

ATHENA XI – MỤC LỤC

VU49. BĂNG CHUYỀN SẢN XUẤT	Tên chương trình: CONVEYOR.CPP	3
VV17. XIN TRỢ GIÚP	Tên chương trình: RESCUE.CPP	9
VV18. PHÒNG ĐỆM	Tên chương trình: BUFFER.CPP	17
VV19. TRÌNH DIỄN	Tên chương trình: SHOWS.CPP	20
VV20. KHOẢNG CÁCH HAMMING	Tên chương trình: HAMMING.CPP	22
VV21. CHIM CU	Tên chương trình: CUCKOO.CPP	25
VV22. CHÂM BÀI	Tên chương trình: TESTING.CPP	30
VV23. PHƯƠNG ÁN THỬ NGHIỆM	Tên chương trình: PLAN.CPP	34
VV24. THỰC ĐƠN	Tên chương trình: MENU.CPP	44
VV25. ĐẦU TƯ	Tên chương trình: INVEST.CPP	46
VV26. BIẾN ĐỔI XÂU	Tên chương trình: STRANSF.CPP	48
VV27. CHỈ SỐ ĐÁNH GIÁ	Tên chương trình: S_INDEX.CPP	51
VV28. TÙ VÀ	Tên chương trình: TRUMPET.CPP	54
VV29. QUẢN LÝ MỨC	Tên chương trình: LEVELS.CPP	56
VV30. BẢNG SỐ	Tên chương trình: SHEETS.CPP	59
VV31. KHOẢNG CÁCH HOÀN THIỆN	Tên chương trình: P_DIST.CPP	63
VV32. ÁO KHOÁC	Tên chương trình: JACKETS.CPP	65
VV33. BIỂU TƯỢNG VUI	Tên chương trình: SMILEY.CPP	72
VV34. KHÓA CHƯƠNG TRÌNH	Tên chương trình: LOCK.CPP	80
VV35. THỨ TỰ	Tên chương trình: ORDER.CPP	83
VV36. HAI THẾ GIỚI	Tên chương trình: TWOWORLD.CPP	87
VV37. PHÓ ĐI BỘ	Tên chương trình: FOOTPATH.CPP	91
VV38. A+B	Tên chương trình: ABPLUS.CPP	97
VV39. BIỂU THỨC	Tên chương trình: EXPR.CPP	103
VV40. SỐ BƯỚC	Tên chương trình: STEPS.CPP	109
VV41. MÁY IN 3D	Tên chương trình: PRN3D.CPP	113
VV42. PHỐI HỢP	Tên chương trình: LIAISE.CPP	115
VV43. ĐIỂM THI ACM	Tên chương trình: ACM.CPP	120
GIẢI THUẬT MANAKER TÌM PALINDROME		122
Bài toán		122

<i>Giải thuật</i>	122
<i>Đánh giá độ phức tạp</i>	124
VV44. SỐ LƯỢNG PALINDROME <i>Tên chương trình: ALL_PAL.CPP</i>	125
VV45. KHẢ NĂNG NHỚ <i>Tên chương trình: MEMORABLE.CPP</i>	128
VV46. ĐÀI NGUYÊN <i>Tên chương trình: TUNDRA.CPP</i>	134
VV47. TIN NHÁN <i>Tên chương trình: SMS.CPP</i>	138
VV48. HÀNG CỘT <i>Tên chương trình: COLUMNS.CPP</i>	142
VV49. BIỂU THỨC NGOẶC <i>Tên chương trình: BR_EXPR.CPP</i>	147
VV50. TRÒ CHƠI NHI PHÂN <i>Tên chương trình: BINGAME.CPP</i>	151
VW01. THI TIN HỌC <i>Tên chương trình: OLYMPIAD.CPP</i>	155
VW02. TRÒ CHƠI 9999 <i>Tên chương trình: G9999.CPP</i>	161

VU49. BĂNG CHUYỀN SẢN XUẤT

Tên chương trình: CONVEYOR.CPP

Nhà máy có n băng chuyền , linh kiện trên mỗi băng chuyền phải qua m công đoạn xử lý. Ban đầu, linh kiện ở băng chuyền nào sẽ qua tất cả các khâu xử lý ở băng chuyền đó.

Các tiến bộ khoa học kỹ thuật mới dần dần được triển khai. Sau một công đoạn xử lý nào đó linh kiện trên băng chuyền **A** và linh kiện trên băng chuyền **B** đổi chỗ cho nhau.

Có q truy vấn, mỗi truy vấn thuộc một trong 2 dạng:

- **0 a b x** – (Truy vấn loại 0), nếu linh kiện ban đầu ở băng chuyền **a**, sau công đoạn x - ở băng chuyền **A**, linh kiện ban đầu ở băng chuyền **b**, sau công đoạn x - ở băng chuyền **B**, với truy vấn này, sau công đoạn x , linh kiện ở các băng chuyền **A** và **B** đổi chỗ cho nhau ($a \neq b$, $1 \leq a, b \leq n$, $1 \leq x \leq m$),
- **1 r x** – (Truy vấn loại 1), yêu cầu cho biết linh kiện ban đầu ở băng chuyền **r** sẽ ở băng chuyền nào sau công đoạn xử lý x , $1 \leq r \leq n$, $1 \leq x \leq m$),

Với mỗi truy vấn loại 1 hãy đưa ra băng chuyền xác định được.

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

- Dòng đầu tiên chứa 3 số nguyên n , m và q ($1 \leq n, m, q \leq 10^5$),
- Mỗi dòng trong q dòng tiếp theo chứa một truy vấn.

Kết quả: Đưa ra file văn bản CONVEYOR.OUT với mỗi truy vấn loại 1 đưa ra băng chuyền xác định được, mỗi kết quả đưa ra trên một dòng.

Ví dụ:

CONVEYOR.INP
3 4 4
1 3 4
0 3 2 2
1 3 2
1 2 4

CONVEYOR.OUT
3
2
3



Giải thuật: Rừng tìm kiếm nhanh và siêu nhanh (x-fast-trie và y-fast-trie), Kỹ thuật OOP.

Nhận xét:

Đối với các cấu trúc dữ liệu phức tạp, việc gắn phép xử lý với dữ liệu (*Kỹ thuật lập trình hướng đối tượng – OOP*) sẽ làm cho việc mô tả và triển khai giải thuật đơn giản và dễ hiểu hơn, tuy nhiên giá phải trả là thời gian thực hiện tăng lên đôi chút.

Hệ thống lập trình C++ cho phép tạo các hàm cùng tên và phân biệt theo danh sách tham số hình thức, lời gọi sẽ được nhận dạng theo danh sách tham số thực tế, điều này cho phép tạo các chương trình dễ hiểu theo *Kỹ thuật lập trình có cấu trúc (Structured Programming)*, chương trình được xây dựng theo cách *mịn hóa* dần dần các phép xử lý dữ liệu.

Ta có bản đồ dữ liệu 2 chiều: *Băng chuyền × Công đoạn*,

Thay vì quản lý tường minh vị trí của sản phẩm, ta có thể gắn sản phẩm với *bộ các phép xử lý* phục cập nhật và nhận dạng vị trí sản phẩm theo công đoạn.

Toàn bộ việc xử lý được đưa về 2 phép *cập nhật* và *tìm kiếm* thông tin trên cây quản lý đoạn. Các hàm xử lý cây được lập trình theo sơ đồ đệ quy để có chương trình ngắn gọn và dễ hiểu.

Tổ chức dữ liệu:

Dựa trên 2 cấu trúc chủ yếu **hasmap** và **perm** phục vụ lưu trữ trạng thái băng chuyền và vị trí sản phẩm và mô tả các phép xử lý cơ sở gắn với dữ liệu.

Độ phức tạp của giải thuật: $O(nlnn)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "conveyor."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef uint64_t ull;
typedef int64_t ll;
#define mk make_pair
#define pii pair<int, int>
const ull p1 = 131;
const ull p2 = 129;
const double eps = 1e-8;
const double pi = acos(-1.0);

const int infi = 1e9 + 7;
const ll inf = 1e18 + 7;
const ll dd = 1e5 + 7;

const int size = (1 << 23);
struct hashMap
{
    int64_t H[size + 7] = {};
    int R[size + 7] = {};

    int64_t mHash(int a, int b) {return a * 100000011 + b + 1;}

    int count(int v, int r)
    {
        ll g = mHash(v, r);
        ll h = g & (size - 1);
        while(1)
        {
            if(H[h] == 0) return 0;
            if(H[h] == g) return 1;
            h++;
            if(h == size) h = 0;
        }
        return 1;
    }

    int &operator[](pii T)
    {
        ll g = mHash(T.first, T.second);
        ll h = g & (size - 1);
        while(1)
        {
            if(H[h] == 0 || H[h] == g)
            {
                H[h] = g;
                return R[h];
            }
            h++;
            if(h == size) h = 0;
        }
    }
}
```

```

        }

    } Go, Inv;

struct perm
{
    int V;
    int get(int v)
    {
        if(Go.count(v, V)) return Go[mk(v, V)];
        return v;
    }
    int unGet(int v)
    {
        if(Inv.count(v, V)) return Inv[mk(v, V)];
        return v;
    }
    void swapPr(int a, int b)
    {
        if(!Go.count(a, V))
        {
            Go[mk(a, V)] = a;
            Inv[mk(a, V)] = a;
        }
        if(!Go.count(b, V))
        {
            Go[mk(b, V)] = b;
            Inv[mk(b, V)] = b;
        }
        int toA = get(a);
        int toB = get(b);
        swap(Go[mk(a, V)], Go[mk(b, V)]);
        swap(Inv[mk(toA, V)], Inv[mk(toB, V)]);
    }
    void add(int v, int to)
    {
        Go[mk(v, V)] = to;
        Inv[mk(to, V)] = v;
    }

    void del(int v)
    {
        Go[mk(v, V)] = v;
        Inv[mk(v, V)] = v;
    }
};

perm P[dd * 4];
vector<int> Qu;

void get(int v, int l, int r, int lq, int rq)
{
    if(lq > rq) return;
    if(lq == l && rq == r)
    {
        Qu.push_back(v);
        return;
    }
}

```

```

    }
    int s = (l + r) / 2;
    get(v * 2, l, s, lq, min(rq, s));
    get(v * 2 + 1, s + 1, r, max(lq, s + 1), rq);
}

pair<pii, pii> upd(int v, int l, int r, int x, int a, int b)
{
    if(l == r)
    {
        P[v].swapPr(a, b);
        return mk(mk(a, P[v].get(a)), mk(b, P[v].get(b)));
    }
    int s = (l + r) / 2;
    if(x <= s)
    {
        auto t = upd(v * 2, l, s, x, a, b);
        int toA = P[v * 2 + 1].get(t.first.second);
        int toB = P[v * 2 + 1].get(t.second.second);
        P[v].add(t.first.first, toA);
        P[v].add(t.second.first, toB);
        return mk(mk(t.first.first, toA), mk(t.second.first, toB));
    }
    else
    {
        auto t = upd(v * 2 + 1, s + 1, r, x, a, b);
        int toA = P[v * 2].unGet(t.first.first);
        int toB = P[v * 2].unGet(t.second.first);
        P[v].add(toA, t.first.second);
        P[v].add(toB, t.second.second);
        return mk(mk(toA, t.first.second), mk(toB, t.second.second));
    }
}

int n, m;
int get(int r, int t)
{
    Qu.clear();
    get(1, 0, m - 1, 0, r);
    for(int i = 0; i < Qu.size(); i++) t = P[Qu[i]].get(t);
    return t;
}

int main()
{ int q;
    fi >> n >> m >> q;
    for(int i = 0; i < 4 * m + 5; i++) P[i].V = i;
    for(int i = 0; i < q; i++)
    { int t, l, r, x;
        fi >> t;
        if(t == 0)
        {
            fi >> l >> r >> x;
            upd(1, 0, m - 1, x - 1, l, r);
        }
        else
        { int a, b;
}

```

```
    fi>>a>>b;
    int t = get(b - 1, a);
    fo<<t<<'\\n';
}
fo<<"\\n*** Time: "<<clock() / (double)1000<<" sec";
}
```



Ở khu Manhattan các đại lộ đều chạy từ bắc xuống nam và được đánh số 0, 1, 2, . . . từ đông sang tây. Tất cả các phố đều chạy từ đông sang tây và được đánh số 0, 1, 2, . . . từ nam lên bắc.

Chú vẹt Polly sống ở Manhattan và có n người bạn trong khu phố. Mỗi người bạn có một khu vực sống và không bao giờ rời khỏi khu vực đó. Người thứ i chỉ bay đi bay lại trong khu vực giữa hai đại lộ x_1^i, x_2^i và giữa 2 phố y_1^i, y_2^i , $i = 1 \dots n$.

Một hôm, khi đang đậu trên cây ở giao đại lộ \mathbf{x}_a với phố \mathbf{y}_a chú nghe thấy tiếng kêu cứu xin giúp đỡ. Theo bản năng, chú xác định ngay được khoảng cách d từ chỗ chú đậu tới nơi kêu cứu. Ở Manhattan âm thanh bị các nhà cao tầng chắn lại và truyền theo các đại lộ và phố, vì vậy khoảng cách giữa 2 điểm (\mathbf{a}, \mathbf{b}) và (\mathbf{c}, \mathbf{d}) sẽ là $|\mathbf{a}-\mathbf{c}| + |\mathbf{b}-\mathbf{d}|$.

Trước khi bay đi trợ giúp Polly nhẩm tính xem có bao nhiêu trong số bạn của mình có thể là người đang cần trợ giúp.

Cho q truy vấn, mỗi truy vấn là một khoảng cách d . Hãy xác định với mỗi trường hợp có thể một trong số bao nhiêu người bạn của Polly xin trợ giúp.

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

- ⊕ Dòng đầu tiên chứa 2 số nguyên n và q ($1 \leq n, q \leq 10^5$),
- ⊕ Dòng thứ 2 chứa 2 số nguyên \mathbf{x}_a và \mathbf{y}_a ,
- ⊕ Dòng thứ i trong n dòng sau chứa 4 số nguyên x_1^i, y_1^i, x_2^i và y_2^i ,
- ⊕ Mỗi dòng trong q dòng sau chứa một số nguyên d .

Xét 2 trường hợp:

- ⊖ Các tọa độ đều có giá trị nằm trong khoảng từ 0 đến 10^6 và $0 \leq d \leq 2 \times 10^6$,
- ⊖ Các tọa độ có giá trị nằm trong khoảng từ 0 đến 10^9 và $0 \leq d \leq 2 \times 10^9$,

Kết quả: Đưa ra file văn bản RESCUE.OUT q dòng, mỗi dòng chứa một số nguyên – kết quả của truy vấn tương ứng.

Ví dụ:

RESCUE.INP	
6	13
1	4
0	7 1 6
3	5 0 3
0	1 3 2
4	6 5 3
8	7 7 4
8	0 7 2
0	
1	6
2	7
3	8
4	9
5	10
	11
	12

RESCUE.OUT
1
1
3
4
3
2
2
1
2
2
2
1
0



VV17 Balt20170503 A

Giải thuật: Tổng tiền tố.

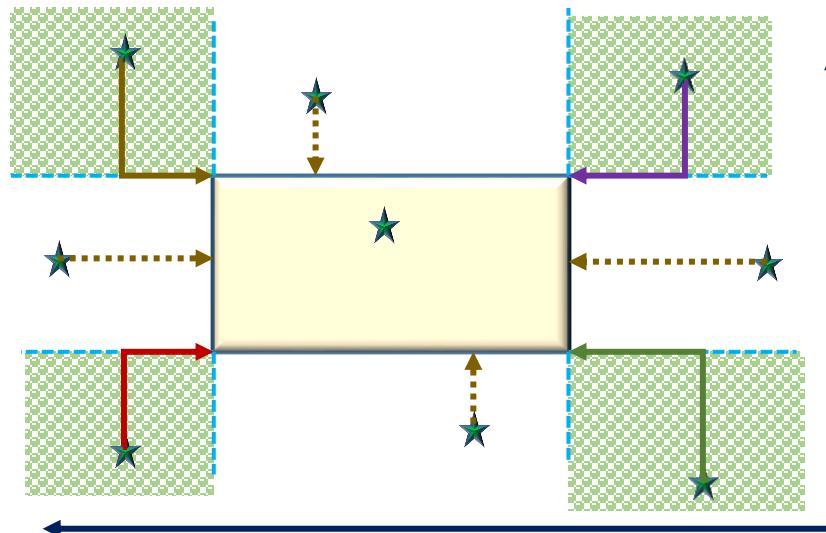
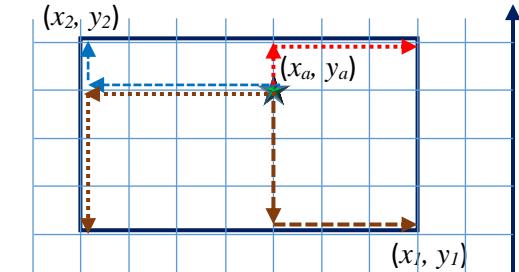
Gọi **A** là điểm có tọa độ (x_a, y_a),

Với mỗi vùng đã cho cần tính khoảng cách manhattan gần nhất và xa nhất từ **A** tới các điểm thuộc vùng đang xét, gọi các khoảng cách đó tương ứng là **dmin** và **dmax**,

Để thuận tiện xử lý, dữ liệu được *chuẩn hóa* để $x_1 \leq x_2, y_1 \leq y_2$.

Không phụ thuộc vào vị trí điểm **A**, **dmax** là giá trị lớn nhất trong số các khoảng cách từ **A** tới 4 đỉnh của vùng đang xét.

Để tính **dmin** cần phân biệt 9 *tình huống*:



Sau khi tính được **dmax** và **dmin** cần đánh dấu để tích lũy tần số.

Có 2 khả năng cần xét:

- ✚ Trường hợp **dmax** là đủ nhỏ (ví dụ, 2×10^6 như trong ràng buộc thứ nhất),
- ✚ Hiệu **dmax-dmin** có thể rất lớn.

Khi **dmax** đủ nhỏ có thể khai báo mảng tần số **frequency** cho *từng giá trị khoảng cách có thể*,

Việc đánh dấu trong trường hợp này đơn thuần là **++frequency[dmin]; --frequency[dmax+1];**

Khi **dmax** lớn: cần tạo mảng lưu các mốc **dmin**, **dmax** khác nhau và mảng tần số **frequency** được phục vụ *ghi nhận sự thay đổi tần số ở các mốc*.

Sau khi ghi nhận với mọi vùng, tiến hành cộng dồn để có mảng tổng tiền tố.

Khi **dmax** bé, mỗi tra cứu với **d** trong truy vấn được thực hiện với chi phí O(1). Trong trường hợp **dmax** lớn – chi phí mỗi truy vấn là O($k \log k$), trong đó **k** là số lượng các mốc khác nhau (không vượt quá $2 \times n$).

Tổ chức dữ liệu:

- Trường hợp **dmax** nhỏ: mảng **int** **p_sum[2*N]={0}** lưu tổng tiền tố các khoảng cách,
- Trường hợp **dmax** lớn:
 - Bảng **map<int, int>** md ghi nhận các mốc thay đổi tổng tiền tố và giá trị thay đổi,
 - Mảng **vector<int>dist** ghi nhận mốc thay đổi,
 - Mảng **vector<int>frequence** ghi nhận giá trị tổng tiền tố ở các mốc.

Trong mọi trường hợp: không lưu lại tọa độ các vùng.

Xử lý:

Phần tính **dmin**, **dmax** là như nhau cho cả 2 trường hợp,

Tính tổng tiền tố và xử lý truy vấn với trường hợp **dmax** nhỏ:

Tích lũy
tổng tiền

```

for (int i=1; i<=dr; ++i) p_sum[i] += p_sum[i-1];

for (int i=0; i<m; ++i)
{
    fi>>d;
    if (d<dl || d>dr) ans=0;
    else ans=p_sum[d];
    fo<<ans<<'\\n';
}

```

Tra cứu kết quả

Độ phức tạp của giải thuật: lớp O(n).

*Trường hợp **dmax** lớn:*

Một trong số các cách ghi nhận và chỉnh lý số liệu thống kê khi gặp mốc trùng nhau là **map<int, int> md**;

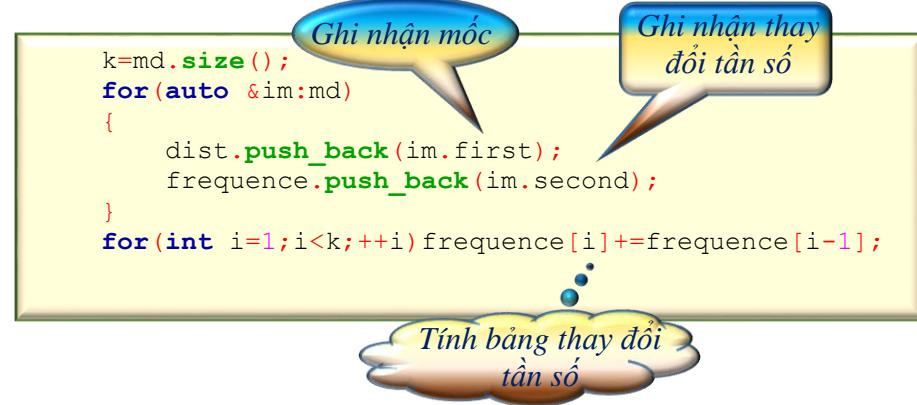
Ghi nhận biến
tìm kiếm

```

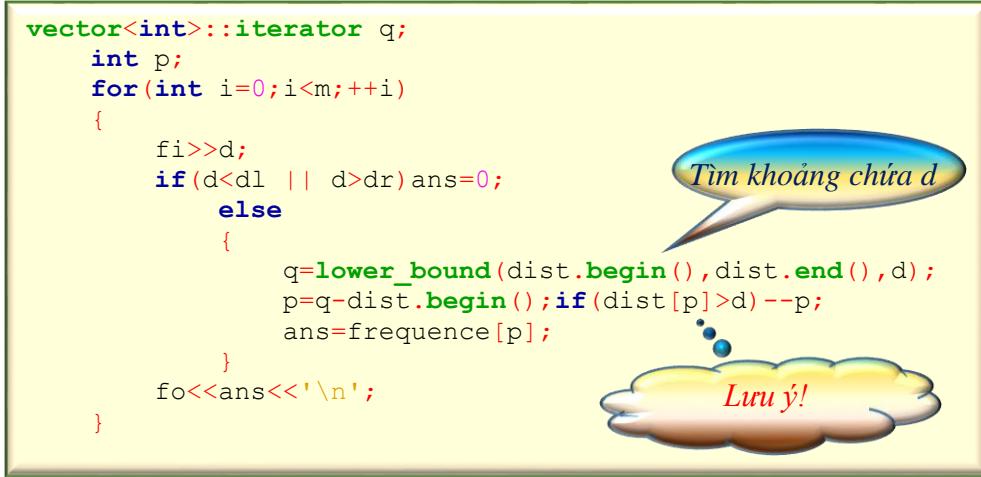
++md[dmin]; --md[dmax+1];
if (dmin<dl) dl=dmin;
if (dmax>dr) dr=dmax;

```

Chuyển dữ liệu sang các mảng phục vụ tra cứu:



Xử lý truy vấn:



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

Chương trình: Trường hợp dmax nhỏ.

```
#include <bits/stdc++.h>
#define NAME "rescue."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N= 100000;
int n,m,d,ans,x1,y1,x2,y2,dmax,dmin;
int a,b,t,dl=2000009,dr=-1;
int p_sum[2*N]={0};

int main()
{
    fi>>n>>m>>a>>b; t=a+b;

    for(int i=0;i<n;++i)
    {
        fi>>x1>>y1>>x2>>y2;
        if(x1>x2) swap(x1,x2);
        if(y1>y2) swap(y1,y2);
        while(1)
        {
            if(x1<=a && a<=x2 && y1<=b && b<=y2) { dmin=0; break; }
            if(a<=x1 && b<=y1) {dmin=abs(x1+y1-t); break; }
            if(a<=x1 && y1<=b && b<=y2) {dmin=x1-a; break; }
            if(a<=x1 && b>=y2) {dmin=x1-a+b-y2; break; }
            if(x1<=a && a<=x2 && b<=y1) {dmin=y1-b; break; }
            if(x1<=a && a<=x2 && b>=y2) {dmin=b-y2; break; }
            if(a>=x2 && b<=y1) {dmin=a-x2+y1-b; break; }
            if(a>=x2 && y1<=b && b<=y2) {dmin=x2-a;break; }
            if(a>=x2 && b>=y2) {dmin=t-(x2+y2);break; }
        }

        dmax=(int) abs(x1-a); dmax+=(int) abs(y1-b);
        dmax=max(dmax,abs(x1-a)+abs(y2-b));
        dmax=max(dmax,abs(x2-a)+abs(y1-b));
        dmax=max(dmax,abs(x2-a)+abs(y2-b));

        ++p_sum[dmin]; --p_sum[dmax+1];
        if(dmin<dl) dl=dmin;
        if(dmax>dr) dr=dmax;
    }

    for(int i=1;i<=dr;++i)p_sum[i]+=p_sum[i-1];

    for(int i=0;i<m;++i)
    {
        fi>>d;
        if(d<dl || d>dr) ans=0;
        else ans=p_sum[d];
        fo<<ans<<'\\n';
    }

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

Chương trình: Trường hợp dmax lớn.

```
#include <bits/stdc++.h>
#define NAME "rescue."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N= 100000;
int n,m,d,ans,x1,y1,x2,y2,dmax,dmin;
int a,b,t,dl=2000009,dr=-1,k;
map<int,int> md;
vector<int> dist,frequence;

int main()
{
    fi>>n>>m>>a>>b; t=a+b;

    for(int i=0;i<n;++i)
    {
        fi>>x1>>y1>>x2>>y2;
        if(x1>x2) swap(x1,x2);
        if(y1>y2) swap(y1,y2);
        while(1)
        {
            if(x1<=a && a<=x2 && y1<=b && b<=y2) { dmin=0; break; }
            if(a<=x1 && b<=y1) {dmin=abs(x1+y1-t); break; }
            if(a<=x1 && y1<=b && b<=y2) {dmin=x1-a; break; }
            if(a<=x1 && b>=y2) {dmin=x1-a+b-y2; break; }
            if(x1<=a && a<=x2 && b<=y1) {dmin=y1-b; break; }
            if(x1<=a && a<=x2 && b>=y2) {dmin=b-y2; break; }
            if(a>=x2 && b<=y1) {dmin=a-x2+y1-b; break; }
            if(a>=x2 && y1<=b && b<=y2) {dmin=x2-a; break; }
            if(a>=x2 && b>=y2) {dmin=t-(x2+y2); break; }
        }

        dmax=(int) abs(x1-a); dmax+=(int) abs(y1-b);
        dmax=max(dmax,abs(x1-a)+abs(y2-b));
        dmax=max(dmax,abs(x2-a)+abs(y1-b));
        dmax=max(dmax,abs(x2-a)+abs(y2-b));

        ++md[dmin]; --md[dmax+1];
        if(dmin<dl) dl=dmin;
        if(dmax>dr) dr=dmax;
    }
    k=md.size();
    for(auto &im:md)
    {
        dist.push_back(im.first);
        frequence.push_back(im.second);
    }
    for(int i=1;i<k;++i) frequence[i]+=frequence[i-1];

    vector<int>::iterator q;
    int p;
    for(int i=0;i<m;++i)
    {
```

```

fi>>d;
if(d<dl || d>dr) ans=0;
else
{
    q=lower_bound(dist.begin() , dist.end() , d);
    p=q-dist.begin(); if(dist[p]>d)--p;
    ans=frequence[p];
}
fo<<ans<<' \n';
}

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

```



VV18. PHÒNG ĐỆM

Tên chương trình: BUFFER.CPP

Hai tiến trình hoạt động song song và sử dụng chung một phòng đệm vòng tròn. Phòng đệm có kích thước m bytes. Với phòng đệm vòng tròn, khi đọc hoặc ghi sau byte số 0 là byte số 1, sau byte số 1 là byte số 2. . . . , sau byte số $m-2$ là byte số $m-1$, sau byte số $m-1$ là byte số 0.

Tiến trình I ghi thông tin vào phòng đệm, mỗi lần ghi một khối w bytes. Tiến trình II đọc thông tin từ phòng đệm, mỗi lần đọc một khối x bytes. Khi một tiến trình đang sử dụng phòng đệm thì tiến trình kia phải chờ.

Khi tiến trình II cần đọc thông tin nhưng trong phòng đệm không có đủ x bytes thì nó sẽ chuyển sang trạng thái chờ, khi nào phòng đệm có không ít hơn x bytes dữ liệu sẽ hoạt động tiếp.

Khi tiến trình I cần ghi thông tin, nếu trong phòng đệm không còn đủ w bytes trống thì nó sẽ chuyển sang trạng thái chờ, khi nào phòng đệm có không ít hơn w bytes trống sẽ hoạt động tiếp.

Có thể xảy ra tình huống tiến trình II phải chờ vì trong phòng đệm không có đủ x bytes dữ liệu và tiến trình I cũng phải chờ vì trong phòng đệm không còn đủ w chỗ trống! Tình huống này được gọi là *bé tắc*.

Với m , x và w cho trước, hãy xác định có thể xảy ra bé tắc hay không. Nếu có thể thì đưa ra thông báo “**DEADLOCK**”. Trong trường hợp bé tắc không thể xảy ra – đưa ra thông báo “**OK**”.

Dữ liệu: Vào từ file văn bản BUFFER.INP gồm một dòng chứa 3 số nguyên m , x và w ($0 < n, x, w \leq 10^{18}$, $x \leq m$, $w \leq m$).

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

Ví dụ:

BUFFER.INP	BUFFER.OUT
5 3 4	DEADLOCK



VV18 Mos_N20161016 B

Giải thuật: Phân tích hoạt động ô tô mát.

Dung lượng thông tin còn lại trong phòng đệm luôn là bội của $\text{USCLN}(\mathbf{r}, \mathbf{w})$ vì dung lượng mỗi lần ghi và đọc đều là bội của số này,

Tính lại $\mathbf{r} = \mathbf{r}/\text{gcd}(\mathbf{r}, \mathbf{w})$; $\mathbf{w} = \mathbf{w}/\text{gcd}(\mathbf{r}, \mathbf{w})$; $\mathbf{m} = \mathbf{m}/\text{gcd}(\mathbf{r}, \mathbf{w})$;

Dung lượng phòng đệm có thể không chia hết cho ước số chung lớn nhất của \mathbf{r} và \mathbf{w} , nhưng phần thừa đó không bao giờ chứa thông tin và bị bỏ qua khi tính lại,

Bây giờ \mathbf{r} và \mathbf{w} nguyên tố cùng nhau.

Nếu trong phòng đệm đang có \mathbf{x} bytes và xuất hiện bέ tẮc thì có nghĩa:

- ✚ Không thể đọc tiếp khi $\mathbf{x} \leq \mathbf{r}-1$,
- ✚ Không thể ghi tiếp khi $\mathbf{m}-\mathbf{x} \leq \mathbf{w}-1 \rightarrow \mathbf{m}-\mathbf{w}+1 \leq \mathbf{x}$.

Như vậy bέ tẮc xảy ra khi tồn tại \mathbf{x} thỏa mãn điều kiện

$$\mathbf{m}-\mathbf{w}+1 \leq \mathbf{x} \leq \mathbf{r}-1 \leftrightarrow \mathbf{m}-\mathbf{w}+1 \leq \mathbf{r}-1 \leftrightarrow \mathbf{r}+\mathbf{w}-2 \geq \mathbf{m}$$

Thật vậy, khi thỏa mãn điều kiện đã đánh dấu ở trên thì sẽ có chuỗi hành động dẫn đến bέ tẮc:

- Đọc các khối thông tin \mathbf{r} từ phòng đệm chừng nào còn có thể,
- Ghi vào phòng đệm *đúng một khối* kích thước \mathbf{w} ,
- Lặp lại hai bước trên.

Sóm hay muộn trong phòng đệm sẽ còn lại $\mathbf{r}-1$ bytes vì quá trình thao tác trên tương ứng với việc nhiều lần thực hiện phép $\mathbf{r}+\mathbf{w}$ theo mô đun \mathbf{r} , ta sẽ có tất cả các lớp đồng dư với \mathbf{r} , trong số đó có $\mathbf{r}-1$.

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "buffer."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t l,r,w,t;

int main()
{
    fi>>l>>r>>w;
    t=__gcd(r,w);
    r/=t; w/=t; l/=t;
    if(r+w-2>=l) fo<<"DEADLOCK"; else fo<<"OK";
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Sự ra đời và phổ biến của camera bay (*drones*) đã làm xuất hiện một môn thể thao nghệ thuật mới – Trình diễn came ra bay đội hình. Các camera xếp thành đội hình và bay đồng bộ với các thao tác phức tạp trên không.

Để rèn luyện kỹ năng phối hợp đồng bộ người ta cho các camera bay vòng tròn theo chiều kim đồng hồ. Độ cao tập hợp đội hình được coi là 0. Trên vòng tròn bay biểu diễn có n điểm, ở điểm thứ i (theo chiều kim đồng hồ) đội hình phải ở độ cao a_i ($i = 1 \div n$). Các độ cao khác nhau từng đội một. Đường bay từ một điểm tới điểm tiếp theo cạnh nó được gọi là khoảng bay.

Ban huấn luyện tổ chức đo chính xác các tham số bay ở 3 điểm i , j và k . Ba điểm này phải thỏa mãn các điều kiện: $a_i < a_j$, $a_j > a_k$, số khoảng bay từ i tới j cộng với số khoảng bay từ j tới k phải là nhỏ nhất (*càng ngắn càng khó điều khiển!*).

Hãy chỉ ra một bộ 3 số i , j , k thỏa mãn điều kiện của Ban huấn luyện.

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

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

Kết quả: Đưa ra file văn bản SHOWS.OUT trên một dòng 3 số nguyên tìm được. Nếu có nhiều phương án thỏa mãn thì đưa ra một phương án tùy chọn.

Ví dụ:

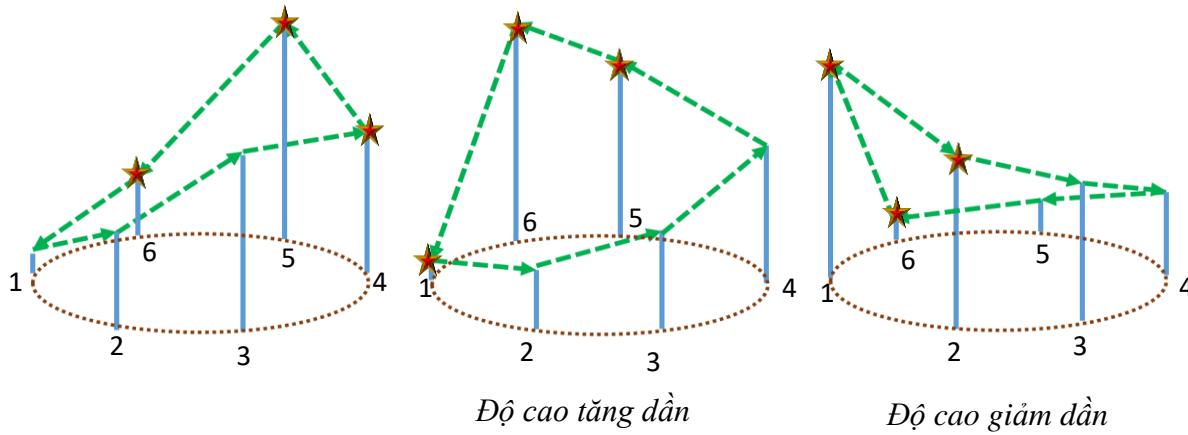
SHOWS.INP
4
2 0 1 6

SHOWS.OUT
3 4 1



Giải thuật: Đoán nhận giải thuật.

Do các độ cao khác nhau từng đôi một nên ta chỉ cần chọn j sao cho a_j là cực đại địa phương, khi đó $i = j-1$ và $k = j+1$. Nếu $j-1 = 0$ thì $i = 1$, nếu $j+1 = n+1$ thì $k = 1$. Độ dài đường đi $i \rightarrow j \rightarrow k$ là 2 và là ngắn nhất.



Tuy vậy việc tìm cực đại địa phương tương đối phức tạp vì liên quan tới độ cao điểm trước và sau, hơn thế nữa phải thực hiện 2 phép so sánh. Ngoài ra cần phân lập trường hợp cực đại địa phương nằm ở cuối (dãy tăng dần) hay ở đầu (dãy giảm dần).

