Để dẫn xuất kết quả cần có mảng đánh dấu các số xuất hiện là đầu trong quá trình duyệt mảng từ trái sang phải.

Độ phức tạp của giải thuật: O(nlogn).

Giải thuật II:

Sử dụng tập hợp `set<int> s`,

Khi nạp một phần tử x vào tập hợp hàm `s.insert(x)` trả về cặp giá trị

(*Vị trí, Kết quả nạp*)

Kết quả nạp = $\begin{cases} \text{True} & \text{nếu nạp được,} \\ \text{False} & \text{nếu không nạp được} \\ & (\text{trong tập hợp đã có giá trị này}). \end{cases}$

Khi quảng cáo x xuất hiện lần đầu tiên, x chưa có trong s và vì vậy nạp được vào tập hợp.

Như vậy, ta dễ dàng phân biệt lần gặp x đầu tiên với các lần gặp còn lại.

Chương trình sẽ ngắn gọn hơn nhiều.

Độ phức tạp của giải thuật: O(nlogn).

Chương trình 1

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

int main()
{
    fi>>n;
    vector<pair<int,int>>a(n);
    for(int i=0; i<n; ++i)
    {
        fi>>a[i].ff; a[i].ss=i;
    }
    sort(a.begin(),a.end());
    vector<int> ans(n,0);
    for(int i=1; i<n; ++i) if(a[i].ff==a[i-1].ff) ans[a[i].ss]=1;
    for(int i:ans) fo<<i<<' ';
    Times;
}
```

Chương trình 2

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

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i)
    {
        fi>>x;
        if(s.insert(x).second) fo<<"0 "; else fo<<"1 ";
    }
    Times;
}
```



BA06. NHÌ BẢNG

Tên chương trình: MAX2.CPP

Giải Cờ vây Quốc tế có nhiều bảng, mỗi bảng có thể có số lượng đấu thủ khác nhau. Các đấu thủ trong bảng thi đấu vòng tròn một lượt, tính điểm chọn ra 2 đấu thủ có điểm cao nhất để vào vòng đấu loại trực tiếp với đấu thủ được chọn ở bảng khác. Một bảng có n đấu thủ, kết thúc vòng bảng người thứ i có tổng điểm là a_i , $i = 1 \div n$.

Hãy xác định điểm của đấu thủ thứ 2 được chọn.

Dữ liệu: Vào từ file văn bản MAX2.INP:

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

Kết quả: Đưa ra file văn bản MAX2.OUT một số nguyên – điểm của đấu thủ thứ 2 được chọn.

Ví dụ:

MAX2.INP	MAX2.OUT
6	
8 2 4 5 3 2	5



BA06 Théoy AXIII

Giải thuật: Cơ sở lập trình .

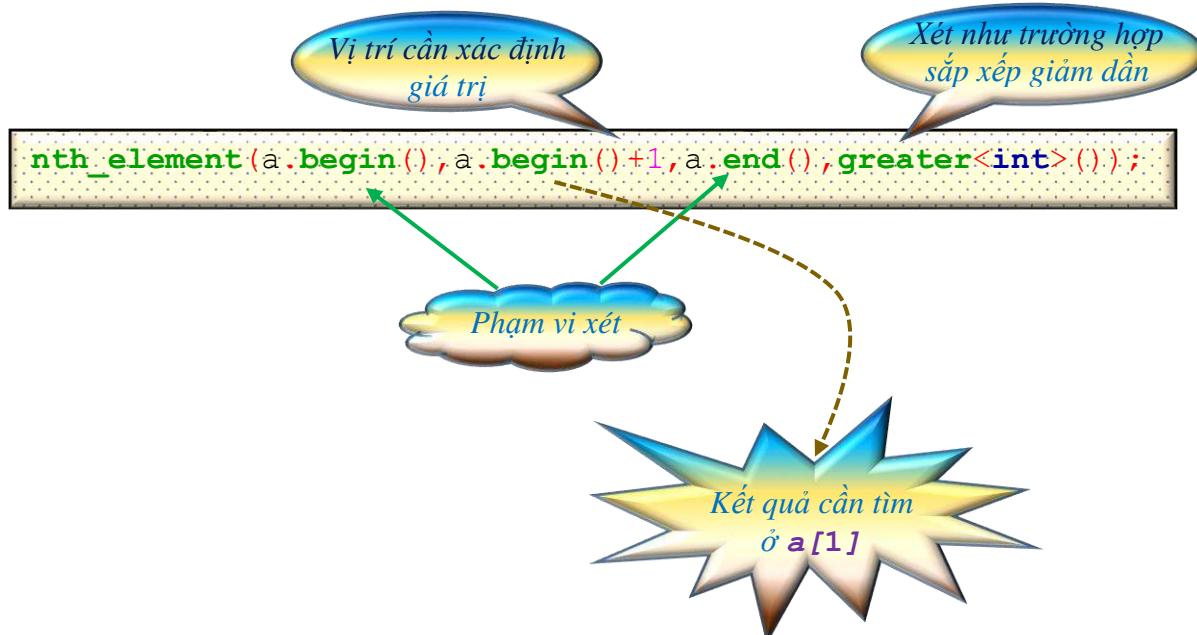
Việc tìm *min/max* thứ 2 là bài toán thường gặp trong thực tế và là trường hợp riêng của bài toán tìm *min/max* thứ **k**.

Dưới đây sẽ xét 2 giải thuật:

Giải thuật thứ I: Giải thuật tổng quát cho lớp bài toán này và cho phép dễ dàng mở rộng cho trường hợp cần tìm *min/max* thứ **k**,

Giải thuật thứ II: Chỉ áp dụng cho trường hợp tìm *min/max* thứ 2.

Giải thuật tổng quát được xây dựng dựa trên hàm ***nth_element*** của C++ cho phép đưa phần tử cần tìm về vị trí **k** trong dãy dữ liệu đang xử lý.



Hàm ***nth_element*** thường dùng để tìm *phần tử đứng tại vị trí giữa* của dãy đã sắp xếp mà ***không sắp xếp toàn dãy***.

Độ phức tạp xử lý trung bình của ***nth_element*** : $O(m)$, trong đó m – độ dài phạm đoạn dữ liệu cần xét.

Độ phức tạp của các giải thuật đã nêu: $O(n)$.

Chương trình 1

```
#include <bits/stdc++.h>
#define NAME "max2."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define ff first
#define ss second
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;

int main()
{
    fi>>n;
    vector<int>a(n);
    for(int i=0; i<n; ++i) fi>>a[i];
    nth_element(a.begin(), a.begin() + 1, a.end(), greater<int>());
    fo<<a[1];
    Times;
}
```

Chương trình 2

```
#include <bits/stdc++.h>
#define NAME "max2."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define ff first
#define ss second
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,x,b[2]={0};

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i)
    {
        fi>>x;
        if(b[1]<x) {b[1]=x; if(b[0]<b[1]) swap(b[0],b[1]);}
    }
    fo<<b[1];
    Times;
}
```



BA07. ĐỘ TĨNH LẶNG

Tên chương trình: SILENCE.CPP

Siêu núi lửa Yellowstone là một trong 5 núi lửa có thể gây thảm họa hủy diệt sự sống trên trái đất. Điều đáng lo ngại là các cơn địa chấn xảy ra ngày càng nhiều, thường xuyên hơn ở Yellowstone làm các nhà khoa học lo ngại. Các thiết bị đo địa chấn tự động ghi nhận và truyền về trung tâm xử lý số liệu đo đạc.



Để lôi cuốn sự chú ý của mọi người, dữ liệu được hiển thị trên một màn hình dạng như một khe hẹp chiếu lên k dữ liệu cuối cùng thu thập được. Cứ mỗi lần có địa chấn, dữ liệu bị đẩy chạy sang trái, dữ liệu cũ nhất bên trái sẽ bị mất đi, dữ liệu mới nhất sẽ xuất hiện bên phải của màn hình.

1	3	2	4	6	5	3	4
1	3	2	4	6	5	3	4
1	3	2	4	6	5	3	4
1	3	2	4	6	5	3	4
1	3	2	4	6	5	3	4

Điều mọi người quan tâm nhất là “Độ tĩnh lặng” của núi lửa đang ngủ này, tức là giá trị rung động nhỏ nhất ghi nhận được trong khoảng k địa chấn liên tiếp ứng với các dữ liệu trên màn hình.

Trong khoảng thời gian quan sát có n địa chấn được ghi nhận, cơn địa chấn thứ i có cường độ a_i , $i = 1 \div n$.

Hãy xác định Độ tĩnh lặng của núi lửa sau mỗi địa chấn kể từ địa chấn thứ k ghi nhận được.

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

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

Kết quả: Đưa ra file văn bản SILENCE.OUT trên một dòng $n-k+1$ số nguyên xác định các độ tĩnh lặng ghi nhận được, dữ liệu đưa ra theo trình tự ghi nhận được.

Ví dụ:

SILENCE.INP	SILENCE.OUT
8 4 1 3 2 4 6 5 3 4	1 2 2 3 3



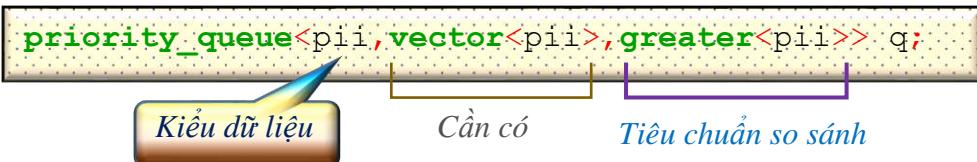
Giải thuật: Cơ sở lập trình.

Để quản lý min của các dữ liệu trong đoạn đã xét có thể dùng phòng đệm tổ chức dưới dạng hàng đợi ưu tiên,

Theo ngầm định, hàng đợi ưu tiên sắp xếp dữ liệu theo chiều giảm dần.

Để có hàng đợi ưu tiên theo chiều tăng dần (phục vụ tìm *min*):

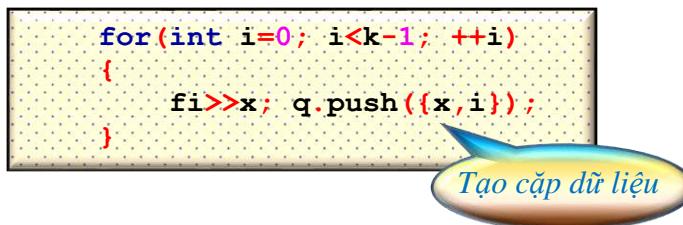
- ✚ Nếu các giá trị của dãy đều dương – nạp vào hàng đợi giá trị **-a[i]**,
- ✚ Trường hợp tổng quát: khai báo tiêu chuẩn so sánh



Xử lý:

Dữ liệu lưu trữ dưới dạng cặp số nguyên (**a_i, i**),

Nạp vào phòng đệm **k-1** giá trị đầu tiên của dãy



Gọi vị trí điểm đầu của đoạn độ dài **k** đang xét là **p**, ban đầu **p = 0**.

Với mỗi **a_i** tiếp theo:

- Nạp (**a_i, i**) vào phòng đệm,
- Chừng nào phần tử đầu phòng đệm có vị trí nhỏ hơn **p** – loại bỏ khỏi phòng đệm,
- Ghi nhận *min*, tăng **p** cho bước tiếp theo.

Độ phức tạp của giải thuật: O(nlogk).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "silence."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define ff first
#define ss second
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
int n,k,x,p=0;
pii t;
priority_queue<pii,vector<pii>,greater<pii>> q;

int main()
{
    fi>>n>>k;
    vector<int> ans(n-k+1);
    for(int i=0; i<k-1; ++i)
    {
        fi>>x; q.push({x,i});
    }
    for(int i=k-1; i<n; ++i)
    {
        fi>>x;
        q.push({x,i});
        t=q.top();
        while(t.ss<p) {q.pop(); t=q.top();}
        ans[p++]=t.ff;
    }
    for(int i:ans) fo<<i<<' ';
    Times;
}
```



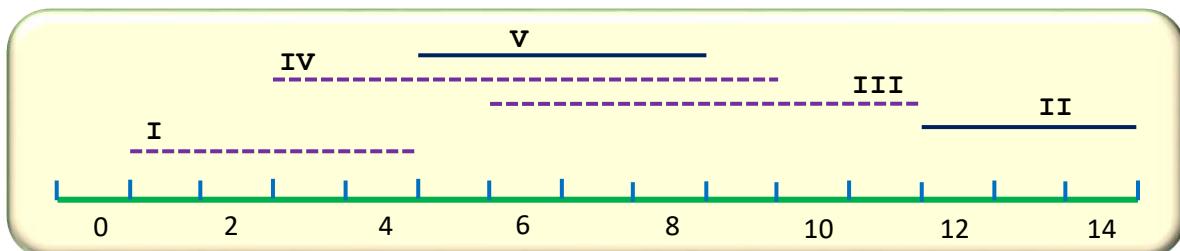
BA08. TẢI CHƯƠNG TRÌNH

Tên chương trình: PROGRAMS.CPP

Steve có một Flash Disk chuyên để lưu các chương trình ứng dụng tải về từ trên mạng. Sector 0 của Flash Disk được dùng để ghi thông tin quản lý đĩa, những sectors còn lại – được đánh số từ 1 trở đi và để ghi các chương trình tải về.

Sau khi học API (*Application Programming Interface – Giao diện lập trình ứng dụng*) Steve rất hào hứng và tự viết một chương trình tải thông tin từ mạng, ghi vào Flash Disk theo địa chỉ tuyệt đối từ sector x đến hết sector y . Với chương trình này Steve đã tải về n phần mềm, theo trình tự từ 1 đến n , phần mềm i được ghi bắt đầu từ sector x_i cho đến hết sector y_i ($i = 1 \div n$). Dung lượng Flash Disk đủ lớn để ghi được thông tin theo yêu cầu.

Do quên quản lý bộ nhớ tự do nên phần mềm tải về sau có thể ghi đè lên thông đã tải trước đó và như vậy, chỉ có các phần mềm tải về nào không bị các phần mềm tải sau ghi đè lên mới có thể sử dụng. Steve khá thất vọng về kết quả nhận được, nhưng trước khi bắt tay vào cải tiến chương trình tải thông tin của mình cần copy những phần mềm đã tải về còn dùng được vào ổ



C.

Hãy xác định số lượng phần mềm được copy vào ổ đĩa C và số thứ tự (theo trình tự tải về) của các phần mềm đó.

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

- ✚ Dòng đầu tiên chứa một 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 ($1 \leq x_i \leq y_i \leq 10^9$).

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

- ✿ Dòng thứ nhất chứa một số nguyên – số lượng phần mềm được copy vào C,
- ✿ Dòng thứ 2 chứa các số nguyên theo thứ tự tăng dần xác định các phần mềm được copy.

Ví dụ:

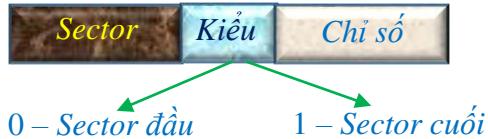
PROGRAMS.INP
5
1 4
12 14
6 11
3 9
5 8

PROGRAMS.OUT
2
2 5



Giải thuật: Hình học tính toán, kỹ thuật tổ chức dữ liệu .

Mỗi số trong cặp dữ liệu (\mathbf{x} , \mathbf{y}) được ghi nhận dưới dạng một nhóm 3:



Với cách đánh số kiểu như trên, khi sắp xếp dữ liệu, nếu có 2 giá trị đầu và cuối giống nhau, phần tử tương ứng với đầu khoảng sẽ đứng trước.

Ghi nhận dữ liệu và sắp xếp tăng dần,

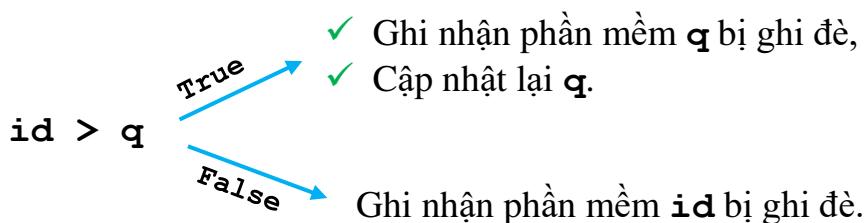
Dùng một mảng đánh dấu các phần mềm bị ghi đè: khi gặp điểm đầu – đánh dấu 2, gặp điểm cuối – đánh dấu 1.

Gọi \mathbf{q} là chỉ số của phần mềm chưa bị ghi đè tại thời điểm xét, ban đầu $\mathbf{q} = -1$ (chưa có phần mềm nào được xét),

Duyệt các dữ liệu đã ghi nhận:

Sector đầu khoảng: chỉ số \mathbf{id}

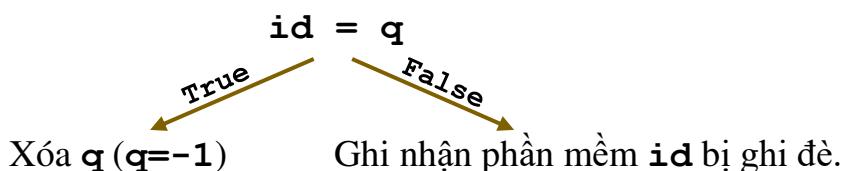
Với $\mathbf{q} \geq 0$:



Trường hợp $\mathbf{q} < 0$ – chưa có phần mềm nào có thể được lưu lại ở sector đang xét: cần xác định chỉ số \mathbf{idx} lớn nhất trong số các phần mềm đè lên sector \mathbf{s} đang xét (những phần mềm này đã có điểm đầu và chưa gặp điểm cuối). Nếu $\mathbf{id} > \mathbf{idx}$ – ghi nhận id là chỉ số phần mềm có thể được lưu, trong trường hợp ngược lại – đánh dấu phần mềm \mathbf{id} bị ghi đè.

Để tìm idx cần tổ chức hàng đợi ưu tiên ghi nhận \mathbf{id} của các phần mềm bị đánh dấu xóa đã gấp.

Sector cuối khoảng: chỉ số \mathbf{id}



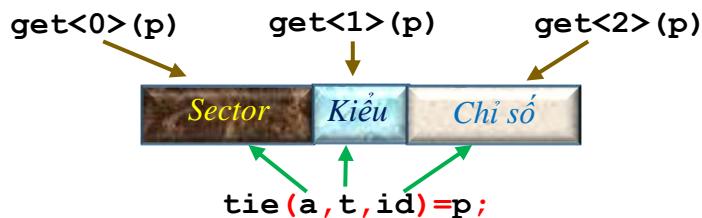
Tổ chức dữ liệu:

- Mảng `vector<ti>s (2*n)` – ghi nhận dữ liệu dưới dạng nhóm 3,
- Mảng `vector<int> ans (n, 0)` – đánh dấu phần mềm bị ghi đè.
- Hàng đợi `priority_queue<int> mx` – ghi nhận `id` các phần mềm đã bị đánh dấu xóa.

Xử lý:

Với **p** – một phần tử của **s**.

Có 2 cách truy nhập tới các trường của **p**:



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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "programs."
#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");
int n,x,y;
priority_queue<int> mx;

int main()
{
    fi>>n;
    vector<tii>s(2*n);
    for(int i=0; i<n; ++i)
    {
        fi>>x>>y;
        s[2*i]=make_tuple(x,0,i);
        s[2*i+1]=make_tuple(y,1,i);
    }
    sort(s.begin(),s.end());
    vector<int> ans(n,0);
    int a,t,tx,id,q=-1;
    for(auto &p:s)
    {
        tie(a,t,id)=p;
        if(t==0)
        {
            if(id>q && q>=0) {ans[q]=2; mx.push(q); q=id; continue;}
            if(id>q && q<0)
            {
                tx=-1;
                while(!mx.empty() && ans[mx.top()]==1) mx.pop();
                if(!mx.empty()) tx=mx.top();
                if(id>tx) {q=id; continue;}
            }
            mx.push(id); ans[id]=2; continue;
        }
        else
        {
            if(id==q) {q=-1; continue;} else ans[id]=1;
        }
    }
    x=n;
    for(int i:ans) x-=i;
    fo<<x<<'\n';
    for(int i=0; i<n; ++i) if(ans[i]==0) fo<<i+1<<' ';
    Times;
}
```



BA09. HOÁN VI

Tên chương trình: PERM.CPP

Hoán vị các số nguyên từ 1 đến n là dãy số nguyên $\mathbf{A} = (a_1, a_2, \dots, a_n)$, trong đó $a_i \neq a_j$ với $i \neq j$ và $1 \leq a_i \leq n$, $1 \leq i, j \leq n$. Ví dụ, $(2, 3, 4, 1)$ là một hoán vị các số từ 1 đến 4.

Xét hoán vị của các số nguyên từ 1 đến n . Hoán vị $\mathbf{A} = (a_1, a_2, \dots, a_n)$ gọi là nhỏ hơn hoán vị $\mathbf{B} = (b_1, b_2, \dots, b_n)$ nếu $a_1 < b_1$ hoặc tồn tại i ($1 < i \leq n$) để $a_j = b_j$ với $1 \leq j < i$ và $a_i < b_i$. Ví dụ, với $\mathbf{A} = (2, 3, 4, 1)$ và $\mathbf{B} = (2, 4, 1, 3)$ ta có $\mathbf{A} < \mathbf{B}$. Thứ tự được xác định như trên được gọi là thứ tự từ điển.

Cho q truy vấn, mỗi truy vấn có dạng $n \ k$ và yêu cầu xác định hoán vị có thứ tự từ điển k của các số từ 1 đến n . Các hoán vị được đánh số thứ tự bắt đầu từ 1.

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

- ⊕ Dòng đầu tiên chứa một số nguyên q ($1 \leq q \leq 1000$),
- ⊕ Mỗi dòng sau chứa 2 số nguyên n và k ($1 \leq n \leq 16$, $1 \leq k \leq n!$).

Kết quả: Đưa ra file văn bản PERM.OUT các hoán vị tìm được, mỗi hoán vị trên một dòng.

Ví dụ:

PERM.INP	PERM.OUT
7	1 2 3 4
4 1	1 2 4 3
4 2	2 1 3 4
4 7	2 1 4 3
4 8	4 3 2 1
4 24	2 1 3 4 5
5 25	2 1 4 3 5
5 27	



Giải thuật: Cơ sở lập trình .

Xét các hoán vị bắt đầu bằng số **d** ($1 \leq d \leq n$).

Dễ dàng thấy rằng có **(n-1)!** hoán vị bắt đầu bằng số **d**.

Từ đây với **k** cho trước có thể xác định được số đầu tiên trong hoán vị cần tìm:

$$d = (k-1) / (n-1)! + 1$$

Đánh dấu loại bỏ số **d** ta có bài toán tương tự ban đầu, kích thước nhỏ hơn, với số thứ tự mới là

$$k \% (n-1)!$$

và số lượng các số là **n-1**.

Với mỗi bài toán con: xác định số đầu tiên là số thứ mấy trong các số còn lại chưa sử dụng.

Tổ chức dữ liệu:

- ▀ Mảng **int64_t p[17]** – lưu bảng giá trị các giai thừa,
- ▀ Mảng **int d[17]** – đánh dấu các số chưa được chọn,
- ▀ Mảng **int r[17]** – lưu hoán vị tìm được.

Xử lý:

Tính bảng giá trị giai thừa:

```
p[0]=1;
for(int i=1; i<=16; ++i)p[i]=p[i-1]*i;
```

Xác định số đầu tiên trong các số còn lại chưa sử dụng ở mỗi bước:

Lưu ý phần tử cuối cùng!

```
fi>>n>>k; --k;
for(int i=0; i<=n; ++i)d[i]=0;
for(int i=1; i< n; ++i)
{
    r[i]=k/p[n-i]; k=k%p[n-i];
}
r[n]=0;
```

Ban đầu chưa có phần tử nào được sử dụng

Xác định hoán vị cần tìm:

```
for(int i=1; i<=n; ++i)
{
    m=0;
    for(int j=1; j<=n; ++j)
        if(d[j]==0){if(r[i]==m){r[i]=j; d[j]=1; break;} else ++m;}
}
```

Số thứ tự của phần tử chưa sử dụng

Độ phức tạp của giải thuật: $\approx O(q)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "perm."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,m,q,r[17],d[17];
int64_t p[17],k;

int main()
{
    fi>>q;
    p[0]=1;
    for(int i=1; i<=16; ++i)p[i]=p[i-1]*i;
    for(int iq=0; iq<q; ++iq)
    {
        fi>>n>>k; --k;
        for(int i=0; i<=n; ++i)d[i]=0;
        for(int i=1; i< n; ++i)
        {
            r[i]=k/p[n-i]; k=k%p[n-i];
        }
        r[n]=0;
        for(int i=1; i<=n; ++i)
        {
            m=0;
            for(int j=1; j<=n; ++j)
                if(d[j]==0) {if(r[i]==m) {r[i]=j; d[j]=1; break;} else ++m;}
        }
        for(int i=1; i<=n; ++i) fo<<r[i]<<' '; fo<<'\n';
    }

    Times;
}
```



VY03. MIỀN LIÊN THÔNG

Tên chương trình: CONNECTED.CPP

Có n người sống trong chung cư, đánh số từ 1 đến n . Quan sát các đối thoại khi đi chung thang máy người nhận biết được người x_i có cùng một ngành chuyên môn với người y_i ($1 \leq x_i, y_i \leq n$, $x_i \neq y_i$). Có m cặp quan hệ như vậy được ghi nhận. Có thể có những người không gặp trong thang máy bao giờ và họ thuộc nhóm những người có ngành không xác định.

Hãy xác định kết quả quan sát cho biết có bao nhiêu ngành nghề khác nhau (kể cả ngành không xác định) được xác định, mỗi ngành có bao nhiêu người và cụ thể là những ai.

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

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

Không có cặp giá trị nào trùng nhau.

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

- Dòng đầu tiên chứa số nguyên k – số ngành nghề khác nhau được xác định,
- k nhóm dòng mỗi nhóm 2 dòng:
 - ✿ Dòng thứ nhất trong nhóm: số người trong ngành tương ứng,
 - ✿ Dòng thứ 2 – nêu theo thứ tự tăng dần những người cùng ngành trong nhóm.

Ví dụ:

CONNECTED.INP
9 4
3 1
1 8
5 4
2 3

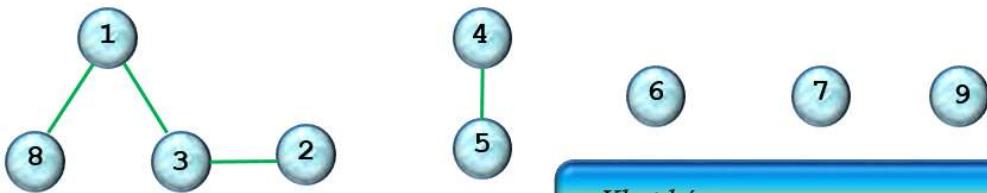
CONNECTED.OUT
3
4
1 2 3 8
2
4 5
3
6 7 9



VY03 Theory AXIII

Giải thuật: Kỹ thuật tổ chức dữ liệu, Loang theo chiều rộng.

Lưu cấu trúc đồ thị:



Đỉnh	Danh sách đỉnh kề
1	$a_1 = \{3, 8\}$
2	$a_2 = \{3\}$
3	$a_3 = \{1, 2\}$
4	$a_4 = \{5\}$
5	$a_5 = \{4\}$
6	$a_6 = \{\emptyset\}$
7	$a_7 = \{\emptyset\}$
8	$a_8 = \{1\}$
9	$a_9 = \{\emptyset\}$

Khai báo:
`vector<vector<int>> a (n+1);`

Nạp dữ liệu
`fi>>x>>y;`
`a[x].push_back(y);`
`a[y].push_back(x);`

Mỗi đỉnh \leftrightarrow vector lưu danh sách các đỉnh kề,

Sử dụng mảng **visited** đánh dấu các đỉnh đã tới thăm,

Ban đầu chưa có đỉnh nào được thăm,

Dùng dòng xếp hàng **q** ghi nhận các đỉnh cần tới thăm,

Loang theo chiều rộng (*Breadth – First Search – BFS*):

- Tìm đỉnh chưa tới thăm,
- Nạp danh sách đỉnh kề vào hàng đợi **q**,
- Đánh dấu đỉnh: đã tới thăm,
- Với mỗi đỉnh trong hàng đợi **q**:
 - ✿ Lấy đỉnh đứng đầu **ra khỏi** hàng đợi,
 - ✿ Nếu đỉnh đó chưa được thăm: Nạp danh sách đỉnh kề vào hàng đợi **q**,
 - ✿ Đánh dấu đã tới thăm.

Lặp lại cho đến khi mọi đỉnh đều được thăm
Lặp lại cho đến khi hàng đợi **q** rỗng

Với mỗi thành phần liên thông: cần có một tập *ghi nhận các đỉnh có thể đi tới được*.

Mỗi thành phần liên thông

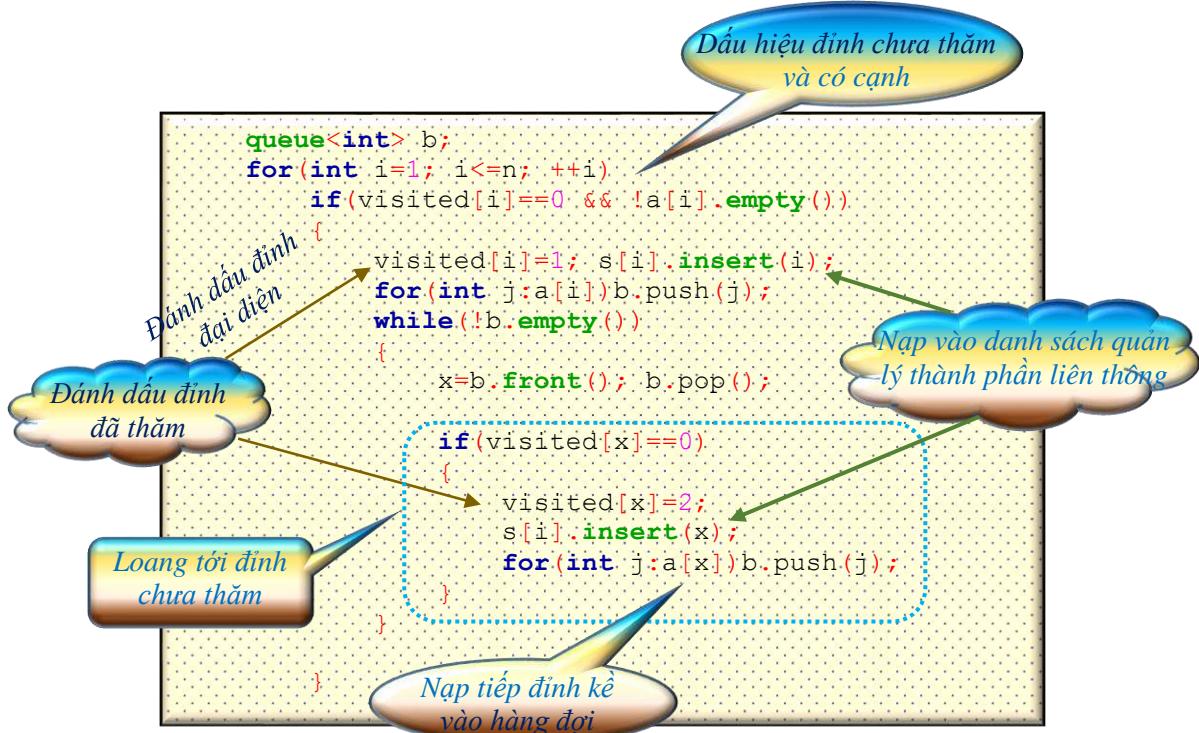
Một *đỉnh đại diện*

Tập *ghi nhận các đỉnh có thể đi tới được*

Tổ chức dữ liệu:

- Mảng hai chiều `vector<vector<int>> a (n+1)` – ghi nhận danh sách đỉnh kề,
- Mảng `vector<int> visited (n+1, 0)` – đánh dấu đỉnh tới thăm,
- Mảng các tập hợp `vector<set<int>> s (n+1)` – ghi nhận thành phần liên thông,
- Hàng đợi `queue<int> b` – phục vụ BFS.

Xử lý:



Loang theo chiều rộng (BFS):

Dẫn xuất kết quả: Cần lưu ý thống kê các đỉnh không nối với đỉnh nào khác.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

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

int main()
{
    fi>>n>>m;
    vector<vector<int>>a(n+1);
    vector<int>visited(n+1,0);
    vector<set<int>>s(n+1);
    for(int i=0; i<m; ++i)
    {
        fi>>x>>y;
        a[x].push_back(y);
        a[y].push_back(x);
    }
    queue<int> b;
    for(int i=1; i<=n; ++i)
        if(visited[i]==0 && !a[i].empty())
    {
        visited[i]=1; s[i].insert(i);
        for(int j:a[i])b.push(j);
        while(!b.empty())
        {
            x=b.front(); b.pop();

            if(visited[x]==0)
            {
                visited[x]=2;
                s[i].insert(x);
                for(int j:a[x])b.push(j);
            }
        }
    }
    int res0=0, res1=0;
    for(int i=1; i<=n; ++i) res1+=visited[i]==1, res0+=visited[i]==0;
    fo<<res1+(res0>0)<<'\n';
    for(int i=1; i<=n; ++i)
        if(visited[i]==0)s[0].insert(i);
        else if(visited[i]==1)
        {
            fo<<s[i].size()<<'\n';
            for(int j:s[i])fo<<j<<' '; fo<<'\n';
        }
    if(res0>0){fo<<res0<<'\n'; for(int i:s[0])fo<<i<<' ';}
    Times;
}
```