Việc lập trình sẽ đơn giản hơn nhiều nếu chọn a_j là *cực đại toàn cục*.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "shows."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n,t,p,q,r,hmx=-2000000001;

int main()
{
    fi>>n;
    for(int i=1;i<=n;++i)
    {
        fi>>t;
        if(t>hmx) hmx=t, q=i;
    }
    if(q==n) r=1; else r=q+1;
    if(q==1) p=n; else p=q-1;
    fo<<p<<' '<<q<<' '<<r;
}
```



VV20. KHOẢNG CÁCH HAMMING

Tên chương trình: HAMMING.CPP

Trên giờ Tin học Hermione được làm quen với một khái niệm khoảng cách mới – khoảng cách Hamming của các số thập phân. Khoảng cách Hamming giữa 2 số nguyên không âm x, y là số lượng các chữ số khác nhau trong dạng biểu diễn thập phân ở cùng một vị trí của 2 số. Nếu một số có ít chữ số hơn số kia thì bổ sung thêm các số 0 không có nghĩa để 2 số có cùng số lượng chữ số.

Ví dụ, $x = 315, y = 245$ thì khoảng cách Hamming $h(x, y) = 2$. Với $x = 315, y = 3$ có $h(315, 3) = h(315, 003) = 3$.

Hermione không có cảm tình lắm với khái niệm khoảng cách này, nhưng sớm hay muộn cũng phải làm bài tập với nó. Vì vậy Hermione quyết định tăng cường thêm khả năng mới cho đứa thần của mình: chỉ cần chỉ đứa thần vào 2 số a và b ($a \leq b$) đứa thần sẽ cho biết ngay khoảng cách lớn nhất có thể đạt được giữa 2 số x và y thuộc đoạn $[a, b]$. Ví dụ, với $a = 11, b = 17$, kết quả sẽ là 1 (có thể chọn $x = 11, y = 16$). Sau một hồi chật vật, cuối cùng Hermione cũng hoàn thành xong việc huấn luyện.

Hãy xác định kết quả Hermione sẽ nhận được khi chỉ vào 2 số a và b .

Dữ liệu: Vào từ file văn bản HAMMING.INP gồm 2 dòng, dòng đầu tiên chứa số nguyên a , dòng thứ 2 chứa số nguyên b ($1 \leq a \leq b \leq 10^{1\,000\,000}$).

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

Ví dụ:

HAMMING.INP
1
11

HAMMING.OUT
2



VV20 Mos_OO20170317 B

Giải thuật: Khả năng phân tích toán học.

Đánh số các vị trí các chữ số của một số từ hàng đơn vị trở đi và bắt đầu từ 1,

Gọi \mathbf{k} là vị trí lớn nhất mà các chữ số tương ứng của \mathbf{a} và \mathbf{b} khác nhau,

Ta có thể chọn $\mathbf{x} = \mathbf{a}$,

Xây dựng \mathbf{y} : các chữ số của \mathbf{a} , \mathbf{b} , \mathbf{x} tại vị trí i ký hiệu là \mathbf{a}_i , \mathbf{b}_i , \mathbf{x}_i ,

➡ Các chữ số ở vị trí $j > k$ (nếu có) $\mathbf{y}_j = \mathbf{b}_j$,

➡ $\mathbf{y}_k = \mathbf{b}_k$,

➡ Với $i = k-1 \div 1$:

$$\mathbf{y}_i = \begin{cases} 0 & \text{nếu } \mathbf{a}_i \neq 0, \\ 1 & \text{nếu } \mathbf{a}_i = 0, \end{cases}$$

Từ cách xây dựng ta có:

➡ $\mathbf{a} < \mathbf{x} < \mathbf{y} = \mathbf{b}$,

➡ Khoảng cách Hamming $\mathbf{h}(\mathbf{x}, \mathbf{y}) = k$.

Dễ dàng thấy rằng không thể tìm được cặp số (\mathbf{x}, \mathbf{y}) có $\mathbf{h}(\mathbf{x}, \mathbf{y})$ lớn hơn.

Cách lập luận này được gọi là *phương pháp chứng minh xây dựng* (*Constructive evidence*) – một phương pháp thường được áp dụng trong toán học.

Dữ liệu rất lớn vì vậy cần lưu trữ dưới dạng xâu.

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "hamming."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
string a,b;
int la,lb,k;
int main()
{
    fi>>a>>b;
    la=a.size(); lb=b.size(); k=lb;
    if(la==lb)
    {
        k=0;
        while(k<la && a[k]==b[k]) ++k;
        k=la-k;
    }
    fo<<k;
    fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
}
```



Chim cu là một loài đặc biệt, chuyên đẻ trứng vào tổ các loại chim khác, nhờ chúng ấp và nuôi con cho mình. Ở mỗi tổ đến nhờ chúng hất bỏ một quả trứng của chim chủ, để vào đó quả trứng của mình với kích thước và màu sắc tương tự! Vì vậy chim chủ không biết đã có sự đột nhập và thay thế.

Để nghiên cứu về đời sống của chim cu người ta đặt trên n cây mỗi cây một tổ chim để thu hút các loài chim khác nhau về sống và kéo theo việc chim cu về đẻ trứng.



Mỗi quả trứng của chim cu được đặc trưng bởi cặp 2 số nguyên khác nhau (x, y). Quả trứng (x, y) chỉ có thể đặt vào tổ x hoặc tổ y , không đặt vào nơi nào khác được. quả (x, y) hoàn toàn giống quả (y, x). Khi chim tới tổ x đẻ trứng (x, y) vào đó. Quá trình sẽ diễn ra như sau: Nếu tổ x không có trứng thì đẻ trứng vào đó và quá trình xử lý kết thúc. Nếu ở đó đã có quả (x, p) thì chim sẽ đẻ trứng (x, y) vào tổ x , tha quả (x, p) đặt sang tổ p , nếu tổ p đã có trứng thì tha quả trứng cũ ở p đi tiếp nơi khác, . . . Cứ như thế cho đến khi quá trình di chuyển kết thúc.

Hiện tại có m tổ chứa trứng và cần phải trả lời q câu hỏi nghiên cứu, mỗi câu có một trong 3 dạng:

1. (*Khảo sát*) Nếu chim đẻ trứng (x, y) vào tổ x thì quá trình di chuyển trứng có kết thúc hay không? Chỉ đơn thuần khảo sát, trên thực tế **trạng thái các tổ không thay đổi**.
2. (*Biến đổi*) Việc thêm quả trứng (x, y) vào tổ x có là một quá trình kết thúc hay không? Nếu có – **thực hiện việc bổ sung** trứng (x, y) vào tổ x theo quy trình đã mô tả ở trên.
3. (*Khảo sát*) Có bao nhiêu **cặp có trình tự** (x, y) cho phép chim đẻ thêm trứng (x, y) vào tổ x với thực trạng các trứng hiện có trong các tổ? Việc xét các cặp độc lập với nhau, dựa trên cấu hình hiện có.

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

- ⊕ Dòng đầu tiên chứa 3 số nguyên n, m và q ($2 \leq n \leq 2 \times 10^5, 0 \leq m \leq n, 1 \leq q \leq 6 \times 10^5$),
- ⊕ Dòng thứ i trong m dòng sau chứa 2 số nguyên $x_i y_i$ cho biết tổ x_i có quả trứng (x_i, y_i), $x_i \neq y_i$,
- ⊕ Dòng thứ j trong q dòng sau chứa thông tin xác định một câu hỏi, bắt đầu bằng số nguyên t_j – loại câu hỏi ($t_j = 1, 2, 3$). Nếu $t_j < 3$ thì tiếp sau là 2 số nguyên x_j và y_j .

Kết quả: Đưa ra file văn bản CUCKOO.OUT, kết quả mỗi câu hỏi đưa ra trên một dòng. Với các câu hỏi loại 1 và 2 câu trả lời là “**Yes**” hoặc “**No**” tùy theo quá trình di chuyển có kết thúc hay không. Với câu hỏi loại 3 – kết quả là một số nguyên.

Ví dụ:

CUCKOO.INP
5 3 8
1 2
5 1
2 4
1 1 2
3
2 1 2
3
2 4 2
2 5 3
3
1 4 5

CUCKOO.OUT
Yes
20
Yes
8
No
Yes
0
No



VV21
Mos_OO20170317 A

Giải thuật: Cấu trúc dữ liệu DSU

Xây dựng đồ thị, trong đó mỗi tổ là một đỉnh, mỗi quả trứng là một cạnh,
Xét các thành phần liên thông,

Trong từng thành phần liên thông mỗi cạnh tương ứng với đỉnh của mình vì vậy số cạnh không thể lớn hơn số đỉnh,

Thành phần liên thông k đỉnh thì có k hoặc $k-1$ cạnh (nếu ít cạnh hơn sẽ không còn liên thông),

Gọi thành phần liên thông có $k-1$ cạnh là *loại 1*, có k cạnh – *loại 2*,

Trứng có thể được bổ sung nếu cạnh tương ứng với nó nối thành phần liên thông loại 1 với thành phần liên thông bất kỳ khác hoặc nối 2 đỉnh loại 1.

Để biết một đỉnh thuộc thành phần liên thông nào cũng như số đỉnh trong mỗi thành phần liên thông cần dùng *cấu trúc các tập không giao nhau* (*Disjoint-Set-Union* – DSU),

Với mỗi tập: lưu thêm thông tin về số cạnh trong tập,

Khi bổ sung cạnh trong thành phần liên thông loại 1 – tăng số lượng cạnh của tập thêm 1, khi nối 2 thành phần liên thông, số cạnh miền liên thông mới bằng tổng số cạnh 2 miền cũ cộng thêm 1,

Gọi **cnt1** là số lượng đỉnh của các thành phần liên thông loại 1, **cnt2** là số lượng đỉnh của các thành phần liên thông loại 2, dễ dàng thấy rằng số lượng cạnh cho phép bổ sung là $2 \times \text{cnt1} \times \text{cnt2} + \text{cnt1} \times (\text{cnt1}-1)$.

Như vậy, với cấu trúc DSU ta luôn có thông tin trả lời cho các loại câu hỏi.

Tổ chức dữ liệu:

Mảng **int** `p` [MAXN] lưu đỉnh đại diện của mỗi miền liên thông,

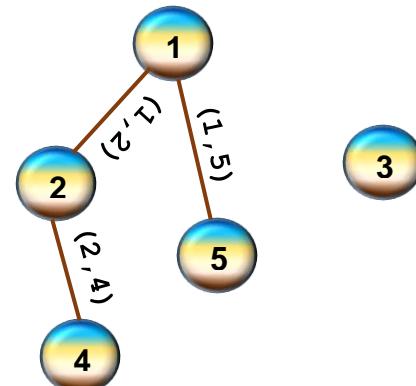
Mảng **int** `size` [MAXN] lưu kích thước các miền liên thông,

Mảng **int** `c` [MAXN] đánh dấu đỉnh nhận dạng các thành phần liên thông,

Biến **int** `tot_size` ghi nhận tổng số đỉnh của các thành phần liên thông loại 2.

Xử lý:

Toàn bộ các xử lý được đặt cơ sở trên hai hàm chính:



- **int get(int v)** – tìm đỉnh đại diện của **v**,
- **void join(int a, int b)** – hợp nhất 2 cây có đỉnh đại diện tương ứng là **a** và **b**.

Hàm **get()** được thực hiện theo đúng sơ chuẩn nêu trong lý thuyết.

Hàm **join()**:

```

void join(int a, int b)
{
    a = get(a);
    b = get(b);

    if (a == b)
    {
        c[a] = true;
        tot_size += size[a];
        return;
    }

    if (c[a] && !c[b]) tot_size += size[b];
    if (!c[a] && c[b]) tot_size += size[a];

    if (size[a] > size[b]) swap(a, b);

    p[a] = b;
    c[b] [= c[a];
    size[b] += size[a];
}

```

Dộ phức tạp của giải thuật: O((m+q)logn).

. Chương trình

```
#include <bits/stdc++.h>
#define NAME "cuckoo."
#include <cstdio>
#include <cassert>
#include <algorithm>

using namespace std;
ifstream fi (NAME"inp");
//ifstream fi ("72");
ofstream fo (NAME"out");

const int MAXN = 200100;

int p[MAXN];
int c[MAXN];
int size[MAXN];
int tot_size=0;

int get(int v)           //int find_set(int v)
{
    if (p[v] != v) return v = get(p[v]);
    return p[v];
}

void join(int a, int b)   // void union_sets (int a, int b)
{
    a = get(a);
    b = get(b);

    if (a == b)
    {
        assert(!c[a]);
        c[a] = true;
        tot_size += size[a];
        return;
    }

    if (c[a] && !c[b]) tot_size += size[b];
    if (!c[a] && c[b]) tot_size += size[a];

    if (size[a] > size[b]) swap(a, b);

    p[a] = b;
    c[b] |= c[a];
    size[b] += size[a];
}

int main()
{
    int n, m, q;

    fi>>n>>m>>q;
```

```

for (int i = 0; i < n; i++) p[i] = i, size[i] = 1;      //void make_set (int
v)
for (int i = 0; i < m; i++)
{
    int a, b;
    fi>>a>>b;
    --a, --b;
    join(a, b);
}

for (int i = 0; i < q; i++)
{
    int ty;
    fi>>ty;
    if (ty == 1 || ty == 2)
    {
        int a, b;
        fi>>a>>b;
        --a, --b;
        if (c[get(a)] + c[get(b)] > 1) fo<<"No\n";
        else
        {
            fo<<"Yes\n";
            if (ty == 2) join(a, b);
        }
    } else
        fo<<2*tot_size*1LL*(n-tot_size)+(n-tot_size)*1LL*(n-tot_size-1)<<'\n';
}
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```



VV22. CHẤM BÀI

Tên chương trình: TESTING.CPP

Một hệ thống tự động chấm bài trong các kỳ thi lập trình được đưa vào thử nghiệm. Để đánh giá kết quả người ta chuẩn bị m tests, đánh số từ 1 đến m .

Ban đầu, tất cả các tests được lấy ra từ cơ sở dữ liệu và nạp vào hàng đợi theo trình tự từ 1 đến m .

Chương trình cần chấm được kiểm tra n lần, mỗi lần được xác định bởi 2 số nguyên a_i và b_i , $a_i \leq b_i$ và cùng chẵn hoặc cùng lẻ, $i = 1 \div n$. Ở lần chấm thứ i , chương trình được kiểm tra với các tests ở vị trí $a_i, a_i+2, a_i+4, \dots, b_i$ trong hàng đợi, sau đó các tests này bị xóa khỏi hàng đợi, các tests còn lại trong hàng đợi sẽ bị dồn lên, đứng sát nhau từ vị trí 1 trở đi. Ví dụ, sau một số lần kiểm tra, trong hàng đợi còn các tests 2, 3, 4, 5, 10, 12, 13, 20. Lần kiểm tra tiếp theo có $a_i = 3$ và $b_i = 7$, khi đó các tests 4, 10 và 13 sẽ được sử dụng, sau đó chúng bị loại khỏi hàng đợi và trong hàng đợi còn lại các tests 2, 3, 5, 12, 20. Test có số nhỏ nhất được sử dụng trong lần kiểm tra này là 4 và test có số lớn nhất là 13.

Với mỗi lần kiểm tra hãy xác định số nhỏ nhất và lớn nhất trong cơ sở dữ liệu của các tests được sử dụng.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n \leq 10^5$, $1 \leq m \leq 10^{18}$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên cùng chẵn hoặc cùng lẻ a_i và b_i ($1 \leq a_i \leq b_i \leq m$), dữ liệu đảm bảo có thể chọn được ít nhất một test từ hàng đợi.

Kết quả: Đưa ra file văn bản TESTING.OUT, với mỗi lần kiểm tra đưa ra rẽn một dòng 2 số nguyên xác định số nhỏ nhất và lớn nhất trong cơ sở dữ liệu của các tests được sử dụng.

Ví dụ:

TESTING.INP
2 10
2 8
1 3

TESTING.OUT
2 8
1 5



VV22 Mos_OO20170317 C

Giải thuật: Ứng dụng Treap (Cây Đè các).

Xem nội dung chi tiết ở phần lý thuyết tương ứng.

Đây là mẫu bài toán minh họa cho lý thuyết đã nêu.

Chương trình

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

struct node
{
    node *l;
    node *r;
    ll x;
    ll msk;
    ll sz;
    int cn;
};

typedef node* pnode;

const int MAXNODES = 100000 * 100;
node nodes[MAXNODES];
int curnode = 0;

inline pnode newnode()
{
    return nodes + curnode++;
}

pnode fullnode(ll sz)
{
    pnode ans = newnode();
    ans->l = ans->r = NULL;
    ans->x = ans->sz = sz;
    ans->cn = ans->msk = 0;
    return ans;
}

void go(pnode a, int tp)
{
    if (a->x == 0) return;
    if ((a->x == 1) && (tp == 1)) return;
    a->x = (a->x + tp) / 2;
    a->msk += (ll)tp << ((a->cn)++);
}
```

```

void push(pnode a)
{
    if (! (a->l))
    {
        a->l = fullnode((a->sz + 1) / 2);
        a->r = fullnode(a->sz / 2);
    }
    for (int i = 0; i < a->cn; i++)
    {
        int v1 = ((a->msk) >> i) & 1;
        int v2 = ((a->l->x & 1) ^ v1);
        go(a->l, v1);
        go(a->r, v2);
    }
    a->msk = a->cn = 0;
}

pnode root;

ll gtid(ll x)
{
    pnode cur = root;
    ll cl = 1;
    while (cur->sz > 1)
    {
        push(cur);
        if (cur->l->x < x)
        {
            cl += cur->l->sz;
            x -= cur->l->x;
            cur = cur->r;
        }
        else cur = cur->l;
    }
    return cl;
}

ll lv, rv;
void modify(pnode p, ll x)
{
    if ((x >= lv) && (x + p->x <= rv + 1))
    {
        if ((x & 1) == (lv & 1)) go(p, 0);
        else go(p, 1);
        return;
    }
    push(p);
    ll md = (x + p->l->x);
    if (md <= rv) modify(p->r, x + p->l->x);
    if (md > lv) modify(p->l, x);
    p->x = p->l->x + p->r->x;
}

int main()
{
    int n;

```

```

long long m;
fi >> n >> m;
root = fullnode(m + 1);
for (int i = 0; i < n; i++)
{
    ll l, r;
    fi >> l >> r;
    fo << gtid(l) << " " << gtid(r) << "\n";
    lv = l, rv = r;
    modify(root, 1);
}
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
return 0;
}

```



Dãy nhà dọc theo trục lộ giao thông được xây dựng từ những năm 60 của thế kỷ XX theo phương pháp lắp ghép bê tông kiểu cũ, nay đã xuống cấp nghiêm trọng, cần phá đi để xây mới. Mỗi nhà được trung bằng một ký tự thể hiện kiểu cấu trúc. Các ký tự có thể là chữ cái la tinh (hoa hoặc thường và phân biệt chữ hoa với chữ thường), ký tự số hệ 10 hoặc các ký tự từ tập “ , !_.- ”. Các ký tự này tạo thành xâu s mô tả kiểu các nhà theo thứ tự từ trái sang phải.

Các nhà khoa học đang xây dựng phương pháp nổ định hướng phá sập nhanh chóng dãy nhà liên tiếp nhau mà các ký tự đặc trưng của nó tạo thành xâu p . Khoảng cách giữa nhà thứ i (kiểu p_i) và nhà thứ $i+1$ (kiểu p_{i+1}) không quan trọng, miễn là giữa chúng không có nhà khác đứng xen.

Thị trưởng thành phố rất sốt ruột với kế hoạch xây dựng lại và ra lệnh lần lượt phá các nhà nguy hiểm hơn cả ở các vị trí x_1, x_2, \dots, x_m tính từ trái sang phải ($1 \leq x_1 < x_2 < \dots < x_m \leq \text{length}(s)$) để kịp xây dựng trước khi mùa mưa tới. Chỉ có một đội thợ có đủ tay nghề và dụng cụ tháo dỡ nhà cũ, giải phóng mặt bằng nên sau khi phá một nhà xong, họ phải mất một khoảng thời gian tập kết thiết bị cho việc tháo dỡ nhà tiếp theo.

Các nhà khoa học muốn đảm bảo chắc chắn phương pháp mới của mình thành công nên mất khá nhiều thời gian để rè soát, tính toán lại các tham số của vụ nổ định hướng trên mô hình số. Họ được phép thực hiện phương án mới khi nào mọi việc đã sẵn sàng, nhưng chỉ được tiến hành trước hoặc sau khi đội tháo dỡ hoàn thành xong một công việc của mình ở một ngôi nhà theo kế hoạch do thị trưởng thành phố giao.

Hãy xác định có bao nhiêu dãy nhà khác nhau cho phép thực nghiệm phương án mới.

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

- ✚ Dòng đầu tiên chứa xâu s độ dài không quá 10^6 ,
- ✚ Dòng thứ 2 chứa xâu p độ dài không quá 10^6 và không vượt quá xâu s ,
- ✚ Dòng thứ 3 chứa số nguyên m , ($0 \leq m \leq \text{length}(s)$)
- ✚ Dòng thứ tư chứa m số nguyên x_1, x_2, \dots, x_m thỏa mãn các điều kiện đã nêu.

Kết quả: Đưa ra file văn bản PLAN.OUT một số nguyên – số dãy nhà khác nhau cho phép thực nghiệm phương án mới.

Ví dụ:

PLAN.INP	PLAN.OUT
<pre>aabbcc abc 3 2 4 5</pre>	<pre>2</pre>



Giải thuật I: Mô phỏng tiến trình, độ phức tạp $O(n^2)$.

Lần lượt xét câu hình dãy nhà ở các thời điểm khi số nhà bị phá là $0, 1, 2, \dots, m$,

Trường hợp chưa có nhà nào bị phá: kiểm tra các vị trí $0, 1, 2, \dots, ls-lp$, đếm số lần xuất hiện xâu p như một xâu con các ký tự liên tiếp nhau trong s :

```
for(int i=0; i<=ls-lp; ++i)
{
    t=s.substr(i, lp);
    ans+=t==p;
}
```

Đếm câu hình sau khi phá i tòa nhà:

- ✚ Xác định vị trí tòa nhà cần phá ở lần thứ i ,
- ✚ Xóa ký tự tương ứng khỏi xâu s ,
- ✚ Gọi vị trí ký tự bị xóa trong xâu s đang xét là k , kiểm tra sự xuất hiện của xâu p trong s bắt đầu từ các vị trí $k, k-1, k-2, \dots, k-lp+1$ và tương ứng cập nhật kết quả. Cần lưu ý là phụ thuộc vào k , một số vị trí đã nêu có thể không tồn tại hoặc không cần kiểm tra.

```
for(int i=0; i<n; ++i) { fi>>b[i]; --b[i]; }
for(int i=0; i<n; ++i)
{
    b[i]+=i;
    s.erase(b[i], 1); --ls;
    u=max(0, b[i]-lp+1); v=min(ls-lp+1, b[i]);
    for(int j=u; j<v; ++j)
    {
        t=s.substr(j, lp);
        ans+=t==p;
    }
}
```

Nhận xét: Không cần thiết lưu mảng giá trị $b[i]$.

Độ phức tạp của giải thuật: $O(ls^2)$.

Chương trình I

```
#include <bits/stdc++.h>
#define NAME "plan."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N=1000002;
int n;
int lt,ls,lp,lb,lpt,lps,ans=0;
string s,t,p,pt,ps;
int tg,m,u,v;

int main()
{
    fi>>s>>p; ls=s.size(); lp=p.size();
    for(int i=0;i<=ls-lp;++i)
    {
        t=s.substr(i,lp);
        ans+=t==p;
    }
    fi>>n;
    for(int i=0;i<n;++i) {fi>>b[i];--b[i];}
    for(int i=0;i<n;++i)
    {
        b[i]-=i;
        s.erase(b[i],1);--ls;
        u=max(0,b[i]-lp+1); v=min(ls-lp+1,b[i]);
        for(int j=u;j<v;++j)
        {
            t=s.substr(j,lp);
            ans+=t==p;
        }
    }
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
}
```

Giải thuật II: Hàm Z, Cây Fenwick.

Gọi **t** là xâu xác định dãy nhả còn lại sau khi thực hiện phá hủy xong **m** nhả theo kế hoạch phá các nhả nguy hiểm,

Số lượng dãy khác nhau phù hợp điều kiện thí nghiệm được xác định từ 3 nguồn:

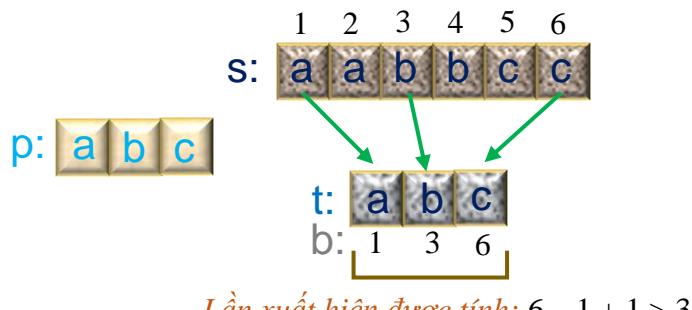
- ✚ Số lần xuất hiện xâu **p** trong **s** như xâu con các ký tự liên tục,
- ✚ Số lần xuất hiện xâu **p** trong **t** như xâu con các ký tự liên tục và không trùng với xâu con nào đã xác định ở bước trước,
- ✚ Số lần xuất hiện xâu **p** trong đó phần đầu của **p** thuộc **t**, phần còn lại thuộc **s** kể từ sau một ký tự bị xóa nào đó trở đi và không trùng với các cấu hình đã xác định ở 2 bước trước.

Việc tìm tất cả các lần xuất hiện xâu **p** trong **s** có thể thực hiện một cách có hiệu quả thông qua hàm **Z**.

Tương tự như vậy với việc tìm các lần xuất hiện xâu **p** trong **t**, nhưng phải loại bỏ các lần xuất hiện trong đó chỉ chứa ký tự ở các vị trí liên tục trong **s**.

Gọi **lp** là độ dài xâu **p**, **lt** là độ dài xâu **t** và **b_i** – vị trí của ký tự **t_i** trong xâu **s**, **zpt_i** là độ dài tiền tố chung lớn nhất của **p** và xâu con của **t** bắt đầu từ vị trí **i** (của **t**). Khi đó, lần xuất hiện của **p** trong **t** từ vị trí **i** sẽ được tính nếu thỏa mãn:

- ▣ **zpt_i = lp**, (*Xuất hiện như xâu con các ký tự liên tiếp trong t*),
- ▣ **b_{i+lp-1}-b_i-1 > lp** (*Tồn tại ký tự bị xóa trong khoảng tương ứng*).



Xử lý nguồn thứ 3:

Gọi **zpt_x** là độ dài *tiền tố chung lớn nhất* của xâu **p** với xâu con của **t** bắt đầu từ vị trí **x**,

Gọi **zps_y** là độ *dài hậu tố chung lớn nhất* của xâu **p** với xâu con của **s** kết thúc tại vị trí **y**,

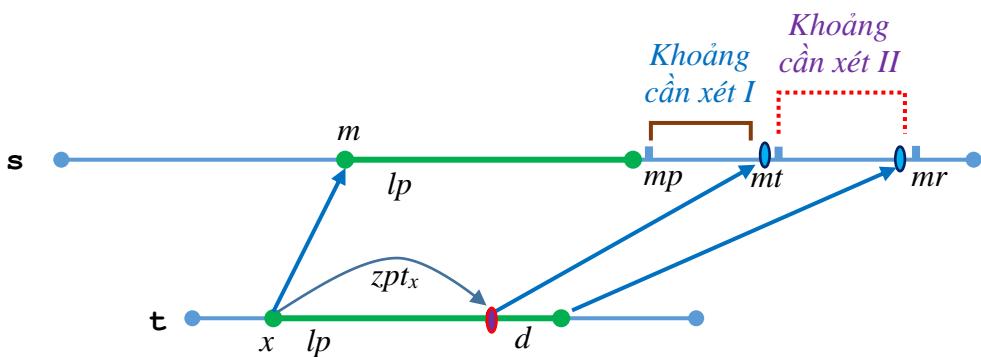
Ký hiệu $f(\mathbf{k})$ – số lượng ký tự còn lại tính từ đầu cho tới \mathbf{k} (kể cả vị trí \mathbf{k}) sau khi xóa các ký tự trong danh sách đã cho có vị trí nhỏ hơn hoặc bằng \mathbf{k} .

Để cắp (\mathbf{x}, \mathbf{y}) xác định một xâu \mathbf{p} cần và đủ là:

$$zpt_{\mathbf{x}} + zps_{\mathbf{y}} \geq 1p,$$

$$\text{Tồn tại } j \mid (f(x_j) - x + 1) + (y - x_j) = 1p$$

Xét ký tự ở vị trí \mathbf{x} trong \mathbf{t} và $zpt_{\mathbf{x}} > 0$. Vị trí tương ứng của \mathbf{x} trong \mathbf{s} là $m = b[\mathbf{x}]$ và vị trí ký tự trái nhất trong \mathbf{s} có thể tạo thành \mathbf{p} sau khi xóa một vài ký tự là $mp = m + 1p$.



Vị trí ký tự phải nhất trong \mathbf{t} có khoảng cách tới \mathbf{x} bằng $1p$ có vị trí tương ứng trong \mathbf{s} nhỏ hơn $mr = b[\mathbf{x}+1p]$.

Gọi mt là vị trí trong \mathbf{s} của phần tử cuối cùng tham gia vào tiền tố của \mathbf{t} bắt đầu từ \mathbf{x} . Ta có $mt = b[\mathbf{x}+zpt_{\mathbf{x}}-1]$.

Lưu ý: Các địa chỉ tính cần đảm bảo thuộc xâu \mathbf{s} .

Số lượng các cấu hình mới được xác định bởi số vị trí \mathbf{y} trong \mathbf{s} thỏa mãn các điều kiện:

- Thuộc khoảng $[mp, mr]$,
- $zps_{\mathbf{y}} \geq 1p - zpt_{\mathbf{x}}$.

Trong sơ đồ tính toán cần phân biệt trường hợp $mt < mp$ và $mt \geq mp$.

Việc tính số lượng các vị trí cần tìm có thể thực hiện một cách có hiệu quả bằng cây Fenwick.

Tổ chức dữ liệu:

Ngoài việc lưu trữ các xâu còn cần có các mảng:

- `int b[N]` – lưu vị trí các ký tự của \mathbf{t} trong \mathbf{s} ,
- `int zpt[N], zps[N]` – lưu độ dài tiền tố và hậu tố đã nêu,

☞ **int ft[N], fs[N]** – phục vụ tổ chức cây Fenwick tính các câu hình mới.

Xử lý:

Tính **zpt_i**:

- Tạo xâu **pt = p + "*" + t**,
- Tính giá trị hàm Z với **pt** từ vị trí **lp**,
- Đưa **zpt_i**, $i = lp \div lp+1t$ về đầu.

Tính **zps_i**:

- Đảo ngược các xâu **p** và **s**,
- Các bước xử lý còn lại: tương tự trường hợp tính **zpt_i**.

Tính các lần xuất hiện của **p** ban đầu trong **s** và **t**: dựa vào các mảng giá trị **zps_i** và **zpt_i**. Khi xử lý **t** cần lưu ý bỏ qua các lần xuất hiện đã tính khi xử lý **s**.

Cần xây dựng riêng 2 hàm **calc_1** và **calc_2** cập nhật kết quả cho 2 phần đã nêu vì giá trị **d** có thể khác nhau với các trường hợp **mt < mp** và **mt ≥ mp**.

Cây Fenwick **ft** phục vụ cho hàm thứ nhất, cây **fs** phục vụ cho hàm thứ 2. Mỗi lần tính cần loại bỏ các phần tử thừa trước đó khỏi cây và bổ sung các phần tử mới ứng với đoạn cần tính vào cây. Với mỗi cây, mỗi phần tử **zps_i** chỉ được nạp vào cây không quá một lần và có thể bị loại bỏ cũng không quá một lần trong toàn bộ quá trình xử lý!

Độ phức tạp của giải thuật: $O((lp+ls)\ln(lp+ls))$.