VY04. NGHỆ THUẬT IKEBANA

Tên chương trình: IKEBANA.CPP

Nghệ thuật cắm hoa Ikebana của Nhật Bản được các bạn trẻ rất ưa chuộng. Một bạn đã chụp ảnh các chậu hoa mình cắm được và chia sẻ tác phẩm của mình trên Facebook.



Trên ảnh có n chậu hoa, mỗi chậu hoa được đặc trưng bằng một số nguyên, chậu hoa thứ i thuộc loại a_i , $i = 1 \div n$. Các chậu hoa giống nhau có cùng

một số đặc trưng. Bức ảnh nhận được rất nhiều Likes, nhưng cũng gây ra một cuộc tranh luận sôi nổi.

Vấn đề ở chỗ tác giả cho biết các chậu hoa được đặt trước một tấm gương để tăng hiệu ứng thị giác. Như vậy trên ảnh *một số chậu hoa* (hoặc tất cả) được nhìn thấy qua gương, tức là trên ảnh có thể có chậu hoa thật, nhưng cũng có thể là ảnh của nó!

Vậy thực tế tác giả đã cắm được bao nhiêu chậu hoa? Tất cả phụ thuộc vào vị trí chụp ảnh trước tấm gương. Ở vị trí I mọi chậu hoa được chụp qua gương và như vậy có 6 chậu hoa thật. Ở vị trí II của gương chậu hoa trái nhất trong ảnh là thực, còn lại – nhìn qua gương, như vậy tác giả đã cắm 5 chậu hoa. Ở vị trí III tất cả các chậu hoa được chụp cả thật lẫn ảnh qua gương, như vậy tác giả chỉ cắm có 3 chậu hoa!

Hãy xác định có bao nhiêu vị trí chụp ảnh có thể và ứng với nó – số chậu hoa thực tế tác giả đã cắm.

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

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

Kết quả: Đưa ra file văn bản IKEBANA.OUT dòng đầu tiên chứa số nguyên k – số vị trí chụp ảnh có thể, dòng thứ 2 chứa k số nguyên theo thứ tự giảm dần – số chậu hoa thực tế đã cắm xác định được ứng với từng vị trí chụp ảnh.

Ví dụ:

IKEBANA.INP
6
1 1 2 2 1 1

IKEBANA.OUT
3
6 5 3



VY04 Theory A XIII

Giải thuật: Hàm băm.

Luôn tồn tại một vị trí chụp ảnh: chụp ngay trước tấm gương , như vậy *mỗi chậu hoa đều chỉ nhìn thấy một lần* qua hình trong gương.

Nếu tồn tại một vị trí chụp nào khác: ảnh chụp tạo ra một *tiền tố độ dài chẵn* của dãy số là một *dãy con palindrome*.



Một trong những phương pháp hiệu quả nhận dạng palindrome là kỹ thuật hàm băm.

Với một p đủ lớn, xây dựng 2 hàm băm $f1$ và fr :

$$f1_i = \sum_{j=0}^i a_j \times p^j \text{ và } fr_i = \sum_{j=n-1}^i a_j \times p^{n-1-j}$$

$a_0 \quad a_1 \dots a_i$	$a_{i+1} \quad a_{i+2} \dots a_{2i+1}$	$a_{2i+2} \dots a_{n-1}$
$u = f1_i$	$v = fr_{i+1} - fr_{2i+2} \times p^{i+1}$	

Nếu $u = v$ thì tiền tố gồm $2(i+1)$ phần từ đầu tiên là một palindrome.

Các giá trị u khác nhau tương ứng với $i = 0 \div n/2$.

Tổ chức dữ liệu:

- Mảng `vector<int64_t> h(n)` – lưu các giá trị p^i ,
- Mảng `vector<int> a(n)` – lưu dãy số ban đầu,
- Các mảng `vector<int64_t> f1(n), fr(n+1)` – lưu giá trị của 2 hàm băm,
- Mảng `vector<int> ans` – lưu kết quả tìm được.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "ikebana."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
#define ff first
#define ss second
using namespace std;
const int64_t p=1e9+7;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;

int main()
{
    fi>>n;
    vector<int64_t> h(n), f1(n), fr(n+1);
    vector<int> a(n), ans;
    h[0]=1;
    for(int i=1; i<n; ++i) h[i]=h[i-1]*p;
    for(int i=0; i<n; ++i) fi>>a[i];
    f1[0]=a[0]; fr[n-1]=a[n-1]; fr[n]=0;
    for(int i=1; i<n; ++i) f1[i]=f1[i-1]*p+a[i];
    for(int i=n-2; i>=0; --i) fr[i]=fr[i+1]*p+a[i];
    ans.push_back(n);
    for(int i=0; i<n/2; ++i)
    {
        if((fr[i+1]-fr[2*i+2]*h[i+1])==f1[i]) ans.push_back(n-i-1);
    }
    fo<<ans.size()<<'\n';
    for(int i:ans) fo<<i<<' ';
    Times;
}
```



VY05. CHUỖI ĐIỀU HÒA

Tên chương trình: HARMONIC.CPP

Tổng vô hạn $\sum_{i=1}^{\infty} \frac{1}{i}$ được gọi là chuỗi điều hòa.

Xét h_n – tổng n số hạng đầu tiên của chuỗi điều hòa.

$$h_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = \frac{P}{Q}$$

trong đó $\frac{P}{Q}$ là phân số tối giản, P, Q – các số nguyên dương.

Ví dụ, với $n = 3$, $h_3 = 1 + \frac{1}{2} + \frac{1}{3} = \frac{11}{6}$, ta có $P = 11$, $Q = 6$.

Với n cho trước, hãy xác định P và Q .

Dữ liệu: Vào từ file văn bản HARMONIC.INP gồm một dòng chứa số nguyên n ($2 \leq n \leq 10^4$).

Kết quả: Đưa ra file văn bản HARMONIC.OUT, dòng đầu tiên chứa số nguyên P , dòng thứ 2 chứa số nguyên Q .

Ví dụ:

HARMONIC.INP	HARMONIC.OUT
25	34052522467 8923714800



VY05 JBOI20180711 A AXIII

Giải thuật: Xác định số nguyên tố, Xử lý số lớn.

Xét $h_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$

Mẫu số chung **ms** của các phân số trong tổng sẽ là tích tất cả các số nguyên tố không vượt quá **n**, mỗi số tham gia vào mẫu số chung với bậc cao nhất gấp khi lần lượt phân tích các số từ 2 đến **n** ra thừa số nguyên tố.

Số **n** cần xét không lớn ($\leq 10^4$), vì vậy có thể dùng các giải thuật đơn giản (như sàng Eratosthenes) để tìm các số nguyên tố.

Mẫu số chung **ms** được tính theo sơ đồ nhân một số lớn với số nhỏ.

Để có tử số cần phải tính các phân số dạng **ms/i** ($i = 1 \div n$) và lấy tổng các kết quả nhận được. Việc tính toán được thực hiện dựa trên giải thuật chia một số lớn cho số nhỏ và giải thuật cộng các số lớn.

Trước khi đưa ra kết quả cần tìm các ước số chung của tử và mẫu số, tiến hành giản ước.

Tổ chức dữ liệu:

- ▀ Mảng **int p[10001]={0}** – để lọc số nguyên tố,
- ▀ Mảng **vector<int>prime** – lưu các số nguyên tố khác nhau tìm được,
- ▀ Mảng **vector<int> pr_p** – lưu bậc của các số nguyên tố tham gia vào mẫu số,
- ▀ Mảng **vector<int> pw** – lưu lũy thừa các số nguyên tố trong mẫu số,
- ▀ Các mảng **vector<int64_t> res_a, res_b** – lưu kết quả.

Xử lý:

Sàng Eratosthenes tìm các số nguyên tố:

```

    if(i>n);
    n2=sqrt(n)+1;
    for(int i=2; i<n2; ++i)
        if(p[i]==0) for(int j=i*i; j<=n; j+=i) p[j]=1;
    for(int i=2; i<=n; ++i) if(p[i]==0) prime.push_back(i);

```

*Đánh dấu
hợp số*

Tính bậc các số nguyên tố trong mẫu số:

```

m=prime.size(); pw.resize(m); pr_p.resize(m);
for(int i=0; i<m; ++i)
{
    k=n, t=0; tp=prime[i]; r=tp;
    while(r<=n) ++t; r*=tp; r/=tp;
    pw[i]=r; pr_p[i]=t;
}

```

Sơ đồ nhân số lớn với số bé để tính mẫu số:

```
void mulp(int x)
{
    int64_t tm;
    int km=res_b.size();
    tm=0;
    for(int i=0; i<km; ++i)
    {
        tm+=res_b[i]*x;
        res_b[i]=tm%b;
        tm/=b;
    }
    if(tm>0) res_b.push_back(tm);
}
```

Phát sinh
số mới

Tính tử số:

```
void get_a()
{
    int kg=res_b.size(); res_a=res_b;
    vector<int64_t> tv;
    for(int i=2; i<=n; ++i)
    {
        tv=res_b;
        dvs(tv,i); add(tv);
    }
}
```

Chú ý

```
void dvs(vector<int64_t> &v, int x)
{
    int64_t tm=0, t1;
    int kd=v.size();
    for(int i=kd-1; i>=0; --i)
    {
        t1=tm*x+v[i];
        tm=t1%x; v[i]=t1/x;
    }
    r=tm;
}
```

Còn được sử dụng để
kiểm tra chia hết

```
void add(vector<int64_t> v)
{
    int d=0, int64_t tm;
    int ka=res_a.size();
    int kv=v.size();
    if(kv<ka)
        for(int i=0; i<ka-kv; ++i)
            v.push_back(0);
    for(int i=0; i<ka; ++i)
    {
        tm=res_a[i]+v[i]+d;
        res_a[i]=tm%b; d=tm/b;
    }
    if(d>0) res_a.push_back(d);
}
```

Cân bằng
độ dài

Sơ đồ giản ước:

```
void norm1()
{
    int d=1;
    vector<int64_t> tv;
    int kp=prime.size();
    for(int i=0; i<kp; ++i)
    {
        int t=prime[i], tp=pr_p[i];
        r=0; tv=res_a;
        while(r==0 && tp) {dvs(tv, t); if(r==0) {d*=t; res_a=tv; --tp;}}
    }
    if(d>1) dvs(res_b, d);
    k=res_b.size()-1;
    while(res_b[k]==0) {res_b.pop_back(); --k;}
}
```

Xóa số 0 không có nghĩa

Dám bảo mẫu số cũng chia hết

Các đưa ra số lớn:

```
void prt(vector<int64_t> v)
{
    k=v.size();
    fo<<v[k-1];
    for(int i=k-2; i>=0; --i) fo<<setw(9)<<setfill('0')<<v[i];
    fo<<'\'n';
}
```

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include<bits/stdc++.h>
#define b 1000000000
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi("harmonic.inp");
ofstream fo("harmonic.out");
typedef pair<int,int> pii;

int n,n2,m,k,t,r,tp,p[10001]={0};
vector<int> prime,pw,pr_p;
vector<int64_t> res_a,res_b;

void prt(vector<int64_t> v)
{
    k=v.size();
    fo<<v[k-1];
    for(int i=k-2; i>=0; --i) fo<<setw(9)<<setfill('0')<<v[i];
    fo<<'\'\n';
}

void dvs(vector<int64_t> &v, int x)
{
    int64_t tm=0,t1;
    int kd=v.size();
    for(int i=kd-1; i>=0; --i)
    {
        t1=tm*b+v[i];
        tm=t1%x; v[i]=t1/x;
    }
    r=tm;
}

void add(vector<int64_t> v)
{
    int d=0; int64_t tm;
    int ka=res_a.size();
    int kv=v.size();
    if(kv<ka) for(int i=0; i<ka-kv; ++i) v.push_back(0);
    for(int i=0; i<ka; ++i)
    {
        tm=res_a[i]+v[i]+d;
        res_a[i]=tm%b; d=tm/b;
    }
    if(d>0) res_a.push_back(d);
}

void get_a()
{
    int kg=res_b.size(); res_a=res_b;
    vector<int64_t> tv;
    for(int i=2; i<=n; ++i)
    {
        tv=res_b;
        dvs(tv,i); add(tv);
    }
}
void mulp(int x)
{
    int64_t tm;
    int km=res_b.size();
```

```

tm=0;
for(int i=0; i<km; ++i)
{
    tm+=res_b[i]*x;
    res_b[i]= tm%b;
    tm/=b;
}
if(tm>0) res_b.push_back(tm);
}

void norml()
{
    int d=1;
    vector<int64_t> tv;
    int kp=prime.size();
    for(int i=0; i<kp; ++i)
    {
        int t=prime[i], tp=pr_p[i];
        r=0; tv=res_a;
        while(r==0 &&tp) {dvs(tv,t); if(r==0) {d*=t; res_a=tv; --tp;}}
    }
    if(d>1) dvs(res_b,d);
    k=res_b.size()-1;
    while(res_b[k]==0) {res_b.pop_back(); --k;}
}

int main()
{
    fi>n;
    n2=sqrt(n)+1;
    for(int i=2; i<n2; ++i)
        if(p[i]==0) for(int j=i*i; j<=n; j+=i) p[j]=1;
    for(int i=2; i<=n; ++i) if(p[i]==0) prime.push_back(i);

    m=prime.size(); pw.resize(m); pr_p.resize(m);
    for(int i=0; i<m; ++i)
    {
        k=n, t=0; tp=prime[i]; r=tp;
        while(r<=n) ++t, r*=tp; r/=tp;
        pw[i]=r; pr_p[i]=t;
    }

    res_b.push_back(pw[0]);
    for(int i=1; i<m; ++i) mulp(pw[i]);
    get_a();
    norml();
    prt(res_a);   prt(res_b);
    Times;
}

```



VY06. SỐ K BÍT 1

Tên chương trình: K_BIN.CPP

Xét các số nguyên dương ở dạng biểu diễn nhị phân có đúng k bit 1 và gọi tập các đó là lớp k . Ví dụ, số $23_{10} = 10111_2$ là một số ở lớp 4.

Cho 2 số nguyên n và k .

Hãy tính tổng các số nhỏ hơn n và thuộc lớp k .

Dữ liệu: Vào từ file văn bản K_BIN.INP gồm một dòng chứa 2 số nguyên n và k ($2 \leq n \leq 10^{15}$, $0 \leq k \leq 50$).

Kết quả: Đưa ra file văn bản K_BIN.OUT một số nguyên – số lượng số tính được theo mô đun 1234567.

Ví dụ:

K_BIN.INP	K_BIN.OUT
15 3	45



VY06 JBOI20180711 B A XIII

Giải thuật: Sơ đồ lắp (Quy hoạch động đơn giản).

Xét i là số nguyên dương có đúng k bit 1 trong dạng biểu diễn nhị phân.

Nhận xét:

- + Nếu i chẵn thì $i/2$ cũng là số nguyên dương có đúng k bit 1 trong dạng biểu diễn nhị phân,
- + Nếu i lẻ, $i/2$ có $k-1$ bit 1 trong dạng biểu diễn nhị phân.

Ký hiệu $x_1 = n-1$, $x_2 = x_1/2$, $x_3 = x_2/2$, ..., $x_{nx} = 1$.

Gọi:

$s_{i,k}$ – tổng các số nhỏ hơn hoặc bằng i có k bit 1,

$c_{i,k}$ – số lượng các số nhỏ hơn hoặc bằng i có k bit 1,

x_i – số lượng bit có nghĩa của i .

Ta có:

$$s_{i,1} = 2^{x_i},$$

$$c_{i,1} = x_i$$

và bộ công thức lắp:

$$s_{i,k} = 2 \times s_{i/2,k} + 2 \times s_{i/2,k-1} + c_{i/2,k-1},$$

$$c_{i,k} = c_{i/2,k} + c_{i/2,k-1},$$

với $i = x_{nx}=1, x_{nx-1}, \dots, x_1$.

Kết quả: Lưu ở $s_{n-1,k}$.

Lưu ý:

Nếu $i+1$ là lẻ và có k bit 1 thì phải bớt $s_{i,k}$ đi $i+1$ và giảm 1 ở $c_{i,k}$,

Để thuận tiện lập trình thay vì giá trị trực tiếp i – sử dụng chỉ số tương ứng trong mảng \mathbf{x} .

Tổ chức dữ liệu:

- ─ Mảng `int64_t` $x[70]$ – lưu các giá trị phục vụ tính lắp,
- ─ Mảng `int64_t` $s[70][70]$ – lưu tổng giá trị các số,
- ─ Mảng `int64_t` $c[70][70]$ – lưu số lượng số.

Độ phức tạp của giải thuật: $O(lnn)$.

Chương trình

```
#include<bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi("k_bin.inp");
ofstream fo("k_bin.out");
const int M = 1234567;
int64_t n,k,S[70][70],C[70][70],i,j,x[70],a,nx,m,p,s;

int main()
{
    fi>>n>>k; m=n-1; nx=0;
    while(m) x[++nx]=m, m=m/2;
    for(p=0,s=0,i=nx;i>=1;i--)
    {
        a=x[i]; p++; s=s+a%2;
        S[i][1]=((1LL<<p)-1)%M;
        C[i][1]=p;
        for(j=2; (1LL<<j)-1<=a && j<=k; j++)
        {
            S[i][j]=(2*S[i+1][j]+2*S[i+1][j-1]+C[i+1][j-1])%M;
            C[i][j]=(C[i+1][j]+C[i+1][j-1])%M;
            if(a%2==0 && s+1==j)
            {
                S[i][j]=(S[i][j]+M-(a+1)%M)%M;
                C[i][j]=(C[i][j]+M-1)%M;
            }
        }
    }
    fo<<S[1][k];
    Times;
}
```

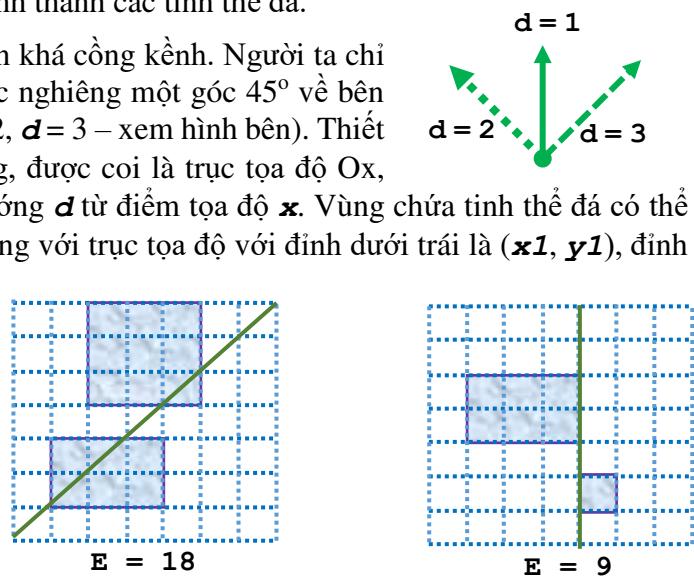


Mưa đá gây thiệt hại lớn cho mùa màng và có thể cho cả nhà cửa. Người ta đang thí nghiệm một phương pháp mới làm tan các đám mây đá, chuyển nó sang trạng thái mây thường không chứa các tinh thể nước đá. Việc thay đổi trạng thái mây được thực hiện bằng cách chiếu một tia laser mạnh vào đó. Các tinh thể đá sẽ khuỷu tảng tia laser, truyền năng lượng sang xung quanh làm mất điều kiện cân bằng để hình thành các tinh thể đá.

Thiết bị đang ở giai đoạn thí nghiệm nên khá cồng kềnh. Người ta chỉ có thể chiếu thẳng tia laser lên trời hoặc nghiêng một góc 45° về bên phải hay bên trái (các hướng $d = 1, d = 2, d = 3$ – xem hình bên). Thiết bị được di chuyển trên đường ray thẳng, được coi là trục tọa độ Ox, mỗi lần thử nghiệm được chiếu theo hướng d từ điểm tọa độ x . Vùng chứa tinh thể đá có thể coi như một hình chữ nhật cạnh song song với trục tọa độ với đỉnh dưới trái là (x_1, y_1) , đỉnh trên phải là (x_2, y_2) .

Năng lượng truyền vào đám mây bằng bình phương tổng độ dài đoạn tia laser tiếp xúc với đám mây (trên biên hoặc bên trong).

Năng lượng hữu ích E của một lần chiếu bằng tổng năng lượng đã truyền vào các đám mây mà tia laser tiếp xúc.



Cho tọa độ cặp đỉnh của n đám mây, tọa độ và hướng của t lần chiếu tia laser. Hãy xác định năng lượng hữu ích của mỗi lần chiếu.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 5 \times 10^4$),
- ✚ Dòng thứ i trong n dòng tiếp theo chứa 4 số nguyên $x_{1i}, y_{1i}, x_{2i}, y_{2i}$ xác định vùng tinh thể đá thứ i ($1 \leq x_{1i}, x_{2i}, y_{1i}, y_{2i} \leq 10^5$, $x_{1i} < x_{2i}$, $y_{1i} < y_{2i}$), hai vùng bất kỳ không giao nhau với diện tích khác 0.
- ✚ Dòng tiếp theo chứa số nguyên t ($1 \leq t \leq 10^5$),
- ✚ Dòng thứ j trong t dòng tiếp theo chứa 2 số nguyên x_j và d_j ($-10^5 \leq x_j \leq 2 \times 10^5$).

Kết quả: Đưa ra file văn bản LAZER.OUT các năng lượng hữu ích tính được, mỗi giá trị trên một dòng dưới dạng số nguyên.

Ví dụ:

LAZER.INP
4
1 1 4 3
2 4 5 7
6 3 10 5
8 1 9 2
2
0 3
9 1

LAZER.OUT
18
9



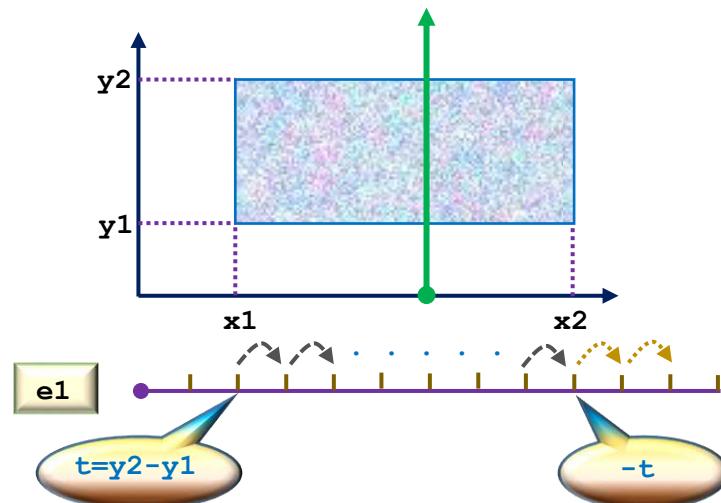
Giải thuật: *Tổng tiền tố (Sơ đồ kép).*

Để thuận tiện cho việc đánh chỉ số trong C++ cần tính tiền tọa độ sang phải 10^5 để tọa độ các điểm tham gia vào tính toán sẽ không âm.

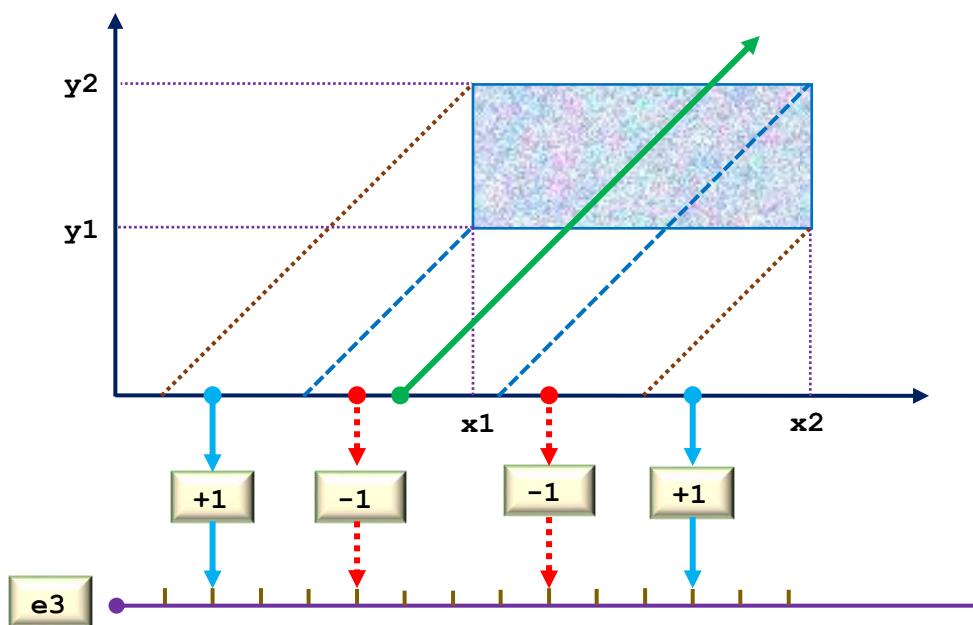
Xét sự ảnh hưởng đến năng lượng hữu ích của một đám mây theo từng hướng chiếu.

Các mảng **e1**, **e2** và **e3**: **ex_i** lưu thông tin về năng lượng hữu ích ở điểm **i** tương ứng với hướng chiếu **x**.

Hướng **d = 1**:



Hướng **d = 3**:



Hướng **d = 2**: Có bức tranh xử lý tương tự trường hợp **d = 3**.

Đối với mảng **e1**: việc tính tổng tiền tố một lần đủ để tổng hợp thông tin cần thiết.

Với các mảng **e2** và **e3**:

- ✳ Tính tổng tiền tố lần thứ I: Xác định bức tranh tích lũy năng lượng,

- 🌟 Tính tổng tiền tố lần thứ II: Xác định giá trị năng lượng cần xác định.
- 🌟 *Lưu ý:* Mỗi đơn vị tích lũy ở đây là $\sqrt{2}$, vì vậy sau khi bình phương, cần *nhân đôi* kết quả nhận được.

Tổ chức dữ liệu:

Các mảng `int e1[300005]={0}, e2[300005]={0}, e3[300005]={0}` – lưu tổng tiền tố cho các hướng.

Độ phức tạp của giải thuật: $\approx O(\max(n, t))$.