Chương trình II:

```
#include <bits/stdc++.h>
#define NAME "plan."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N=1000002;
int n,b[N],fw[N]={0},zpt[N]={0},zps[N]={0} ;
int lt,ls,lp,lb,lpt,lps,ans=0;
string s,t,p,pt,ps;
int tg,m,u,v,uf=0,vf=0;
int ut=0,vt=0,ft[N]={0};

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

void insert_ft(int x)
{
    while (x<=lp) {++ft[x]; x+=(x&(-x));}
}

int sum_f(int x)
{
    int r=0;
    while (x>0) {r+=fw[x]; x&=(x-1);}
    return r;
}

int sum_t(int x)
{
    int r=0;
    while (x>0) {r+=ft[x]; x&=(x-1);}
    return r;
}

void erase_f(int x,int y)
{
    for (int i=x;i<y;++)
    {
        tg=zps[i];
        if (tg>0) while (tg<=lp) {--fw[tg]; tg+=(tg&(-tg));}
    }
}

void erase_t(int x,int y)
{
    for (int i=x;i<y;++)
    {
        tg=zps[i];
        if (tg>0) while (tg<=lp) {--ft[tg]; tg+=(tg&(-tg));}
    }
}
```

```

void add_f(int x,int y)
{
    for(int i=x;i<y;++i)
    {
        tg=zps[i];
        if(tg>0) insert_fw(tg);
    }
}

void add_t(int x,int y)
{
    for(int i=x;i<y;++i)
    {
        tg=zps[i];
        if(tg>0) insert_ft(tg);
    }
}

void calc_z(string s,int x, int ls, int * z)
{int l,r;
z[x] = 0;
for (int i = x+1, l = 0, r = 0; i<ls; ++i) {
    z[i] = min(z[i - 1], max(0, r - i + 1));
    while (s[z[i]] == s[i + z[i]])
        z[i]++, l = i, r = i + z[i] - 1;
}
}

void calc_1(int x,int y, int q)
{
    int lf,rh,tg;
    if(x>=y) return;
    if(ut==0) add_t(x,y);
    else
    {
        if(vt<=x) {erase_t(ut,vt); add_t(x,y);}
        else {erase_t(ut,x); add_t(vt,y);}
    }
    ut=x;vt=y;
    tg=q-1;if(tg>0)tg= sum_t(tg); else tg=0;
    ans+=sum_t(lp)-tg;
}

void calc_2(int x,int y, int q)
{
    int lf,rh,tg;
    if(x>=y) return;
    if(uf==0) add_f(x,y);
    else
    {
        if(vf<=x) {erase_f(uf,vf); add_f(x,y);}
        else {erase_f(uf,x); add_f(vf,y);}
    }
    uf=x; vf=y;
    tg=q-1;if(tg>0)tg= sum_f(tg); else tg=0;
    ans+=sum_f(lp)-tg;
}

```

```

int main()
{
    fi>>s>>p; ls=s.size(); lp=p.size(); t=s;
    fi>>n;

    for(int i=0;i<=ls;++i)b[i]=i;

    for(int i=0;i<n;++i)
    {
        int tmp;
        fi>>tmp;
        b[tmp]=0;
        tmp-=1+i;
        t.erase(tmp,1);
    }
    lb=0;
    for(int i=0;i<=ls;++i) if(b[i]) b[++lb]=b[i];

    lt=ls-n; lpt=lp+lt+1; lps=lp+ls+1;
    pt=p+'*'+t; b[lt+1]=ls+1;
    for(int i=lt+2;i<N;++i)b[i]=0;

    calc_z(pt,lp,lpt,zpt);

    reverse(p.begin(),p.end()); reverse(s.begin(),s.end());
    ps=p+'*'+s;
    calc_z(ps,lp,lps,zps);

    reverse(zps,zps+lp+ls+2);
    for(int i=ls+1;i<=ls+lp+1;++i) zps[i]=0;

    for(int i=0;i<=lt;++i) zpt[i]=zpt[i+lp];
    for(int i=lt+1;i<=lt+lp+1;++i) zpt[i]=0;

    for(int i=1;i<=ls;++i) if(zps[i]==lp) ++ans;

    for(int i=1;i<=lt;++i)
        if(zpt[i]==lp && b[i+lp-1]-b[i]>=lp) ++ans;

    int tr,d,mp,mt,k,mk,mr;
    for(int i=1;i<=lt;++i)
        if(zpt[i]!=0)
        {
            m=b[i]; mp=m+lp;
            if(mp>ls) break;
            if(b[i+lp-1]-m+1==lp) continue;

            tg=i+zpt[i]-1; mt=b[tg];
            if(mt>ls) --mt;
            if(mt>=mp)
            {
                tg=(ut>i)? ut:i;
                while(b[tg]<mp && tg<lt) ++tg;
                d=lp-min(tg-i,zpt[i]);
                calc_1(mp,mt,d);
            }
        }
}

```

```
d=lp-zpt[i]; mr=i+lp; if(mr>lt)mr=lt+1; mr=b[mr];
if(d>0){mk=mp; mk=max(mk,vt); calc_2(mk,mr,d);}
}
fo<<ans;
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VV24. THỰC ĐƠN

Tên chương trình: MENU.CPP

Công ty “Hải sản Biển Đông” có dịch vụ bán hàng qua mạng. Có n thực đơn chuẩn nấu lẩu khác nhau được đóng gói sẵn, mỗi thực đơn có đúng m thành phần (rau ngổ, mùi tàu, nghêu lưỡi đỏ, tôm châm tráng, mực trứng Phú Quốc, . . .) với số lượng khác nhau. Thành phần thứ j trong thực đơn i có số lượng a_{ij} ($i = 1 \div n$, $j = 1 \div m$).

Để chọn, khách hàng có một dòng m ô trống và chỉ cần ghi số lượng loại thứ j mình cần vào ô j . Thành phần nào số lượng bao nhiêu cũng được thì ghi -1 vào ô tương ứng. Việc đáp ứng yêu cầu của khách hàng sẽ nhanh hơn nếu trong số đã đóng gói có và còn loại mà khách hàng muốn. Ví dụ, với $m = 3$ và yêu cầu là -1, 3, 2, mọi thực đơn đã đóng gói có số lượng thành phần thứ 2 là 3 và thành phần thứ 3 là 2 đều có thể gửi cho khách hàng.

Hiện tại trong giỏ hàng còn q yêu cầu chưa được xử lý. Với mỗi yêu cầu hãy xác định số thực đơn chuẩn phù hợp với yêu cầu đó.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n, m \leq 10^3$),
- ✚ Dòng thứ i trong n dòng tiếp theo chứa m số nguyên dương, mỗi số không vượt quá 10^6 , xác định một thực đơn chuẩn,
- ✚ Dòng $n+2$ chứa số nguyên q ($1 \leq q \leq 100$),
- ✚ Mỗi dòng trong q dòng tiếp theo chứa m số nguyên xác định một yêu cầu, các số có thể là -1 hoặc nguyên dương và không vượt quá 10^6 .

Kết quả: Đưa ra file văn bản MENU.OUT số lượng thực đơn chuẩn phù hợp với yêu cầu của khách hàng, mỗi số trên một dòng.

Ví dụ:

MENU.INP	MENU.OUT
3 8	
6 5 97 99 82 50 95 1	
85 62 11 64 94 84 88 19	
43 99 11 64 94 84 31 19	
3	
-1 -1 11 64 94 84 -1 19	
-1 -1 -1 99 -1 -1 -1 1	
95 -1 -1 -1 -1 80 -1 -1	
	2
	1
	0



VV24 Cr7_20170304 A

Giải thuật: Mảng hai chiều, Duyệt vét cạn.

Với mỗi đơn đặt hàng: kiểm tra tất cả các thực đơn, đếm số lượng phù hợp.

Độ phức tạp của giải thuật: O(q×n×m).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "menu."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N=1000;

int n, m, q,ans;
int a[N][N], b[N];
bool good;

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

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

    fi >> q;
    for (int i=0;i<q;i++)
    {
        for (int j=0;j<m;j++) fi>>b[j];
        ans = 0;
        for (int j=0;j<n;j++)
        {
            good = 1;
            for (int k=0;k<m;k++)
                if (b[k] != -1 && b[k] != a[j][k]) {good = 0; break; }

            ans+=good;
        }
        fo <<ans<<'\
';
    }

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



VV25. ĐẦU TƯ

Tên chương trình: INVEST.CPP

Một dự án nghiên cứu khoa học thu hút được sự chú ý của nhiều nhà đầu tư. Đây là một dự án đầy rủi ro nhưng nếu thành công sẽ mang lại lợi nhuận vô cùng lớn. Sau một thời gian thảo luận các nhà đầu tư quyết định các vốn góp sẽ không chênh lệch nhau quá nhiều, cụ thể là nếu có k nhà đầu tư thì số vốn góp vào là $x, x+1, x+2, \dots, x+k-1$ tỷ đồng.

Tổng cộng dự án đã thu hút được n tỷ đồng. Các nhà đầu tư không muốn lộ diện trong dự án này. Vì vậy giới truyền thông chỉ có thể đoán già đoán non mọi việc. Người ta bình luận nhiều về vốn đầu tư ít nhất và nhiều nhất của những thành viên tham gia.

Hãy xác định các khả năng có thể về vốn đầu tư ít nhất và nhiều nhất.

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

Kết quả: Đưa ra file văn bản INVEST.OUT các khả năng có thể về vốn đầu tư ít nhất và nhiều nhất, mỗi khả năng đưa trên một dòng gồm 2 số nguyên – số vốn góp ít nhất và số vốn góp nhiều nhất của phương án đầu tư có thể.

Ví dụ:

INVEST.INP	INVEST.OUT
27	13 14 8 10 2 7



VV25 Cr7_20170304 B

Giải thuật: Phân tích mô hình toán học, đoán nhận giải thuật.

Nếu có k nhà đầu tư và số vốn nhỏ nhất là x thì

$$x + (x+1) + (x+2) + \dots + (x+k-1) = n$$

$$kx + 1+2+\dots+k-1 = n$$

$$kx + k(k-1)/2 = n$$

Từ đây suy ra $n - kx(k-1)/2$ chia hết cho k .

$$n - kx(k-1)/2 \geq 0 \rightarrow k \leq \sqrt{2n}$$

Về phải của bất đẳng thức có thể là một số thực, vì vậy cần làm tròn lên khi lấy biên để duyệt.

Ván đè còn lại chỉ là duyệt mọi k từ 2 đến $\sqrt{2n}$ và dẫn xuất kết quả với những k phù hợp.

Độ phức tạp của giải thuật: $O(\sqrt{2n})$.

Chương trình:

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

int64_t n,m,k;

int main ()
{
    fi >>n;
    m=(int64_t)(sqrt(2*n)+1);
    for(int64_t i=2;i<=m;++i)
    {
        k=n-(i+1)*i/2;
        if(k%i==0) fo<<k/i+1<<' ' <<k/i+i<<' \n';
    }
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



Để tạo hứng thú trong học tập thầy giáo cho số nguyên n và yêu cầu mỗi bạn trong lớp tạo một xâu độ dài n theo ý riêng của mình từ 3 ký tự a, b, c , trong xâu không nhất thiết phải có mặt đủ cả 3 loại ký tự nêu trên, thậm chí có thể chỉ sử dụng một trong số các ký tự đã nêu. Giải thưởng đặc biệt sẽ dành cho 2 bạn có cặp xâu mà số người đưa ra lời giải đúng trong lớp là ít nhất với dữ liệu vào là 2 xâu đó. Những người có lời giải đúng với chương trình chạy nhanh nhất cũng sẽ được trao giải riêng!

Nội dung bài cần giải là như sau: Gọi \mathbf{A} là xâu thứ nhất, \mathbf{B} là xâu thứ 2 (do các bạn cung cấp). Hãy tìm xâu \mathbf{C} và xâu \mathbf{D} có cùng độ dài với \mathbf{A} và thỏa mãn các tính chất sau:

- Các xâu \mathbf{C} và \mathbf{D} chỉ được chứa các ký tự khác nhau trong \mathbf{A} và phải có đầy đủ các ký tự khác nhau trong \mathbf{A} , nếu \mathbf{A} chỉ chứa một loại ký tự a thì trong trường hợp cần thiết – có thể sử dụng thêm ký tự b , nếu \mathbf{A} chỉ chứa một loại ký tự b – có thể sử dụng thêm ký tự c , nếu \mathbf{A} chỉ chứa một loại ký tự c – có thể sử dụng thêm ký tự a ,
- Gọi $h(\mathbf{X}, \mathbf{Y})$ là khoảng cách Hamming của 2 xâu \mathbf{X} và \mathbf{Y} , cần có $h(\mathbf{C}, \mathbf{B}) = h(\mathbf{D}, \mathbf{B}) = n$,
- \mathbf{C} có thứ tự từ điển nhỏ nhất và $h(\mathbf{A}, \mathbf{C})$ là nhỏ nhất,
- \mathbf{D} là xâu có thứ tự từ điển nhỏ nhất và thỏa mãn thỏa mãn 2 điều kiện đầu.

Với các điều kiện đã cho bài toán luôn có lời giải.

Có thể, bạn cũng muốn thử sức trong cuộc thi này?

Dữ liệu: Vào từ file văn bản STRANSF.INP, dòng thứ nhất chứa xâu \mathbf{A} khác rỗng, dòng thứ 2 chứa xâu \mathbf{B} cùng độ dài và chỉ hứa các ký tự trong tập $\{a, b, c\}$, độ dài xâu không vượt quá 10^5 .

Kết quả: Đưa ra file văn bản STRANSF.OUT các xâu \mathbf{C} và \mathbf{D} tìm được, mỗi xâu trên một dòng.

Ví dụ:

STRANSF.INP
aaabc
abcba

STRANSF.OUT
baaac
baaac



Giải thuật: Phân tích hoạt động ô tô mát.

Chương trình

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

int n,kz,ky,z[3]={0},y[3]={0};
string a,b,c,d;
char t;

void transf_3()
{
    int k;
    for(int i=n-1;i>=0;--i)
        if(b[i]!='c') {c[i]='c';k=i; break;} else c[i]=(b[i]=='a')? 'b':'a';
    for(int i=0;i<k;++i) c[i]=(b[i]=='a')? 'b':'a';
}

void transf_2()
{
    char c0,c1,c2;
    string present="abc";
    int p;
    for(int i=0;i<3;++i) if(z[i]==0) {c0=97+i; break;}
    p=present.find(c0); present.erase(p,1);
    c1=present[0]; c2=present[1];
    for(int i=0;i<n;++i)c[i]=(b[i]==c1)? c2:c1;
}

void transf_1()
{
    char c1,c2;
    string patt="abca";
    int p;
    p=patt.find(a[0]); c1=patt[p];c2=a[0];
    if(c1>c2) swap(c1,c2);
    for(int i=0;i<n;++i)c[i]=(b[i]==c1)? c2:c1;
}

void xly3()
{
    for(int i=n-1;i>=0;--i)
        if(c[i]=='d' && (a[i]!='c')) {c[i]='c'; break;}
    for(int i=0;i<n;++i)
        if(c[i]=='d') c[i]=(a[i]!='a')? c[i]=='a': c[i]=='b';
}

void xly2()
{
    char c0,c1,c2;
    string present="abc";
    int p;
    for(int i=0;i<3;++i) if(z[i]==0) {c0=97+i; break;}
```

```

p=present.find(c0); present.erase(p,1);
c1=present[0]; c2=present[1];
for(int i=0;i<n;++i)
    if(c[i]=='d') c[i]=(a[i]==c1)? c2:c1;
}

void xly1()
{
    char c1;
    string patt="abca";
    int p;
    p=patt.find(a[0]); c1=patt[p];
    for(int i=0;i<n;++i) if(c[i]=='d') c[i]=c1;
}

int main ()
{
    fi>>a>>b;
    c=a; n=a.size();
    for(int i=0;i<n;++i)
    {
        t=a[i]; z[t-97]=1;
        if(t==b[i]) c[i]='d'; else y[t-97]=1;
    }
    kz=z[0]+z[1]+z[2]; ky=y[0]+y[1]+y[2];
    if(kz==1)xly1(); else {if(kz==2)xly2();else xly3();}
    fo<<c<<'\n';

    if(kz==1)transf_1(); else {if(kz==2)transf_2();else transf_3();}
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}

```



VV27. CHỈ SỐ ĐÁNH GIÁ

Tên chương trình: S_INDEX.CPP

Làm thế nào để đánh giá sự thành công của một nhà khoa học? Dựa vào số bài báo được công bố hay dựa vào số lần một bài báo được trích dẫn tới ở công trình của những người khác? Cả hai tham số đó đều quan trọng.

Nói một bài báo có điểm số trích dẫn là c nếu nó được trích dẫn tới c lần trong các công trình của những nhà khoa học khác. Một trong số các cách đánh giá sự thành công của một nhà khoa học là tính chỉ số thành công s_index dựa trên sự kết hợp giữa số lượng bài báo và chỉ số trích dẫn của các bài báo đó.

Chỉ số s_index của một nhà khoa học bằng k lớn nhất nếu người đó có k bài báo, mỗi bài có điểm số trích dẫn không nhỏ hơn k . Ví dụ, một người có 10 bài báo, mỗi bài báo được trích dẫn không dưới 10 lần thì s_index của người đó ít nhất là bằng 10.

Một người có n bài báo, bài báo thứ i có điểm trích dẫn là c_i , $i = 1 \div n$. Hãy xác định s_index của người đó.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 5 \times 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên c_1, c_2, \dots, c_n ($0 \leq c_i \leq 10^6$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản S_INDEX.OUT một số nguyên – s_index tìm được.

Ví dụ:

S_INDEX.INP
5
8 5 3 4 10

S_INDEX.OUT
4



VV27 Cr_20170204 A

Giải thuật: Kỹ năng phân tích và đoán nhận giải thuật.

Để đánh giá cần dựa trên các bài báo có nhiều trích dẫn nhất,
Như vậy cần sắp xếp các chỉ số trích dẫn c_i theo trình tự giảm dần,
Đánh số các phần tử trong dãy bắt đầu từ 1,
Nếu $c_k > k$ thì s_index sẽ không nhỏ hơn k ,
Kết quả cần tìm sẽ là k lớn nhất thỏa mãn điều kiện $c_k \leq k$.

Lưu ý:

- Có thể sắp xếp c_i theo thứ tự tăng dần, sau đó xử lý từ cuối về đầu và thay đổi tương ứng điều kiện kiểm tra,
- Có thể sắp xếp c_i theo thứ tự tăng dần, sau đó dùng hàm **revers** để nhận được dãy sắp xếp giảm dần,
- Cần có giá trị hàng rào đủ lớn ở cuối dãy để các dữ liệu đã cho nằm bắt đầu từ vị trí 1 trở đi.

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

Chương trình I:

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

int main()
{
    fi>>n;
    vector<int>c(n+1);
    for(int i=0;i<n;++i) fi>>c[i]; c[n]=1000001;
    sort(c.begin(),c.end(),greater<int>());
    ans=n;
    for(int i=1;i<=n;++i)
        if(c[i]<i) {ans=i-1; break;}
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```

Chương trình II:

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

int main()
{
    fi>>n;
    vector<int>c(n);
    for(int i=0;i<n;++i) fi>>c[i];
    sort(c.begin(),c.end());
    ans=n;
    for(int i=1;i<=n;++i)
        if(c[n-i]<i) {ans=i-1; break;}
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Trước khi có điện thoại việc truyền tin đi xa được thực hiện bằng chim bồ câu đưa thư hoặc hệ thống các trạm chuyển tiếp. Thông tin được truyền từ một trạm tới trạm kế cận bằng người đưa tin cưỡi ngựa, bằng cách đốt lửa tạo khói bốc cao hoặc mã hóa và truyền đi bằng tù và, trống. Ở thế kỷ XVI một chiếc tàu của Tây Ban Nha bị đánh ở vùng biển phía đông châu lục. Chỉ hai hôm sau những thợ dân sống ở bờ tây của lục địa đã biết tin này qua hệ thống truyền tin bằng trống và tù và của họ!

Trong hệ thống phòng thủ của một vương quốc có n trạm truyền tin bằng tù và từ biên giới tới thủ đô được thành lập, các trạm đặt cách đều nhau, trạm số 1 ở biên giới và trạm thứ n – ở thủ đô. Khi một trạm có tin báo thì các trạm có khoảng cách không quá d cũng nghe thấy, tức là khi trạm i thôi tù và thì các trạm j nghe thấy nếu $|i-j| \leq d$ và truyền tiếp tin nếu cần. Sau một thời gian dài tồn tại và ít hoạt động một số trạm bị bỏ trống, không có người trông coi. Riêng trạm ở biên giới và ở thủ đô vẫn hoạt động. Quốc vương mới lên ngôi và rất quan tâm đến giao thông, liên lạc. Ông quyết định khôi phục lại sự thông suốt của việc truyền tin từ biên giới về thủ đô bằng cách khôi phục lại một số ít nhất các trạm đang bị bỏ trống.

Cho biết trạng thái s_i của trạm i (bằng 1 nếu đang hoạt động và bằng 0 – bị bỏ trống), $i = 1 \dots n$. Hãy xác định số lượng ít nhất các trạm cần khôi phục.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và d ($1 \leq n \leq 3 \times 10^5$, $1 \leq d \leq n$),
- ✚ Dòng thứ 2 chứa n số nguyên s_1, s_2, \dots, s_n – trạng thái của các trạm.

Kết quả: Đưa ra file văn bản TRUMPET.OUT một số nguyên – số lượng ít nhất các trạm cần khôi phục.

Ví dụ:

TRUMPET.INP	TRUMPET.OUT
<pre>8 2 1 1 0 0 1 0 0 1</pre>	<pre>2</pre>



Giải thuật: Kỹ năng phân tích và đoán nhận giải thuật, kỹ thuật 2 con trỏ.

Gọi **p** là trạm cuối cùng còn hoạt động mà thông tin có thể truyền từ trạm **i** tới,
Ban đầu **p = 1**,

Xét đoạn **p+1, p+2, ..., q = p+d**,

Nếu tồn tại **i ∈ [p+1, q]** và **s_i = 1** thì cho **p = i**, tính **q** mới tương ứng,

Nếu không tồn tại **i** thỏa mãn các điều kiện nêu trên – bổ sung trạm mới vào vị trí **p+d**, gán **p+d** cho **p**, tính **q** mới và tiếp tục xử lý,

Quá trình xử lý kết thúc khi **n ∈ [p+1, q]**.

Lưu ý:

- Không cần lưu trữ tường minh **q**,
- Có thể kết thúc xử lý sớm hơn khi **n-p ≤ d**.

Độ phức tạp của giải thuật: O(n).

Chương trình

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

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

    for(int i=0;i<n-d+1;++i)
        if(s[i]) p=i; else if(i-p==d) ++ans, p=i;

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



Trong các cuộc thi tin học đồng đội một người trong đội luôn có nhiệm vụ tạo tests và kiểm tra chương trình.

Một bài toán cần giải đòi hỏi phải làm việc với cây nhị phân có 2^n nút lá, nút khác lá được gọi là nút trong. Giá trị ở nút trong là max giá trị 2 nút con của nó. Các nút của cây được chia thành n mức. Mức 0 chứa nút gốc, các nút con của gốc có mức 1, nút con của các nút mức i thuộc mức $i+1$, $i = 0, 1, \dots, n-2$. Như vậy các nút lá thuộc mức $n-1$.

Steve có nhiệm vụ kiểm tra chương trình với bộ dữ liệu a_1, a_2, \dots, a_m , trong đó $m = 2^n$. Các giá trị này sẽ được ghi vào nút lá của cây, với mỗi i có một nút lá chứa giá trị a_i , $i = 1 \div m$ và không có nút lá nào trống.

Điều mà Steve quan tâm hiện nay là với những cách điền nút lá khác nhau mỗi số a_i ($i = 1 \div m$) có thể đạt tới mức có số nhỏ nhất là bao nhiêu?

Hãy xác định mức có số nhỏ nhất có thể đạt được với mỗi số trong bộ dữ liệu đã cho.

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

- + Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 20$),
- + Dòng thứ 2 chứa $m = 2^n$ số nguyên a_1, a_2, \dots, a_m ($1 \leq a_i \leq 10^9$, $i = 1 \div m$).

Kết quả: Đưa ra file văn bản LEVELS.OUT trên một dòng m số nguyên xác định các mức có thể đạt được của mỗi số trong bộ dữ liệu đã cho.

Ví dụ:

LEVELS.INP	LEVELS.OUT
2	2 0 1 1
1 4 3 2	



Giải thuật: Cây nhị phân, xử lý bít.

Giá trị lớn nhất trong bộ dữ liệu sẽ đạt tới mức 0 không phụ thuộc và cách bố trí dữ liệu ở các nút lá,

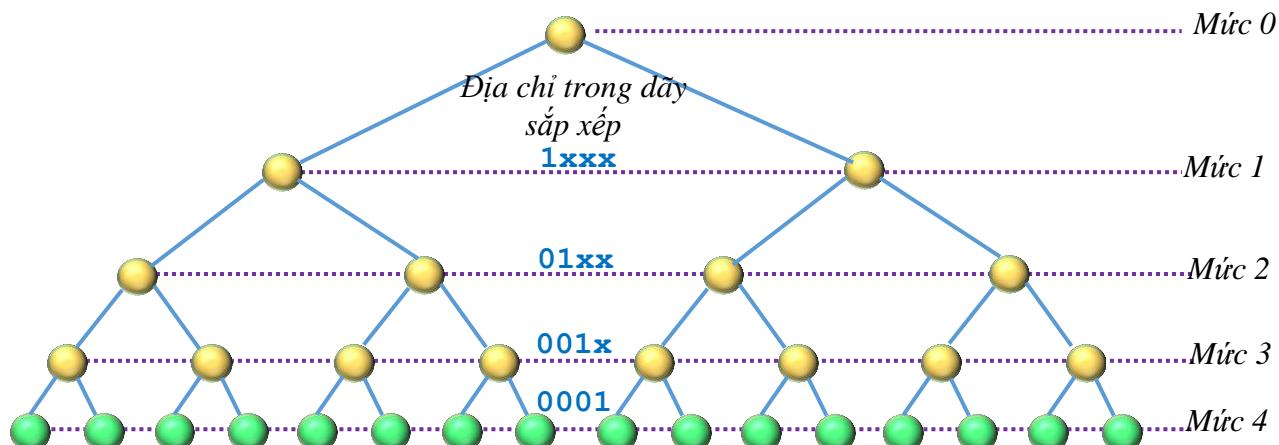
Các phần tử có *giá trị giống nhau* sẽ có *cùng mức nhỏ nhất* có thể đạt được,

Nút gốc chia cây thành 2 cây con, mỗi cây con có 2^{n-1} nút,

Một phần tử muốn chiếm được vị trí gốc cây con này (mức 1) thì phải có không ít hơn $2^{n-1}-1$ phần tử nhỏ hơn hoặc bằng nó, nói một cách khác, trong dãy được sắp xếp theo giá trị tăng dần các phần tử của bộ dữ liệu ban đầu, địa chỉ của phần tử gốc cây con phải có bít thứ $n-1$ bằng **1**,

Tương tự như vậy một phần tử muốn chiếm được vị trí mức 2 phải có địa chỉ trong dãy đã sắp xếp có bít thứ $n-2$ bằng **1**,

Tổng quát hóa, ta có một phần tử muốn chiếm được vị trí mức **i** phải có địa chỉ trong dãy đã sắp xếp có bít thứ $n-i$ bằng **1**,



Như vậy toàn bộ quá trình xác định mức được dựa trên việc xử lý địa chỉ của các phần tử có giá trị khác nhau.

Tổ chức dữ liệu:

- Mảng `vector<pair<int, int>>` a (m) – lưu các phần tử và số thứ tự của nó trong bộ dữ liệu ban đầu,
- Mảng `vector<int>` ans (m, n) – lưu kết quả (với giá trị khởi tạo là n).

Độ phức tạp của giải thuật: O(nlnn).

Chương trình

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

int main()
{
    fi>>n; m=1<<n;
    vector<int>ans (m,n);
    vector<pair<int,int>>a (m);

    for(int i=0;i<m;++i) {fi>>t;a[i]={t,i};}
    sort(a.begin(),a.end());
    k=m-1; lv=0; t=a[k].first;
    while(1)
    {
        while(k>=0 && a[k].first==t) ans[a[k--].second]=lv;
        if(k<=0) break;
        t=a[k].first;p=k+1;
        for(int i=n-1;i>=0;--i)
            if((l<<i)&p) {lv=n-i;break;}
    }
    for(int i=0;i<m;++i) fo<<ans[i]<<' ';
    fo<<"\nTime: "<<clock () / (double) 1000<<" sec";
}
```

Xử lý các phần tử
giống nhau

Xử lý phần tử mới



VV30. BẢNG SỐ

Tên chương trình: SHEETS.CPP

Xét bảng số kích thước $n \times n$ ô, trong đó $n = 2^k$, mỗi ô của bảng chứa một số nguyên trong phạm vi từ 1 đến n , các số khác nhau từng đôi một. Các hàng đánh số từ 1 từ trên xuống dưới, các cột đánh số từ 1 từ trái sang phải.

Người ta gấp bảng đè cho mép dưới trùng lên mép trên của bảng, nửa dưới nằm đè lên nửa trên, sau đó gấp nửa bên phải đè lên nửa bên trái. Sau k lần thao tác như vậy ta được cột các ô độ cao n . Các ô của cột được đánh số từ 1, từ dưới lên trên, ô ở vị trí p chứa số x .

Yêu cầu xử lý các truy vấn với các bảng khác nhau, mỗi truy vấn có một trong 2 dạng:

- 1 k x – Truy vấn loại 1: tìm vị trí p chứa số x với bảng kích thước $n = 2^k$,
- 2 k p – Truy vấn loại 2: xác định số x ghi ở ô p với bảng kích thước $n = 2^k$.

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

- Đòng đầu tiên chứa một số nguyên q – số lượng truy vấn ($1 \leq q \leq 10^4$),
- Mỗi dòng trong q dòng sau chứa một truy vấn với $1 \leq k \leq 31$, x và p đã cho đảm bảo có lời giải.

Kết quả: Đưa ra file văn bản SHEETS.OUT, kết quả mỗi truy vấn đưa ra trên một dòng dưới dạng số nguyên.

Ví dụ:

SHEETS.INP
3
1 1 4
2 2 14
1 2 16

SHEETS.OUT
3
15
3



Giải thuật: Phân tích hoạt động và mô phỏng ô tô mát.

Hai truy vấn có tính chất **thuận - nghịch**, vì vậy có thể dùng kết quả truy vấn loại 1 để kiểm tra kết quả xử lý truy vấn loại 2 và ngược lại,

Mỗi số **v** trong phạm vi từ 1 đến $n = 2^k$ tương ứng với ô tọa độ (**row**, **col**),

Giá trị **v** có thể xem như địa chỉ tuyến tính của các ô trong bảng, (**row**, **col**) – địa chỉ vật lý của ô,

Để thuận tiện xử lý nên đánh số địa chỉ tuyệt đối bắt đầu từ 0, hàng và cột của bảng cũng đánh số bắt đầu từ 0,

Các phép xử lý đều là biến đổi nhị phân: giá trị các tham số giảm một nửa hoặc tăng gấp đôi sau mỗi bước biến đổi, vì vậy cần chuẩn bị các tham số $b_i = 2^i$, $i = 0 \div 31$, $a_i = 2^{i+1} - 1$, $i = 0 \div 30$;

Xử lý truy vấn loại 1:

Đánh số địa chỉ tuyệt đối từ 0: **--v**;

Tính địa chỉ vật lý tương ứng:

- $n = 2^k = b[k]$,
- $\text{row} = v/n$, $\text{col} = (v-1) \% n$;

Mỗi phép biến đổi gồm 2 bước:

- ✚ Bước 1: gấp mép dưới đè lên trên mép trên,
- ✚ Bước 2: gấp mép phải đè lên trên mép trái.

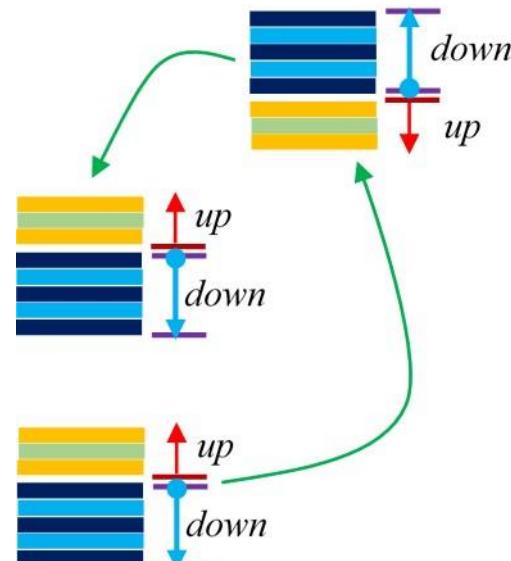


Mỗi ô (**i**, **j**) tương ứng với 2 giá trị nguyên: **down** – số lớp **kết từ nó** xuống đáy, **up** – số lớp ở trên nó.

Ở bước 1, khi gấp từ dưới lên trên, nếu ô (**i**, **j**) có $i < n/2$, vị trí của nó không thay đổi, số lớp bên trên sẽ tăng thêm một lượng bằng **up+down**, tham số **down** không thay đổi,

Nếu $i \geq n/2$ ô (**i**, **j**) sẽ chuyển vào vị trí ô (**i'**, **j**), trong đó **i'** có các bit nghịch đảo nhị phân với các bit của **i**. Giá trị **up** mới sẽ bằng **down-1**, giá trị **down** mới bằng $2 * up + down + 1$ của các giá trị **up** và **down** cũ.

Số dòng của bảng giảm một nữa.



Tương tự như vậy ở bước 2 với các cột. Số cột sẽ giảm một nữa.

Từ (**row**, **col**) ban đầu việc biến đổi được thực hiện cho đến khi **row=col=0**. Kết quả cần tìm sẽ là giá trị **down**.

Xử lý truy vấn loại 2:

Đặt cột vào vị trí $(0, 0)$,

Mỗi phép biến đổi gồm 2 bước:

- + Bước 1: Chia đôi các cột, xét bảng có **số cột gáp đôi**,
- + Bước 2: Chia đôi các cột, xét bảng có **số dòng gáp đôi**.

Tham số quản lý ở mỗi ô:

- + **down** – số của lớp *ở đáy cột*,
- + **up** – số của lớp *trên cùng* của cột.

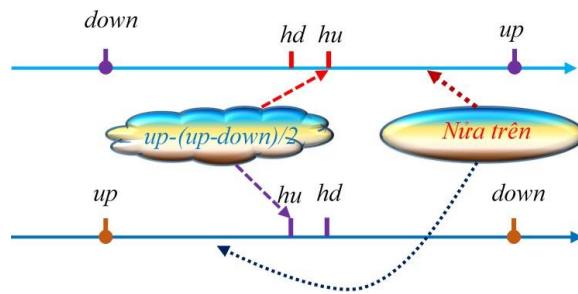
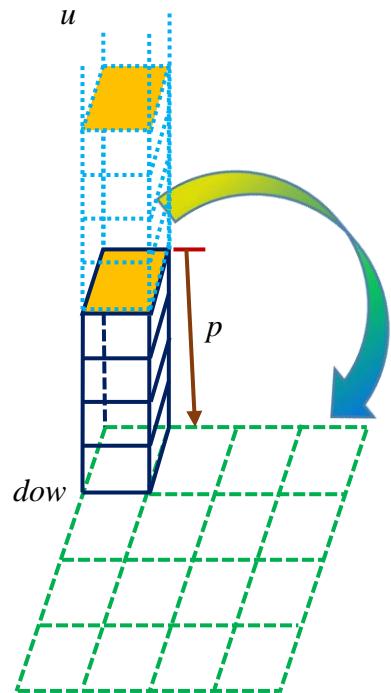
Giá trị xuất phát:

down=0, **up**=n-1; --**p**;

row=0, **col**=0;

Khi chia đôi cột, nếu **p** ở nửa dưới thì giữ nguyên **row** và **col**, chỉnh lý lại các biên, nếu **p** ở nửa trên – chỉnh lý **row** và **col** (chỉ sang ô mới), cập nhật các biên **down** và **up**.

Chia đôi cột: Phân biệt 2 trường hợp: **up > down** và **up < down**.



Kiểm tra **p** thuộc nửa dưới: $(p-v) * (p-hu) > 0$.

Các phép cập nhật **col**, **row**, **up** và **down** – theo phương thức tương tự như khi xử lý truy vấn loại 1.

Điều kiện dừng: **p==down**. Từ tọa độ ô tìm được dễ dàng tính giá trị chưa trong ô.

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "sheets."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int32_t m,k,v,q,ans,b[32],a[32];

void calc_1()
{
    int32_t row,col,up,down,tup,n;
    n=b[k];
    row=(v-1)/n; col=v%n; if(col==0) col=n; --col;
    up=0; down=1;
    for(int i=k-1;i>=0;--i)
    {
        if(row&b[i]) tup=down-1, down+=2*up+1, up=tup, row^=a[i];
        else up=2*up+down;
        if(col&b[i]) tup=down-1, down+=2*up+1, up=tup, col^=a[i];
        else up=2*up+down;
        if(row==0 && col==0) break;
    }
    fo<<down<<'\
';
}

void calc_2()
{
    int32_t row=0,col=0,up,down,hd,hu,n;
    down=0; up=a[k+1]; --v; n=b[k];
    for(int i=0;i<k;++i)
    {
        if(down==v) break;
        hu=up-(up-down)/2; hd=(up>down)? hu-1:hu+1;
        if((v-up)*(v-hu)>0) up=hd;
        else { down=up; up=hu; col^=a[i]; }
        hu=up-(up-down)/2; hd=(up>down)? hu-1:hu+1;
        if((v-up)*(v-hu)>0) up=hd;
        else { down=up; up=hu; row^=a[i]; }
    }
    hd=row*n+col+1; fo<<hd<<endl;
}

int main()
{
    fi>>q;
    for(int i=0;i<32;++i)b[i]=1<<i; a[0]=1;
    for(int i=1;i<31;++i)a[i]=b[i]|a[i-1];
    for(int i=0;i<q;++i)
    {
        fi>>m>>k>>v;
        if(m==1) calc_1(); else calc_2();
    }

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



VV31. KHOẢNG CÁCH HOÀN THIỆN

Tên chương trình: P_DIST.CPP

Một số nguyên dương (số tự nhiên) được gọi là hoàn thiện nếu tổng các ước khác chính nó bằng số đó. Ví dụ, 28 là một số hoàn thiện vì $28 = 1 + 2 + 4 + 7 + 14$.

Khoảng cách hoàn thiện của một số tự nhiên x là giá trị tuyệt đối của hiệu số đó với tổng các ước khác x và ký hiệu là $f(x)$.

Ví dụ, với $x = 12$ ta có $f(12) = |12 - (1 + 2 + 3 + 4 + 6)| = 4$.

Cho 2 số tự nhiên a và b ($a \leq b$). Hãy tính tổng $f(x)$ với mọi $x \in [a, b]$.

Dữ liệu: Vào từ file văn bản P_DIST.INP gồm một dòng chứa 2 số nguyên a và b ($1 \leq a \leq b \leq 10^7$).

Kết quả: Đưa ra file văn bản P_DIST.OUT một số nguyên – tổng tìm được.

Ví dụ:

P_DIST.INP	P_DIST.OUT
5 15	54



VV31 Cr6_20170204 4

Giải thuật: Bảng phương án.

Lập bảng t tích lũy tổng các ước: t_x – tổng các ước của x với $1 \leq x \leq b$,

Việc tích lũy được tiến hành theo ước số: với mỗi số d , các số $2d, 3d, \dots, kd \leq b$ đều lớn hơn d và có ước là d , ước này được tích lũy vào $t_{2d}, t_{3d}, \dots, t_{kd}$.

Các ước cần xét: $d = 1, 2, 3, \dots, b/2$.

Việc tích lũy tổng ước số sẽ có độ phức tạp là:

$$b + b/2 + b/3 + \dots = b \times \left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots\right) \approx b \ln b$$

Kết quả cần tìm: $\text{ans} = \sum_{i=a}^b |t_i - i|$.

Chương trình

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

int main()
{
    fi >> a >> b;
    vector<int> t(b+1);
    for (int i=1; i<=b/2; ++i)
    {
        d=i+i;
        while (d<=b) t[d] += i, d+=i;
    }
    for (int i=a; i<=b; ++i)
        ans += abs(t[i] - i);
    fo << ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VV32. ÁO KHOÁC

Tên chương trình: JACKETS.CPP

Đang thời kỳ chuyển mùa, nhiệt độ trong ngày thay đổi rất nhiều, ra ngoài đường ai cũng phải mặc áo khoác và Hermione không phải là ngoại lệ. Nếu tin theo dự báo thời tiết trên TV thì giai đoạn chuyển mùa này sẽ kéo dài n ngày, ngày thứ i có nhiệt độ trung bình là t_i , $i = 1 \dots n$.

Trong tủ của Hermione có m áo khoác, áo thứ j thích hợp mặc khi nhiệt độ của ngày nằm trong phạm vi từ u_j đến v_j , $j = 1 \dots m$. Cũng như các cô gái trẻ khác cùng thời, Hermione không bao giờ mặc cùng một áo trong 2 ngày liên tiếp. Ra đường thì phải mặc áo khoác, nhưng dĩ nhiên Hermione chưa chập mạch tới mức khoác một lúc 2 hay nhiều áo.

Hãy xác định áo mỗi ngày cần mặc trong suốt giai đoạn chuyển mùa n ngày.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n, m \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên t_1, t_2, \dots, t_n ($0 \leq t_i \leq 10^9$, $i = 1 \dots n$),
- ✚ Dòng thứ j trong m dòng sau chứa 2 số nguyên u_j và v_j ($0 \leq u_j \leq v_j \leq 109$).

Kết quả: Đưa ra file văn bản JACKETS.OUT thông báo “**Yes**” hoặc “**No**” nếu có thể xác định được lịch mặc áo khoác trong suốt khoảng thời gian đã nêu hoặc không có cách lên lịch phù hợp. Trong trường hợp có thể xác định được lịch thì ở dòng thứ hai đưa ra n số nguyên, số thứ i cho biết áo cần mặc trong ngày i ($i = 1 \dots n$). Trong trường hợp có nhiều lịch phù hợp thì đưa ra lịch tùy chọn.

Ví dụ:

JACKETS.INP
4 4
25 25 30 50
10 40
20 30
70 100
50 50

JACKETS.OUT
Yes
2 1 2 4



VV31 Mos_op_20170310 4

Giải thuật: Quy hoạch động.

Để lên được lịch trước hết cần xác định với mỗi ngày những áo có thể mặc, Mỗi ngày có thể có rất nhiều áo phù hợp, nhưng để lên lịch ta chỉ cần **ghi nhận 3** trong số đó và gọi là các **áo được chọn**,

Gọi **fits_{i,j}** là các áo được chọn cho ngày thứ **i**,

Lần lượt xét các ngày đã cho, ở ngày **i** cần tìm **j ∈ fits_i** cho phép sử dụng để lập lịch đến ngày **i**, biến **dp_{i,j}** chứa mốc nối tới áo sử dụng ngày **i-1**, giá trị **-1** chỉ mốc nối rỗng,

Giá trị xuất phát: **dp_{0,j}=0** ∀ **j ∈ fits₀**,

Công thức lặp:

$$dp_{i,j} = \begin{cases} k \in fits_{i-1} \text{ nếu } dp_{i-1,k} \neq -1 \text{ và } fits_{i-1,k} \neq fits_{i,j} \\ -1 \text{ trong trường hợp ngược lại.} \end{cases}$$

Kết quả:

- ✚ Không tồn tại lịch nếu **dp_{n-1,j} = -1** ∀ **j**,
- ✚ Trong trường hợp ngược lại: tồn tại lịch phù hợp, dựa theo các mốc nối và **fits_i** (**i = n-1 ÷ 0**) xác định áo cho mỗi ngày.

Xác định các tập áo được chọn:

Mỗi nhiệt độ đã cho được lưu lại dưới dạng bản ghi là nhóm 3 đại lượng:

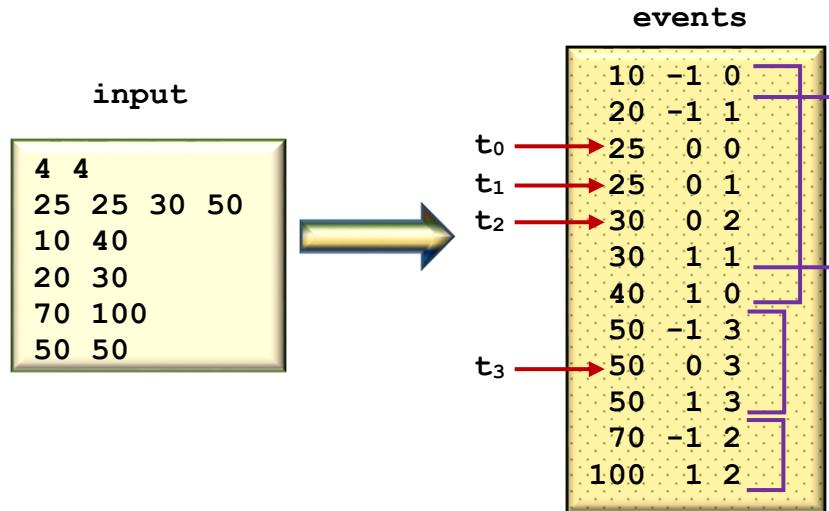
t_i → (**t_i**, 0, **i**),

u_j → (**u_j**, -1, **j**),

v_j → (**v_j**, 1, **j**).

Lưu các nhóm 3 này vào trong mảng sự kiện **events** và sắp xếp theo thứ tự tăng dần. Cặp bản ghi tương ứng với **u_j** và **v_j** xác định khoảng chứa các **t_i**, cho biết áo **j** phù hợp với ngày **i**.

Các cặp **u_j, v_j** được quản lý theo nguyên tắc quản lý cặp ngoặc khi duyệt biểu thức ngoặc.

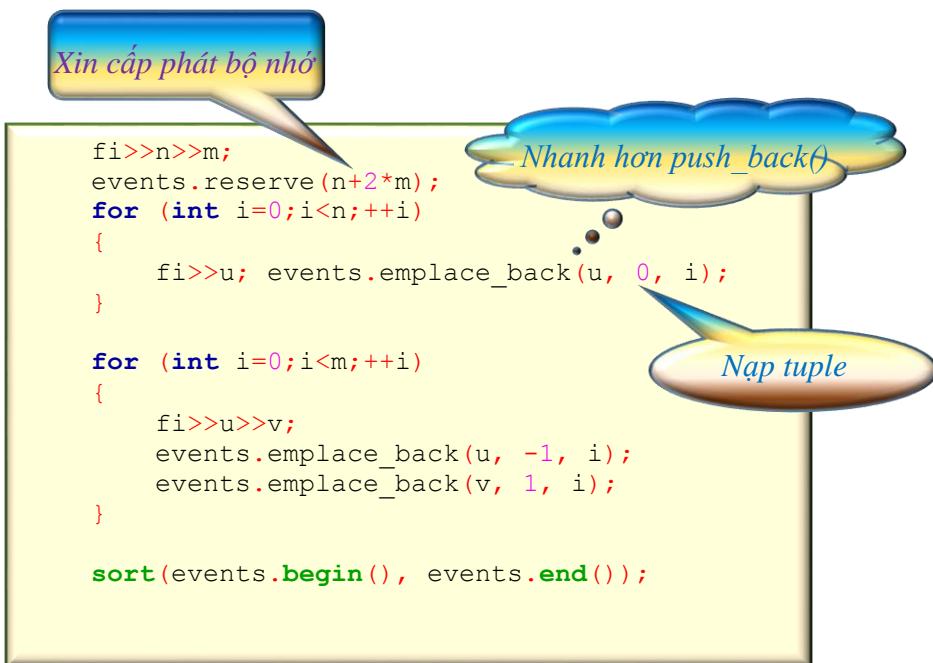


Tổ chức dữ liệu:

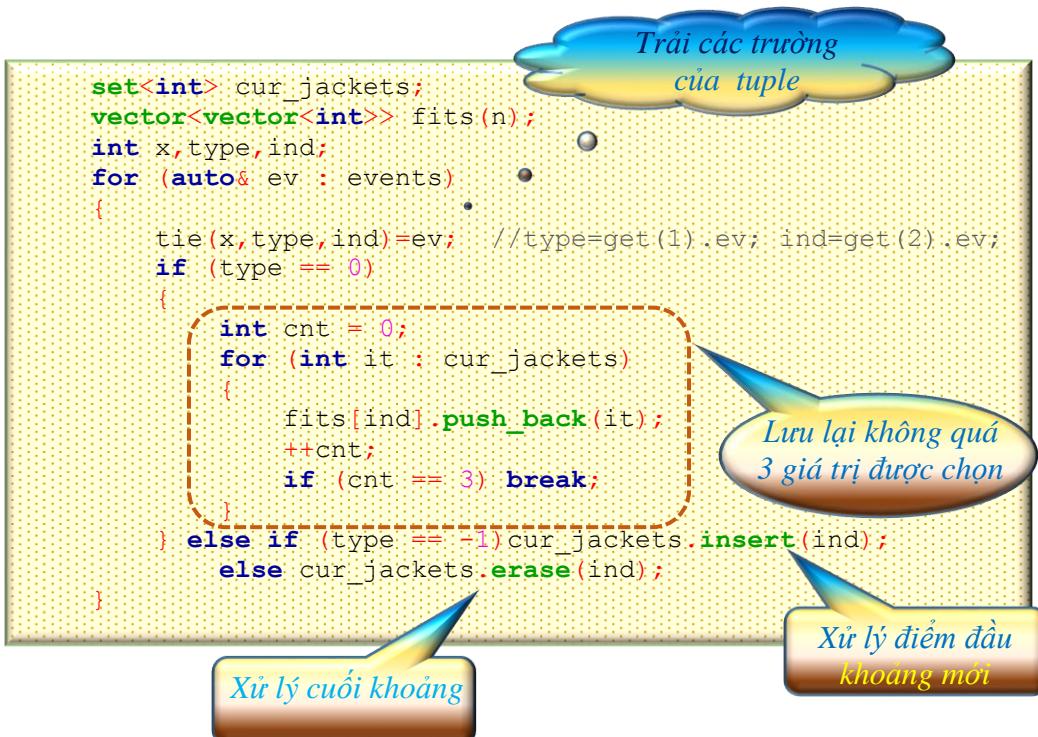
- Dữ liệu nhóm 3 được lưu trữ theo kiểu `tuple<int, int, int>`,
- Mảng `vector<ti3> events` – lưu các bản ghi nhóm 3 đã nêu ở trên,
- Tập `set<int> cur_jackets` – lưu các áo đang xét,
- Mảng hai chiều `vector<vector<int>> fits(n)` – lưu các áo được chọn,
- Mảng `vector<int> ans(n)` – lưu kết quả về lịch cần tìm,
- Mảng hai chiều `vector<vector<int>> dp(n)` – phục vụ tích lũy thông tin theo sơ đồ quy hoạch động.

Xử lý:

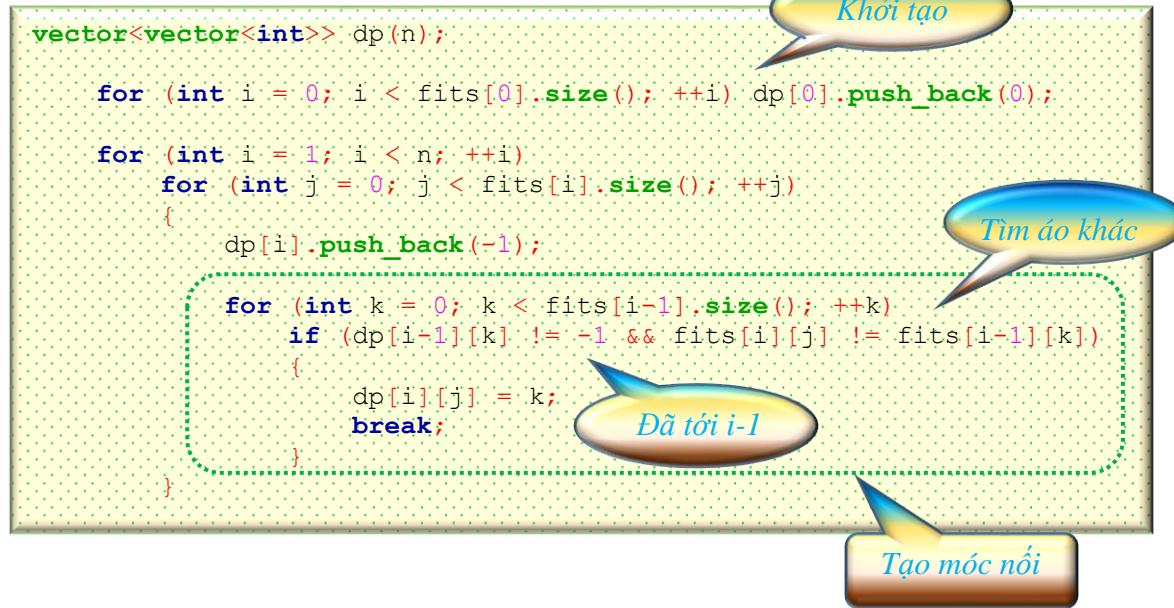
Tạo mảng quản lý các sự kiện:



Xác định tập được chọn:



Xây dựng lịch:



Chương trình

```
#include <bits/stdc++.h>
#define NAME "jackets."
#define times fo<<"\nTime: "<<clock()/(double)1000<<" sec";
using namespace std;
typedef tuple<int,int,int> ti3;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
vector<ti3> events;

int main()
{
    int n, m, u, v;
    fi>>n>>m;
    events.reserve(n+2*m);

    for (int i=0;i<n;++i)
    {
        fi>>u; events.emplace_back(u, 0, i);
    }

    for (int i=0;i<m;++i)
    {
        fi>>u>>v;
        events.emplace_back(u, -1, i);
        events.emplace_back(v, 1, i);
    }

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

    set<int> cur_jackets;
    vector<vector<int>> fits(n);
    int x,type,ind;
    for (auto& ev : events)
    {
        tie(x,type,ind)=ev;
        if (type == 0)
        {
            int cnt = 0;
            for (int it : cur_jackets)
            {
                fits[ind].push_back(it);
                ++cnt;
                if (cnt == 3) break;
            }
        } else if (type == -1) cur_jackets.insert(ind);
        else cur_jackets.erase(ind);
    }

    vector<vector<int>> dp(n);

    for (int i = 0; i < fits[0].size(); ++i) dp[0].push_back(0);

    for (int i = 1; i < n; ++i)
        for (int j = 0; j < fits[i].size(); ++j)
```

```

{
    dp[i].push_back(-1);
    for (int k = 0; k < fits[i-1].size(); ++k)
        if (dp[i-1][k] != -1 && fits[i][j] != fits[i-1][k])
    {
        dp[i][j] = k;
        break;
    }
}

for (int i = 0; i < dp[n-1].size(); ++i)
{
    if (dp[n-1][i] == -1) continue;
    fo << "Yes\n";
    vector<int> ans(n);
    int cur = i;
    for (int pos = n-1; pos >= 0; --pos)
    {
        ans[pos] = fits[pos][cur];
        cur = dp[pos][cur];
    }
    for (int it : ans) fo << it+1 << ' ';
    fo << '\n'; times;
    return 0;
}
fo << "No"; times;
}

```



Chương trình tìm kiếm tài năng Tin học đưa ra trên mạng một lúc n bài toán, dựa vào kết quả nhận được để đánh giá những người dự thi. Số người dự thi có thể rất lớn nên việc nhận dạng bài được gửi tới cần phải thực hiện cực nhanh. Tên bài thứ i có a_i ký tự, $i = 1 \div n$. Để tạo sự hưng phấn và tâm lý vui vẻ thoải mái cho người thi Ban tổ chức yêu cầu tên các bài được công bố dưới dạng chuỗi biểu tượng smiley có độ dài (số biểu tượng) tương ứng với số ký tự trong tên ban đầu của bài toán. Ngoài ra, nếu đánh số các biểu tượng một cách thích hợp thì tên các bài dưới dạng biểu tượng phải tạo thành danh sách có thứ tự từ điển tăng dần. Lưu ý là không được thay đổi trình tự tên bài trong danh sách đã nêu của Ban tổ chức.

Việc nghĩ ra và vẽ các biểu tượng phù hợp không phải là chuyện đơn giản, vì vậy nhóm họa sỹ làm nhiệm vụ này quyết định giảm đến mức nhỏ nhất số các biểu tượng khác nhau cần sử dụng.

Hãy xác định số lượng nhỏ nhất các biểu tượng khác nhau cần thiết kê.

Dữ liệu: Vào từ file văn bản SMILEY.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 SMILEY.OUT một số nguyên – số lượng nhỏ nhất tìm được.

Ví dụ:

SMILEY.INP
5
2 2 2 2 2

SMILEY.OUT
3



VV33 Mos_op_20170310 5

Giải thuật: Thứ tự từ điển, Tìm kiếm nhị phân.

Nếu độ dài các tên theo trình tự đã cho tạo thành **dãy số tăng dần** thì chỉ cần **một** biểu tượng là đủ để có dãy tên theo thứ tự từ điển tăng dần, trong trường hợp ngược lại – phải dùng ít nhất 2 biểu tượng,

Có bao nhiêu tên có độ dài 1 thì **ít nhất phải có** bấy nhiêu biểu tượng khác nhau (điều kiện cần),

Nếu dãy số độ dài đã cho **không bắt đầu bằng 1** thì phải có **thêm ít nhất một biểu tượng** phục vụ biểu diễn phần đầu,

Ta thấy $2^{16} < 10^5 < 2^{17}$, vì vậy trong trường hợp phải dùng nhiều hơn một biểu tượng các tên có độ dài lớn hơn 17 sẽ không kéo theo việc phải tạo biểu tượng mới. Nhận xét này sẽ làm đơn giản hóa đáng kể quá trình xử lý.

Nếu không thể dùng một biểu tượng để biểu diễn thì cần:

- + Xác định sơ bộ số lượng tối thiểu các biểu tượng phải có theo điều kiện cần,
- + Nếu số lượng xác định được lớn hơn hoặc bằng 317 (cho trường hợp xấu nhất, các độ dài đều không vượt quá 2) thì số lượng này cũng thỏa mãn điều kiện đủ để tạo dãy tăng dần theo thứ tự từ điển,
- + Chuẩn hóa dữ liệu bằng cách loại bỏ các giá trị lớn hơn 17, giữ nguyên trình tự các số còn lại,
- + Kiểm tra số lượng biểu tượng hiện có đã đủ để xây dựng dãy tên theo thứ tự từ điển tăng dần hay chưa, nếu chưa đủ thì mở rộng số lượng một cách thích hợp.

Gọi số lượng biểu tượng tối thiểu cần thiết là **ans** và các biểu tượng tương ứng là **b₁, b₂, …, b_{ans}**. Mỗi biểu tượng có thể coi như một ký tự.

Tên đầu tiên sẽ có dạng

$$s_1 = \underbrace{b_1 b_1 \dots b_1}_{a_1} = (b_1, a_1)$$

Tên tiếp theo sẽ là xâu có thứ tự từ điển nhỏ nhất lớn hơn xâu trước nó. Mỗi tên sẽ tương ứng với một nhóm các cặp (**Ký tự**, **Số lượng**).

Có 3 trường hợp xảy ra khi xác định tên thứ i trong danh sách:

- + **a_i > a_{i-1}** → Bổ sung vào cuối s_{i-1} phần tử (b₁, a_i-a_{i-1}), nếu s_{i-1} kết thúc bằng các ký tự b₁ thì phần tử mới sẽ được hợp với phần tử cuối bằng cách **chỉnh lý số lượng**,

-  $a_i = a_{i-1} \rightarrow$ Bớt số lượng phần tử cuối của s_{i-1} một đơn vị, thêm phần tử mới với ký tự tiếp theo và số lượng bằng 1. Có thể xảy ra các tình huống:
 -  Số lượng sau khi giảm là 0, khi đó phần tử này sẽ bị xóa trong s_i ,
 -  Phần tử cuối cùng có dạng (b_{ans} , k), ta không có ký tự lớn hơn để bổ sung phần tử mới, khi đó phải xóa phần tử này, bổ sung **một ký tự** lớn hơn tiếp theo và phần tử (b_1 , $k-1$). Có thể sẽ cần chỉnh lý cách lưu trữ 3 phần tử cuối nhận được.
-  $a_i < a_{i-1} \rightarrow$ Cần xóa $a_{i-1}-a_i$ ký tự cuối của s_{i-1} và xử lý kết quả nhận được như trong trường hợp độ dài bằng nhau đã xét ở trên.

Tuy vậy, ngay từ đầu giá trị ans không xác định được trước (trừ trường hợp dãy đơn điệu tăng).

Có 2 cách xử lý:

-  Tìm kiếm nhị phân trong khoảng từ 1 đến n ,
-  Xác định cận dưới của ans , dần xuất kết quả cho các trường hợp đáp ứng điều kiện đủ, trong trường hợp cần xác định tiếp ans :
 - Tìm kiếm nhị phân trong khoảng từ ans đến 327,
 - Xác định kết quả bằng cách tăng dần ans .

Trên thực tế, ta có thể tìm kiếm và chỉnh lý ans trong các đoạn nằm giữa **2 độ dài 1 liên tiếp** nhau.

Việc lưu trữ s_i dưới dạng các phần tử (**Ký tự**, **Số lượng**) là cách xử lý vạn năng cho mọi khả năng của **Số lượng**.

Bài toán đang xét cho phép lọc dữ liệu và chỉ cần xử lý với các $a_i \leq 17$. Như vậy chỉ cần lưu trữ và xử lý **một số 17 chữ số trong cơ số ans**. Việc chuyển sang số tiếp theo đơn thuần là cộng 1 và chuẩn hóa độ dài về giá trị a_i cần thiết hoặc bổ sung thêm các số 0 để có số độ dài a_i .

Tổ chức dữ liệu:

-  Mảng **int** $a[N]$ – lưu trữ các độ dài tên ($N = 10^5 + 1$),
-  Mảng **int** $s[21] = \{0\}$ – lưu trữ số tương ứng với xâu các biểu tượng.

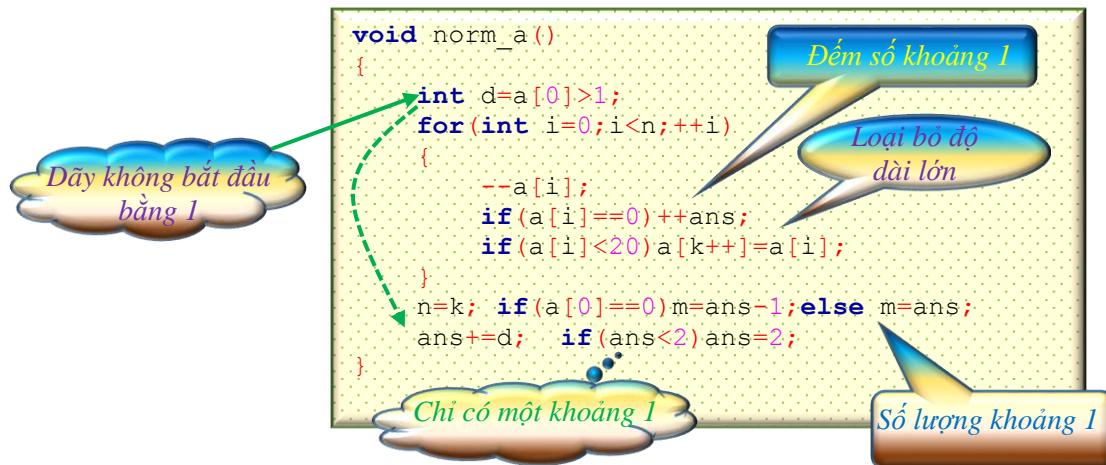
Xử lý:

Nhập dữ liệu và kiểm tra trường hợp đơn điệu tăng:

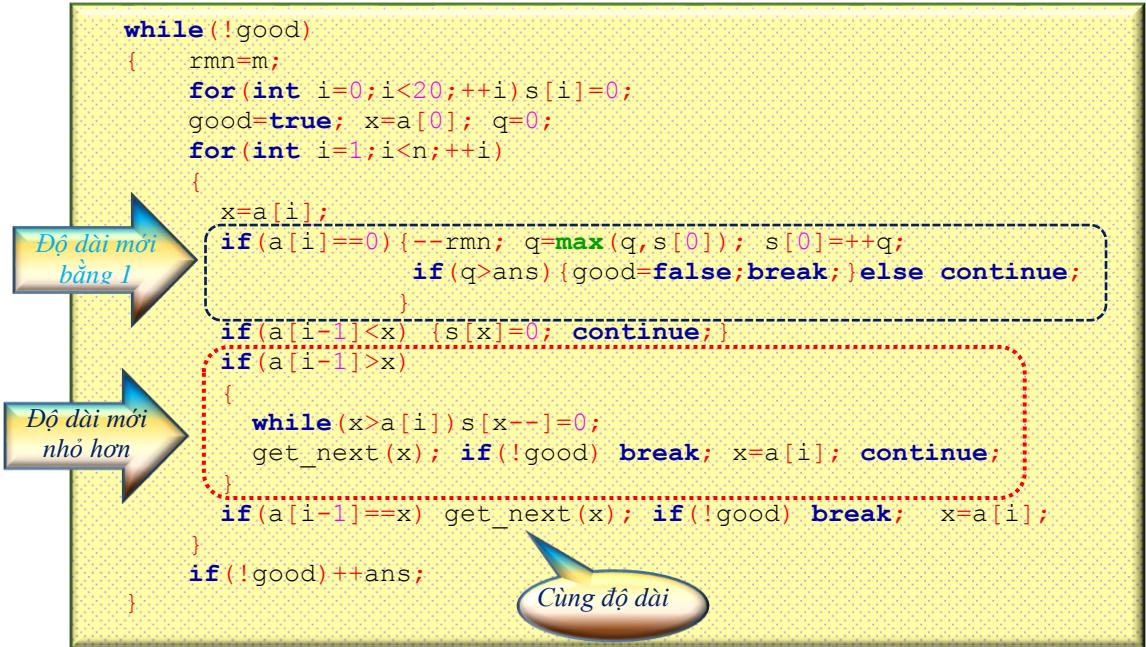
```
fi>>n;
for (int i = 0; i < n; i++) fi>>a[i];
bool incr = true;
for (int i = 1; i < n; i++) incr &= a[i] > a[i - 1];
if (incr)
{
    fo<<"1"; times; return 0;
}
```

Trường hợp **ans** > 1:

Lọc dữ liệu và kiểm tra trường hợp **ans** < 317:



Xác định **ans** bằng phương pháp điều chỉnh tăng dần:



Hàm tính xâu cùng độ dài có thứ tự từ điển tiếp theo:

- ✚ Mô phỏng phép tăng 1 trong cơ số **ans**,
- ✚ Sau khi nhận được mã biểu diễn mới, dựa vào chữ số bậc cao nhất (**s[0]**) kiểm tra xem còn đủ mã dự trữ cho các khoảng 1 còn lại, nếu không còn đủ thì phải tăng **ans** và kiểm tra lại toàn dãy với số lượng biểu tượng mới.

Việc điều chỉnh **ans** có thể thực hiện theo sơ đồ tìm kiếm nhị phân:

```
int l=ans-1, r=317;
while(l<r-1)
{
    ans=(l+r)/2; rmn=m;
    for(int i=0;i<20;++i)s[i]=0;
    good=true; x=a[0]; q=0;
    for(int i=1;i<n;++i)
    {
        x=a[i];
        if(a[i]==0){--rmn; q=max(q,s[0]); s[0]=++q;
                      if(q>ans){good=false;break;}else continue;
                    }
        if(a[i-1]<x) {s[x]=0; continue;}
        if(a[i-1]>x)
        {
            while(x>a[i])s[x--]=0;
            get_next(x); if(!good) break; x=a[i]; continue;
        }
        if(a[i-1]==x) get_next(x); if(!good) break; x=a[i];
    }
    if(!good) l=ans; else r=ans;
}
```

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

Chương trình

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

int const N = 1234567;
int s[21]={0};
int a[N], ans=0, n, k, x, m, q, rmn;
bool good=false;

void norm_a()
{
    int d=a[0]>1;
    for(int i=0; i<n; ++i)
    {
        --a[i];
        if(a[i]==0) ++ans;
        if(a[i]<20) a[k++]=a[i];
    }
    n=k;  if(a[0]==0) m=ans-1; else m=ans;
    ans+=d;  if(ans<2) ans=2;
}

void get_next(int p)
{
    int q, t;
    ++s[p];
    while(p>0 && s[p]==ans) {s[p]=0; --p; ++s[p];}
    if(s[0]>=ans-rmn)
    {
        good=false; return;
    }
}

int main()
{
    fi>>n;
    for (int i = 0; i < n; i++) fi>>a[i];
    bool incr = true;
    for (int i = 1; i < n; i++) incr &= a[i] > a[i - 1];
    if (incr)
    {
        fo<<"1"; times;
        return 0;
    }
    k=ans; norm_a();
    if(ans>316){fo<<ans; times; return 0;}

    while(!good)
    {
        rmn=m;
        for(int i=0; i<20; ++i) s[i]=0;
        good=true; x=a[0]; q=0;
```

```

for (int i=1; i<n; ++i)
{
    x=a[i];
    if (a[i]==0) {--rmn; q=max(q, s[0]); s[0]=++q;
                    if (q>ans) {good=false; break; } else continue;
                }
    if (a[i-1]<x) {s[x]=0; continue; }
    if (a[i-1]>x)
    {
        while (x>a[i]) s[x--]=0;
        get_next(x); if (!good) break; x=a[i]; continue;
    }
    if (a[i-1]==x) get_next(x); if (!good) break; x=a[i];
}
if (!good) ++ans;
}

fo<<ans; times;
}

```



VV34. KHÓA CHƯƠNG TRÌNH

Tên chương trình: **LOCK.CPP**

Titan là vệ tinh lớn nhất của sao Thổ và là vệ tinh lớn thứ hai trong hệ mặt trời (đứng sau vệ tinh Ganimed của Thiên vương tinh). Titan là vệ tinh duy nhất có bầu khí quyển dày đặc bao gồm chủ yếu là ni-tơ, có đường kính 5 512 km, gấp rưỡi mặt trăng và có khối lượng lớn hơn 80% so với mặt trăng. Đó là nơi hy vọng nhất tìm thấy sự sống ngoài trái đất trong hệ mặt trời.

Một tàu thăm dò được phóng lên Titan. Theo dự kiến, tàu thăm dò sẽ hạ cánh xuống điểm có tọa độ $(0, 0)$, khởi động robot tiên hành tiến hành đo đạc, khảo sát theo lộ trình đã cài đặt dưới xâu s độ dài n chứa các ký tự trong tập $\{L, U, R, D\}$, mỗi ký tự xác định hướng chuyển động tiếp theo của robot tương ứng theo các hướng sang Trái, lên Trên, sang Phải, xuống Dưới, mỗi bước bằng một đơn vị độ dài.

Trong quá trình tàu di chuyển tới Titan các nhà khoa học tiếp tục phân tích dữ liệu từ các trạm thăm dò trước đó, phát hiện ra điểm tọa độ (x, y) có nhiều triển vọng cung cấp những thông tin hết sức quan trọng và muốn đưa robot tới dừng ở điểm đó.

Việc cài lại chương trình trong robot là hết sức phức tạp vì xâu s không được lưu trữ tường minh. Cuối cùng người ta đi đến giải pháp là khóa tín hiệu điều khiển hướng L, U, R, D sau một số lần phát nhất định, có thể là 0 nếu khóa tác động ngay từ đầu hoặc là k sau lần phát thứ k . Để đảm bảo chắc chắn robot không rời khỏi điểm (x, y) , sau lần phát tín hiệu thứ n , mọi hướng chưa bị khóa tác động cũng sẽ được báo tín hiệu khóa bằng giá trị n . Dĩ nhiên có nguy cơ robot không thể tới đích (x, y) theo cách xử lý trên. Trong trường hợp đó cần khóa toàn bộ chương trình ngay từ đầu bằng một giá trị -1 duy nhất, đảm bảo robot đứng yên tại chỗ hạ cánh chờ cài đặt chương trình mới.

Hãy xác định các lệnh điều khiển cần gửi đi.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^6$),
- ✚ Dòng thứ 2 chứa xâu s độ dài n ,
- ✚ Dòng thứ 3 chứa 2 số nguyên x và y ($|x|, |y| \leq 10^6$).

Kết quả: Đưa ra file văn bản LOCK.OUT số -1 hoặc 4 số nguyên trên một dòng xác định thời điểm khóa các tín hiệu điều khiển. Nếu có nhiều cách khóa khác nhau thì đưa ra cách tùy chọn.

Ví dụ:

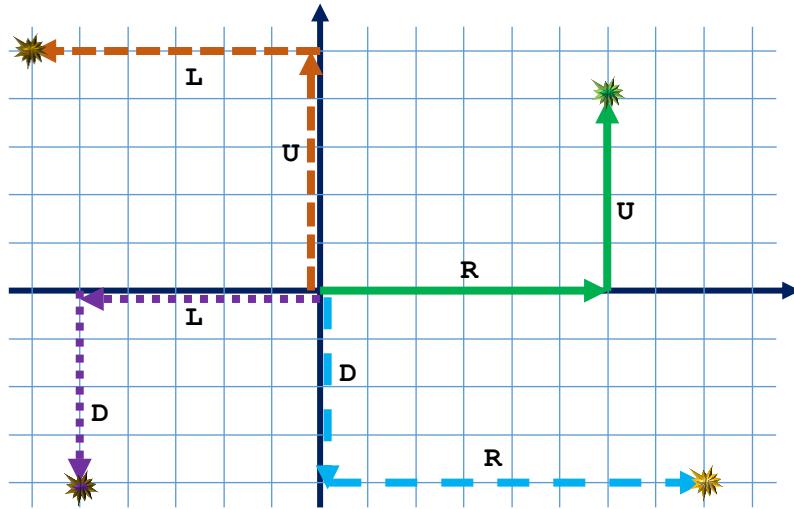
LOCK.INP	LOCK.OUT
8	1
LLURDRRD	7
1 -1	6 8



VV33 Mos_op_20170310 5

Giải thuật: Xử lý xâu.

Từ gốc tọa độ $(0, 0)$ tới điểm (x, y) chỉ cần dùng **tối đa** không quá 2 loại lệnh,
Những lệnh còn lại: khóa ngay từ đầu.



Với những lệnh cần dùng: mỗi lần thực hiện giá trị tuyệt đối của tọa độ tương ứng giảm 1.

Quá trình xử lý kết thúc khi có tọa độ kết quả là $(0, 0)$ hoặc khi duyệt hết xâu s .