Chương trình

```
#include<bits/stdc++.h>
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi("lazer.inp");
ofstream fo("lazer.out");
int n,k,t,x1,x2,y1,y2,xx1,xx2,yy1,yy2;
int e1[300005]={0},e2[300005]={0},e3[300005]={0};
int64_t E;
int main()
{
    fi>>n;
    for(int i=1;i<=n;i++)
    {
        fi>>xx1>>yy1>>xx2>>yy2;
        xx1+=100000;xx2+=100000;
        // d = 3
        x1=xx1-yy1; x2=xx2-yy1; y2=yy2-yy1; y1=0;
        k=min(x2-x1,y2);
        e3[(x1-y2)+1]++;e3[(x1-y2)+1+k]--;
        e3[x2-(k-1)]--;e3[x2+1]++;
        // d = 2
        x1=xx1+yy1; x2=xx2+yy1; y2=yy2-yy1; y1=0;
        k=min(x2-x1,y2);
        e2[x1+1]++;e2[x1+1+k]--;
        e2[x2+y2-(k-1)]--;e2[x2+y2+1]++;
        // d = 1
        e1[xx1]+=(yy2-yy1);
        e1[xx2+1]-=(yy2-yy1);
    }
    for(int i=1;i<=300000;i++)
    {
        e1[i]=e1[i]+e1[i-1];
        e2[i]=e2[i]+e2[i-1];
        e3[i]=e3[i]+e3[i-1];
    }
    for(int i=1;i<=300000;i++)
    {
        e2[i]=e2[i]+e2[i-1];
        e3[i]=e3[i]+e3[i-1];
    }
    fi>>t;
    for(int i=1;i<=t;i++)
    {
        fi>>x1>>k;
        x1+=100000;
        switch (k)
        {
            case 1: E=(int64_t)e1[x1]*e1[x1]; break;
            case 2: E=(int64_t)e2[x1]*e2[x1]*2; break;
            case 3: E=(int64_t)e3[x1]*e3[x1]*2; break;
        }
        fo<<E<<"\n";
    }
    Times;
}
```



Đường dẫn vào khách sạn nơi thường tổ chức những cuộc gặp gỡ của các tổ chức quốc tế được trang trí bằng một dãy n chậu hoa, mỗi loại hoa một màu được đánh số từ 1 đến n tương ứng với trình tự xuất hiện của màu trong bảng màu cầu vòng. Khi có hội nghị về một nước nào đó, các chậu hoa được bố trí theo trình tự thể hiện màu cờ của nước đó.

Hôm nay người ta chuẩn bị cho hội nghị khoa học về bảo vệ môi trường, Ban Tổ chức quyết định bố trí lại các chậu hoa phản ánh trạng thái cầu vòng, tức là phải sắp xếp các chậu hoa từ trái sang phải theo trình tự tăng dần của màu hoa (từ 1 đến n). Hiện tại, ở vị trí i đang có chậu hoa màu c_i , $i = 1 \dots n$, $c_i \neq c_j$ với $i \neq j$.

Ở thời đại cách mạng KHKT 4.0 mọi thứ đều được tự động hóa. Một robot được tách ra để bố trí lại các chậu hoa. Cứ mỗi đơn vị thời gian robot thực hiện được một thao tác đưa chậu hoa từ vị trí i về vị trí j với chi phí năng lượng là $(i-j)^2$. Các thực hiện nhanh nhất là xét và đưa một số chậu về vị trí đúng của mình. Ví dụ, với trạng thái hiện tại 1, 7, 3, 4, 5, 2, 6: Đưa chậu 2 về vị trí 2 với chi phí $(6-2)^2$, có trạng thái 1, 2, 7, 3, 4, 5, 6, sau đó – đưa chậu 7 về vị trí cuối cùng với chi phí $(7-3)^2$, kết quả là có dãy chậu hoa 1, 2, 3, 4, 5, 6, 7. Đây là cách thực hiện nhanh nhất về thời gian với chi phí năng lượng là $4^2 + 4^2 = 32$. Đáng tiếc pin của robot chỉ còn e đơn vị năng lượng.

Hãy xác định khoảng thời gian ngắn nhất (tức là số thao tác ít nhất) robot hoàn thành nhiệm vụ được giao với phạm vi năng lượng hiện có.

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

- ✚ Dòng đầu tiên chứa hai số nguyên n và e ($1 \leq n \leq 10^5$, $1 \leq e \leq 10^{15}$),
- ✚ Dòng thứ 2 chứa các số nguyên c_1, c_2, \dots, c_n – một hoán vị các số từ 1 đến n .

Kết quả: Đưa ra file văn bản RAINBOW.OUT một số nguyên – số thao tác ít nhất xác định được. Nếu không có cách nào sắp xếp được trong phạm vi năng lượng e hiện có thì đưa ra một nguyên số -1.

Ví dụ:

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



Giải thuật: Cây Fenwick, Tìm kiếm nhị phân .

Tạm gác yêu cầu ràng buộc về năng lượng và thời gian, có thể tính được 2 phương án cực trị:

- ✚ Phương án tiết kiệm thời gian nhất,
- ✚ Phương án tiết kiệm năng lượng nhất.

Lời giải cần tìm sẽ nằm giữa các lời giải của hai phương án trên.

Phương án tiết kiệm thời gian nhất:

Nếu chậu với loại hoa i có số nghịch thế $q > 0$, bằng *một thao tác* với chi phí năng lượng q^2 chậu hoa này sẽ được đưa về đúng vị trí của nó trong dãy kết quả.

Giả thiết tồn tại tất cả là k chậu có số nghịch thế lớn hơn 0, tương ứng là $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k$ thì thời gian nhỏ nhất để sắp xếp lại là \mathbf{k} và chi phí năng lượng sẽ là $\sum_{i=1}^k x_i^2$.

Phương án tiết kiệm năng lượng nhất:

Nếu chậu với loại hoa i có số nghịch thế $q > 0$, ta thực hiện q lần đổi chỗ với chậu bên cạnh, mỗi lần đổi chỗ: chi phí năng lượng là 1.

Như vậy thời gian và chi phí năng lượng có cùng trị số:

$$\mathbf{y} = \mathbf{x}_1 + \mathbf{x}_2 + \dots + \mathbf{x}_k$$

Tối ưu hóa thời gian với chi phí năng lượng không vượt quá e:

Nhận xét:

Nếu có a và b thỏa mãn điều kiện $b-a>1$ thì $a^2 + b^2 > (a+1)^2 + (b-1)^2$.

Từ đây suy ra: để có chi phí năng lượng nhỏ cần phân tích \mathbf{y} thành \mathbf{k} số hạng sao cho $|\mathbf{x}_i - \mathbf{x}_j| \leq 1$ với mọi $1 \leq i, j \leq k$. Như vậy với mọi i , \mathbf{x}_i chỉ có thể nhận một trong 2 giá trị: t hoặc $t+1$.

Gọi số lần xuất hiện của $t+1$ là r , ta có:

$$\mathbf{y} = (k-r) \times t + r \times (t+1) = k \times t + r,$$

trong đó $0 \leq r < k \rightarrow t = \mathbf{y}/k, r = \mathbf{y}\%k$.

Chi phí năng lượng sẽ là: $d_e(\mathbf{y}, k) = (k-r) \times t^2 + r \times (t+1)^2$.

Ta còn có:

- ✚ $d_e(y+1, k) \geq d_e(y, k),$
- ✚ $d_e(y, k+1) \leq d_e(y, k).$

Dường lối giải:

Tính năng lượng cần thiết trong phương án tiết kiệm năng lượng nhất,

Nếu năng lượng đã cho nhỏ hơn năng lượng cần trong phương án tiết kiệm năng lượng nhất – bài toán vô nghiệm,

Bằng phương pháp tìm kiếm nhị phân, tìm cách thay mỗi phép chuyển chỗ trong phương án đang xét bằng nhóm các phép chuyển với các tính chất đã nêu ở trên để có năng lượng cần thiết cho việc thực hiện không vượt quá năng lượng cho trước và tổng số lượng phép chuyển là nhỏ nhất.

Tổ chức dữ liệu:

- Mảng `vector<int>` fw – Phục vụ tổ chức cây Fenwick tính số nghịch thế,
- Mảng `vector<int>` y – Lưu kích thước mỗi phép chuyển đổi ban đầu,
- Mảng `vector<int>` a – Lưu số lượng phép chuyển đổi cùng kích thước.

Xử lý:

Các hàm phục vụ cây Fenwick:

```
// Nạp giá trị vào cây
void ins_fw(int x)
{
    while(x <= n) ++fw[x], x+=x&(-x);
}

//Tính số thuận thế
int sum_fw(int x)
{
    int r=0;
    while(x>0) {r+=fw[x]; x&=(x-1);}
    return r;
}
```

Tính các tham số phục vụ điều khiển giải thuật:

```
fw.resize(n+2,0); a.resize(n+2,0);
for(int i=1; i<=n; ++i)
{
    fi>>k; ins_fw(k);
    m=i-sum_fw(k);
    ++a[m];
}
a[0]=0;
for(int i=n; i>0; --i) if(a[i]) {p=i; break;}
for(int i=1; i<=n; ++i) if(a[i])
    {mine+= (int64_t) (i)*a[i]; y.push_back(i);}
for(int i=p; i>0; --i) if(a[i]) cur_t+=(int64_t) (i)*i*a[i];
```

Phần còn lại: Tìm kiếm nhị phân theo thời gian và dẫn xuất kết quả.

Lưu ý: Tồn tại *sơ đồ khác* tích luỹ thông tin vào cây Fenwick và dẫn xuất số nghịch thế.

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("rainbow.inp");
ofstream fo("rainbow.out");
int n,k,m,p;
int64_t E,cur_t=0,t,t1,ans=0,mine=0;
vector<int> fw,a,y;

int64_t d_e(int64_t x, int tm)
{
    int64_t b = x / tm;
    int64_t r = x % tm;
    return (tm - r)*b*b + r*(b + 1)*(b + 1);
}

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

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

int main()
{
    fi>>n>>E;
    fw.resize(n+2,0); a.resize(n+2,0);
    for(int i=1; i<=n; ++i)
    {
        fi>>k;ins_fw(k);
        m=i-sum_fw(k);
        ++a[m];
    }
    a[0]=0;
    for(int i=n; i>0; --i) if(a[i]) {p=i; break;}
    for(int i=1; i<=n; ++i) if(a[i])
        {mine+=(int64_t)(i)*a[i]; y.push_back(i);}
    for(int i=p; i>0; --i) if(a[i]) cur_t+=(int64_t)(i)*i*a[i];

    if(E<mine) {fo<<-1; return 0;}

    int u,v;
    int64_t lf,rt,mid,e,cnt,s;

    ans=-1; lf=1; rt=(int64_t)(n)*n;

    while(rt-lf>=0)
    {
        mid=(lf+rt)/2; cnt=0; e=0; s=0; u=1; v=1;
        for(int i=0; i<y.size(); ++i)
        {
            while (u<y[i] && d_e(y[i], u)-d_e(y[i], u + 1)>mid) u++;
            v = max(v, u);
            while (v < y[i] && d_e(y[i], v)-d_e(y[i], v+1)>= mid) v++;
            e += a[y[i]]*d_e(y[i], v);
            s += a[y[i]]*v;
        }
    }
}
```

```

    if (v>1 && d_e(y[i],v-1)-d_e(y[i],v)==mid)
        cnt+= a[y[i]]*(v-u);
}
if (e <= E)
{
    ans = s - min((E - e)/mid, cnt);
    lf = mid + 1;
} else rt = mid - 1;
}
fo<<ans;
Times;
}

```



VY09. BÁN VÉ TÀU NHANH

Tên chương trình: TICKETS.CPP

Tuyến tàu nhanh Bắc – Nam có $n+1$ ga, đánh số từ 0 đến n . Toàn bộ đoàn tàu có s chỗ ngồi. Với hệ thống bán vé điện tử hành khách có thể đặt chỗ qua mạng hoặc tại các thiết bị bán vé tự động. Mỗi yêu cầu đặt chỗ có dạng $x \ y \ k$ – mua k vé lên từ ga x và xuống ở ga y . Ở một ga nào đó, nếu có hành khách xuống thì chỗ đó có thể được dùng để phục vụ cho các yêu cầu đặt chỗ khác.

Các yêu cầu đặt chỗ được chuyển về server của hệ thống để xử lý. Nếu còn chỗ – yêu cầu sẽ được đáp ứng, trong trường hợp ngược lại – đưa ra thông báo từ chối.

Cho đến trước giờ khởi hành hệ thống nhận được q yêu cầu đặt chỗ, không có 2 yêu cầu nào xuất hiện cùng thời điểm.

Với mỗi yêu cầu đặt chỗ trong danh sách yêu cầu theo trình tự tăng dần của thời gian xuất hiện hãy xác định có thể đáp ứng được yêu cầu đó hay không.

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

- ✚ Dòng đầu tiên chứa 3 số nguyên n , s và q ($1 \leq n \leq 2 \times 10^5$, $1 \leq s, q \leq 10^5$),
- ✚ Mỗi dòng sau q dòng sau chứa 3 số nguyên x , y và k ($0 \leq x < y \leq n$, $1 \leq k \leq s$).

Kết quả: Đưa ra file văn bản TICKETS.OUT q số nguyên, mỗi số trên một dòng – kết quả xử lý các yêu cầu, 1 – chấp nhận, 0 – từ chối.

Ví dụ:

TICKETS.INP	TICKETS.OUT
5 3 5	1
0 4 2	1
2 3 1	0
3 5 2	1
3 5 1	1
4 5 1	



Giải thuật: Ứng dụng Treap với khóa ẩn.

Chương trình

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

struct Node
{
    int y;
    int sz;
    int val,vmin,inc;
    Node *l, *r;
};

Node *new_node (int val)
{
    Node *result = new Node;
    result->y = rand();
    result->sz = 1;
    result->val = val;
    result->vmin = val;
    result->inc = 0;
    result->l = result->r = nullptr;
    return result;
}

Node *tp=nullptr;

int get_Vmin (Node *t)
{
    return t != NULL ? t->vmin : (int)1e9;
}

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);
    t->vmin = min(t->val,min(get_Vmin(t->l),get_Vmin(t->r)));
}

Node *merge (Node* t1, Node *t2)
{
    get_Inc(t1); get_Inc(t2);
    if (t1 == nullptr) return t2;
```

```

    if (t2 == nullptr) return t1;
    get_Inc(t1); get_Inc(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;
    }
}

void split(Node *t, int x, Node *&t1, Node *&t2)
{
    if (t == nullptr)
    {
        t1 = t2 = nullptr;
        return;
    }
    get_Inc(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 *add(Node *t, int pos, int val)
{
    Node *t1, *t2;
    split(t, pos, t1, t2);
    Node* new_tree = new_node(val);
    return merge(merge(t1, new_tree), t2);
}

Node* remove_1(Node *t, int pos)
{
    Node *t1, *t2, *t3, *t4;
    split(t, pos, t1, t2);
    split(t2, 1, t3, t4);
    t = merge(t1, t4);
    delete t3;
    return t;
}

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;
    }

Node *from_vector(const vector<int>& v)
{
    Node *result = nullptr;
    for (int i = 0; i < v.size(); ++i)
        result = merge(result, new_node(v[i]));
    return result;
}

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) return t->val;
    else
        return get_value(t->r, pos - my_idx - 1);
}

int get_value_r(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) return t->val;
    else
        return get_value(t->r, pos - my_idx - 1);
}

Node *to_front(Node *t, int l, int r)
{
    Node *t1, *t2, *t3, *t4;
    split(t, r + 1, t1, t2);
    split(t1, l, t3, t4);
    return merge(merge(t4, t3), t2);
}

void calc_Param(Node *t, int l, int r, int &r1)
{
    Node *t1, *t2, *t3;
    split(t, l, t1, t2);
    split(t2, r-l+1, t2, t3);
    r1=t2->vmin;
    merge(t1, merge(t2, t3));
}

void print_tree(Node *t)
{
    get_Inc(t);
    if (t == nullptr) return;
    print_tree(t->l);
    fo<< t->val << " ";
    print_tree(t->r);
}

void increase(Node *t, int l, int r, int val)
{
    Node *t1, *t2, *t3;
    split(t, l, t1, t2);