Cần đưa ra kết quả -1 khi sau khi duyệt hết xâu s ta vẫn không nhận được tọa độ $(0, 0)$.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình

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

int main()
{
    fi>>n;
    int l=n,u=n,r=n,d=n;
    fi>>s>>x>>y;
    if(x>=0)l=0; if(x<=0)r=0;
    if(y>=0)d=0; if(y<=0)u=0;
    for(int i=0;i<n;++i)
    {
        switch(s[i])
        {
            case 'L':{if(x<0)++x,l=i+1; break;}
            case 'U':{if(y>0)--y,u=i+1; break;}
            case 'R':{if(x>0)--x,r=i+1; break;}
            case 'D':{if(y<0)++y,d=i+1; break;}
        }
        if(x==0 && y==0)break;
    }
    if(x!=0 || y!=0) fo<<l<<' '<<u<<' '<<r<<' '<<d;
    Times;
}
```



Xét cây nhị phân mỗi nút có 2 nút con. Các nút của cây được phân thành lớp. Nút gốc thuộc lớp 0, những nút con của các nút lớp $i-1$ thuộc lớp i . Các nút của cây được đánh số từ 1 trở đi bắt đầu từ nút gốc theo trình tự duyệt các lớp 0, 1, 2, . . . , mỗi lớp duyệt từ trái sang phải.

Mỗi nút của cây chứa một phân số. Nếu một nút chứa phân số $\frac{p}{q}$ thì nút con trái của nó chứa phân số $\frac{p}{p+q}$, nút phải chứa phân số $\frac{p+q}{q}$. Nút gốc chứa phân số 1/1, tương ứng với $p = 1$ và $q = 1$.

Có m truy vấn, mỗi truy vấn thuộc một trong 2 dạng sau:

- **1 p q** – Yêu cầu xác định số thứ tự của nút chứa phân số $\frac{p}{q}$,
- **2 n** – Yêu cầu xác định p và q của nút thứ n .

Với mỗi truy vấn hãy đưa ra kết quả tìm được.

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

- ✚ Dòng đầu tiên chứa một số nguyên m ($1 \leq m \leq 10^6$),
- ✚ Mỗi dòng trong m dòng sau chứa thông tin về một truy vấn.

Các giá trị n , p , q không vượt quá 10^{18} , đảm bảo tồn tại lời giải và các giá trị tìm được không vượt quá 10^{18} .

Kết quả: Đưa ra file văn bản ORDER.OUT, kết quả mỗi truy vấn đưa ra trên một dòng.

Ví dụ:

ORDER.INP
3
1 1 1
1 3 4
2 11

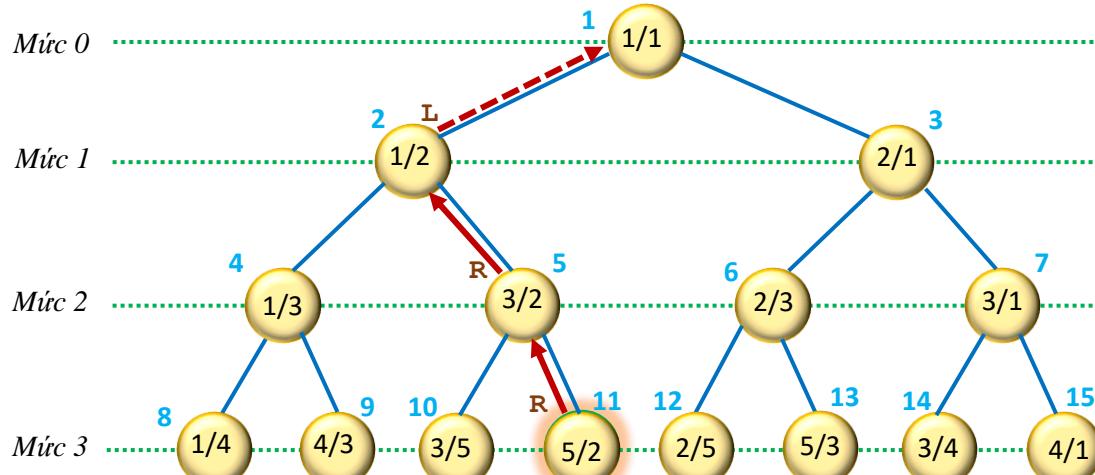
ORDER.OUT
1
14
3 5



Giải thuật: Xử lý bút.

Ta thấy mức i có 2^i nút,

Một nút có thứ tự là k thì 2 nút con trái và phải của nó có số thứ tự tương ứng là $2k$ và $2k+1$. Số thứ tự nút cha của nó là $k/2$.



Xét nút chứa phân số (p/q) . Nếu $p > q$ thì nút này là nút con trái của nút cha trực tiếp, trong trường hợp ngược lại – là nút con phải,

Một nút có $p = 1$ thì nó và các nút cha của nó trên đường di chuyển về gốc đều là nút con trái,

Một nút có $q = 1$ thì nó và các nút cha của nó trên đường di chuyển về gốc đều là nút con phải.

Xét đường đi từ nút chứa phân số (p/q) về nút có $p = 1$ hoặc $q = 1$, đường đi này tương ứng với xâu s chỉ chứa các ký tự thuộc tập $\{L, R\}$. Dựa vào quan hệ so sánh của p và q ta dễ dàng xác định được xâu s .

Ví dụ, với $p = 5, q = 2$ ta có:

$$\begin{array}{ccccccc} (5, 2) & \rightarrow & (3, 2) & \rightarrow & (1, 2) \\ s = & R & R & \longrightarrow & s = 11 \end{array}$$

Để tiện xử lý, ta thay L bằng 0 và R bằng 1.

Bằng cách duyệt đường đi từ cuối về đầu xâu s ta dễ dàng tính được số thứ tự của phần tử. Lưu ý là phần còn lại của đường đi về nút gốc không được lưu trữ tường minh trong s . Nếu việc xác định s dừng lại ở nút $(1/u)$ thì nút đó có số thứ tự là 2^{u-1} , nếu nút chứa giá trị $(v/1)$ thì số thứ tự là $2^v - 1$.

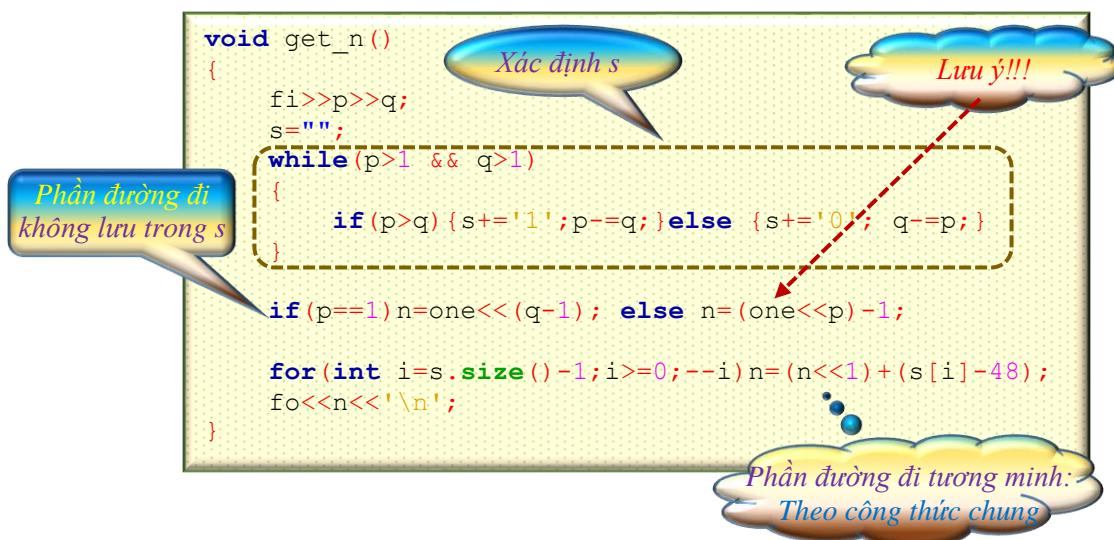
Tương tự như vậy với việc xác định **p**, **q** theo số thứ tự **n**. Nếu **n** là chẵn thì nút đang xét là nút con trái, trong trường hợp ngược lại – nút con phải. Dựa vào đường đi ta dễ dàng chỉnh lý **p** và **q**.

Tổ chức dữ liệu:

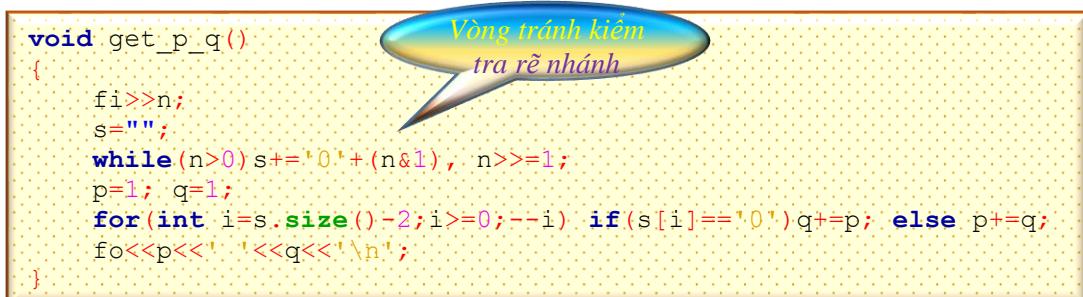
Để thuận tiện xử lý ta lưu đường đi dưới dạng xâu các ký tự 0 và 1. Có thể tránh việc sử dụng xâu này, nhưng điều đó sẽ làm chương trình phức tạp lên đôi chút một cách không cần thiết. Trong trường hợp xâu nhất độ dài xâu **s** cũng không vượt quá 64.

Xử lý:

Truy vấn loại 1 (*Tính n*):



Truy vấn loại 2 (*Tính p, q*): Lưu trữ tường minh toàn bộ đường đi!



Độ phức tạp của giải thuật: O(m).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "order."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n,p,q,one=1;
int k,t;
string s;

void get_n()
{
    fi>>p>>q;
    s="";
    while(p>1 && q>1)
    {
        if(p>q) {s+='1';p-=q;} else {s+='0'; q-=p;}
    }
    if(p==1) n=one<<(q-1); else n=(one<<p)-1;
    for(int i=s.size()-1;i>=0;--i)n=(n<<1)+(s[i]-48);
    fo<<n<<'\n';
}

void get_p_q()
{
    fi>>n;
    s="";
    while(n>0)s+='0'+(n&1), n>>=1;
    p=1; q=1;
    for(int i=s.size()-2;i>=0;--i) if(s[i]=='0') q+=p; else p+=q;
    fo<<p<<' '=<<q<<'\n';
}

int main()
{
    fi>>t;
    for(int i=0;i<t;++i)
    {
        fi>>k;
        if(k==2) get_p_q(); else get_n();
    }
    Times;
}
```



Cuộc cách mạng khoa học kỹ thuật lần thứ IV đã làm xuất hiện các cộng đồng dân cư sử dụng chủ yếu hệ cơ số 2 trong cuộc sống hàng ngày (*Cộng đồng 0*) bên cạnh những cộng đồng sử dụng chủ yếu hệ cơ số 10 truyền thống (*Cộng đồng 1*).

Khi Bob nói cái áo sơ mi mình mới mua giá 100 USD, Alice, một người thuộc cộng đồng sử dụng cơ số 2 nghĩ rằng Bob đang nói ở hệ thập phân và có nghĩa là cái áo đó có giá 110 0100 ở hệ cơ số 2 mình đang dùng, còn Shue thì cho rằng Bob đang dùng cơ số 2 và như vậy áo có giá 4 USD theo cơ số thập phân. Tóm lại, sẽ có nhiều rắc rối và hiểu lầm xảy ra khi không biết người nói chuyện với mình thuộc cộng đồng nào.

Khu dân cư có dạng lưới ô vuông r dòng và c cột. Ở mỗi ô, cộng đồng nào chiếm đa số thì dần dần mọi người sống ở đó đều hòa chung vào cộng đồng đa số và trở thành nơi mọi người cùng dùng một cơ số như nhau. Trên bản đồ mỗi ô được ghi nhận là 0 hoặc 1 phụ thuộc vào ở đó người ta dùng cơ số 2 hay cơ số thập phân. Từ một ô người ta có thể chuyển sang ô kề cạnh.

Một nhân viên tiếp thị muốn đi giới thiệu hàng từ ô $(r1, c1)$ tới ô $(r2, c2)$. Để tránh mọi rắc rối có thể xảy ra người đó chỉ muốn đi trong phạm vi một loại cộng đồng và chuẩn bị trước số liệu phù hợp với cộng đồng trên đường đi của mình, loại **Binary** hoặc **Decimal**. Nếu không thể đi chỉ giới hạn trong một cộng đồng thì buộc phải chuẩn bị tới 2 loại tài liệu khác nhau và bộ tài liệu đó được gọi là **Neither**.

Có m nhân viên tiếp thị được cử đi giới thiệu sản phẩm mới, mỗi người có điểm xuất phát và điểm đích của mình. Hãy xác định bộ tài liệu cần chuẩn bị đối với mỗi người.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên r và c ($1 \leq r, c \leq 1\,000$),
- ✚ Trong r dòng sau, mỗi dòng chứa một xâu độ dài c chỉ bao gồm các ký tự { 0, 1 } xác định bản đồ khu dân cư,
- ✚ Dòng tiếp theo chứa số nguyên m ($1 \leq m \leq 1\,000$),
- ✚ Mỗi dòng trong m dòng tiếp theo chứa 4 số nguyên $r1, c1, r2$ và $c2$ xác định điểm xuất phát và điểm đích của một nhân viên tiếp thị ($1 \leq r1, r2 \leq r, 1 \leq c1, c2 \leq c$).

Kết quả: Đưa ra file văn bản TWOORLD.OUT loại tài liệu mà mỗi người phải chuẩn bị, mỗi dữ liệu đưa ra trên một dòng.

Ví dụ:

TWOORLD.INP
1 4
1100
2
1 1 1 4
1 1 1 1

TWOORLD.OUT
Neither
Decimal



VV36 Bayl 2016

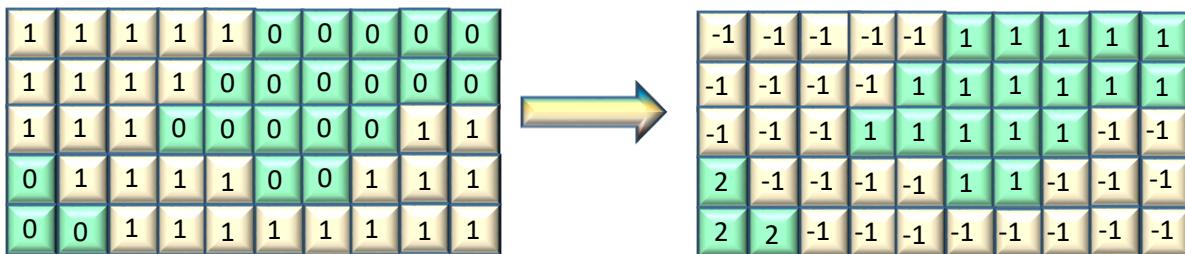
Giải thuật: Loang theo chiều rộng (BFS).

Ván đề cần giải quyết khá đơn giản: xác định xem ô xuất phát và ô đích có cùng một miền liên thông hay không và nếu cùng thì thuộc loại nào – 0 hay 1.

Vì phải xử lý nhiều truy vấn nên cần xác định trước các miền liên thông mỗi loại, xây dựng bản đồ liên thông để tra cứu.

Các miền liên thông chứa ô ‘0’ được đánh số 1, 2, 3, . . .

Các miền liên thông chứa ô ‘1’ được đánh số -1, -2, -3, . . .



Việc xác định miền liên thông được xác định bằng phương pháp loang. Với giới hạn kích thước đã cho việc *loang theo chiều sâu là không thích hợp* vì mất nhiều thời gian và cần rất nhiều bộ nhớ, vì vậy cần phải loang theo chiều rộng.

Tổ chức dữ liệu:

- Mảng **char** `d[N][N]` – lưu trữ trạng thái miền cần xử lý,
- Mảng **int** `a[N][N]={0}` – lưu trữ bản đồ miền liên thông,
- Hàng đợi **queue<pii>** `q` – lưu trữ vết để loang theo chiều rộng, mỗi phần tử là một cặp dữ liệu (**i**, **j**).

Xử lý:

Tạo hàng rào quanh vùng dữ liệu cần xử lý để đơn giản hóa quá trình loang,

Tìm điểm đầu của miền liên thông mới và loang theo chiều rộng,

Hàm **bfs_a** (**char** **x**) xác định miền liên thông theo ký tự **x**:

```
void bfs_a (char x)
{ int u,v;
  while (!q.empty ())
  {
    t= q.front (); u=t.first; v=t.second;
    a[u][v]=k; q.pop ();
    if (d[u][v-1]==x && a[u][v-1]==0) {q.push ({u,v-1}); a[u][v-1]=MX;}
    if (d[u][v+1]==x && a[u][v+1]==0) {q.push ({u,v+1}); a[u][v+1]=MX;}
    if (d[u-1][v]==x && a[u-1][v]==0) {q.push ({u-1,v}); a[u-1][v]=MX;}
    if (d[u+1][v]==x && a[u+1][v]==0) {q.push ({u+1,v}); a[u+1][v]=MX;}
  }
}
```

Dành dấu đã nạp
vào hàng đợi

Độ phức tạp của giải thuật: $O(r \times c)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "twoworld."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
const int N = 1002;
const int MX = 1000000000;
int r,c,n,m,r1,r2,c1,c2,k=0,a[N][N]={0};
char d[N][N];
queue<pii> q;
pii t;
string s;
void bfs_a(char x)
{ int u,v;
  while(!q.empty())
  {
    t=q.front(); u=t.first; v=t.second;
    a[u][v]=k; q.pop();
    if(d[u][v-1]==x && a[u][v-1]==0) {q.push({u,v-1}); a[u][v-1]=MX;}
    if(d[u][v+1]==x && a[u][v+1]==0) {q.push({u,v+1}); a[u][v+1]=MX;}
    if(d[u-1][v]==x && a[u-1][v]==0) {q.push({u-1,v}); a[u-1][v]=MX;}
    if(d[u+1][v]==x && a[u+1][v]==0) {q.push({u+1,v}); a[u+1][v]=MX;}
  }
}
int main()
{ fi>>r>>c;
  for(int i=0;i<=c+1;++i)d[0][i]='2',d[r+1][i]='2';
  for(int i=1;i<=r;++i)
  {
    fi>>s;
    d[i][0]='2'; d[i][c+1]='2';
    for(int j=0;j<c;++j) d[i][j+1]=s[j];
  }
  for(int i=1;i<=r;++i)
    for(int j=1;j<=c;++j)
      if(d[i][j]=='0' && a[i][j]==0)
        { ++k; q.push({i,j}); bfs_a('0'); }
  k=0;
  for(int i=1;i<=r;++i)
    for(int j=1;j<=c;++j)
      if(d[i][j]=='1' && a[i][j]==0)
        { --k; q.push({i,j}); bfs_a('1'); }
  fi>>m;
  for(int i=0;i<m;++i)
  {
    fi>>r1>>c1>>r2>>c2;
    if(a[r1][c1]>0 && a[r1][c1]==a[r2][c2]) {fo<<"Binary\n"; continue;}
    if(a[r1][c1]<0 && a[r1][c1]==a[r2][c2]) {fo<<"Decimal\n"; continue;}
    fo<<"Neither\n";
  }
  Times;
}
```



Hệ thống tàu điện ngầm phát triển làm cho việc khai thác tuyến đường sắt trên cao trở nên không kinh tế. Chính quyền thành phố quyết định biến nó thành tuyến phố đi bộ, trên đó cho thuê kinh doanh các ki ốt ăn uống giải khát.

Toàn bộ tuyến đường được chia thành k đoạn bằng nhau, đánh số từ 1 trở đi bắt đầu từ trái. Mỗi cá nhân hay tổ chức muốn thuê phải thuê nguyên một đoạn hay một số đoạn liên tục. Khi được thuê người ta phải dựng ki ốt và mang các trang thiết bị cần thiết lên đó (bàn, ghế, máy nướng thịt, kho chứa nước uống, v. v. . .) và tạo ra một tải trọng nhất định lên mặt đường.

Có n đơn xin cấp phép kinh doanh. Đơn thứ i xin sử dụng các đoạn từ lf_i đến rt_i và tạo tải trọng lên mặt đường là p_i . Chính quyền có thể từ chối hay đáp ứng đơn xin. Một đơn khi được đáp ứng sẽ có quyền sử dụng toàn bộ các đoạn đã nêu trong đơn. Một đoạn đường có thể thuộc nhiều đơn được cấp phép, nhưng khi đó áp lực lên mặt đường là tổng tải trọng nêu trong các đơn tương ứng. Tất cả các đoạn đều phải được cho thuê. Để đảm bảo an toàn, người ta muốn áp lực trên đoạn có tổng tải trọng lớn nhất phải là nhỏ nhất.

Hãy đưa ra áp lực trên đoạn có tổng tải trọng lớn nhất.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và k ($1 \leq n \leq 10^5$, $1 \leq k \leq 10^9$),
- ✚ Dòng thứ i trong n dòng sau chứa 3 số nguyên lf_i , rt_i và p_i ($1 \leq lf_i \leq rt_i \leq k$, $1 \leq p_i \leq 10^9$).

Kết quả: Đưa ra file văn bản FOOTPATH.OUT một số nguyên – áp lực lớn nhất tìm được. Nếu không tồn tại phương án cho thuê thì đưa ra số -1.

Ví dụ:

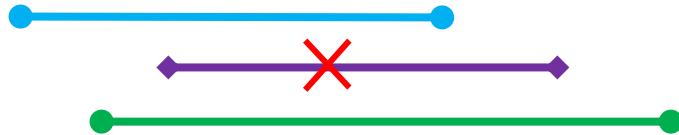
FOOTPATH.INP
4 5
1 4 3
4 5 5
1 1 3
1 2 1

FOOTPATH.OUT
8



Giải thuật: Cây quản lý đoạn, Quy hoạch động.

Dễ dàng thấy rằng nếu bài toán có nghiệm thì đoạn tải trọng lớn nhất thuộc không quá 2 đơn xin cấp phép. Thật vậy, nếu có nhiều hơn 2 đơn được đáp ứng trên một đoạn thì ít nhất tồn tại một đơn chứa các đoạn nằm gọn trong hợp các đoạn của 2 đơn nào đó khác, khi loại đơn này tải trọng lớn nhất sẽ giảm.



Ta sẽ lắp ráp các đoạn thẳng, kéo dài dần để phủ toàn bộ đường theo nguyên tắc quy hoạch động,

Sắp xếp các đoạn thẳng theo đầu phải,

Loại bỏ các đoạn lồng nhau,

Xét tập chấp nhận được: tập các đoạn thẳng có thể xem xét để kéo dài lời giải, đó là tập các đoạn thẳng có đầu trái nhỏ hơn hoặc bằng $r+1$, trong đó r là đầu phải của đoạn cuối cùng,

Đoạn được chọn để kéo dài lời giải là đoạn:

- ✚ Tạo ra áp lực p trên đường nhỏ nhất,
- ✚ Có đầu phải nhỏ nhất trong số các đoạn thỏa mãn điều kiện đầu.

Để thực hiện được việc này ta cần có *cây quản lý min* theo đoạn, lưu trữ tọa độ phải nhất và giá trị min,

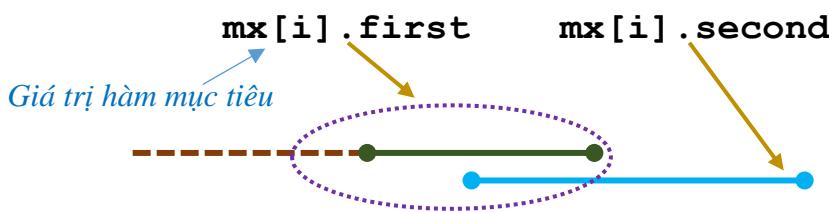
Việc bổ sung vào cây là gán giá trị tương ứng vào nút lá, xóa khỏi cây – gán giá trị vô cực.

Gọi $\text{mx}[i]$ là đại lượng lưu trữ kết quả tối ưu đạt được khi phủ đoạn i của đường và các đoạn trước đó cũng đã được phủ.

$\text{mx}[i]$ cần chứa 2 thông tin:

- ✚ Áp lực tối đa lên con đường trên các đoạn đã phủ,
- ✚ Áp lực lên đường ở đầu phải đoạn đã phủ.

Giá trị thứ 2 là cần thiết khi phần tiếp theo cần xét đè lên phần đã phủ.



Công thức lắp có dạng:

- $\text{mx}[i].first = \min(\max(\text{mx}[j].first, p_i + p_j))$ với mọi $j | l_j \leq l_i \leq r_j \leq r_i$,
- $\text{mx}[i].first = \min(\max(\text{mx}[j].first, p_i + p_j))$ với mọi $j | r_j = l_i - 1$.

Trường thứ 2 của $\text{mx}[i]$ chứa giá trị p_i .

Để quản lý tập hợp các đơn đăng ký chấp nhận được phục vụ tính giá trị hàm mục tiêu trên tập chấp nhận được cần tổ chức một cây nhị phân quản lý min theo đoạn.

Do giá trị k có thể rất lớn, trong lúc đó – giá trị n ở phạm vi chấp nhận được vì vậy cần ánh xạ thông tin của đơn đặt hàng về phạm vi tỷ lệ với n để có thể xây dựng cây quản lý min, trong ánh xạ cần giữ nguyên quan hệ lồng nhau, giao nhau hoặc tiếp xúc nhau của thông tin ban đầu.

Việc loại bỏ các đơn có khoảng lồng trong đơn khác có thể thực hiện ngay trong quá trình xác định tập chấp nhận được.

Tổ chức dữ liệu:

- Mảng nhóm 3 $t113 \times [\text{MAXN}]$ – lưu trữ thông tin ban đầu về các đơn xin phép kinh doanh,
- Mảng $l1 q [\text{MAXN}]$ – phục vụ ánh xạ co với các đơn xin phép,
- Cây $\text{map } <l1, l1> mp$ – phục vụ đánh số lại các đoạn trong đơn ban đầu,
- Mảng $\text{pair } <l1, l1> t [\text{MAXN}]$ – phục vụ tổ chức cây quản lý min,
- Mảng $\text{pair } <l1, l1> mx [\text{MAXN}]$ – lưu giá trị hàm mục tiêu phục vụ lắp.

Xử lý:

Nhập dữ liệu và sắp xếp theo điểm cuối:

```
fi >> n >> k;
for (int i=1; i<=n; i++) { fi>> l>>r>>p; x[i]=make_tuple(l,r,p);}
```

```
bool comp(t113 a, t113 b) {return get<1>(a) < get<1>(b); }
```

Ánh xạ co:

```
for (int i=1; i<=n; i++)
{
    tie(l,r,p)=x[i];
    q[++kl]=l; q[++kl]=r;
    q[++kl]=l-1; q[++kl]=r-1;
    q[++kl]=l+1; q[++kl]=r+1;
}
q[++kl] = k; q[++kl] = l;
sort(q + 1, q + kl + 1);
```

Tạo bản đồ

```
ll m = 0;
for (int i=1; i<=kl; i++) if (q[i] > 0) if (mp[q[i]]==0) mp[q[i]]+=+m;
for (int i=1; i<=n; i++)
    {tie(l,r,p)=x[i]; l=mp[l], r=mp[r]; x[i]=make_tuple(l,r,p);}
```

Ánh xạ

Tính lắp theo quy hoạch động:

Khởi tạo
giá trị đầu

```
for (int i=1; i<=4 * m; i++) t[i] = {MAXLL, MAXLL}, mx[i]=t[i];

for (int i=1; i<=n; i++)
{
    tie(l,r,p)=x[i];

    if (mx[l-1].fi < MAXLL) mx[r]=min(mx[r], {max(mx[l-1].fi,p), p});
    xx = get(l, l, m, l, r);
    if (xx.fi < MAXLL) mx[r]=min(mx[r], {max(xx.se+p, xx.fi), xx.se+p});

    modify(l, l, m, r, mx[r]);
}
```

Một cách mở
gói dữ liệu

Nạp giá trị hàm mục tiêu
vào cây quản lý đoạn

Các hàm hỗ trợ:

Chương trình

```
#include <bits/stdc++.h>
#define NAME "footpath."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
#define fi first
#define se second

using namespace std;
ifstream fi (NAME"inp");
//ifstream fi ("63");
ofstream fo (NAME"out");
typedef int64_t ll;
typedef tuple<ll,ll,ll> tll3;
const int MAXN = 3000000;
const ll MAXLL = 4611686018427387900LL;

tll3 x[MAXN];
ll q[MAXN];
pair <ll,ll> t[MAXN], mx[MAXN];

bool comp(tll3 a, tll3 b) {return get<1>(a) < get<1>(b);}

void modify(ll v,ll tl,ll tr,ll pos,pair<ll,ll> val)
{
    if (tl == tr) t[v]=val;
    else
    {
        ll tm = (tl+tr)/2;
        if (pos <= tm) modify(v*2, tl, tm, pos, val);
        else modify(v*2 + 1, tm+1, tr, pos, val);
        t[v] = min(t[v*2], t[v*2+1]);
    }
}

pair < ll, ll > get(ll v,ll tl,ll tr,ll l,ll r)
{
    if (l > r) return {MAXLL, MAXLL};
    if (tl == l && tr == r) return t[v];
    ll tm = (tl+tr)/2;
    return min(get(v*2,tl,tm,l,min(tm, r)),
               get(v*2+1,tm+1,tr,max(tm+1,l), r));
}

int main()
{
    map < ll, ll > mp;
    ll ans = MAXLL,now,n,k,l,r,p,kl = 0;
    pair <ll,ll> xx;
    fi >> n >> k;
    for (int i=1; i<=n; i++) {fi>> l>>r>>p;x[i]=make_tuple(l,r,p);}
    sort(x+1,x+n+1, comp);
    for (int i=1; i<=n; i++)
    {
        tie(l,r,p)=x[i];
        q[++kl]=l; q[++kl]=r;
```

```

        q[++kl]=l-1; q[++kl]=r-1;
        q[++kl]=l+1; q[++kl]=r+1;
    }
    q[++kl] = k; q[++kl] = 1;
    sort(q + 1, q + kl + 1);

ll m = 0;
for (int i=1; i<=kl; i++) if (q[i] > 0) if (mp[q[i]] == 0) mp[q[i]]==+m;
for (int i=1; i<=n; i++)
    {tie(l,r,p)=x[i]; l= mp[l], r=mp[r]; x[i]=make_tuple(l,r,p);}

for (int i=1; i<=4 * m; i++) t[i] = {MAXLL, MAXLL}, mx[i]=t[i];
for (int i=1; i<=n; i++)
{
    tie(l,r,p)=x[i];
    if (mx[l-1].fi < MAXLL) mx[r]=min(mx[r],{max(mx[l-1].fi,p),p});
    xx = get(l, 1, m, l, r);
    if (xx.fi < MAXLL) mx[r]=min(mx[r],{max(xx.setp, xx.fi),xx.setp});
    modify(l, 1, m, r, mx[r]);
}

k = mp[k];
if (mx[k].fi == MAXLL) mx[k].fi=-1;
fo << mx[k].fi << endl;
Times;
}

```



Cho n số nguyên a_1, a_2, \dots, a_n . Với dãy số nguyên này có thể tồn tại nhóm 3 (i, j, k) thỏa mãn điều kiện $a_i + a_j = a_k$.

Ví dụ, với dãy số 1, 1, 3, 3, 4, 6 ta có $a_1 + a_3 = a_5$ ($1 + 3 = 4$).

Hãy xác định có bao nhiêu nhóm 3 thỏa mãn điều kiện $a_i + a_j = a_k$. Hai nhóm 3 (i, j, k) và (j, i, k) được coi là khác nhau. Với dãy số đã nêu tồn tại 10 nhóm 3:

1 3 5	3 1 5
1 4 5	3 1 5
2 3 5	3 2 5
2 4 5	4 2 5
3 4 6	4 3 6

Dữ liệu: Vào từ file văn bản ABPLUS.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 ($-50\ 000 \leq a_i \leq 50\ 000$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản ABPLUS.OUT một số nguyên – số lượng nhóm 3 tìm được.

Ví dụ:

ABPLUS.INP	ABPLUS.OUT
6	10
1 1 3 3 4 6	



Giải thuật: Phân tích tình huống lô gic, Kỹ thuật tổ chức dữ liệu.

Tư tưởng chung của giải thuật rất đơn giản:

- ✚ Tính $t = a_i + a_j$ với $i = 1 \div n, j = 1 \div n, i \neq j$,
- ✚ Với mỗi t tính được kiểm tra sự tồn tại k sao cho $k \neq i, k \neq j$ và $a_k = t$.

Độ phức tạp của giải thuật sẽ là $O(n^3)$, *vượt quá giới hạn thời gian* cho phép với n đủ lớn.

Cần giảm độ phức tạp của giải thuật:

- ✚ Giảm số lần cần tính giá trị t ,
- ✚ Giảm thời gian tìm kiếm k với mỗi t .

Ta thấy, nếu nhóm 3 (i, j, k) thỏa mãn điều kiện cần tìm thì nhóm 3 (j, i, k) cũng thỏa mãn.

Với mỗi nhóm 3 (i, j, k) cần tìm có 4 trường hợp có thể xảy ra:

- ✚ Cả 2 số a_i và a_j đều dương,
- ✚ Cả 2 số a_i và a_j đều âm,
- ✚ Có một số nguyên và một số âm,
- ✚ Có ít nhất một trong 2 số a_i, a_j bằng 0.

Việc phân biệt các trường hợp sẽ cho phép tổ chức dữ liệu một cách thích hợp và làm giảm đáng kể độ phức tạp của giải thuật.

Không có yêu cầu phải đưa ra các chỉ số i, j, k nên ta chỉ cần lưu các giá trị khác nhau của phần tử thuộc dãy cùng với tần số xuất hiện và lưu trữ ở 2 phiên bản: một phiên bản *các giá trị khác nhau* của dãy, phục vụ tính t và một phiên bản lưu trữ *tần số xuất hiện*, phục vụ kiểm tra, xác định sự tồn tại của k và cập nhật kết quả.

Số lượng phần tử 0 được lưu trữ riêng.

Gọi a_i là tần số xuất hiện giá trị $i > 0$ trong dãy ban đầu, \mathbf{va}_j là phần tử thứ j trong dãy các giá trị dương khác nhau, sắp xếp theo thứ tự tăng dần, b_i là tần số xuất hiện giá trị $-i$ ($i > 0$) trong dãy ban đầu, \mathbf{ba}_j là phần tử thứ j trong dãy giá trị giá trị tuyệt đối các phần tử âm khác nhau, sắp xếp theo thứ tự tăng dần, \mathbf{zr} – số lượng phần tử bằng 0.

Xét các phần tử dương:

- ✚ Xét $t = \mathbf{va}_i + \mathbf{va}_j$, $j \geq i$, $t = \mathbf{va}_j + \mathbf{va}_i$ tồn tại khi và chỉ khi $a_j > 1$,
- ✚ Nếu $a_t \neq 0 \rightarrow$ tồn tại k và số lượng nhóm 3 mới tìm được sẽ là $a_i \times a_j \times a_k \times 2$.

Xử lý tương tự như vậy với các phần tử âm,

Trường hợp một số hạng dương và một số hạng âm: xét mọi $\mathbf{t} = \mathbf{v}\mathbf{a}_i + \mathbf{v}\mathbf{b}_j$ và xử lý tương tự như trên,

Trường hợp một số hạng bằng 0: (xảy ra khi $\mathbf{zr} > 0$), \mathbf{t} bằng $\mathbf{v}\mathbf{a}_i$ hoặc $\mathbf{v}\mathbf{b}_j$, \mathbf{k} chỉ tồn tại khi $\mathbf{v}\mathbf{a}_i$ hoặc $\mathbf{v}\mathbf{b}_j$ có tần số xuất hiện lớn hơn 1, khi đó số lượng nhóm 3 mới sẽ là $\mathbf{zr} \times \mathbf{a}[\mathbf{v}\mathbf{a}_i] \times 2$ hoặc $\mathbf{zr} \times \mathbf{b}[\mathbf{v}\mathbf{b}_j] \times 2$.

Trường hợp $\mathbf{t} = 0$ – chỉ cần xét khi $\mathbf{zr} > 2$, số lượng nhóm 3 mới sẽ là $\mathbf{zr} \times (\mathbf{zr} - 1)$.

Tổ chức dữ liệu:

- Các mảng `int` $\mathbf{a}[\mathbf{N}] = \{0\}$, $\mathbf{b}[\mathbf{N}] = \{0\}$ – lưu trữ tần số xuất hiện,
- Các mảng `vector<int>` $\mathbf{va}(\mathbf{na})$, $\mathbf{vb}(\mathbf{nb})$ – lưu trữ giá trị tuyệt đối các số khác nhau trong dãy (tương ứng dương và âm),
- Biến \mathbf{zr} – lưu trữ số lượng số 0.

Xử lý:

Nhập dữ liệu và tính tần số:

```
fi>>n;
for(int i=0; i<n; ++i)
{
    fi>>t;
    if(t==0){++zr; continue;}
    if(t>0)++a[t]; ++na; else ++b[-t], ++nb;
}
```

Tích lũy tần số với số âm

Xây dựng mảng các số khác nhau:

```
vector<int> va(na), vb(nb);
for(int i=0; i<=N; ++i)
{
    if(a[i]) va[ka++]=i;
    if(b[i]) vb[kb++]=i;
}
na=ka; nb=kb;
```

Lưu giá trị

Chỉnh lý kích thước

Xử lý số dương:

```
int mxa=va[na-1], mxb=vb[nb-1];
for(int i=0; i<na-1; ++i)
    for(int j=i; j<na; ++j)
    {
        if(i==j && a[i]==1) continue;
        t=va[i]+va[j];
        if(t>mxa) break;
        if(a[t]>0) if(i==j) ans+=a[vb[i]]*(a[vb[i]-1]);
                     else d=a[vb[j]], ans+=(d*a[vb[i]]*a[t])*2;
    }
```

Ngắt khi tổng vượt quá max

Trường hợp $i = j$

Cập nhật ans
với $i \neq j$

Xử lý kết hợp các số âm và dương:

```
for(int i=0; i<na; ++i)
    for(int j=0; j<nb; ++j)
    {
        t=va[i]-vb[j];
        if(t==0 && zr>0)
            {d=b[vb[j]]; ans+=(d*zr*a[vb[i]])*2; continue;}
        if(t>0 && a[t]>0)
            {d=b[vb[j]]; ans+=(d*a[t]*a[vb[i]])*2; continue;}
        if(t<0 && b[-t]>0)
            {d=a[vb[j]]; ans+=(d*b[-t]*b[vb[j]])*2; continue;}
    }
```

Tổng dương và
tồn tại k

Tổng bằng 0 và
tồn tại 0

Tổng âm và
tồn tại k

Xử lý các phần tử bằng 0:

```
if(zr>0)
{
    if(zr>2) ans+=zr*(zr-1);
    for(int i=0; i<na; ++i) if(a[vb[i]]>1) ans+=zr*a[vb[i]]*2;
    for(int i=0; i<nb; ++i) if(b[vb[i]]>1) ans+=zr*b[vb[i]]*2;
}
```

Điều kiện tồn tại
nhóm giá trị 0

Điều kiện phần tử dương
tạo nhóm với 0

Độ phức tạp của giải thuật:

Gọi m là max số phần tử dương khác nhau và số phần tử âm khác nhau.

Độ phức tạp của giải thuật là $O(m^2)$.

Lưu ý: Có thể thu hẹp miền xử lý dựa vào nhận xét sau: nếu cần tìm $\mathbf{x} = \mathbf{u} + \mathbf{v}$ thì chỉ cần xét với $\mathbf{u} \leq \mathbf{x}/2$ và $\mathbf{v} \geq \mathbf{x}/2$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "abplus."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
const int N = 50001;
int n,t,na,nb,ka=0,kb=0,a[N]={0},b[N]={0};
int64_t zr=0,d,ans=0;

int main()
{
    fi>>n;
    for(int i=0;i<n;++i)
    {
        fi>>t;
        if(t==0) {++zr; continue;}
        if(t>0) ++a[t], ++na; else ++b[-t], ++nb;
    }
    vector<int> va(na),vb(nb);
    for(int i=0; i<=N; ++i)
    {
        if(a[i]) va[ka++]=i;
        if(b[i]) vb[kb++]=i;
    }
    na=ka; nb=kb;
    int mxa=va[na-1], mxb=vb[nb-1];
    for(int i=0;i<na-1;++i)
        for(int j=i;j<na;++j)
        {
            if(i==j && a[i]==1) continue;
            t=va[i]+va[j];
            if(t>mxa) break;
            if(a[t]>0) if(i==j) ans+=a[vb[i]]*(a[vb[i]]-1);
                         else d=a[vb[j]], ans+=(d*a[vb[i]]*a[t])*2;
        }

    for(int i=0;i<nb-1;++i)
        for(int j=i;j<nb;++j)
        {
            if(i==j && b[i]==1) continue;
            t=vb[i]+vb[j];
            if(t>mxa) break;
            if(b[t]>0)
                if(i==j) ans+=b[vb[i]]*(b[vb[i]]-1);
                else d=b[vb[j]], ans+=(d*b[vb[i]]*b[t])*2;
        }

    for(int i=0;i<na;++)
        for(int j=0;j<nb;++)
        {
            t=va[i]-vb[j];
            if(t==0 && zr>0)

```

```

        {d=b[vb[j]]; ans+=(d*zr*a[vb[i]])*2; continue;}
    if(t>0 && a[t]>0)
        {d=b[vb[j]]; ans+=(d*a[t]*a[vb[i]])*2; continue;}
    if(t<0 && b[-t]>0)
        {d=a[vb[j]]; ans+=(d*b[-t]*b[vb[i]])*2; continue;}
}
if(zr>0)
{
    if(zr>2)ans+=zr*(zr-1);
    for(int i=0;i<na;++i)if(a[vb[i]]>1)ans+= zr*a[vb[i]]*2;
    for(int i=0;i<nb;++i)if(b[vb[i]]>1)ans+= zr*b[vb[i]]*2;
}

fo<<ans;
Times;
}

```



Cuộc cách mạng khoa học kỹ thuật lần thứ IV đã đưa tin học tới mọi ngõ nghách của cuộc sống. Một ngôn ngữ đơn giản được xây dựng để dạy và kiểm tra kiến thức thực hiện các phép cộng trừ cho học sinh lớp 1. Ngôn ngữ chỉ bao gồm 3 loại câu lệnh:

- ✚ **def** tên_bien giá_trị_nguyên_k
- ✚ **calc** biểu_thức =
- ✚ **clear**

Câu lệnh thứ nhất xác định giá trị biến là **k**, ví dụ **def bar 4**. Tên biến chỉ chứa ký tự la tinh thường và có độ dài không quá 30. Giá trị **k** nằm trong khoảng [-1000, 1000]. Trước và sau tên biến có đúng một dấu cách. Không có 2 biến nào có giá trị giống nhau.

Trong câu lệnh thứ 2 *biểu_thức* chứa các tên biến và các phép +, -. Câu lệnh đòi hỏi tìm tên biến có giá trị bằng giá trị *biểu_thức* đã nêu và ghi vào bên phải dấu bằng. Dấu phép tính tách khỏi tên biến bằng một dấu cách.

Câu lệnh thứ 3 xóa tất cả các phép xác định giá trị đã nêu trước đó.

Với mỗi câu lệnh thứ hai yêu cầu đưa ra đẳng thức dưới dạng *biểu_thức* = *tên_bien*. Nếu *biểu_thức* chứa biến chưa xác định giá trị hay giá trị *biểu_thức* không tương ứng với biến nào thì đưa ra đẳng thức với tên ở về phải là **unknown**.

Dữ liệu: Vào từ file văn bản EXPR.INP mỗi dòng chứa một câu lệnh thuộc một trong 3 dạng đã nêu. Tổng số lượng câu lệnh trong file dữ liệu là không quá 2 000.

Kết quả: Đưa ra file văn bản EXPR.OUT các đẳng thức tương ứng với câu lệnh loại hai, mỗi đẳng thức đưa ra trên một dòng.

Ví dụ:

EXPR.INP	EXPR.OUT
<pre>def foo 3 calc foo + bar = def bar 7 def programming 10 calc foo + bar = def is 4 def fun 8 calc programming - is + fun = def fun 1 calc programming - is + fun = clear def foo 4 def bar 6 def newid 10 calc foo + bar =</pre>	<pre>foo + bar = unknown foo + bar = programming programming - is + fun = unknown programming - is + fun = bar foo + bar = newid</pre>



Giải thuật: Xử lý xâu, Cấu trúc dữ liệu cơ sở.

Việc xử lý thông tin kết thúc theo dấu hiệu **eof ()** (*End_of_File*),

Việc xuống dòng và ghi tiếp thông tin vào đầu dòng mới trong C/C++ được thực hiện bằng ký tự **Xuống dòng** hoặc tổ hợp ký tự **Xuống dòng, Lùi về đầu dòng** vì vậy cần rất thận trọng khi đọc và xử lý từng ký tự.

Để tiện xử lý ta sẽ đọc thông tin của cả một dòng vào xâu s bằng câu lệnh **getline (fi , s)**, tuy vậy cần lưu ý là cờ *End_of_file* chỉ thực sự được kích hoạt ở lệnh đọc dòng tiếp theo và kết quả là xâu rỗng!

Việc nhận dạng câu lệnh được thực hiện dựa vào **ký tự thứ 2** của xâu (ký tự **s [1]**).

Thông tin về mỗi biến được lưu ở 2 nơi:

Cây **map<string, int> mp** – lưu ánh xạ *tên* → *giá trị*,

Mảng **string sid[20002]** – lưu ánh xạ *giá trị* → *tên*. Nếu số lượng biến hiện có là m thì độ phức tạp của việc tìm giá trị tương ứng là O(*lnm*), còn việc tìm tên theo giá trị có độ phức tạp O(1).

Lệnh **def** kéo theo việc bổ sung phần tử vào mp và cập nhật **sid**.

Lệnh **clear** sẽ xóa rỗng mp và xóa các tên trong **sid**.

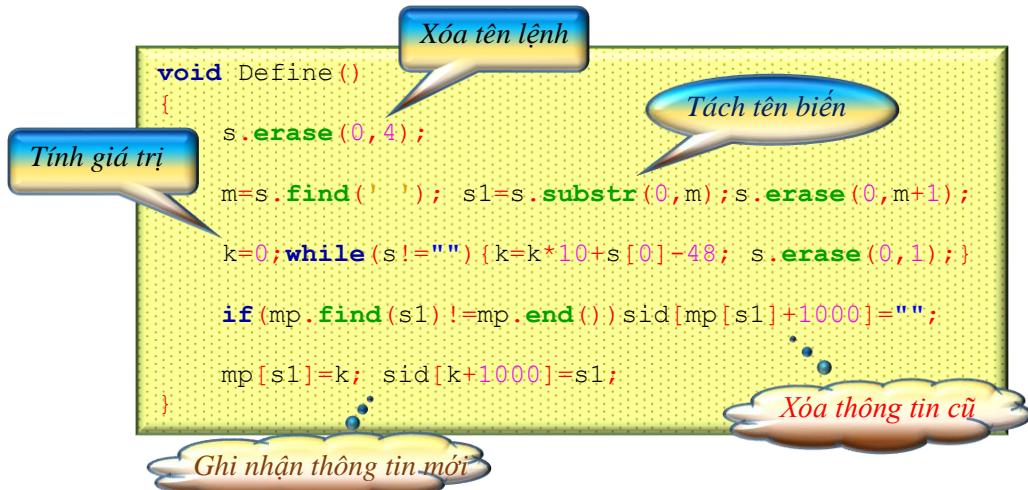
Lệnh **calc** thực hiện các tra cứu tên sang giá trị và ngược lại.

Tổ chức dữ liệu:

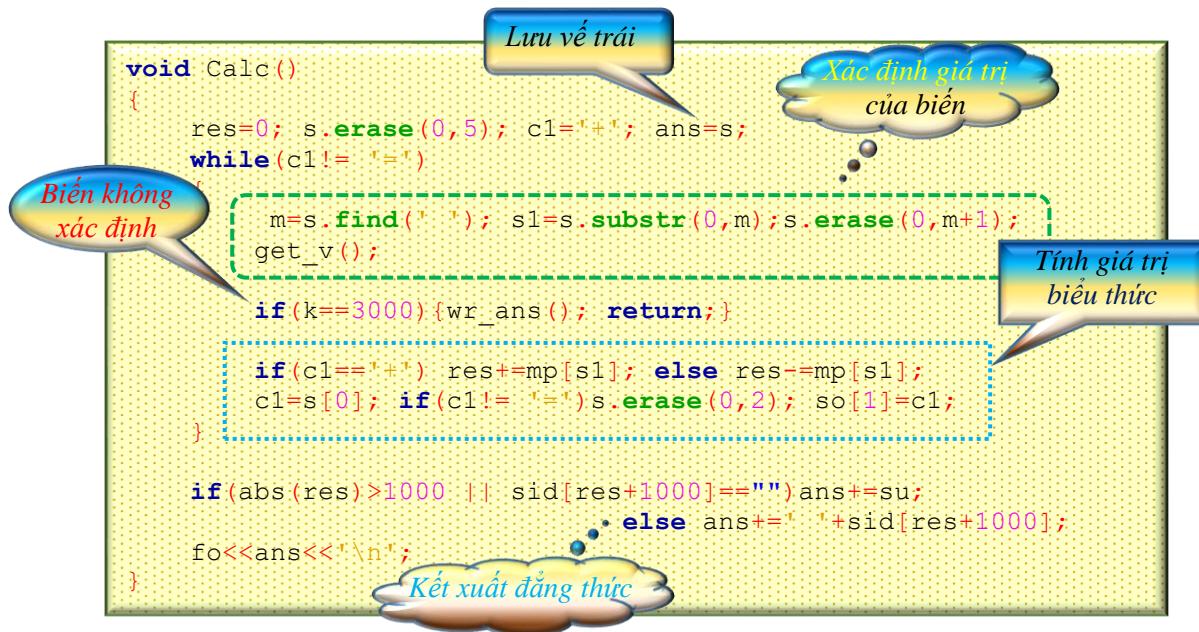
- Cây mp và mảng sid lưu thông tin như đã nêu ở trên,
- Các xâu s và ans lưu dòng cần xử lý và kết quả đưa ra.

Xử lý:

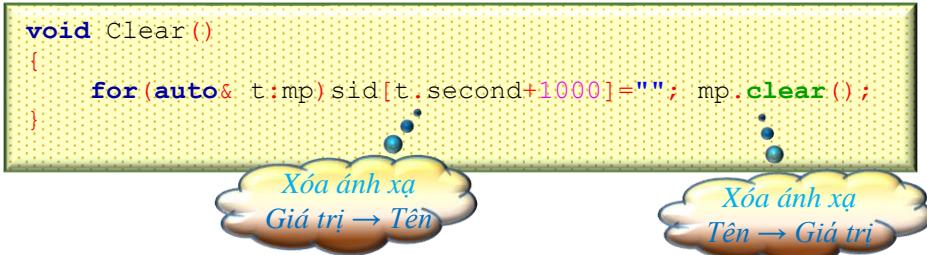
Xử lý câu lệnh **def**: cần phân biệt trường hợp biến mới và trường hợp giá trị mới cho biến cũ.



Xử lý lệnh **calc**:



Xử lý lệnh **clear**:



Độ phức tạp của giải thuật: $\approx O(m \ln m)$, trong đó m – số câu lệnh.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "expr."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const string su=" unknown";
map<string,int> mp;
string sid[2002]={" "}, so=" * ";
string s,s1,ans;
int k,res,t,m;
char c1,c2;

void Define()
{
    s.erase(0,4); m=s.find(' ');
    s1=s.substr(0,m); s.erase(0,m+1);
    k=0; while(s!="") {k=k*10+s[0]-48; s.erase(0,1);}
    if(mp.find(s1)!=mp.end()) sid[mp[s1]+1000]="";
    mp[s1]=k; sid[k+1000]=s1;
}

void get_v()
{
    if(mp.find(s1)==mp.end()) {k=3000; return;}
    k=mp[s1];
}

void wr_ans()
{
    ans+=su;
    fo<<ans<<'\\n';
}

void Calc()
{
    res=0; s.erase(0,5); c1='+'; ans=s;
    while(c1!= '=')
    {
        m=s.find(' '); s1=s.substr(0,m); s.erase(0,m+1);
        get_v();
        if(k==3000) {wr_ans(); return;}
        if(c1=='+') res+=mp[s1]; else res-=mp[s1];
        c1=s[0]; if(c1!=' ') s.erase(0,2); so[1]=c1;
    }
    if(abs(res)>1000 || sid[res+1000]== "") ans+=su;
    else ans+=' '+sid[res+1000];
    fo<<ans<<'\\n';
}

void Clear()
{
    for(auto& t:mp) sid[t.second+1000]="" ; mp.clear();
```

```
}
```



```
int main()
{
    while (!fi.eof())
    {
        getline(fi,s);
        if(s!="")
        {
            switch(s[1])
            {
                case 'e': {Define(); break;}
                case 'a': {Calc(); break;}
                case 'l': {Clear(); break;}
            }
        }
    }
    Times;
}
```



Cho xâu s chỉ chứa các ký tự trong tập $\{0, 1, ?\}$. Giả thiết trong s có m ký tự $?$. Nếu thay mỗi ký tự $?$ lần lượt bằng các ký tự 0 và 1 ta được 2^m xâu nhị phân chỉ chứa các ký tự trong tập $\{0, 1\}$.

Ta có thể biến đổi một xâu nhị phân để nhận được xâu có *thứ tự từ điển nhỏ hơn* bằng cách đổi chỗ hai ký tự cạnh nhau, tức là đổi chỗ 2 ký tự s_i và s_{i+1} . Một phép đổi chỗ được gọi là một bước. Sau một số bước biến đổi ta sẽ nhận được xâu nhị phân có thứ tự từ điển nhỏ nhất. Nếu xâu nhị phân ban đầu đã có thứ tự từ điển nhỏ nhất trong số các xâu nhị phân có cùng số lượng ký tự 0 và cùng số lượng ký tự 1 thì dĩ nhiên số phép biến đổi sẽ là 0 .

Ví dụ, với $s = "11010"$ ta cần 5 phép biến đổi để nhận được xâu tương ứng có thứ tự từ điển nhỏ nhất. Một trong số các cách biến đổi đó là như sau:

11010 → 10110 → 01110 → 01101 → 01011 → 00111

Hãy xác định tổng số lượng phép biến đổi cần thực hiện để đưa mỗi xâu trong số 2^m xâu nhận được về xâu tương ứng có thứ tự từ điển nhỏ nhất.

Dữ liệu: Vào từ file văn bản STEPS.INP gồm một dòng chứa xâu s có độ dài không quá 5×10^5 .

Kết quả: Đưa ra file văn bản STEPS.OUT một số nguyên – tổng số lượng phép biến đổi tìm được theo mô đun $10^9 + 7$.

Ví dụ:

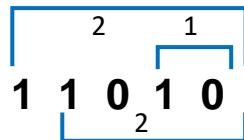
STEPS.INP	STEPS.OUT
10100?0?	39



Giải thuật: Tính tổ hợp.

Xâu nhị phân có thứ tự từ điển nhỏ nhất khi các ký tự 0 (nếu có) đứng ở đầu xâu và các ký tự 1 (nếu có) - ở cuối xâu.

Xét xâu chỉ chứa các ký tự 0 và 1, số bước chuyển cần thực hiện với mỗi ký tự 1 bằng số ký tự 0 đứng sau nó.



Xét xâu s độ dài n chỉ chứa các ký tự từ tập $\{0, 1, ?\}$ và trong đó có m ký tự $?$. Xâu này xác định tập X gồm 2^m xâu nhị phân độ dài n .

Xét ký tự s_i . Giả thiết đứng sau nó có q ký tự 1 và k ký tự $?$ ($k > 1$). Số lượng ký tự 0倜ng minh sẽ là $n-1-i-q-k = d-k$, trong đó $d = n-1-i-q$.

Khi thay mỗi ký tự $?$ bằng 0 và 1, số lượng số 0 mới ở sau vị trí i có thể thay đổi từ 0 đến k .

Số lượng 0 mới	Số lượng xâu
0	$C_k^0 = 1$
1	$C_k^1 = k$
2	C_k^2
3	C_k^3
...
$k-1$	C_k^{k-1}
k	C_k^k

Nếu $s_i = 0$ – không cần di chuyển,

Nếu $s_i = 1$ – cần di chuyển về cuối để tất cả các ký tự 1 tạo thành một nhóm liên tục cuối xâu.

Tổng số lượng phép di chuyển ở tất cả 2^k xâu sẽ là:

$$r = \sum_{j=0}^k (d-j) \times C_k^j$$

$$= d \times \sum_{j=0}^k C_k^j - \sum_{j=0}^k j \times C_k^j = d \times 2^k - k \times 2^{k-1} = (d \times 2^k) - (k \times 2^{k-1}).$$

Nếu $s_i = ?$ – cần tổng hợp 2 trường hợp **0** và **1** ở vị trí **i**.

Các giá trị **r** cũng như kết quả tổng hợp là rất lớn, vì vậy cần lấy mô đun theo $10^9 + 7$.

Cần lưu ý khi gặp ký tự **?** đầu tiên trong quá trình duyệt từ cuối xâu về đầu, giá trị **k** đang bằng 0, 2^{k-1} sẽ tham gia vào công thức như giá trị 1.

Ngoài cần lưu ý, nếu **?** đúng cuối xâu việc thay thế nó bằng **0** hay **1** đều không làm thay đổi số phép xử lý cần thực hiện.

Giá trị 2^{k-1} được sử dụng tăng dần theo **k**, vì vậy có thể tích lũy dần trong quá trình tính toán.

Tổ chức dữ liệu:

Chỉ cần lưu trữ xâu **s** và một số biến đơn lưu trữ kết quả trung gian, kết quả cuối cùng cần tìm.

Độ phức tạp của giải thuật: O(n) vì mỗi ký tự của s được xử lý với độ phức tạp O(1).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "steps."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int MD = 1000000007;

string s;
int n,d,ic,q=0,k=0,p=1;
int64_t r,res=0;

int main()
{
    fi>>s; n=s.size(); ic=n-1;

    for(int i=n-1;i>=0;--i)
    {
        if(s[i]=='0') continue;
        d=ic-i-q;
        if(k==0 && s[i]=='?' && i!=ic) r=(d*2-1)*p%MD; else r=(d*2-k)*p%MD;
        if(s[i]=='1') {res=(res+r)%MD; ++q; }
        else {res=(2*res+r)%MD; ++k; }
        if(k>1) p=(p*2)%MD;
    }

    fo<<res;
}
```



Công nghệ in 3D đã phát triển tới mức nhiều máy móc, thiết bị có thể sản xuất trực tiếp bằng máy in 3D.

Phòng thí nghiệm Công nghệ cao nhận được đơn đặt hàng sản xuất n máy phun thuốc cao áp và yêu cầu hoàn thành trong thời hạn ngắn nhất có thể để phục vụ ngăn chặn một bệnh dịch đang có nguy cơ lan rộng. Máy phun thuốc có thể sản xuất theo công nghệ in 3D, mỗi ngày một máy in 3D có thể cho ra một máy phun thuốc. Hiện tại trong Phòng thí nghiệm chỉ có một máy in 3D. Tuy nhiên máy in 3D có thể tự sản xuất ra máy in 3D mới với công suất một máy mỗi ngày. Máy in mới có thể đưa vào khai thác ngày hôm sau và có thể sử dụng để chế tạo máy phun hoặc chế tạo máy in mới.

Hãy xác định số ngày ít nhất cần thiết để sản xuất không ít hơn n máy phun thuốc cao áp.

Dữ liệu: Vào từ file văn bản PRN3D.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 10^{15}$).

Kết quả: Đưa ra file văn bản PRN3D.OUT một số nguyên – số ngày ít nhất cần thiết để thực hiện đơn đặt hàng.

Ví dụ:

PRN3D.INP	PRN3D.OUT
5	4



Giải thuật: Phương pháp trượt nhanh nhất (Steepest descent).

Nếu chỉ sử dụng một máy in để sản xuất máy phun cao áp thì cần **n** ngày,

Nếu **n** > 3 thì khi có thêm máy in, thời gian hoàn thành đơn đặt hàng sẽ giảm,

Các máy in 3D mới cần sản xuất ngay từ những ngày đầu để tận dụng tối đa hiệu quả của mỗi máy in,

Không cần thiết tính toán chính xác số lượng máy in cần có để ngày cuối cùng cho ra đúng n máy phun cao áp với sự tham gia sản xuất của tất cả máy in 3D hiện có lúc đó, tức là ở ngày cuối cùng có thể không cần sử dụng hết mọi máy in (điều này có thể dễ dàng chứng minh).

Như vậy, **k** ngày đầu tiên chỉ dành để chế tạo máy in, những ngày còn lại – sản xuất máy phun cao áp.

Số lượng máy in chế tạo được trong **k** ngày là 2^k , số ngày cần thiết để hàn thành đơn đặt hàng là $(n+2^k-1)/2^k+k$.

Xuất phát từ thời gian thực hiện ban đầu **res** = **n**, lần lượt tính lại **res** theo **k** chừng nào **res** còn không tăng.

Độ phức tạp của giải thuật: O(lnn).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "prn3d."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int64_t n,p=1,d;
int64_t res;

int main()
{
    fi>>n; res=n;
    for(int i=1;i<n;++)
    {
        p<<=1;
        d=(n+p-1)/p+i;
        if(d<res)res=d;
        if(d>res)break;
    }
    fo<<res;
    Times;
}
```



Toán và Tin có quan hệ chặt chẽ với nhau, vì vậy Câu lạc bộ học sinh giỏi Toán và Câu lạc bộ học sinh giỏi Tin của nhà trường cũng thường có các buổi sinh hoạt chung, giao lưu với nhau. Trong các buổi giao lưu, ngoài phần trao đổi về học thuật các bạn còn thường thi giải một số bài toán giữa 2 đội Toán và Tin, mỗi bài toán đòi hỏi sự phân công công việc và phối hợp hoạt động một cách tốt nhất giữa các thành viên trong đội.

Bài thi hôm nay là cho số nguyên dương n . Mỗi đội phải đưa ra kết quả phân tích số n thành một tích dưới dạng $n = (\mathbf{y}_1+1)(\mathbf{y}_2+1)\dots(\mathbf{y}_k+1)$, trong đó \mathbf{y}_i là các số nguyên dương, khác nhau từng đôi một. Số điểm nhận được sẽ là k . Ví dụ, với $n = 1024$, một đội có thể đưa ra kết quả là $1024 = 2 \times 512$ và được 2 điểm, đội khác đưa ra kết quả là $2 \times 16 \times 32$ và được 3 điểm. Trên thực tế, điểm cao nhất có thể nhận được là 4 ($1024 = 2 \times 4 \times 8 \times 16$).

Với n cho trước, hãy xác định số điểm tối đa có thể đạt.

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

- ✚ Dòng đầu tiên chứa một số nguyên q – số lượng tests ($1 \leq q \leq 100$),
- ✚ Mỗi dòng trong q dòng sau chứa một số nguyên n ($10^3 \leq n \leq 10^{15}$).

Kết quả: Đưa ra file văn bản LIAISE.OUT các số điểm tối đa có thể đạt, mỗi số trên một dòng.

Ví dụ:

LIAISE.INP	LIAISE.OUT
4	8
1099511627776	3
127381	10
497664000	11
12441600000	



VV42 SpC 2010

Giải thuật: Phân tích ra thừa số nguyên tố, Đếm tổ hợp.

Cần phân tích mỗi số n nguyên dương đã cho thành tích của nhiều thừa số lớn hơn 1 khác nhau nhất,

Có nhiều cách phân tích, nhưng yêu cầu đặt ra là chỉ cần xác định số lượng thừa số, Vì n là đủ lớn nên việc phân tích ra thừa số nguyên tố cần thực hiện theo giải thuật Leman,

Giả thiết $n = a_1^{p_1} \times a_2^{p_2} \times \dots \times a_m^{p_m}$, trong đó a_i là các số nguyên tố, $i = 1 \div m$.

Do $n \leq 10^{15}$ nên $\sum_{i=1}^m p_i < 64$.

Xét các thừa số tham gia vào kết quả phân tích cần tìm hình thành từ một số nguyên tố a_i . Các thừa số đó là $a_i, a_i^2, a_i^3, \dots, a_i^{k_i}$, trong đó $k_i = \max\{j \mid \sum_{i=1}^j i \leq p_i\}$. Như vậy còn $p_i - k_i$ thừa số a_i chưa tham gia vào kết quả phân tích.

Như vậy, trong tích kết quả, nếu mỗi thừa số là lũy thừa của một số nguyên tố (trong số các số tìm được khi phân tích n ra số nguyên tố), ta có $\text{ans} = \sum_{i=1}^m k_i$.

Bậc còn lại của các thừa số nguyên tố là $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m)$, trong đó $\mathbf{x}_i = \mathbf{p}_i - \mathbf{k}_i$.

Dễ dàng chứng minh là $\mathbf{x}_i \leq 10$.

Nếu tồn tại, các thừa số khác nhau và khác nhau những thừa số đã xác định có dạng $a_i^u a_j$ hoặc $a_i^u a_j a_v$, $i, j, v = 1 \div m$, $i \neq j$, $i \neq v$, $j \neq v$, $1 \leq u \leq \mathbf{x}_i$.

Tổ chức dữ liệu:

- Mảng động `vector<uint64_t>` `divisors` – lưu kết quả phân tích n ra thừa số nguyên tố,
- Mảng `int` `p[65]` – lưu bậc của các thừa số nguyên tố.

Xử lý:

Phân tích n ra thừa số nguyên tố theo giải thuật Leman,

Tính bậc của mỗi thừa số nguyên tố:

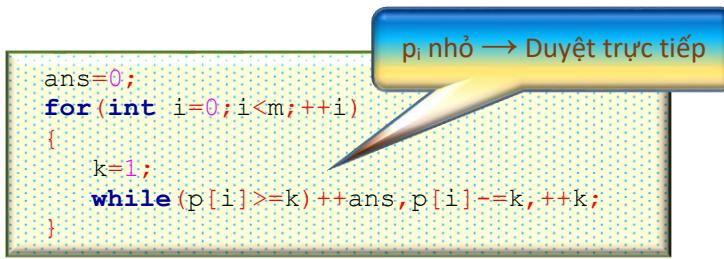
Hàng rào

```

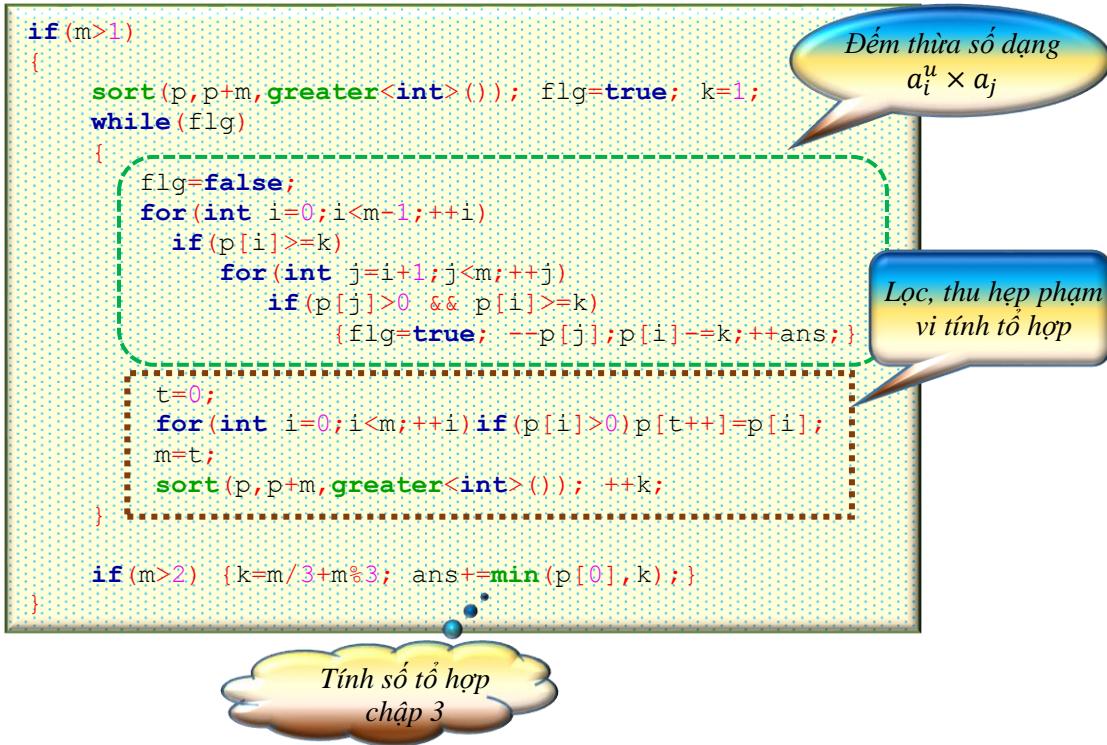
a=divisors[0]; b=0; m=0; divisors.push_back(0);
for (int i = 0; i < divisors.size(); ++i)
{
    if(a==divisors[i])++b;
    else { p[m++]=b;a=divisors[i]; b=1; }
}

```

Tính số lượng thừa số là lũy thừa của một số nguyên tố:



Tính số thừa số là tổ hợp chap 2 và chap 3:



Độ phức tạp của giải thuật: $O(q \times n^{1/3})$.

Chương trình

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

vector<uint64_t> find_prime_divisors_leman(uint64_t n)
{
    vector<uint64_t> divisors;
    uint64_t sqrt3_n_up = ceil(pow((long double)n, (long double)1.0 / 3)) + 2;

    // finding divisors <= n ^ (1 / 3)
    uint64_t m = 2;
    while (m * m <= n && m < sqrt3_n_up) {
        while (n % m == 0) {
            divisors.push_back(m);
            n /= m;
        }
        ++m;
    }
    if (m * m > n)
    {
        if (n != 1)
            divisors.push_back(n);
        return divisors;
    }

    // either n is prime or n = p * q, n ^ (1 / 3) < p <= q < n ^ (2 / 3)
    long double sqrt6_n = pow((long double)n, (long double)1.0 / 6);
    long double sqrt_n = sqrt((long double)n);
    for(int k = 1; k < sqrt3_n_up; ++k) {
        long double sqrt_k = sqrt((long double)k);
        uint64_t sqrt_4nk = ceil(2 * sqrt_k * sqrt_n) + 2;

        // diff = ([sqrt(4nk)]^2 - 4nk)
        uint64_t diff = sqrt_4nk * sqrt_4nk - 4 * k * n;
        while (diff >= 2 * sqrt_4nk - 1) {
            diff = diff - 2 * sqrt_4nk + 1;
            --sqrt_4nk;
        }
        int max_d = ceil(sqrt6_n / (4 * sqrt_k)) + 3;
        for (int d = 0; d < max_d; ++d) {
            uint64_t a = sqrt_4nk + d;
            uint64_t b = round(sqrt((long double)diff));
            if (b * b == diff) {
                // diff is a perfect square, a ^ 2 = b ^ 2 mod n
                uint64_t d1 = __gcd(a + b, n);
                uint64_t d2 = __gcd(a - b, n);
                if (1 < d1 && d1 < n || 1 < d2 && d2 < n) {
                    uint64_t p = (1 < d1 && d1 < n) ? d1 : d2;
                    divisors.push_back(p);
                    divisors.push_back(n / p);
                }
            }
        }
    }
    return divisors;
}
```

```

        }
    }
    diff += 2 * a + 1;
}
}

// n is prime
divisors.push_back(n);
return divisors;
}

int main() {
    uint64_t n, m, q;
    fi >> q;
    for (int i = 0; i < q; ++i)
    {
        fi >> n;
        vector<uint64_t> divisors = find_prime_divisors_leman(n);
        int k, d[65], p[65], a, b, t, ans;
        bool flg;

        a = divisors[0]; b = 0; m = 0; divisors.push_back(0);
        for (int i = 0; i < divisors.size(); ++i)
        {
            if (a == divisors[i]) ++b;
            else { d[m] = a; p[m++] = b; a = divisors[i]; b = 1; }
        }

        sort(p, p + m); ans = 0;
        for (int i = 0; i < m; ++i)
        {
            k = 1;
            while (p[i] >= k) ++ans, p[i] -= k, ++k;
        }
        k = 0; for (int i = 0; i < m; ++i) if (p[i] > 0) p[k++] = p[i]; m = k;
        if (m > 1)
        {
            sort(p, p + m, greater<int>()); flg = true; k = 1;
            while (flg)
            {
                flg = false;
                for (int i = 0; i < m - 1; ++i)
                    if (p[i] >= k)
                        for (int j = i + 1; j < m; ++j)
                            if (p[j] > 0 && p[i] >= k)
                                { flg = true; --p[j]; p[i] -= k; ++ans; }

                t = 0;
                for (int i = 0; i < m; ++i) if (p[i] > 0) p[t++] = p[i]; m = t;
                sort(p, p + m, greater<int>()); ++k;
            }
            if (m > 2) { k = m / 3 + m % 3; ans += min(p[0], k); }
        }
        fo << ans << '\n';
    }

    Times;
}

```



ICPC là cuộc thi lập trình đồng đội giữa các trường đại học trên thế giới, được tổ chức hàng năm và thường được gọi tắt là thi ACM. Mỗi đội tham gia có 3 người, cùng giải chung một số bài toán. Đội thi có thể nộp bài tùy chọn và thời điểm bất kỳ để chấm. Thông báo phản hồi của hệ thống chấm là *Đúng (right)* hoặc *Sai (wrong)*. Hệ thống lưu lại thời điểm nộp bài, tên bài và kết quả đã phản hồi. Thời điểm nộp bài là số nguyên, xác định đó là phút thứ máy tính từ đầu cuộc thi. Trong biên bản, tên bài là các chữ cái la tinh in hoa. Mỗi lần nộp bài tương ứng với một dòng biên bản.

Với mỗi lần có kết quả sai, thời gian giải bài đó bị cộng thêm đại lượng phạt là 20. Nếu kết quả đúng thì thời gian giải bài đó được tính bằng tổng thời điểm lúc nạp chấm được kết quả đúng với tổng giá trị phạt (nếu có) của bài đó.

Kết quả chung cuộc của một đội là số lượng bài giải đúng và tổng thời gian giải các bài đó. Các đội được xếp hạng theo số lượng bài giải được. Nếu 2 đội có cùng số lượng bài giải được thì đội nào có tổng thời gian giải nhỏ hơn sẽ đứng trên.

Cho biên bản nộp bài của một đội. Hãy xác định số bài giải được và tổng thời gian giải những bài đó.

Dữ liệu: Vào từ file văn bản ACM.INP, mỗi dòng tương ứng với một dòng của biên bản, các dòng được đưa ra theo thứ tự tăng dần của thời điểm nộp bài, thời điểm nộp không quá 300. Dữ liệu vào kết thúc bằng dòng chứa một số -1.

Kết quả: Đưa ra file văn bản ACM.OUT trên một dòng 2 số nguyên – số bài giải được và tổng thời gian giải những bài đó.

Ví dụ:

ACM.INP	ACM.OUT
3 E right 10 A wrong 30 C wrong 50 B wrong 100 A wrong 200 A right 250 C wrong 300 D right -1	3 543



Giải thuật: Thống kê đơn giản.