```

```

        split (t2, r-1+1, t2, t3);
        t2->inc=val;
        t = merge (t1, t2);
        t= merge (t, t3);
    }

int main()
{
    int n,s,q;
    fi>>n>>s>>q;
    int x,y,k,r;
    Node *tree=nullptr;
    for (int i = 0; i <=n; ++i)
        tree = merge(tree, new_node(s));
    for(int i=0; i<q; ++i)
    {
        fi>>x>>y>>k;
        calc_Param(tree,x,y,r);
        if(r<k) { fo<<"0\n"; continue; }
        fo<<"1\n";
        increase(tree,x,y-1,-k);
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```



Cho số nguyên dương m và bảng \mathbf{A} kích thước $n \times n$ với các phần tử nguyên dương.

Bảng con kích thước $w \times h$ là các phần tử của \mathbf{A} nằm trên giao giữa các cột $x, x+1, \dots, x+w-1$ với các dòng $y, y+1, \dots, y+h-1$. Số lượng phần tử của bảng con này là $w \times h$.

Hãy xác định số phần tử nhỏ nhất của bảng con thỏa mãn điều kiện nếu thực hiện phép tính bít OR với tất cả các phần tử của bảng con ta được \mathbf{x} .

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

- ✚ Dòng đầu tiên chứa 2 số nguyên m và n ($1 \leq m < 2^{31}$, $2 \leq n \leq 500$),
- ✚ Dòng thứ i trong n dòng sau chứa n số nguyên dương xác định một dòng của bảng \mathbf{A} , các số có giá trị nhỏ hơn 2^{31} .

Dữ liệu đảm bảo bài toán có nghiệm và nghiệm không nhỏ hơn 2.

Kết quả: Đưa ra file văn bản OR_SUM.OUT một số nguyên – số phần tử nhỏ nhất của bảng con tìm được.

Ví dụ:

OR_SUM.INP	OR_SUM.OUT
<pre>11 4 5 9 1 8 7 7 3 1 2 3 1 9 5 5 8 7</pre>	<pre>3</pre>



Giải thuật: Xử lý bít, Tổng tiền tố 2 chiều.

Xét bảng con \mathbf{B} xác định bởi phần tử góc trên trái (x, y), độ rộng w và độ cao h .
Như vậy phần tử góc dưới phải sẽ là ($x+w-1, y+h-1$).

Gọi diện tích của \mathbf{B} là số lượng phần tử trong bảng con, diện tích của \mathbf{B} là $w \times h$.

Gọi $Q(x, y, w, h)$ – tổng or các phần tử
của \mathbf{B} .

Độ phức tạp của giải thuật phụ thuộc vào giải
thuật tính $Q(x, y, w, h)$.

Khác với việc tích lũy kết quả theo phép cộng
bình thường, khi tích lũy kết quả theo phép
or, nếu một bít nào đó của kết quả được xác
lập bằng 1 thì nó sẽ *luôn luôn bằng 1* cho đến khi kết thúc quá trình tích lũy.

Như vậy không được phép sử dụng các phần tử ngoài bảng \mathbf{B} trong quá trình tích lũy.

Kỹ thuật tạo *ma trận tích lũy thura* cho phép giải quyết vấn đề nêu trên: Xây dựng
bảng 4 chiều \mathbf{T} với $T_{p,q,x,y}$ – tổng or cho bảng con kích thước $2^p \times 2^q$ với góc
trên trái ở vị trí (x, y) .

$$T_{0,0,x,y} = A_{x,y},$$

$$T_{p,q,x,y} = \begin{cases} T_{p,q-1,x,y} \text{ or } T_{p,q-1,x,y+2^{(q-1)}} \text{ với } q > 0, \\ T_{p-1,q,x,y} \text{ or } T_{p-1,q,x+2^{(p-1)},y} \text{ với } p > 0, \end{cases}$$

Thời gian xây dựng bảng \mathbf{T} có bậc $O(n^2 \log^2(n))$.

Tính $Q(x, y, w, h)$ dựa trên \mathbf{T} :

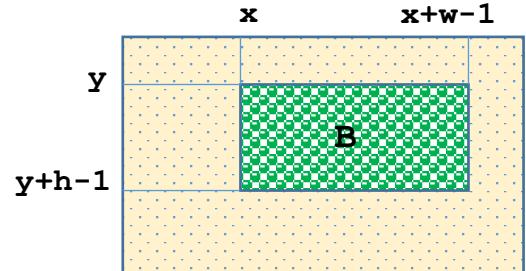
Ký hiệu L_d – số nguyên k lớn nhất thỏa mãn điều kiện $2^k \leq d$,

$$\begin{aligned} Q(x, y, w, h) &= T_{L[w], L[h], x, y} \\ &\text{or } T_{L[w], L[h], x+w-2^L[w], y} \\ &\text{or } T_{L[w], L[h], x+w-2^L[w], y+h-2^L[h]} \\ &\text{or } T_{L[w], L[h], x, y+h-2^L[h]} \end{aligned}$$

Độ phức tạp của việc tính Q : $O(1)$.

- ✓ Xuất phát từ phần tử trên trái của \mathbf{A} :
- ✓ Xét các bảng con \mathbf{B} độ cao $h = 1, 2, \dots, n$,
- ✓ Trượt dần cửa sổ chứa \mathbf{B} sang phải,
- ✓ Xét các \mathbf{B} với độ rộng tăng dần,
- ✓ Bỏ qua các cửa sổ làm xuất hiện bít 1 thừa (không có trong \mathbf{m}),
- ✓ Khi gấp \mathbf{B} cho kết quả \mathbf{m} : Cố gắng thu hẹp \mathbf{B} bằng cách *tăng biên trái*.

Độ phức tạp của giải thuật: $O(n^3)$.



Chương trình

```
#include <bits/stdc++.h>
#define NAME "or_sum."
#define nmax 502
#define logn 9
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int T[logn][logn][nmax][nmax], L[nmax];

inline int query(int i, int j, int aa, int bb)
{
    return T[L[aa]][L[bb]][i][j] |
           T[L[aa]][L[bb]][i+aa-(1<<L[aa])][j] |
           T[L[aa]][L[bb]][i+aa-(1<<L[aa])][j+bb-(1<<L[bb])] |
           T[L[aa]][L[bb]][i][j+bb-(1<<L[bb])];
}

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

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            fi >> T[0][0][i][j];

    for (int p = 0; p < logn; p++)
        for (int q = 0; q < logn; q++)
            for (int i = 0; i + (1 << p) <= n; i++)
                for (int j = 0; j + (1 << q) <= n; j++)
                    if (p)
                        T[p][q][i][j]=T[p-1][q][i][j] | T[p-1][q][i+(1 << (p-1))][j];
                    else if (q)
                        T[p][q][i][j]=T[p][q-1][i][j] | T[p][q-1][i][j+(1 << (q-1))];

    for (int i = 2; i <= n; i++) L[i] = L[i-1] + ((i & (i-1)) == 0);

    int ans = n*n + 1;
    for (int h = 1; h <= n; h++)
        for (int i = 0; i + h <= n; i++)
            for (int u = 0, v = 0, s = 0; u < n; u++)
            {
                s |= query(i, u, h, 1);
                if (s & ~m) { v = u+1; s = 0; continue; }
                while (s == m)
                {
                    ans = min(ans, (u-v+1)*h);
                    v++;
                    s = query(i, v, h, u-v+1);
                }
            }
    fo << ans;
    cerr<<clock()/(double)1000;
    return 0;
}
```



BÀI 11. LŨY THỪA CỦA 2

Tên chương trình: POWER2.CPP

Cho n số nguyên dương a_1, a_2, \dots, a_n .

Hãy xác định:

- Số lượng các số a_i có dạng 2^k ,
- Số lượng các k khác nhau trong số các $a_i = 2^k$,
- Gọi m là số lượng các k khác nhau tìm được. Hãy đưa ra giá trị k ở giữa trong trường hợp m lẻ hoặc cặp giá trị k ở giữa trong trường hợp m chẵn trong dãy sắp xếp các k theo giá trị giảm dần.

Dữ liệu: Vào từ file văn bản POWER2.INP:

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

Kết quả: Đưa ra file văn bản POWER2.OUT:

- ✿ Dòng đầu tiên chứa 2 số nguyên – số lượng các số a_i có dạng 2^k và số lượng các k khác nhau,
- ✿ Dòng thứ 2 chứa số hoặc cặp số nguyên ở giữa của dãy các số k khác nhau.

Ví dụ:

POWER2.INP	POWER2.OUT
11	7 4
3 4 2 6 4 5 8 4 32 15 8	2 3

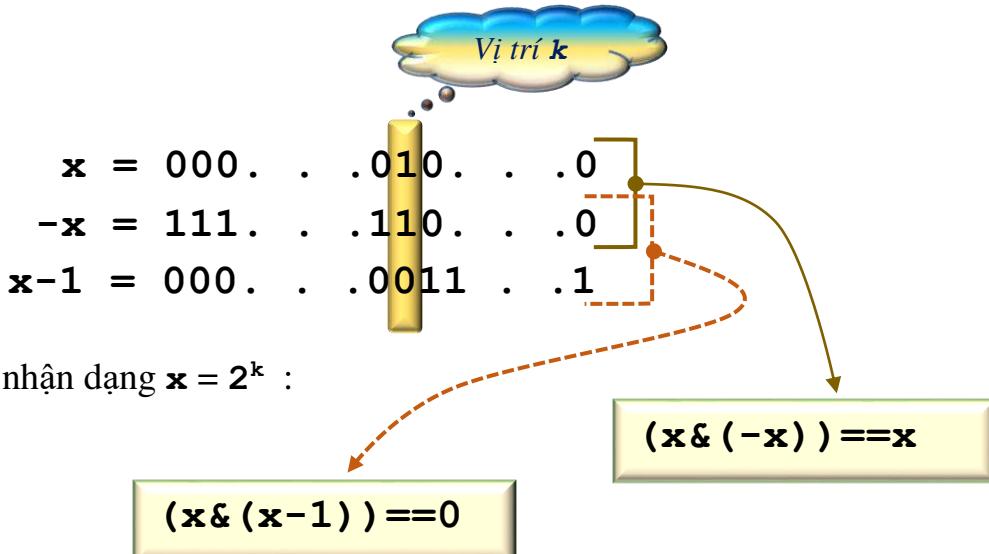


Giải thuật: Xử lý bít.

Số nguyên dương và là lũy thừa của 2 trong dạng biểu diễn nhị phân chỉ có một bít bằng 1.

Xét $x = 2^k$ ($k \geq 0$).

Trong dạng biểu diễn nhị phân:



Tìm vị trí bít 1:

```

int pos(int64_t y)
{
    int p;
    for(int i=0; i<64; ++i)
        if((y>>i)&1){p=i; break;}
    return p;
}

```

Từ điều kiện $a_i \leq 10^{18}$: $0 \leq k \leq 63$.

Lập bảng đánh dấu các k xuất hiện và từ đó dễ dàng liệt kê các k khác nhau theo trình tự tăng hoặc giảm dần.

Từ bảng liệt kê các k khác nhau: dãy xuất giá trị/các giá trị ở giữa.

Độ phức tạp của giải thuật: $O(n)$.

Ứng dụng: Quản lý chỉ số trong cây Fenwick.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "power2."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t x;
int n,m=0,u,v=0,b[64]={0};
vector<int64_t> a;

int pos(int64_t y)
{
    int p;
    for(int i=0; i<64; ++i)
        if((y>>i)&1){p=i; break;}
    return p;
}

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i)
    {
        fi>>x;
        if( (x&(-x))==x)           //if( (x&(x-1))==0)
        {
            u=pos(x); b[u]=1; ++v;
        }
    }
    for(int i=0; i<64; ++i) if(b[i])b[m++]=i;

    fo<<v<<' '<<m<<'\\n';
    u=b[m/2-1]; v=b[m/2];
    if(m&1) fo<<v; else fo<<u<<' '<<v;
}

}
```



Tàu hành khách mới hoạt động theo nguyên lý của hệ thống cáp treo: chuyển động liên tục không dừng. Tại các ga sẽ có băng tải chuyển động nhanh dần đưa hành khách vào toa và băng tải chuyển động chậm dần đưa hành khách rời toa xuống ga. Tàu gồm một toa có n ghế ngồi bố trí theo hàng dọc, đánh số từ 1 đến n . Hành khách được bố trí ngồi ở các ghế liên tiếp nhau bắt đầu từ 1. Hành khách mới vào được bố trí ở ghế trống sát với các hành khách khác, nếu toa đang trống thì được bố trí ngồi ở ghế 1. Hành khách vào ở cửa sau và ra ở cửa trước của toa.

Khi đến mỗi ga, hệ thống sẽ đưa hành khách xuống trước, sau đó mới tiếp nhận hành khách mới. Các hành khách trong toa sẽ được tự động dồn lên trước, đảm bảo các ghế liên tục từ 1 trở đi có người ngồi (nếu còn khách).



Do việc đưa khách ra theo băng chuyền nên nếu ở một ga nào đó hành khách ngồi ở ghế k xuống thì mọi hành khách ở các ghế từ 1 đến $k-1$ cũng bị đưa xuống theo ngay cả khi chưa tới ga định đến. Giá vé đi từ ga x đến ga y phụ thuộc vào nhiều yếu tố như quãng đường, mức trợ giá cho các loại khách, có mang theo hành lý, kèm trẻ em, . . . Toàn tuyến có m ga. Hiện tại có n khách đã mua vé. Hành khách thứ i lên ở ga x_i và xuống ở ga y_i với vé giá c_i , $i = 1 \div n$. Dĩ nhiên, không ai muốn xuống trước khi tới ga mình cần đến. Có nhiều tàu chạy nên nếu không tiện lên ở chuyến này thì chờ chuyến sau.

Bộ phận điều độ muốn lên lịch tiếp nhận khách sao cho tổng chi phí thu được ở mỗi chuyến là lớn nhất đồng thời mọi hành khách lên tàu đều xuống được đúng ga mình muốn.

Hãy xác định tổng thu lớn nhất của chuyến đầu tiên, số hành khách được phục vụ và danh sách trình tự vào của hành khách.

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

- + Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n \leq 10^5$, $1 \leq m \leq 2 \times 10^9$),
- + Dòng thứ i trong n dòng tiếp theo chứa 3 số nguyên x_i , y_i và c_i – thông tin về người thứ i ($1 \leq x_i < y_i \leq m$, $1 \leq c_i \leq 10^4$).

Kết quả: Đưa ra file văn bản NON_STOP.OUT, dòng thứ nhất chứa số nguyên xác định tổng chi phí lớn nhất thu được, dòng thứ 2 chứa số nguyên xác định số hành khách được lên tàu, dòng thứ 3 chứa các số nguyên xác định trình tự vào của hành khách (xem ví dụ).

Ví dụ:

NON_STOP.INP	
4	10
1	3 3
1	10 2
2	5 3
1	2 5

NON_STOP.OUT	
11	
3	
4	1 3



Giải thuật: Quy hoạch động, Cây Fenwick quản lý max.

Sơ đồ nguyên lý quy hoạch động áp dụng để giải:

Gọi \mathbf{dp}_i – thu nhập lớn nhất có được khi xét i hành khách đầu tiên và *hành khách thứ i được lên tàu*,

Công thức lặp:

$$\mathbf{d}_i = \mathbf{c}_i + \max\{\mathbf{dp}_j\},$$

trong đó:

- ✚ $1 \leq j < i$,
- ✚ $\mathbf{y}_j \leq \mathbf{y}_i$,
- ✚ $\mathbf{d}_1 = \mathbf{c}_1$.

Với mỗi i cần xét tất cả các phần tử trước đó vì vậy giải thuật có độ phức tạp $O(n^2)$.

Để giảm độ phức tạp của giải thuật cần cải tiến việc tìm $\max\{\mathbf{dp}_j\}$, điều này đòi hỏi phải tổ chức cây tìm kiếm nhị phân quản lý các giá trị \max cũng như trình tự duyệt dữ liệu để tính \mathbf{dp}_i .

Số lượng ga lớn sẽ gây khó khăn trong việc tổ chức cây nhị phân. Tuy số lượng ga có thể rất lớn nhưng chỉ có n yêu cầu xếp chỗ, vì vậy có thể thực hiện ánh xạ co, đưa về trường hợp có n ga, đánh số từ 1 đến n , bao toàn quan hệ trước – sau trong hành trình của mỗi khách và giữa các khách,

Việc thực hiện ánh xạ co được gọi là *chuẩn hóa* dữ liệu.

Thông tin cần lưu giữ về mỗi hành khách: nhóm 4 giá trị

$$(Ga_lên, Ga_xuống, Giá_vé, Số_thứ_tự_trong_input) \equiv (\mathbf{s}, \mathbf{d}, \mathbf{cost}, \mathbf{id})$$

Thông tin về hành khách được sắp xếp tăng dần và chuẩn hóa.

Với 2 hành khách **a** và **b**, **a** sẽ lên trước **b** nếu $s_a \leq s_b$ và $d_a \leq d_b$.

Các hàm hỗ trợ trong quá trình giải:

- ✿ **query** – tìm $\max dp_j$,
- ✿ **upd** – nạp dp_i vào cây tìm kiếm,
- ✿ **best** – ghi nhận kết quả tối ưu.

Bài toán đòi hỏi dẫn xuất các thông tin về phương án tối ưu, vì vậy cần ghi nhận các thông tin về các phương án tốt nhất trong quá trình tìm kiếm.

Tổ chức dữ liệu:

- ─ Mảng **t4i** $a[N_{max}]$ – Lưu dữ liệu vào dưới dạng nhóm 4 số nguyên,
- ─ Mảng **pii** $aib[N_{max}]$ – Cây Fenwick lưu cặp giá trị $\{dp_i, i\}$,
- ─ Mảng **int** $dp[N_{max}]$ – Lưu các giá trị dp_i ,

- Mảng `int come[Nmax]` – Lưu trình tự tiếp nhận khách (để dẫn xuất kết quả),
- Mảng `map<int, int> mp` – Phục vụ đánh số lại ga đến.

Xử lý:

Đánh số lại ga đến:

```

Danh dấu ga đến
Danh số lại

void normalize()
{
    map<int, int> mp;
    int ss, dd, cc, id;
    for(int i=1; i<=n; ++i) mp[get<1>(a[i])] = i;

    int cnt = 0;
    for(auto &it : mp) it.second = ++cnt;
    for(int i=1; i<=n; ++i)
    {
        tie(ss, dd, cc, id)=a[i];
        dd= mp[dd];
        a[i]=make_tuple(ss, dd, cc, id);
    }
}

```

Xử lý cây Fenwick:

Nap giá trị

```

Tím max
void upd(int pos, pii x)
{
    for(; pos; pos-=(pos&(-pos))) best(aib[pos], x);
}

```

```

pii query(int pos)
{
    pii ans = {0, 0};
    for(; pos<=n; pos+=(pos&(-pos))) best(ans, aib[pos]);
    return ans;
}

```

```

void best(pii &A, pii B)
{
    if(B.first > A.first) A = B;
}

```

Tìm kiếm theo sơ đồ quy hoạch động:

```
void solve()
{
    pii S;
    ans = {0, 0};
    int dd;
    for(int i=n; i> -i)
    {
        dd = get<1>(a[i]);
        S = query(dd);
        come[i] = S.second;
        dp[i] = S.first + get<2>(a[i]);
        dd = get<1>(a[i]);
        upd(dd, {dp[i], i});
        best(ans, {dp[i], i});
    }
}
```

Lưu ý

Người đang xét

Dẫn xuất kết quả:

```
void print_solution()
{
    fo << ans.first << '\n';
    int who = ans.second, cnt = 0;
    while(who)
    {
        ++cnt;
        who = come[who];
    }
    fo << cnt << '\n';
    who = ans.second;
    while(who)
    {
        fo << get<3>(a[who]) << ' ';
        who = come[who];
    }
}
```

Tính số lượng

Dẫn xuất danh sách

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

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
typedef tuple<int,int,int,int> t4i;
typedef pair<int,int> pii;
ifstream fi ("non_stop.inp");
ofstream fo ("non_stop.out");

const int Nmax = 1e5 + 5;
int n,M,dp[Nmax],come[Nmax],s,d,cost;

t4i a[Nmax];
pii ans, aib[Nmax];

void best(pii &A, pii B)
{
    if(B.first > A.first) A = B;
}

pii query(int pos)
{
    pii ans = {0, 0};
    for(; pos<=n; pos+=(pos&(-pos))) best(ans, aib[pos]);
    return ans;
}

void upd(int pos, pii x)
{
    for(; pos; pos-=(pos&(-pos))) best(aib[pos], x);
}

void normalize()
{
    map<int,int> mp;
    int ss,dd,cc,id;
    for(int i=1; i<=n; ++i) mp[get<1>(a[i])] = 1;

    int cnt = 0;
    for(auto &it : mp) it.second = ++cnt;
    for(int i=1; i<=n; ++i)
    {
        tie(ss,dd,cc,id)=a[i];
        dd= mp[dd];
        a[i]=make_tuple(ss,dd,cc,id);
    }
}

void print_solution()
{
    fo << ans.first << '\n';
    int who = ans.second, cnt = 0;

    while(who)
    {
        ++cnt;
        who = come[who];
    }
    fo << cnt << '\n';

    who = ans.second;
```

```

while (who)
{
    fo << get<3>(a[who]) << ' ';
    who = come[who];
}
}

void solve()
{
    pii S;
    ans = {0,0};
    int dd;
    for(int i=n; i; --i)
    {
        dd=get<1>(a[i]);
        S = query(dd);
        come[i] = S.second;
        dp[i] = S.first + get<2>(a[i]);
        dd=get<1>(a[i]);
        upd(dd, {dp[i], i});
        best(ans, {dp[i], i});
    }
}

int main()
{
    fi >> n >> M;
    for(int i=1; i<=n; ++i)
    {
        fi>>s>>d>>cost;
        a[i]=make_tuple(s,d,cost,i);
    }
    sort(a+1, a+n+1);
    normalize();
    solve();
    print_solution();
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}

```



BA12. TRUNG BÌNH

Tên chương trình: AVERAGE.CPP

Bài thi được chấm theo thang điểm 100 và điểm của mỗi bài là nguyên. Kết quả thi của n bài là a_1, a_2, \dots, a_n . Trong báo cáo về kỳ thi có mục nêu điểm trung bình của các thí sinh.

Thay vì tính trung bình theo công thức $\frac{1}{n} \sum_{i=1}^n a_i$, người viết báo cáo đã ghi điểm các bài thành một hàng dài, chọn 2 điểm x và y trong hàng, xóa chúng và ghi vào cuối hàng điểm trung bình của 2 giá trị bị xóa (tức là $(x+y)/2$). Quá trình trên được thực hiện $n-1$ lần thì trong dãy chỉ còn 1 số và người ta viết số đó vào báo cáo ở mục điểm trung bình.

Phụ thuộc vào trình tự chọn cặp số x và y kết quả cuối cùng nhận được có thể nhỏ hơn, bằng hoặc lớn hơn giá trị trung bình của cả dãy số.

Hãy xác định giá trị lớn nhất có thể nhận được và đưa ra với 6 chữ số sau dấu chấm thập phân.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($0 \leq a_i \leq 100$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản AVERAGE.OUT một số thực với độ chính xác 10^{-6} – giá giá trị lớn nhất có thể nhận được.

Ví dụ:

AVERAGE.INP	AVERAGE.OUT
3 1 5 3	3.500000



BA12 Cr720180303 A AXIII

Giải thuật: Cơ sở lập trình. Phương án dùng hàng đợi ưu tiên .

Cách tổ chức vun đống,

Giới thiệu khả năng và cách sử dụng hàng đợi ưu tiên.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "average."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
priority_queue<double, vector<double>, greater<double>>q;
int n;
double x,y,z;

int main()
{
    fi>>n;
    for(int i=0;i<n;++i){fi>>x; q.push(x);}

    for(int i=0;i<n-1;++i)
    {
        x=q.top(); q.pop();
        y=q.top(); q.pop();
        q.push((x+y)/2);
    }
    fo<<fixed<<setprecision(6)<<q.top();
    Times;
}
```



DNA là phân tử mang thông tin di truyền mã hóa cho hoạt động sinh trưởng, phát triển, chuyên hóa chức năng và sinh sản của các sinh vật và thực vật.

Bốn base trong DNA là adenine (viết tắt A), cytosine (C), guanine (G) và thymine (T). Bốn base này gắn với nhóm đường/phosphat để tạo thành nucleotide hoàn chỉnh, như adenosine monophosphate. Adenine ghép cặp với thymine và guanine ghép cặp với cytosine, ký hiệu bằng các cặp base A-T và G-C.

Các nhà khoa học khảo sát một loại cây có DNA xác định bởi xâu s độ dài n , trong đó $s_i = 'C'$ tương ứng với cặp base **G-C** và $s_i = 'T'$ tương ứng với cặp base **A-T**, $i = 1 \div n$.

Để nghiên cứu tính trội của cặp **G-C** người ta loại bỏ trong đoạn từ **1f** đến **rt** một số cặp **A-T** sao cho khi quan sát từ trái sang phải và từ phải sang trái trong đoạn đang xét số lượng cặp **G-C** bao giờ cũng không ít hơn số cặp **A-T**.

Có q dự án nghiên cứu, dự án thứ j nghiên cứu đoạn từ **1f_j** đến **rt_j**, $j = 1 \div q$. Một trong số các dự án đó sẽ được chọn.

Hãy xác định số lượng ít nhất cặp base **A-T** cần loại bỏ trong mỗi dự án.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 5 \times 10^5$),
- ✚ Dòng thứ 2 chứa xâu s độ dài n chỉ gồm các ký tự trong tập {‘C’, ‘T’},
- ✚ Dòng thứ 3 chứa số nguyên q ($1 \leq q \leq 5 \times 10^5$),
- ✚ Dòng thứ j trong q dòng sau chứa 2 số nguyên **1f_j** và **rt_j** ($1 \leq 1f_j \leq rt_j \leq n$).

Kết quả: Đưa ra file văn bản GENETECH.OUT số lượng ít nhất cặp base **A-T** cần loại bỏ trong mỗi dự án, mỗi số trên một dòng, dưới dạng số nguyên.

Ví dụ:

GENETECH.INP
14
CTCTTTCCCTTCTTT
5
3 12
1 14
3 8
2 9
3 11

GENETECH.OUT
4
6
2
4
3



Giải thuật: Cây quản lý đoạn, Tổng tiền tố.

Để tiện xử lý: thay xâu s bằng mảng $\mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n)$:

$$\mathbf{v}_i = \begin{cases} 1 & \text{nếu } s_{i-1} = 'C', \\ -1 & \text{nếu } s_{i-1} = 'T', \end{cases}$$

Mỗi truy vấn (lf , rt) đòi hỏi duyệt xâu con 2 lần:

Lần I:

- ✚ Duyệt các \mathbf{v}_i , với $i = lf \div rt$,
- ✚ Gọi b – số lượng $\mathbf{v}_i = 1$ gấp trong quá trình duyệt, ban đầu $b = 0$,
- ✚ Gấp $\mathbf{v}_i = 1 \rightarrow$ tăng b lên 1,
- ✚ Gấp $\mathbf{v}_i = 0$: nếu $b > 0 \rightarrow$ giảm b một đơn vị, trong trường hợp ngược lại – gán $\mathbf{v}_i = 0$.

Ví dụ: $n = 14$, $s = \text{"CTCTTTCCCTTCTTT"}$, $lf = 3$, $rt = 11$

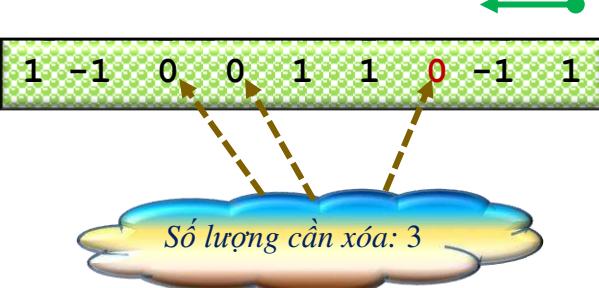
$s:$	C T	C T T T C C T T C T T T
------	--------	--

$v:$	1 -1	1 -1 -1 -1 1 1 -1 -1 1 -1 -1 -1
------	---------	--

Kết quả duyệt I: 1 -1 0 0 1 1 -1 -1 1

Lần II: Xử lý tương tự với $i = rt \div lf$.