Sử dụng một mảng nguyên để tích lũy thời gian phạt,

Nhận dạng kết quả phản hồi: theo ký tự đầu của xâu.

Một trong số các cách xử lý dữ liệu theo giá trị đặc biệt của biến báo kết thúc dữ liệu cần xử lý: Tạo chu trình với số lần lặp vô hạn và thoát ra khỏi chu trình khi gặp giá trị đặc biệt.

Độ phức tạp của giải thuật: O(n).

Chương trình

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

int k=0,ans=0,t,v[27]={0};
char c;
string s;

int main()
{
    while(1)
    {
        fi>>t; if(t<0) break;
        fi>>c>>s;
        if(s[0]=='w') v[c-65]+=20;
        else ans+=v[c-65]+t, ++k;
    }
    fo<<k<<' '<<ans;
    Times;
}
```



GIẢI THUẬT MANAKER TÌM PALINDROME

Bài toán

Cho xâu s độ dài n . Xâu u tạo thành từ dãy các ký tự liên tiếp nhau của s được gọi là xâu con của s . Hai xâu con u và v được gọi là khác nhau nếu tồn tại ít nhất một i để ký tự s_i tham gia vào u và không tham gia vào v hoặc tham gia vào v và không tham gia vào u .

Hãy xác định số lượng xâu con độ dài lớn hơn 1 là palindrome của s .

Giải thuật

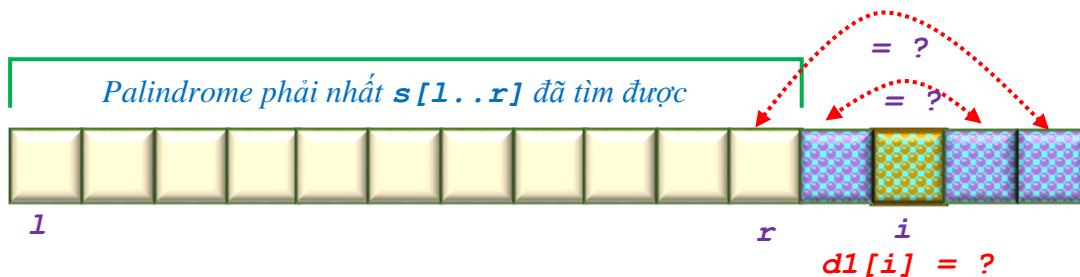
Xét xâu con $u = s[i..j]$. Nếu $j-i+1$ là lẻ thì $p = (i+j)/2$ được gọi là vị trí trung tâm (gọi ngắn gọn là *tâm*) của u , nếu $j-i+1$ chẵn thì tâm là $p = (i+j+1)/2$.

Gọi $d1[i]$ là số lượng các xâu con palindrome độ dài lẻ lớn hơn 1, có tâm là i , $d2[i]$ là số lượng các xâu con palindrome độ dài chẵn lớn hơn 1, có tâm là i . Giả thiết $s[1..r]$ là palindrome phải nhất trong số các palindrome tìm được.

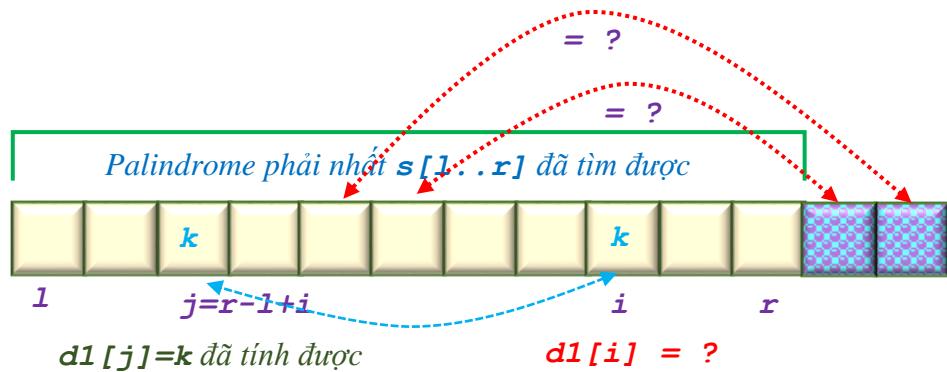
Giả thiết $d1[j]$ và $d2[j]$ đã tính được với mọi $j < i$.

Xét cách tính $d1[i]$.

Có 2 trường hợp xảy ra:



- ✚ $i > r \rightarrow$ so sánh trực tiếp s_{i-1} với s_{i+1} , s_{i-2} với s_{i+2}, \dots để tìm ra palindrome lớn nhất có tâm là i .
- ✚ $i \leq r \rightarrow s_i$ thuộc palindrome đã xác định. Do tính đối xứng của palindrome, nếu không tính đến giá trị r , ta có $d1[r-1+i] = k = d1[i]$, nhưng phần còn lại từ i đến r có thể nhỏ hơn k . Khả năng có thể mở rộng biên r được kiểm tra bằng cách so sánh trực tiếp các ký tự tiếp theo.



Trong mọi trường hợp, sau khi tính **d1 [i]** cần cập nhật **l** và **r**.

Hàm tính **d1**:

```
vector<int> calc_1()
{
    vector<int> d(n, 0);
    int l=0, r=-1;
    for(int i=0; i<n; ++i)
    {
        int k=0;
        if(i<=r) k=min(r-i, d[r-i+1]);
        while(i+k+1<n && i-k-1>=0 && s[i+k+1] == s[i-k-1]) ++k;
        d[i]=k;
        if(i+k>r) l=i-k, r=i+k;
    }
    return d;
}
```

Với sự điều chỉnh chỉ số thích hợp, ta có hàm tính **d2**:

```
vector<int> calc_2()
{
    vector<int> d(n, 0);
    int l=0, r=-1;
    for(int i=0; i<n; ++i)
    {
        int k=0;
        if(i<=r) k=min(r-i+1, d[r-i+1+1]);
        while(i+k+1<n && i-k-1>=0 && s[i+k] == s[i-k-1]) ++k;
        d[i]=k;
        if(i+k-1>r) l=i-k, r=i+k-1;
    }
    return d;
}
```

Các palindrome đếm được sẽ không bị lặp hoặc có tâm khác nhau hoặc khác nhau về độ dài với các palindrome cùng tâm.

Đánh giá độ phức tạp

Để tính $d1[i]$ cần duyệt với $i = 0 \div n-1$,

- ⊕ Với $i > r$, chu trình lặp ở trong thực hiện bao nhiêu lần thì r sẽ tăng lên bấy nhiêu,
- ⊕ Với $i \leq r$ có thể xảy ra 2 trường hợp:
 - ⊖ $i+d1[j]-1 \leq r \rightarrow$ chu trình trong sẽ có số lần lặp bằng không,
 - ⊖ $i+d1[j]-1 > r \rightarrow$ chu trình lặp ở trong thực hiện bao nhiêu lần thì r sẽ tăng lên bấy nhiêu.

Như vậy, r tăng tuyến tính theo số lần lặp của chu trình trong. r không thể vượt quá $n-1$ vì vậy độ phức tạp của giải thuật sẽ là $O(n)$.



VV44. SỐ LƯỢNG PALINDROME

Tên chương trình: ALL_PAL.CPP

Cho xâu s độ dài n . Người ta có thể xóa một số ký tự liên tiếp (có thể là 0) ở đầu xâu và xóa một số ký tự liên tiếp (có thể là 0) ở cuối xâu, nhưng không xóa rỗng xâu. Hai cách xóa gọi là khác nhau nếu tồn tại ít nhất một ký tự bị xóa ở cách thứ nhất nhưng không bị xóa ở cách thứ hai. Lưu ý rằng xâu chỉ chứa một ký tự cũng là xâu palindrome.

Hãy xác định số cách xóa khác nhau để nhận được xâu palindrome.

Dữ liệu: Vào từ file văn bản ALL_PAL.INP gồm một dòng chứa xâu s độ dài không quá 10^6 . Các ký tự trong s đều có mã ASCII lớn hơn 32.

Kết quả: Đưa ra file văn bản ALL_PAL.OUT một số nguyên – số cách xóa khác nhau có thể thực hiện.

Ví dụ:

ALL_PAL.INP	ALL_PAL.OUT
aabacaba	14



VV44

Giải thuật: Sơ đồ lặp (Quy hoạch động đơn giản), Vết cạn.

Mỗi cách xóa cho ta một xâu con khác nhau các ký tự liên tiếp của **s**,

Bài toán dẫn về việc xác định có bao nhiêu xâu con là palindrome.

Gọi **d1 [i]** là số lượng các xâu con palindrome khác nhau độ dài lẻ lớn hơn 1, có tâm là **i**, **d2 [i]** là số lượng các xâu con palindrome khác nhau độ dài chẵn lớn hơn 1, có tâm là **i**, **n** là độ dài xâu **s**.

Kết quả cần tìm **ans = n + $\sum_{i=0}^{n-1} (d1_i + d2_i)$** .

Theo giải thuật Manacher, **d1_i** và **d2_i**, **i = 0 ÷ n-1** có thể tính với độ phức tạp O(n).

Bản chất giải thuật tính **d1_i** và **d2_i** là sơ đồ tính lặp kết hợp với việc vết cạn bộ phận khi cần thiết,

Các mảng **d1** và **d2** không cần lưu trữ vì chỉ được sử dụng với một mục đích – tích lũy kết quả.

Độ phức tạp của giải thuật: O(n).

Lưu ý: Các hàm tính **d1** và **d2** chỉ khác nhau một đôi chỗ ở cách xác định chỉ số. Có thể gộp 2 hàm thành một bằng cách truyền tham số xác định cách tính chỉ số. Tuy vậy việc này sẽ làm tăng **độ phức tạp lập trình** một cách không cần thiết.

Chương trình

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

int n;
string s;int64_t ans;

void calc_1()
{
    vector<int> d(n, 0);
    int l=0, r=-1;
    for(int i=0; i<n; ++i)
    {
        int k=0;
        if(i<=r) k=min(r-i, d[r-i+1]);
        while(i+k+1<n && i-k-1>=0 && s[i+k+1] == s[i-k-1]) ++k;
        d[i]=k;
        if(i+k>r) l=i-k, r=i+k;
    }
    for(int i=0; i<n; ++i) ans+=d[i];
    return ;
}

void calc_2()
{
    vector<int> d(n, 0);
    int l=0, r=-1;
    for(int i=0; i<n; ++i)
    {
        int k=0;
        if(i<=r) k=min(r-i+1, d[r-i+1+1]);
        while(i+k<n && i-k-1>=0 && s[i+k] == s[i-k-1]) ++k;
        d[i]=k;
        if(i+k-1>r) l=i-k, r=i+k-1;
    }
    for(int i=0; i<n; ++i) ans+=d[i];
    return;
}

int main()
{
    fi>>s;
    n=s.size(); ans=n;
    calc_1();
    calc_2();
    fo<<ans;
    Times;
}
```



Tên đầy đủ các thành phố ở một số nước là rất dài. Ví dụ, tên đầy đủ của thành phố Bangkok, thủ đô Thái Lan phải ghi thành 2 cuốn sách. Dĩ nhiên, khó có ai nhớ tên đầy đủ. Mọi người, nhất là khách du lịch, chỉ nhớ một phần của tên. Nói một cách khác, nếu tên đầy đủ là xâu s , thì người ta chỉ nhớ một xâu con t các ký tự liên tiếp nhau của s . Những người khác nhau thì nhớ tên dưới dạng các xâu con khác nhau, nhưng có một điểm chung là xâu con càng chứa trong đó nhiều xâu con là palindrome thì càng được nhiều người nhớ.

Kinh nghiệm cho thấy số người sử dụng tên t thường tương ứng với số xâu con là palindrome của t . Báo cáo tổng hợp các phiếu đăng ký du lịch trong năm cho thấy có m tên được sử dụng, mỗi tên tương ứng với 2 số nguyên lf và rt ($lf \leq rt$) xác định xâu con $s[lf..rt]$. Từ đây người ta có thể ước lượng được số khách du lịch thông qua việc tính số người nhớ và sử dụng nó ở mỗi tên.

Với mỗi tên được sử dụng hãy xác định số người nhớ tên này.

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

- ✚ Dòng đầu tiên chứa xâu s chỉ bao gồm các chữ cái la tinh thường độ dài n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa số nguyên m ($1 \leq m \leq 10^5$),
- ✚ Mỗi dòng trong m dòng sau chứa 2 số nguyên lf và rt ($1 \leq lf \leq rt \leq n$).

Kết quả: Đưa ra file văn bản MEMORABLE.OUT, với mỗi tên xác định theo dữ liệu đã nêu đưa ra một số nguyên – số người nhớ tên đó, mỗi số trên một dòng.

Ví dụ:

MEMORABLE.INP
aabacab
5
1 7
1 4
3 7
2 5
5 7

MEMORABLE.OUT
11
6
7
5
3



Giải thuật: Sơ đồ lặp (Quy hoạch động đơn giản), Vết cạn, Tổng tiền tố.

Mỗi palindrome tương ứng với một tâm của nó (xem Giải thuật Manaker), Số lượng palindrome nhận một ký tự là tâm được gọi là **trọng số** của tâm, Để trả lời các truy vấn trước hết phải tìm **tâm** của các palindrome trong **s** cùng với **trọng số** của nó,

Xâu con **s[lf..rt]** chứa một palindrome khi và chỉ khi nó chứa cả điểm bắt đầu (**điểm vào**) và điểm kết thúc (**điểm ra**) của palindrome,

Một tâm có trọng số khác 1 sẽ chứa các palindrome **lồng nhau** (một palindrome nằm gọn trong palindrome khác) với **các điểm vào ở các vị trí liên tiếp nhau** trước tâm và **các điểm ra ở các vị trí liên tiếp nhau** sau tâm (với palindrome độ dài lẻ) hoặc bắt đầu từ tâm (với palindrome độ dài chẵn).

Điều này cho phép ta tính số điểm vào và số điểm ra chứa trong **s[lf..rt]** và **min** của 2 giá trị tính được sẽ là số palindrome chứa trong xâu con này.

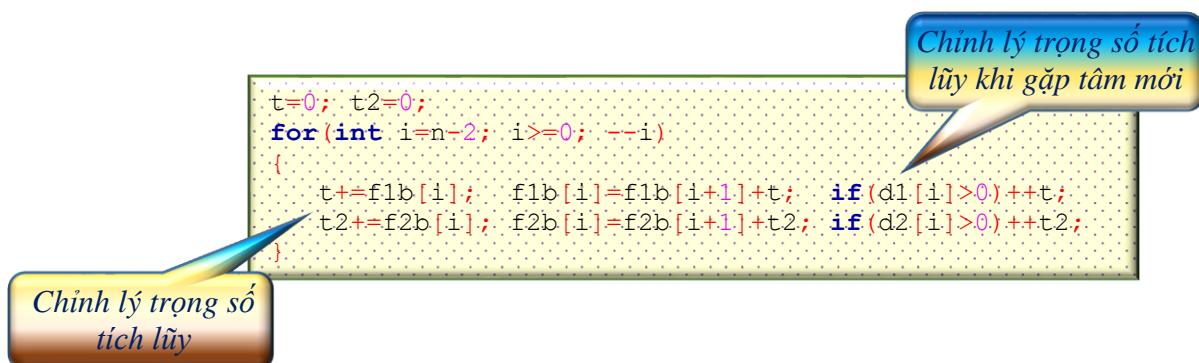
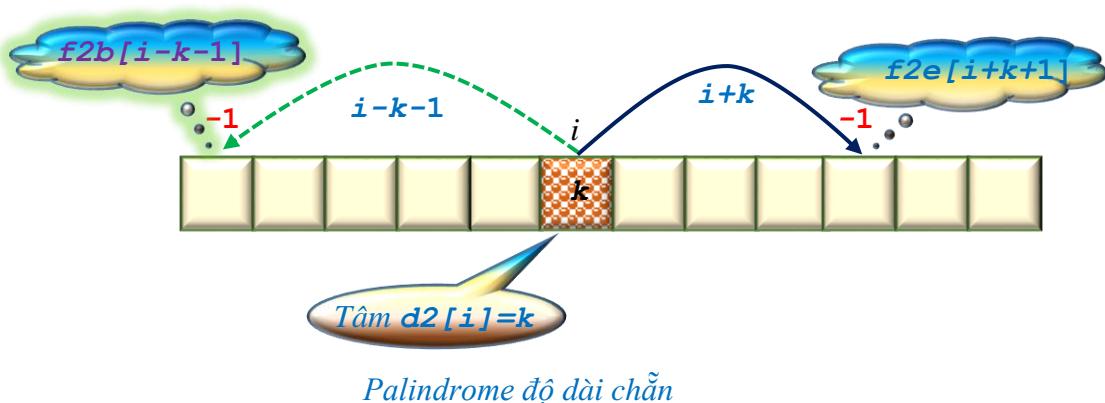
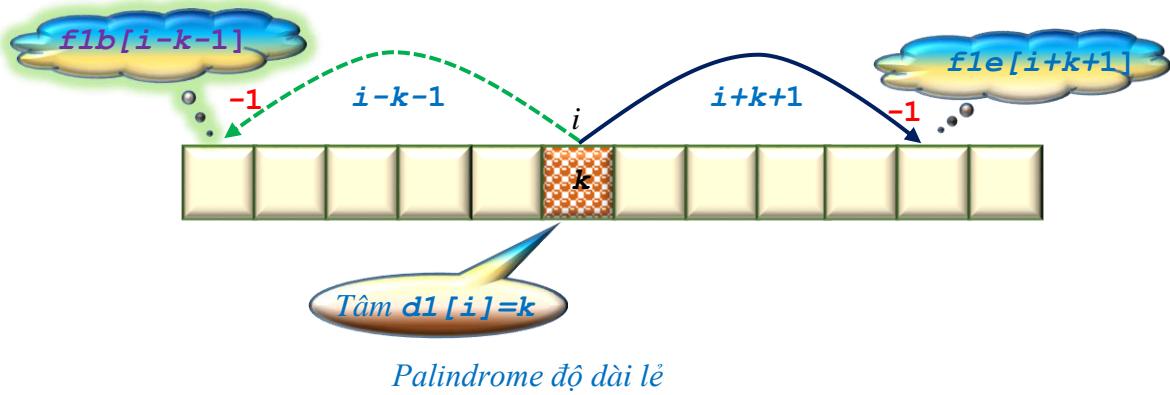
Tổ chức dữ liệu:

- Mảng **vector<int>** **d1(n)** – lưu trọng số tâm các palindrome độ dài lẻ,
- Mảng **vector<int>** **d2(n)** – lưu trọng số tâm các palindrome độ dài chẵn,
- Các mảng **vector<int>** **f1b(n+1, 0), f1e(n+1, 0)** – tích lũy tần số xuất hiện các điểm vào và ra của palindrome độ dài lẻ,
- Các mảng **vector<int>** **f2b(n+1, 0), f2e(n+1, 0)** – tích lũy tần số xuất hiện các điểm vào và ra của palindrome độ dài chẵn.

Xử lý:

Nhập xâu **s** và tính giá trị các mảng **d1, d2**: theo giải thuật Manaker đã trình bày ở trên,

Đánh dấu các điểm vào/ra:

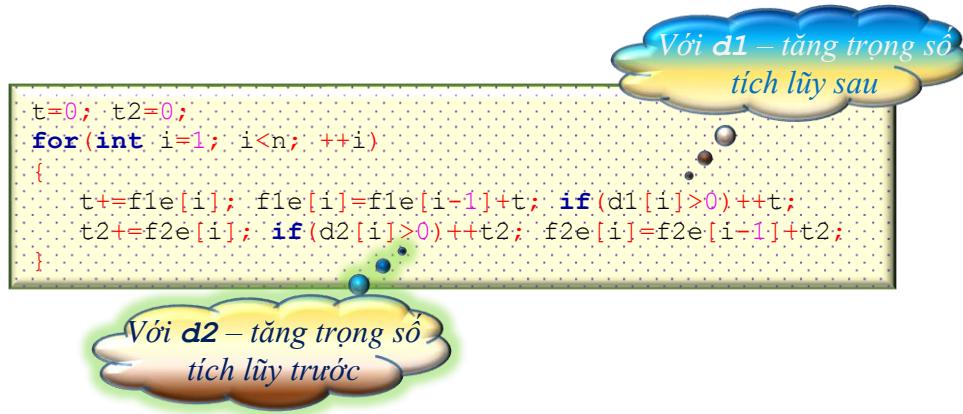


Tính tổng tiền tố các điểm ra:

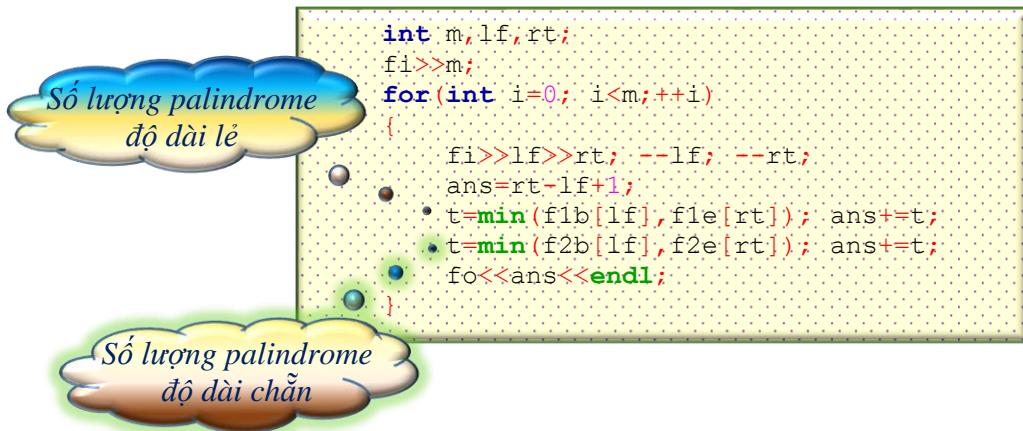
```
for(int i=0; i<n; ++i)
{
    if(d1[i]>0) {--f1e[i+d1[i]+1]; t=i-d1[i]-1; if(t>=0)--f1b[t];}
    if(d2[i]>0) {--f2e[i+d2[i]]; t=i-d2[i]-1; if(t>=0)--f2b[t];}
}
```

Lưu ý: *Ở đầu trái kiểm tra điều kiện tồn tại sẽ thuận tiện hơn việc tổ chức rào chẵn.*

Tính tổng tiền tố các điểm vào:



Xử lý truy vấn:



Độ phức tạp của giải thuật:

Độ phức tạp của việc tính trọng số các tâm: $O(n)$ (*Giải thuật Manacher*),

Độ phức tạp xử lý một truy vấn: $O(1)$,

Độ phức tạp chung: $O(n+m)$.

Chương trình

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

int n,t,t2;
string s;int64_t ans;

vector<int> calc_1()
{
    vector<int> d(n,0);
    int l=0, r=-1;
    for(int i=0;i<n;++i)
    {
        int k=0;
        if(i<=r) k=min(r-i,d[r-i+1]);
        while(i+k+1<n && i-k-1>=0 && s[i+k+1] == s[i-k-1]) ++k;
        d[i]=k;
        if(i+k>r) l=i-k, r=i+k;
    }
    return d;
}

vector<int> calc_2()
{
    vector<int> d(n,0);
    int l=0, r=-1;
    for(int i=0;i<n;++i)
    {
        int k=0;
        if(i<=r) k=min(r-i+1,d[r-i+1]);
        while(i+k<n && i-k-1>=0 && s[i+k] == s[i-k-1]) ++k;
        d[i]=k;
        if(i+k-1>r) l=i-k, r=i+k-1;
    }
    return d;
}

int main()
{
    fi>>s; n=s.size();
    vector<int> d1(n),d2(n);
    vector<int>f1b(n+1,0),f1e(n+1,0),f2b(n+1,0),f2e(n+1,0);
    d1=calc_1();
    d2=calc_2();

    for(int i=0;i<n;++i)
    {
        if(d1[i]>0) {--f1e[i+d1[i]+1]; t=i-d1[i]-1; if(t>=0)--f1b[t];}
        if(d2[i]>0) {--f2e[i+d2[i]]; t=i-d2[i]-1; if(t>=0)--f2b[t];}
    }
}
```

```

t=0; t2=0;
for(int i=1; i<n; ++i)
{
    t+=f1e[i]; f1e[i]=f1e[i-1]+t; if(d1[i]>0)++t;
    t2+=f2e[i]; if(d2[i]>0)++t2; f2e[i]=f2e[i-1]+t2;
}
t=0; t2=0;
for(int i=n-2; i>=0; --i)
{
    t+=f1b[i]; f1b[i]=f1b[i+1]+t; if(d1[i]>0)++t;
    t2+=f2b[i]; f2b[i]=f2b[i+1]+t2; if(d2[i]>0)++t2;
}

int m,lf,rt;
fi>>m;
for(int i=0; i<m; ++i)
{
    fi>>lf>>rt; --lf; --rt;
    ans=rt-lf+1;
    t=min(f1b[lf],f1e[rt]); ans+=t;
    t=min(f2b[lf],f2e[rt]); ans+=t;
    fo<<ans<<endl;
}
Times;
}

```



Vùng đài nguyên mênh mông dân cư khá thưa thớt. Mạng lưới giao thông ở thủ phủ trung tâm có n giao lộ và có m đường hai chiều nối các giao lộ với nhau. Mỗi đường nối 2 giao lộ và giữa 2 giao lộ có không quá một đường nối trực tiếp, cũng như không có đường nối một giao lộ với chính nó. Từ một giao lộ có thể tới bất kỳ giao lộ nào khác của trung tâm. Phương tiện đi lại phổ biến là xe do tuần lộc kéo. Chỉ khi nào qua giao lộ người ta mới phải điều khiển tuần lộc chọn đường, vì vậy dân ở đây quen tính khoảng cách theo số lượng đường khác nhau phải đi.

Đát đai rộng rãi nên mỗi gia đình đều ở một nhà riêng và nhà được xây dựng cạnh giao lộ. Số người ở giao lộ i là a_i , $i = 1 \dots n$. Nếu $a_i = 0$ thì có nghĩa là ở giao lộ i không có nhà. Khi thăm nhau, người ta thường đánh ngồi cờ vây, nói chuyện và uống trà theo từng cặp – một chủ và một khách. Chính vì vậy chỉ có các gia đình có số người giống nhau mới đi lại thăm hỏi nhau, nếu không, những người thừa ra sẽ không biết làm gì và sẽ rất bất tiện.

Với sự phát triển của kỹ thuật số, bây giờ nếu có người thừa họ có thể xúm quanh máy tính bảng, xem phim hoặc cùng nhau chơi một trò chơi online nào đó trong khi thế hệ bậc cha chú bàn công chuyện bên bàn cờ vây truyền thống.

Tiên phong trong phong trào giao lưu mới này là 2 gia đình có số người khác nhau và gần nhau nhất trong số các cặp gia đình có số người khác nhau.

Hãy xác định khoảng cách giữa 2 gia đình đó.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($2 \leq n \leq 10^6$, $1 \leq m \leq 10^6$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($0 \leq a_i \leq 10^9$, $i = 1 \dots n$), đảm bảo có ít nhất một cặp số nguyên dương khác nhau,
- ✚ Mỗi dòng trong m dòng sau chứa 2 số nguyên u và v xác định một đường nối 2 giao lộ u và v ($1 \leq u, v \leq n$, $u \neq v$), các dòng khác nhau ứng với những đường khác nhau.

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

Ví dụ:

TUNDRA.INP	TUNDRA.OUT
5 4 1 0 2 1 0 1 5 4 5 5 2 2 3	3



Giải thuật: Kỹ thuật tổ chức dữ liệu, Loang theo chiều rộng.

Việc tiến hành tìm kiếm chỉ bắt đầu từ các giao lộ có nhà, vì vậy cần lưu trữ số của các giao lộ này,

Cần có các mảng lưu trữ danh sách đỉnh kè nếu coi mạng lưới giao thông như một đồ thị, mỗi giao lộ là một đỉnh và mỗi đường – một cạnh của đồ thị,

Khi loang từ đỉnh **i** tới đỉnh **j** cần mang theo thông tin **độ dài đường đi** và **số người** ở đỉnh **i**, **i** và **j** thay đổi trong quá trình loang, vì vậy các thông tin này nên lưu trữ dưới dạng cặp dữ liệu,

Trong quá trình loang, nếu từ **i** ta tới đỉnh **j** nào đó thì sẽ có trong tay độ dài đường đi ngắn nhất từ **i** tới **j**. Nếu **a_i = a_j** thì việc *loang tiếp* theo hướng này là **không cần thiết** vì nếu gặp **a_k ≠ a_i** thì đường đi từ **i** → **k** sẽ dài hơn đường đi từ **j** → **k**.

Tổ chức dữ liệu:

- ▀ Mảng **int** **q[MAXN]** – lưu các giao lộ có nhà,
- ▀ Các mảng **vector<int>** **G[MAXN]** – lưu danh sách đỉnh kè,
- ▀ Mảng **pair<int, int>** **d[MAXN]** – phục vụ loang theo chiều rộng.

Xử lý:

Ghi nhận dữ liệu:

```
int n, m;
fi>>n>>m;
for(int i = 1; i <= n; i++)
{
    fi>>p[i];
    if(p[i])
        q[r++] = i, d[i].second = p[i];
    else
        d[i].first = -1;
}
for(int i = 0; i < m; i++)
{
    fi>>a>>b;
    G[a].push_back(b);
    G[b].push_back(a);
}
```

Với giao lộ có nhà

Danh sách cạnh kè

Loang theo chiều rộng (BFS) tìm nghiệm:

```
int ans = n + 1;
while(l <= r)
{
    int v = q[l++];
    for(auto u: G[v])
    {
        if(d[u].first == -1)
        {
            d[u].first = d[v].first + 1;
            d[u].second = d[v].second;
            q[r++] = u; ←
        }
        else if(d[u].second != d[v].second)
        {
            ans = min(ans, d[v].first + d[u].first + 1);
            if(d[u].first == d[v].first)
            {
                fo<<ans;
                Times; return 0;
            }
        }
    }
}
}
```

Trường hợp “Điểm giữa”:

Trong quá trình duyệt, giá trị hàm mục tiêu **ans** *không giảm*,

Khi đồ thị có chu trình và *chu trình* này *chứa nghiệm* của bài toán thì sẽ tồn tại một đỉnh **u** có khoảng cách tới đỉnh xuất phát và đỉnh đích không quá 1,

Ngược lại, khi tồn tại **u** với tính chất nói trên – ta *đã nhận được nghiệm cần tìm!*

Cần lưu ý xử lý để tránh việc duyệt vô hạn trong chu trình.

Độ phức tạp của giải thuật: Mỗi đỉnh của đồ thị được duyệt không quá 2 lần, vì vậy giải thuật có độ phức tạp $O(n)$.

Chương trình

```
#include <bits/stdc++.h>
#define NAME "tundra."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
typedef long long ll;
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int MAXN = 1000001;
int p[MAXN], q[MAXN], l=0, r=0, a, b;
pair<int, int> d[MAXN]={make_pair(0,0)};
vector<int> G[MAXN];
int main()
{
    int n, m;
    fi>>n>>m;
    for(int i = 1; i <= n; i++)
    {
        fi>>p[i];
        if(p[i])
            q[r++] = i, d[i].second = p[i];
        else d[i].first = -1;
    }
    for(int i = 0; i < m; i++)
    {
        fi>>a>>b;
        G[a].push_back(b);
        G[b].push_back(a);
    }
    int ans = n + 1;
    while(l <= r)
    {
        int v = q[l++];
        for(auto u: G[v])
        {
            if(d[u].first == -1)
            {
                d[u].first = d[v].first + 1;
                d[u].second = d[v].second;
                q[r++] = u;
            }
            else if(d[u].second != d[v].second)
            {
                ans = min(ans, d[v].first + d[u].first + 1);
                if(d[u].first == d[v].first)
                {
                    fo<<ans; Times; return 0;
                }
            }
        }
    }
    fo<<ans;
    Times;
}
```



Sinh viên rất quan tâm tới việc khi nào sẽ có bài kiểm tra, trọng tâm môn thi tới là gì, . . . Khi một người biết thông tin gì đó sẽ báo lại cho bạn mình qua điện thoại hoặc tin nhắn. Việc hình thành tự phát mạng lưới truyền tin như thế hoạt động không hiệu quả, cùng một tin có người nhận được rất nhiều lần, có người lại không nhận được.

Sắp sửa bước vào mùa thi đầu tiên, mọi người đều hồi hộp và lo lắng. Lớp trưởng cũng đã tiếp xúc nhiều với các bạn trong lớp và nhận thấy rằng mỗi người có một khả năng hòa hợp riêng của mình. Khả năng hòa hợp có thể đánh giá bằng một số nguyên dương. Lớp có n bạn, bạn thứ i có khả năng hòa hợp là a_i , $i = 1 \div n$. Ước số chung của a_i và a_j càng lớn bao nhiêu thì hai bạn i và j càng thoải mái hơn bấy nhiêu khi trao đổi, trò chuyện. Ước số chung của a_i và a_j được gọi là hiệu quả thông báo. Lớp trước quyết định đề xuất một mạng lưới thông báo, khi một người biết chuyện sẽ nhắn cho người khác. Mạng lưới thông báo phải chứa ít nhất số thông báo cần thực hiện, khi một người biết tin thì mọi người trong lớp cũng sẽ được biết và tổng hiệu quả thông báo là lớn nhất.

Hãy xác định tổng hiệu quả thông báo lớn nhất có thể đạt được.

Dữ liệu: Vào từ file văn bản SMS.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 ($1 \leq a_i \leq 10^6$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản SMS.OUT một số nguyên – tổng hiệu quả thông báo lớn nhất có thể đạt được.

Ví dụ:

SMS.INP	SMS.OUT
5	
2 6 15 24 12	23



Giải thuật: Tìm ước số chung lớn nhất của một số với nhiều số.

Điều kiện đầu bài cho thấy:

- Một số có thể tham gia và nhiều cặp khác nhau,
- Cần có đúng $n-1$ cặp số,
- Không có cặp số giống nhau, hai cặp số (a_i, a_j) và (a_j, a_i) được coi là giống nhau.

Nhận xét đầu cho phép tính hàm mục tiêu theo phương pháp tham lam: tìm và ghi nhận cặp có hiệu quả nhất, đánh dấu các số để tránh tạo cặp giống nhau, lặp lại công việc trên cho đến khi có đủ $n-1$ cặp số.

Cần tận dụng thông tin nhận được khi tìm ước số chung lớn nhất (GCD) của 2 số để phục vụ việc tìm GCD của 2 số đó với các số còn lại. Điều này sẽ làm giảm độ phức tạp của giải thuật.

Mỗi số cần xét đều không vượt quá 10^6 , vì vậy mỗi số sẽ có không quá 21 ước số, mỗi ước số lưu dưới dạng một cặp (*Uớc_số*, *Số*),

Sắp xếp các cặp này theo thứ tự giảm dần (không tăng).

Vấn đề còn lại chỉ là chọn từ trên xuống dưới 2 cặp số có cùng ước và ít nhất một trong 2 số tương ứng chưa được sử dụng, cập nhật hàm mục tiêu và đánh dấu sử dụng số.

Các phương pháp tạo và lưu trữ bảng cặp (*Uớc_số*, *Số*) sẽ tương ứng với những giải thuật khác nhau. Phương pháp hiệu quả nhất là tạo bảng tra cứu với 10^6 số từ 1 đến 10^6 trên cơ sở phân tích ra thừa số nguyên tố. Độ phức tạp của giải thuật sẽ $\approx O(n)$, nhưng độ phức tạp lập trình tương đối cao.

Phương pháp dễ lập trình và có độ phức tạp $O(n \ln n)$ là lập bảng lưu các ước của những số đã cho.

Tổ chức dữ liệu:

- Mảng `int` *a* [N] – lưu dữ liệu vào,
- Mảng `vector<pair<int, int>>` *c* – lưu các cặp dữ liệu (*Uớc_số*, *Số*),
- Mảng `int` *mp* [N] – đánh dấu việc sử dụng dữ liệu vào.

Xử lý:

Xây dựng bảng (*Uớc_số*, *Số*):

Với mỗi số nguyên *t* chỉ cần tìm các ước không vượt quá $t^{0.5}$, các ước còn lại – dẫn xuất thông qua các ước đã tìm được.