Kết quả duyệt II:



Nhận xét:

- ✿ Xử lý *mỗi truy vấn*: độ phức tạp là $O(n)$,
- ✿ Các truy vấn có cùng lf : Độ dài phần đầu của kết quả duyệt lần I trùng nhau, như vậy *chỉ cần xây dựng kết quả duyệt lần I một lần* ứng với rt lớn nhất trong các truy vấn có cùng lf , ví dụ, nếu đã xử lý truy vấn (3, 11) ta có thể dễ dàng tính kết quả của truy vấn (3, 10):

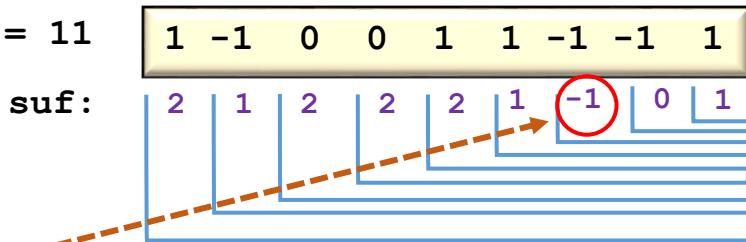
1 -1 0 0 1 1 -1 -1	1
---	---

1 -1 0 0 1 1 0 0	1
---------------------------------------	---

Cần xóa 4

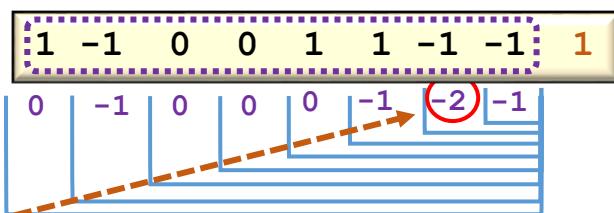
Gọi suf_i là tổng hậu tố của mảng nhận được sau lần duyệt I, tính từ vị trí rt đến vị trí i , $i = \text{rt}, \text{rt}-1, \dots, \text{lf}$.

$$\text{lf} = 3, \text{rt} = 11$$



- $\min(0, \min\{\text{suf}_i\})$ – số lượng cần xóa thêm khi duyệt từ phải sang trái

$$\text{lf} = 3, \text{rt} = 10$$



- $\min(0, \min\{\text{suf}_i\})$ – số lượng cần xóa thêm khi duyệt từ phải sang trái

Từ các nhận xét trên có thể rút ra các kết luận:

- ✓ Cần xử lý truy vấn theo ché độ offline: đọc hết các truy vấn và phân nhóm theo điểm đầu **lf**,
- ✓ Để tìm kiếm nhanh $\min\{\text{suf}_i\}$ cần tổ chức **cây quản lý đoạn**, quản lý các giá trị tổng hậu tố và min của nó trên đoạn,
- ✓ Việc xử lý cây quản lý đoạn có thể thực hiện theo sơ đồ **đệ quy** hoặc **sơ đồ lắp**,
- ✓ Để xác định số lượng phần tử 0 (số lượng cần xóa trong lần duyệt I) cần sử dụng các **công cụ tìm kiếm nhị phân**,
- ✓ **Độ phức tạp của giải thuật** sẽ là $O((n+q)\times \log n)$.

Tổ chức dữ liệu:

- ☒ Xâu **string s** – lưu xâu dữ liệu vào,
- ☒ Mảng **vector<int> v** – Ánh xạ xâu sang mảng số,
- ☒ Mảng **vector<vector<pii>> qr** – Lưu các truy vấn theo nhóm, mỗi phần tử là cặp giá trị (**rt, i**), trong đó **i** – số thứ tự của truy vấn,
- ☒ Mảng **vector<pii> tr** – Quản lý đoạn lưu tổng hậu tố và min tương ứng,
- ☒ Mảng **vector<int> st** – Mô phỏng stack quản lý các vị trí đánh dấu 0 ở lần duyệt I,
- ☒ Mảng **vector<int> ans** – Lưu kết quả ra.

Xử lý: *Giải thuật không sử dụng đệ quy*

Chuẩn bị xử lý truy vấn và xin cấp phát bộ nhớ:

Tính vị trí nút lá

```
fi>>n>>s;
for(int i=20; i>=0; --i)
    if((n>>i)&1){k=i+1; break;}
p0=1<<k;
v.resize(n+1);
for(int i=1; i<=n; ++i) v[i]=(s[i-1]=='.')? 1:-1;
tr.resize(4*n+5, {0, n+1});
fi>>q;
vector<vector<pii>>qr(n+1); ans.resize(q+1);
```

Chuẩn bị bộ nhớ và giá trị đầu cho cây

Cấu trúc phần tử của cây:



Cập nhật cây để nút lá **x** chứa giá trị **val**:

```
void upd_tr(int x, int val)
{
    int p=p0+x-1;
    tr[p].ff=val; tr[p].ss=tr[p].ff;
    while(p>1)
    {
        p/=2;
        tr[p].ff=tr[2*p].ff+tr[2*p+1].ff;
        tr[p].ss=min(tr[2*p].ss+tr[2*p+1].ff, tr[2*p+1].ss);
    }
}
```

Địa chỉ nút lá

Chuẩn bị giá trị nút lá

Cập nhật giá trị suffix

Cập nhật giá trị min suffix

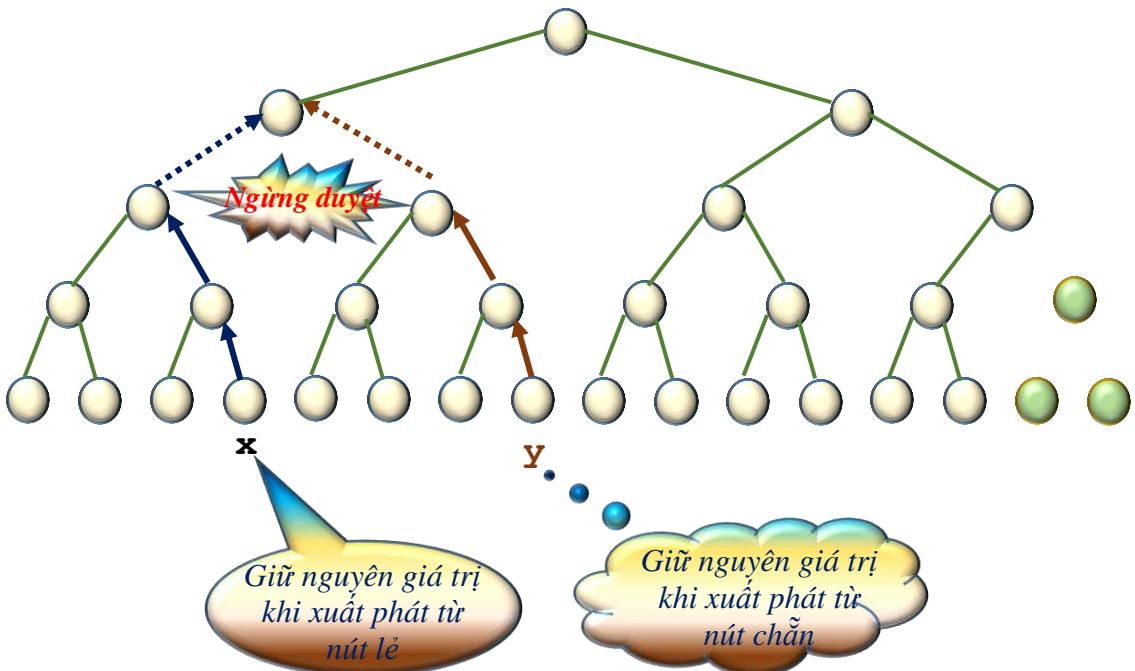
Phân nhóm truy vấn: Tương tự lưu danh sách đỉnh kề của đồ thị.

```
vector<vector<pii>>qr(n+1); ans.resize(q+1);

for(int i=1; i<=q; ++i)
{
    fi>>x>>y;
    qr[x].push_back({y, i});
}
```

Tìm min giá trị suffix trong đoạn $[x, y]$:

Miền tìm min được giới hạn bởi 2 đường đi từ các nút lá x và y về nút cha chung gần nhất



```

int get_min(int x, int y)
{
    pii tx,ty,tm;
    x+=p0-1; y+=p0-1;
    tx=tr[x]; ty=tr[y];
    if(x==y) return min(0,tx.ss);
    while(y-x>1)
    {
        tm=tx;
        if((x&1)==0) tx.ff+=tr[x+1].ff,
                      tx.ss=min(tm.ss+tr[x+1].ff,tr[x+1].ss);
        tm=ty;
        if(y&1) ty.ff+=tr[y-1].ff, ty.ss=min(tr[y-1].ss+tm.ff,tm.ss);
        x/=2; y/=2;
    }
    return min(0,min(tx.ss+ty.ff,ty.ss));
}

```

Nhánh trái

Cập nhật giá trị khi xuất phát từ nút chẵn

Nhánh phải

Cập nhật giá trị khi xuất phát từ nút lẻ

Xử lý truy vấn:

```
for(int i=n; i; --i)
{
    if(v[i]==-1)
    {
        st.push_back(i); upd_tr(i,0);
    }
    else
    {
        upd_tr(i,1);
        if(!st.empty()) { upd_tr(st.back(), -1); st.pop_back(); }
    }
}

for(auto j:qr[i])
{
    int l = i, r = j.ff;
    ans[j.ss]=upper_bound(st.rbegin(), st rend(), r) - st.rbegin() - get_min(l,r);
}
}
```

Lần duyệt I

Số lượng xóa ở lần I

Lần duyệt II

Số lượng xóa ở lần II

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

Chương trình 1

```
#include <bits/stdc++.h>
#define NAME "genetech."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
#define ss second
#define ff first
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
int n,k,p0,q,x,y,lf,rt,r1,r2,sum=0;
string s;
vector<int> v,st,ans;
vector<pii> tr;

void upd_tr(int x,int val)
{
    int p=p0+x-1;
    tr[p].ff=val; tr[p].ss=tr[p].ff;
    while(p>1)
    {
        p/=2;
        tr[p].ff=tr[2*p].ff+tr[2*p+1].ff;
        tr[p].ss=min(tr[2*p].ss+tr[2*p+1].ff,tr[2*p+1].ss);
    }
}

int get_min(int x, int y)
{
    pii tx,ty,tm;
    x+=p0-1; y+=p0-1;
    tx=tr[x]; ty=tr[y];
    if(x==y) return min(0,tx.ss);
    while(y-x>1)
    {
        tm=tx;
        if((x&1)==0)
            tx.ff+=tr[x+1].ff, tx.ss=min(tm.ss+tr[x+1].ff,tr[x+1].ss);
        tm=ty;
        if(y&1) ty.ff+=tr[y-1].ff, ty.ss=min(tr[y-1].ss+tm.ff,tm.ss);
        x/=2; y/=2;
    }
    return min(0,min(tx.ss+ty.ff,ty.ss));
}

int main()
{
    fi>>n>>s;
    for(int i=20; i>=0; --i)
        if((n>i)&1) {k=i+1; break;}
    p0=1<<k;
    v.resize(n+1);
    for(int i=1; i<=n; ++i) v[i]=(s[i-1]=='C')? 1:-1;
    tr.resize(4*n+5,{0,n+1});
    fi>>q;
    vector<vector<pii>>qr(n+1); ans.resize(q+1);

    for(int i=1; i<=q; ++i)
    {
        fi>>x>>y;
        if(x>=y) continue;
        if(x>=p0) ans[i]=min(get_min(x,y),ans[i]);
        else
        {
            int l=0,r=q;
            while(l<r)
            {
```

```

        qr[x].push_back({y, i});
    }
    for(int i=n; i; --i)
    {
        if(v[i]==-1)
        {
            st.push_back(i); upd_tr(i, 0);
        }
        else
        {
            upd_tr(i, 1);
            if(!st.empty()) { upd_tr(st.back(), -1); st.pop_back(); }
        }
    }

    for(auto j:qr[i])
    {
        int l = i, r = j.ff;
        ans[j.ss] = upper_bound(st.rbegin(), st.rend(), r)
                    - st.rbegin() - get_min(l, r);
    }
}

for(int i=1; i<=q; ++i) fo<<ans[i]<<'\\n';
Times;
}

```

Chương trình 2

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

const int NMAX = 500000 + 5;
int N, v[NMAX];

struct Node
{
    int l, r;
    int sum, minSuf;
    Node(int _l = 0, int _r = 0, int _sum = 0, int _minSuf = 0):
        l(_l), r(_r), sum(_sum), minSuf(_minSuf) {}

    static Node singleNode(const int l, const int r, const int val) {
        return Node(l, r, val, min(val, 0));
    }
    friend Node operator+(const Node &arg0, const Node &arg1) {
        return Node(arg0.l, arg1.r, arg0.sum + arg1.sum, min(arg0.minSuf + arg1.sum, arg1.minSuf));
    }
} tree[4 * NMAX];

void build(int node, int l, int r)
{
    if (l == r)
    {
        tree[node] = Node :: singleNode(l, l, v[l]);
        return ;
    }
    const int mid = (l + r) / 2;
    build(2 * node, l, mid);
    build(2 * node + 1, mid + 1, r);
    tree[node] = tree[2 * node] + tree[2 * node + 1];
}

Node query(int node, int l, int r)
{
    if (tree[node].l == l && tree[node].r == r)
        return tree[node];
    const int mid = (tree[node].l + tree[node].r) / 2;
    if (r <= mid)
        return query(2 * node, l, r);
    else if (l > mid)
        return query(2 * node + 1, l, r);
    else
        return query(2 * node, l, mid) +
               query(2 * node + 1, mid + 1, r);
}

void update(int node, int where, int val)
{
    if (tree[node].l == tree[node].r)
    {
        v[where] += val;
        tree[node] = Node :: singleNode(where, where, v[where]);
        return ;
    }
    const int mid = (tree[node].l + tree[node].r) / 2;
    if (where <= mid)
        update(2 * node, where, val);
```

```

    else
        update(2 * node + 1, where, val);
    tree[node] = tree[2 * node] + tree[2 * node + 1];
}

struct Query
{
    int r, pos;
    Query(int _r = 0, int _pos = 0):
        r(_r), pos(_pos) {}
};

vector <Query> queries[NMAX];
int answers[NMAX];

int main() {
    ifstream cin("genetech.inp");
    ofstream cout("genetech.out");
    cin >> N;
    string S;
    cin >> S;

    for (int i = 1; i <= N; ++ i) v[i] = S[i - 1] == 'C' ? 1 : -1;
    build(1, 1, N);

    int Q = 0;
    cin >> Q;
    assert(1 <= Q && Q <= 500000);

    for (int i = 1; i <= Q; ++ i)
    {
        int l, r;
        cin >> l >> r;
        queries[l].push_back(Query(r, i));
    }

    vector <int> st;
    for (int i = N; i; -- i)
    {
        if (v[i] == -1)
            st.push_back(i), update(1, i, 1);
        else
            if (!st.empty())
                update(1, st.back(), -1), st.pop_back();

        for (auto qr: queries[i])
        {
            const int l = i, r = qr.r;
            answers[qr.pos] = upper_bound(st.rbegin(), st.rend(), r)
                - st.rbegin() - query(1, l, r).minSuf;
        }
    }

    for (int i = 1; i <= Q; ++ i)
        cout << answers[i] << '\n';
    cout << "Time: " << clock() / (double)1000;
}

```