```

for (int i = 0; i < n; i++)
{
    int t = a[i];
    for (int j = 1; j * j <= t; j++)
        if (t % j == 0)
        {
            c.push_back(make_pair(j, i));
            if (j * j != t) c.push_back(make_pair(t/j, i));
        }
}
for (int i = 0; i < n; i++) mp[i] = i;
sort(c.rbegin(), c.rend());

```

Ghi nhận ước liên hợp

Một cách sắp xếp theo thứ tự giảm dần

Tích lũy kết quả:

```

int64_t ans = 0;
int size = c.size();
for (int i = 0; i < size;)
{
    int avl = -1, j;
    for (j = i; j < size && c[j].first == c[i].first; j++)
        if (avl == -1) avl = findSet(c[j].second);
        else if (avl != findSet(c[j].second))
        {
            int type = avl;
            ans += c[j].first;
            unite(c[j].second, type);
        }
    i = j;
}

```

Tìm ước giống nhau của các số khác nhau

Cập nhật mảng đánh dấu dữ liệu đã dùng

C++ cho phép thay đổi tham số điều khiển chu

Độ phức tạp của giải thuật: bậc O(nlnn).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "sms."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N = 1e5 + 100;
int n, a[N], mp[N];
vector<pair<int,int>> c;
inline int findSet(int v)
{
    if (mp[v] != v) mp[v] = findSet(mp[v]);
    return mp[v];
}
inline void unite(int a, int b)
{
    a = findSet(a), b = findSet(b);
    mp[a] = b;
}
int main()
{
    fi>>n;
    for (int i = 0; i < n; i++) fi>>a[i];
    for (int i = 0; i < n; i++)
    {
        int t = a[i];
        for (int j = 1; j * j <= t; j++)
            if (t % j == 0)
            {
                c.push_back(make_pair(j, i));
                if (j * j != t) c.push_back(make_pair(t/j, i));
            }
    }
    for (int i = 0; i < n; i++) mp[i] = i;
    sort(c.rbegin(), c.rend());

    int64_t ans = 0;
    int size = c.size();
    for (int i = 0; i < size;)
    {
        int avl = -1, j;
        for (j = i; j < size && c[j].first == c[i].first; j++)
            if (avl == -1) avl = findSet(c[j].second);
            else if (avl != findSet(c[j].second))
            {
                int type = avl;
                ans += c[j].first;
                unite(c[j].second, type);
            }
        i = j;
    }
    fo << ans << endl;
    Times;
}
```



Đọc theo lối đi nối 2 tòa nhà chính của một khu vui chơi – giải trí người ta trồng 2 hàng cột song song, cột hàng bên phải cùng độ cao với cột tương ứng ở hàng bên trái. Có n cột ở mỗi hàng. Cột thứ i có độ cao so với mặt phẳng sàn của tầng 1 là a_i , $i = 1 \dots n$. Ví dụ $a_3 = 2$, $a_4 = -1$ có nghĩa là cặp cột thứ 3 cao hơn sàn 3m, còn cặp cột thứ 4 – thấp hơn sàn 1m. Theo dự kiến ban đầu, người ta lợp lối đi bằng tôn nhựa xanh tạo thành hình sóng uốn lượn trên đầu du khách. Một người bạn của Giám đốc điều hành Khu vui chơi khi được giới thiệu về công trình sắp khánh thành nêu ý kiến sao không biến trần hành lang thành lối đi thứ 2 phục vụ các bà mẹ đẩy xe nôi và những người tàn tật phải ngồi trên xe lăn? Giám đốc điều hành rất thích ý tưởng này và ra lệnh thay đổi độ cao hàng cột để chênh lệch độ cao ở 2 cặp cột liên tiếp nhau là như nhau, tức là với mọi $1 \leq i, j < n$ có $a_{i+1} - a_i = a_{j+1} - a_j$. Chi phí để nâng hoặc giảm độ cao cột một đơn vị là như nhau. Dĩ nhiên, việc cải tạo phải được thực hiện với chi phí nhỏ nhất. Các cột mới vẫn phải có độ cao nguyên.

Hãy xác định chi phí nhỏ nhất cần có, độ cao của cặp cột đầu tiên và chênh lệch của nó so với cặp cột tiếp theo.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($2 \leq n \leq 3 \times 10^6$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($-10^9 \leq a_i \leq 10^9$, $i = 1 \dots n$).

Kết quả: Đưa ra file văn bản COLUMNS.OUT trên một dòng chi phí nhỏ nhất cần có dưới dạng số nguyên, dòng tiếp theo chứa 2 số nguyên – cao của cặp cột đầu tiên và chênh lệch của nó so với cặp cột tiếp theo.

Dữ liệu đảm bảo tồn tại lời giải và các số cần đưa ra không vượt quá 10^{16} theo giá trị tuyệt đối.

Ví dụ:

COLUMNS.INP
5
3 8 10 13 20

COLUMNS.OUT
5
3 4



Giải thuật: Tìm kiếm nhị phân, Cấp số cộng.

Sau khi sửa đổi, độ cao các cột tạo thành một cấp số cộng,

Bài toán có 2 tham số: số hạng đầu \mathbf{x} của cấp số cộng và công bội \mathbf{d} .

Với mỗi công bội \mathbf{d} được lựa chọn tồn tại một \mathbf{x} tương ứng để tổng chi phí giá trị tuyệt đối các chênh lệch là nhỏ nhất.

Xuất phát từ độ cao cột đầu tiên là 0 , ta có các cột với độ cao $0, \mathbf{d}, 2\mathbf{d}, \dots, (\mathbf{n}-1)\mathbf{d}$. Chênh lệnh ở cột thứ i sẽ là $\mathbf{b}_i = \mathbf{a}_i - i \times \mathbf{d}$, $i = 0 \div \mathbf{n}-1$.

Nếu xuất phát từ độ cao ban đầu là \mathbf{x} , thì chênh lệch ở cột i sẽ là $\mathbf{b}_i - \mathbf{x}$. Ta cần tìm \mathbf{x} để $\mathbf{t} = \sum_{i=0}^{\mathbf{n}-1} |\mathbf{b}_i - \mathbf{x}|$ là nhỏ nhất. Đây là bài toán quen thuộc với nghiệm \mathbf{x} bằng giá trị phần tử ở giữa của dãy \mathbf{b}_i được sắp xếp.

Như vậy bài toán cần giải chỉ còn phụ thuộc vào tham số \mathbf{d} . Trong phạm vi từ 0 đến chênh lệch hai cực trị \min, \max của \mathbf{a}_i hàm mục tiêu là đơn điệu, vì vậy có thể dùng phương pháp tìm kiếm nhị phân.

Do đó phải nhận giá trị nguyên nên ở mỗi bước tìm kiếm nhị phân ta phải *xét 2 giá trị* nhận được từ việc *làm tròn lên* và *làm tròn xuống* của điểm giữa.

Kết quả tìm kiếm nhị phân là nhóm 3 giá trị (*Hàm mục tiêu, \mathbf{d}, \mathbf{x}*).

Tổ chức dữ liệu:

- Sử dụng 2 mảng \mathbf{a} và \mathbf{b} với ý nghĩa như đã nêu ở trên để phục vụ tìm kiếm nhị phân,
- Biến \mathbf{ans} kiểu $\text{tuple} < 11, 11, 11 >$ lưu trữ kết quả tìm kiếm.

Xử lý:

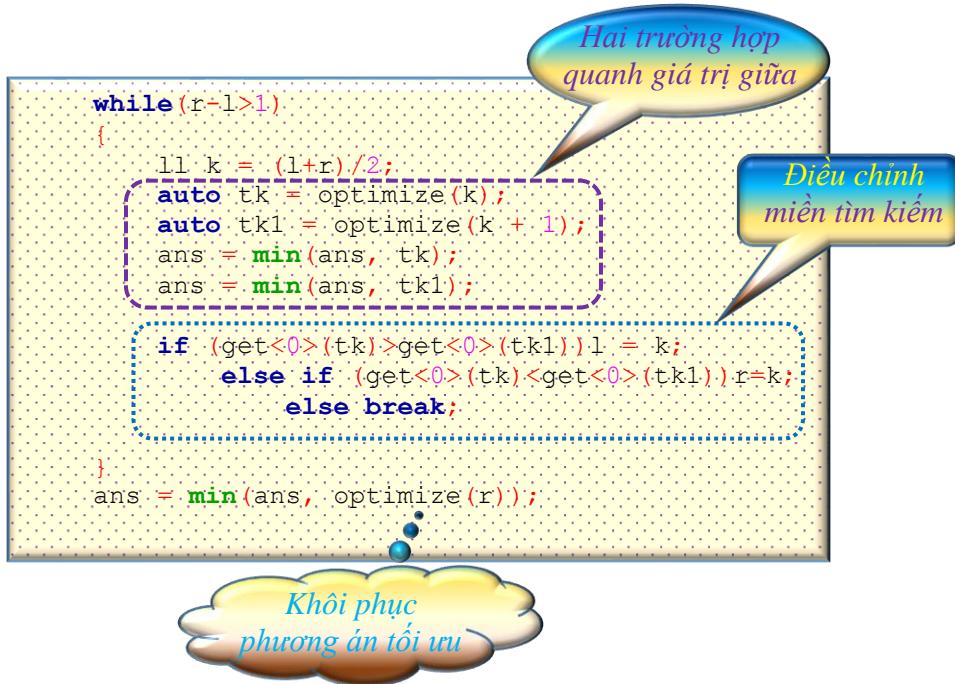
Nhập dữ liệu và chuẩn bị tham số tìm kiếm:

Do khối lượng dữ liệu cần nhập có thể rất lớn (3×10^6), việc sử dụng các chỉ thị vào ra theo quy cách tường minh sẽ tiết kiệm đáng kể thời gian nhập xuất dữ liệu,

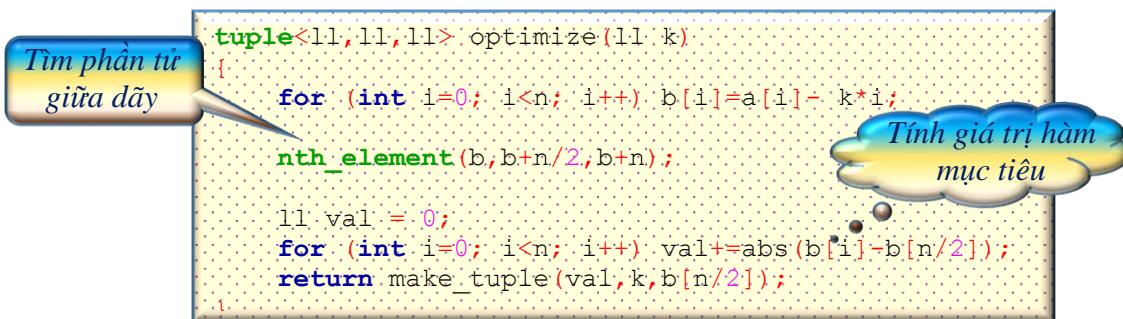
Chỉ cần chuẩn bị giá trị trường đầu tiên của \mathbf{ans} là một số cực lớn, hai trường còn lại – không quan trọng,

Việc xác định các cực trị của \mathbf{a}_i có thể thực hiện ngay trong quá trình nhập dữ liệu.

Tìm kiếm nhị phân:



Hàm tính giá trị hàm mục tiêu:



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

Chương trình I filestream

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

typedef int64_t ll;

const int N = 3000500;
const int amn=-1000000001;
int a[N],t;
ll b[N],mx,mn,l,r;

int n;
tuple<ll,ll,ll> optimize(ll k)
{
    for (int i=0; i<n; i++) b[i]=a[i]- k*i;
    nth_element(b,b+n/2,b+n);
    ll val = 0;
    for (int i=0; i<n; i++) val+=abs(b[i]-b[n/2]);
    return make_tuple(val,k,b[n/2]);
}

int main()
{
    fi>>n; mx=amn; mn=N;
    for (int i = 0; i < n; i++)
        {fi>>t; a[i]=t; if(t>mx) mx=t; if(t<mn) mn=t; }
    r = 2*(mx-mn+n-1)/n; l=-r;
    auto ans = make_tuple((ll)1e16, -5000, -5000);
    while(r-l>1)
    {
        ll k = (l+r)/2;
        auto tk = optimize(k);
        auto tk1 = optimize(k + 1);
        ans = min(ans, tk);
        ans = min(ans, tk1);
        if (get<0>(tk)>get<0>(tk1)) l = k;
        else if (get<0>(tk)<get<0>(tk1)) r = k;
        else break;
    }
    ans = min(ans, optimize(r));
    fo<<get<0>(ans)<<'\
';
    fo<<get<2>(ans)<<' ' <<get<1>(ans);
    Times;
}
```

Chương trình II stdio

```
#include<bits/stdc++.h>
#define NAME "columns."
#define Times printf("\n%f sec",clock() / (double)1000)
using namespace std;

typedef int64_t ll;

const int N = 3000500;
const int amn=-1000000001;
int a[N],t;
ll b[N],mx,mn,l,r;

int n;
tuple<ll,ll,ll> optimize(ll k)
{
    for (int i=0; i<n; i++) b[i]=a[i]-k*i;
    nth_element(b, b+n/2, b+n);
    ll val = 0;
    for (int i=0; i<n; i++) val += abs(b[i]-b[n/2]);
    return make_tuple(val, k, b[n/2]);
}

int main()
{
    freopen(NAME"inp","r",stdin);
    freopen(NAME"out","w",stdout);

    scanf("%d", &n);
    mx=amn; mn=N;
    for (int i = 0; i < n; i++) {scanf("%lld", &t); a[i]=t; if(t>mx)mx=t;
    if(t<mn)mn=t;}
    r = 2 * (mx-mn+n-1)/n, l = -r;
    auto ans = make_tuple((ll)1e16, -4211, -4211);
    while (r - l > 1)
    {
        ll k = (l + r) / 2;
        auto tk = optimize(k);
        auto tk1 = optimize(k + 1);
        ans = min(ans, tk);
        ans = min(ans, tk1);
        if (get<0>(tk) > get<0>(tk1)) l = k;
        else if (get<0>(tk) < get<0>(tk1)) r = k;
        else break;
    }
    ans = min(ans, optimize(r));
    printf("%lld\n", get<0>(ans));
    printf("%lld %lld\n", get<2>(ans), get<1>(ans));

    Times;
}
```



Bob hào hứng kể cho Alice giải thuật mìn mới tìm được, cho phép từ 2 xâu **s** và **t** chỉ chứa các ký tự trong tập { (,) } tìm cực nhanh xâu con chung dài nhất là một biểu thức ngoặc đúng. Trong trường hợp này, xâu con của xâu **a** là xâu nhận được từ **a** bằng cách xóa một số ký tự (có thể là 0) ở những vị trí tùy chọn.

Biểu thức ngoặc đúng được xác định như sau:

- ⊕ Xâu rỗng,
- ⊕ Nếu **A** là biểu thức ngoặc đúng thì **(A)** cũng là biểu thức ngoặc đúng,
- ⊕ Nếu **A** và **B** là 2 biểu thức ngoặc đúng thì **AB** cũng là biểu thức ngoặc đúng.

Thấy Alice nhìn có vẻ thiếu tin tưởng, Bob đề nghị Alice đưa ra 2 xâu **s** và **t** tùy chọn. Bob sẽ chạy chương trình của mình, tìm ra xâu con chung dài nhất là một biểu thức ngoặc đúng. Alice có thể kiểm tra xác thực đó là biểu thức đúng dài nhất có thể rút ra.

Hãy xác định kết quả Bob đưa ra cho Alice.

Dữ liệu: Vào từ file văn bản BR_EXPR.INP, dòng thứ nhất chứa xâu **s**, dòng thứ 2 chứa xâu **t**, độ dài mỗi xâu không quá 700. Một trong 2 xâu nối trên hoặc cả 2 có thể là xâu rỗng.

Kết quả: Đưa ra file văn bản BR_EXPR.OUT xâu con chung dài nhất là một biểu thức ngoặc đúng. Nếu tồn tại nhiều xâu con khác nhau cùng là biểu thức ngoặc đúng dài nhất thì đưa ra biểu thức tùy chọn.

Ví dụ:

BR_EXPR.INP	BR_EXPR.OUT
() (() () ()	
) (() (())	(() ()



Giải thuật: Quy hoạch động.

Bài toán yêu cầu *dẫn xuất phương án tối ưu* (biểu thức ngoặc đúng dài nhất) tìm được,

Để dẫn xuất phương án tối ưu cần có các cấu trúc dữ liệu hỗ trợ để xác định những ký tự tham gia vào kết quả cuối cùng,

Kích thước bài toán không quá lớn ($n \leq 700$) vì vậy, để đơn giản trong xử lý, có thể tổ chức bảng mốc nối ngược hai chiều,

Công thức lặp của quy hoạch động chứa 2 thông tin:

- ✚ Độ dài lớn nhất của biểu thức chung *tiềm năng đúng* (tính từ trái sang phải số ngoặc mở luôn luôn lớn hơn hoặc bằng số ngoặc đóng),
- ✚ Độ dài lớn nhất của *biểu thức ngoặc đúng* đã xác định được.

Để thuận tiện dẫn xuất kết quả dùng mảng đánh dấu xác định loại ngoặc tham gia vào biểu thức chung.

Kết quả tìm kiếm ở mỗi bước chỉ phụ thuộc vào kết quả bước trước, vì vậy chỉ cần giữ 2 dòng cho bảng độ dài và bảng mốc nối ký tự,

Dùng kỹ thuật *con lắc* để đổi vai trò các dòng trong quá trình xử lý.

Tổ chức dữ liệu:

- ─ Mảng **int** `dp[2][MAXN+1]` – lưu độ dài max ở từng bước,
- ─ Mảng **int** `ptr[2][MAXN+1]` – lưu mốc nối tới ký tự tham gia vào xâu con chung,
- ─ Bảng **char** `link[MAXN+1][MAXN+1]` – xác định loại ký tự tham gia vào xâu con chung,
- ─ Các mảng **int** `next_l[MAXN], next_r[MAXN]` – lưu mốc nối tới ngoặc cùng loại tiếp theo trong xâu thứ nhất.

Xử lý:

Nhập dữ liệu: thời gian xử lý phụ thuộc nhiều vào độ dài xâu thứ 2, vì vậy cần chuẩn hóa dữ liệu để độ dài xâu thứ 2 không vượt quá độ dài xâu thứ nhất,

Tạo mốc nối tới ngoặc cùng loại ở xâu thứ nhất,

Duyệt theo sơ đồ quy hoạch động tìm biểu thức ngoặc đúng chung dài nhất, độ dài biểu thức được lưu ở `dpp,0` (phần tử đầu dòng mới).

Dẫn xuất kết quả: truy vết bảng **link**.

Độ phức tạp của giải thuật: $O(m^2)$, trong đó m – độ dài xâu thứ hai.

Chương trình

```
#include<bits/stdc++.h>
#define NAME "br_expr."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int MAXN = 700;

int dp[2][MAXN+1], ptr[2][MAXN+1], p=0, q=1, lt;
int next_l[MAXN], next_r[MAXN];
char link[MAXN+1][MAXN+1];

int main()
{
    string s, t;
    fi >> s >> t; if(s.size()<t.size()) swap(s,t);
    lt=t.size();
    int first_l = -1;
    int first_r = -1;
    {
        int c_next_l = -1;
        int c_next_r = -1;
        for (int i = s.size() - 1; i >= 0; --i)
        {
            next_l[i] = c_next_l;
            next_r[i] = c_next_r;
            if (s[i] == '(') c_next_l = i;
            else c_next_r = i;
        }
        first_l = c_next_l;
        first_r = c_next_r;
    }

    for(int i=0;i<=lt+1;++i) dp[0][i]=dp[1][i]=-MAXN;
    for (int tt = 0; tt <= lt; ++tt)
    {
        for (int bal = 0; bal <= tt; ++bal)
        {
            int &curval = dp[q][bal];
            int &curptr = ptr[q][bal];
            char &lnk = link[tt][bal];
            curval = -MAXN;
            curptr = -1;
            if (tt == 0 && bal == 0) curval = 0;
            if (tt > 0 && dp[p][bal] != -MAXN)
            {
                curval = dp[p][bal];
                curptr = ptr[p][bal];
                lnk = 1;
            }
            if (tt>0 && t[tt-1]==')' && dp[p][bal+1] != -MAXN && dp[p][bal+1] + 1 > curval)
            {
                curval = dp[p][bal+1] + 1;
                curptr = ptr[p][bal+1];
                lnk = 0;
            }
        }
    }
}
```

```

        int newptr=(ptr[p][bal+1]>=0)?next_r[ptr[p][bal+1]]:first_r;
        if (newptr != -1)
        {
            curval = dp[p][bal+1] + 1;
            curptr = newptr;
            lnk = 2;
        }
    }
    if (tt > 0 && t[tt-1] == '('
        && bal > 0 && dp[p][bal-1] != -MAXN
        && dp[p][bal-1] + 1 > curval)
    {
        int newptr=(ptr[p][bal-1]>=0)?next_l[ptr[p][bal-1]]:first_l;
        if (newptr != -1)
        {
            curval = dp[p][bal-1] + 1;
            curptr = newptr;
            lnk = 3;
        }
    }
    p^=1; q^=1;
}

int tt = lt, bal = 0;
string ans;
ans.reserve(dp[p][0]);

while (tt || bal)
{
    char &lnk = link[tt][bal];
    if(lnk==2) ans+=')', ++bal;
    else if(lnk==3) ans+='(', --bal;
    --tt;
}

reverse(ans.begin(), ans.end());
fo << ans << endl;
Times;
}

```



VV50. TRÒ CHƠI NHI PHÂN

Tên chương trình: BINGAME.CPP

Alice và Bob phụ trách mục Giải trí của Tạp chí Tin học với nhiệm vụ đề xuất cho độc giả các bài toán vui, nhẹ nhàng với mọi người nhưng có hàm lượng kiến thức cao cho những ai muốn đi sâu, nắm vững giải thuật. Bài toán trong số ra hôm nay có nội dung như sau.

Cho n xâu nhị phân (chỉ chứa các ký tự trong tập $\{0, 1\}$) f_1, f_2, \dots, f_n và gọi là những xâu cám,

Cho xâu nhị phân s độ dài m , được gọi là *xâu xuất phát*, s không chứa các xâu cám như những xâu con các ký tự liên tiếp nhau. Xâu s có thể là rỗng ($m = 0$).

Bắt đầu từ s , hai người lần lượt đi. Mỗi người, khi đến lượt mình, kéo dài xâu bằng cách viết thêm một ký tự 0 hoặc 1 vào cuối xâu nhận được ở bước trước. Ai làm cho xâu kết quả chứa một trong số các xâu cám như một xâu con các ký tự liên tiếp nhau là thua. Nếu với cách đi tối ưu trò chơi có thể kéo dài vô hạn thì kết quả là hòa. Alice đi trước.

Hãy đưa ra thông báo tên người thắng (**Alice** hay **Bob**) hoặc **Friendship** nếu kết quả là hòa.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($0 \leq n \leq 10^5, 0 \leq m \leq 10^6$),
- ✚ Dòng thứ i trong n dòng sau chứa xâu f_i . Tổng độ dài các xâu cám không vượt quá 10^6 .
- ✚ Nếu $m > 0$ – dòng cuối cùng chứa xâu s độ dài m .

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

Ví dụ:

BINGAME.INP
3 1
000
001
011
0

BINGAME.OUT
Alice



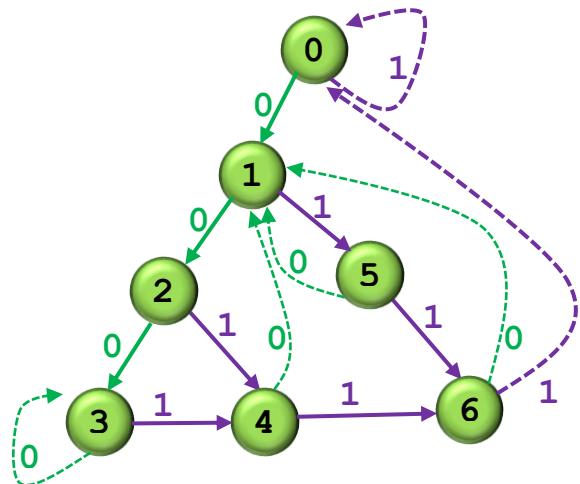
VV50 ZOM20170110 H

Giải thuật: Kỹ thuật bảng phương án.

Xây dựng đồ thị, mỗi nút chứa 2 đường ra, rẽ 2 nhánh theo 0 và 1, xác định đường đi theo các xâu f_i , $i = 1 \div n$,

Nút sau mỗi đường đi được móc nối với nút cuối của đường đi.

Ví dụ, với $f_1=000$, $f_2=001$ và $f_3=011$ ta có cây:



Tạo các mảng xác định danh sách đỉnh kề ở mỗi nút,

Đánh dấu các đỉnh cuối của mỗi đường đi.

Từ mỗi đỉnh cuối của đường đi tiến hành loang theo chiều rộng cho đến khi gặp lại một đỉnh cuối nào đó,

Mỗi nút của đồ thị được tới thăm *không quá 2 lần!*

Ở mỗi nút đã đi qua trên đường đi dẫn tới một điểm cuối – ghi nhận dấu hiệu kết thúc: tính chẵn – lẻ của độ dài đường đi tới điểm cuối.

Để xác định kết quả cần truy vết đường đi theo xâu đã cho. Nút kết thúc cho biết từ đó không có đường đi tới nút kết thúc (Friendship) hoặc người thăng phụ thuộc vào tính chẵn lẻ của dấu hiệu kết thúc.

Tổ chức dữ liệu:

Mảng **int** go [MX] [2] – ghi nhận cấu trúc đồ thị đường đi,

Mảng **int** fl [MX] – đánh dấu nút kết thúc,

Mảng **vector<int>** be [MX] – ghi nhận danh sách đỉnh kề,

Mảng **int** rs [MX] – ghi nhận dấu hiệu kết thúc,

Mảng **int** fr [MX] – kiểm soát số lần duyệt đỉnh,

Mảng **int** was [MX] – đánh dấu vết đường đi.

Dộ phức tạp của giải thuật: O(m), trong đó m – tổng độ dài các \mathbf{f}_i , $i = 1 \dots n$.

Chương trình

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

typedef long long ll;
typedef long double ld;

const int MX = 1000007;
int n, m;
vector<int> be[MX];
string buf;
int go[MX][2];
int pp[MX], pc[MX], fr[MX], fl[MX];
int was[MX], rs[MX], sf[MX];
int cc = 1;

int newn(int p, int c)
{
    pp[cc] = p; pc[cc] = c;
    return cc++;
}

void add(string s)
{
    int n = s.size(), now = 0;
    for (int i = 0; i < n; ++i)
    {
        int x = s[i] - '0';
        if (go[now][x]) now = go[now][x];
        else now = go[now][x] = newn(now, x);
    }
    fl[now] = 1;
}

void build_ak()
{
    queue<int> qu;
    qu.push(0);
    while (!qu.empty())
    {
        int x = qu.front();
        qu.pop();
        if (go[x][0]) qu.push(go[x][0]);
        if (go[x][1]) qu.push(go[x][1]);
        if (x == 0) continue;
        if (pp[x] != 0) sf[x] = go[sf[pp[x]]][pc[x]];
        if (fl[sf[x]]) fl[x] = 1;
        if (!go[x][0]) go[x][0] = go[sf[x]][0];
        if (!go[x][1]) go[x][1] = go[sf[x]][1];
    }
}
```



```

        }
    }

int main()
{
    fi>>n>>m;
    for (int i = 0; i < n; ++i)
    {
        fi>>buf; add(buf);
    }
    build_ak();
}

```

Xác định danh sách đỉnh
kè và chuẩn bị loang

```

for (int i = 0; i < cc; ++i)
    for (int j = 0; j < 2; ++j)
        be[go[i][j]].push_back(i);
for (int i = 0; i < cc; ++i) fr[i] = 2;

queue<int> qu;
for (int i = 0; i < cc; ++i)
    if (fl[i]) {was[i] = 1; rs[i] = 1; qu.push(i);}
}

```

```

while (!qu.empty())
{
    int x = qu.front();
    qu.pop();
    if (rs[x] == 0)
    {
        for (int j: be[x])
            if (!was[j]) {was[j] = 1; rs[j] = 1; qu.push(j);}
    }
    else
        for (int j: be[x])
            if (!was[j])
            {
                --fr[j];
                if (fr[j] == 0) {was[j] = 1; rs[j] = 0; qu.push(j);}
            }
}

fi>>buf;
int now = 0;
for (int i = 0; i < m; ++i)
    now = go[now][buf[i] - '0'];
if (!was[now])
    fo<< "Friendship\n";
else if (rs[now] == 1)
    fo<< "Alice\n";
else
    fo<< "Bob\n";
Times; return 0;
}

```

BFS



Kỳ thi olympic Tin học quốc gia được tổ chức thành hai vòng. Vòng loại được tổ chức ở các địa phương qua hình thức thi online. Có n đơn vị tham gia, đơn vị thứ i có p_i học sinh đăng ký tham gia, $i = 1 \dots n$. Đề thi có 4 bài, mỗi bài có số điểm tối đa là 100. Điểm thí sinh nhận được ở mỗi bài có thể là một số thực với 2 chữ số sau dấu chấm thập phân. Đơn vị thứ i được phép cử q_i thí sinh có tổng điểm cao nhất tham dự vòng II. Tuy vậy, nếu có nhiều em có cùng điểm như người thứ q_i thì tất cả đều được chọn vào vòng II. Ví dụ một đơn vị có 5 em điểm cao nhất là 301, 285.5, 285.5, 285.5 và 278, đơn vị được cử 2 em có tổng điểm cao nhất, nhưng trong trường hợp này được phép chọn 4 em!

Những người có tổng điểm bằng 0 không được chọn, ngay cả khi còn thừa chỉ tiêu cử đi.

Trong biên bản online, mỗi thí sinh tương ứng với một dòng thông tin gồm 6 số: số nguyên id – mã thí sinh, số nguyên gr – mã đơn vị và 4 số thực – điểm các bài thi. Thông tin ở biên bản không được sắp xếp.

Dựa vào biên bản online hãy xác định tổng số thí sinh được chọn vào vòng II và đưa ra danh sách các thí sinh này bao gồm mã thí sinh cùng tổng điểm theo thứ tự giảm dần của tổng điểm.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($2 \leq n \leq 200$),
- ✚ Dòng thứ 2 chứa n số nguyên p_1, p_2, \dots, p_n , $2 \leq p_i \leq 1000$, $i = 1 \dots n$, $m = \sum_{i=1}^n p_i \leq 10^5$,
- ✚ Dòng thứ 3 chứa n số nguyên q_1, q_2, \dots, q_n , $0 < q_i < p_i$, $i = 1 \dots n$,
- ✚ Mỗi dòng trong m dòng sau chứa 6 số: id, gr và 4 số thực – điểm từng bài, $0 < id \leq 10^6$, $1 \leq gr \leq n$.

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

- ✚ Dòng đầu tiên chứa số nguyên k – tổng số thí sinh được gọi vào vòng II,
- ✚ Mỗi dòng trong k dòng sau chứa 2 số: số nguyên id của thí sinh được chọn và một số thực – tổng điểm vòng I của thí sinh đó. Thông tin được đưa ra theo thứ tự tổng điểm giảm dần.

Ví dụ:

OLYMPIAD.INP	OLYMPIAD.OUT
5	8
4 2 3 3 3	401 332.00
2 1 1 2 1	503 315.34
101 1 12 86 100 100	502 315.34
102 1 13.48 15.97 100 38	101 298.00
403 4 11 86 98 0	302 296.00
201 2 0 0 5 0	104 278.00
104 1 57 35 91 95	402 224.00
202 2 48 11.95 99.99 30	202 189.94
103 1 37 28 93 47	
401 4 91 41 100 100	
301 3 0 5 0 0	
303 3 2 0 100 10	
302 3 47 71 100 78	
402 4 18 32 100 74	
503 5 55.16 95 100 65.18	
502 5 100 100 100 15.34	
501 5 0 18 0 0	



VW01 Bel2016 I

Giải thuật: Tổ chức dữ liệu, xử lý số thực.

Các điểm số có dạng biểu diễn thực, để tránh ảnh hưởng của sai số làm tròn cần nhân điểm số với 100 và tiến hành làm tròn lên trước khi chuyển sang dạng lưu trữ nguyên,

Gọi tổng điểm của một thí sinh là **evl**, mỗi dòng trong biên bản online sẽ tương ứng với một nhóm 3 {**gr**, **evl**, **id**},

Thông tin này có thể lưu trữ một bảng nhóm 3 hay lưu trữ trong **vector rtab[gr]**.

Trong mọi trường hợp, thông tin được sắp xếp theo thứ tự giảm dần của tổng điểm evl trong từng mảng riêng hoặc theo đơn vị (trong bảng chung).

Các thông tin tương ứng với mỗi đơn vị hoặc đứng liên tiếp nhau theo trình tự giảm dần trong bảng chung hoặc có thứ tự điểm giảm dần trong từng mảng riêng.

Không khó khăn để rút ra các bản ghi tương ứng với các thí sinh được chọn và ghi vào mảng kết quả riêng.

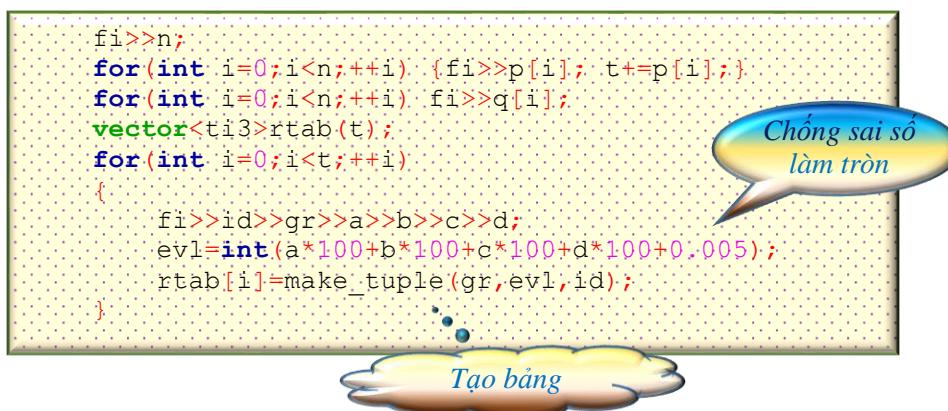
Để đưa kết quả ra cần sắp xếp mảng kết quả theo thứ tự giảm dần của điểm số.

Tổ chức dữ liệu:

- Các mảng **int** p[201], q[201] – lưu số người và chỉ tiêu được phân phối,
- Bảng **vector<ti3>rtab(t)** hoặc **vector<pii>rtab[n]** các mảng – lưu thông tin về tổng điểm của mỗi thí sinh,
- Mảng **vector<pair<int, int>>res** – lưu kết quả các thí sinh được chọn.

Xử lý:

Ghi nhận dữ liệu:



```
fi>>n;
for(int i=0;i<n;++i) { fi>>p[i]; t+=p[i]; }
for(int i=0;i<n;++i) fi>>q[i];
vector<ti3>rtab(t);
for(int i=0;i<t;++i)
{
    fi>>id>>gr>>a>>b>>c>>d;
    evl=int(a*100+b*100+c*100+d*100+0.005);
    rtab[i]=make_tuple(gr,evl,id);
}
```

Chống sai số làm tròn

Tạo bảng

Sắp xếp giảm dần theo tổng điểm:

```
sort(rtаб.begin(), rtаб.end(),
      [](ti3 a, ti3 b){return (get<0>(a)<get<0>(b) || (get<0>(a)==get<0>(b) && get<1>(a)>=get<1>(b));});
```

Hàm xác định điều kiện
sắp xếp

Tách các đối tượng được chọn:

```
u=0; v=p[0];
for(int i=0;i<n;++i)
{
    t=u+q[i]-1;k=0;
    int x=get<1>(rtаб[t]);
    if(x==0) while(t>=u && get<1>(rtаб[t])==0) --k,--t;
    else {++t; while(t<v && get<1>(rtаб[t])!=x) ++k,++t;}
    t=q[i]+k;

    for(int j=0;j<t;++j)
        res.push_back({get<1>(rtаб[u+j]),get<2>(rtаб[u+j])});
    u=v; v+=p[i+1];
}

sort(res.rbegin(),res.rend());
```

Xác định số lượng
tíng nhóm

Tách đối tượng

Sắp xếp giảm dần

Chú ý các đưa ra số thực ở dạng dấu chấm tĩnh và với độ chính xác cho trước!

Độ phức tạp của giải thuật: O(nlnn).

Lưu ý: Số lượng dữ liệu vào có thể rất lớn vì vậy nếu dùng vào theo quy cách (**fscanf**, **fprintf**) sẽ giảm đáng kể thời gian thực hiện chương trình.

Chương trình I

```
#include<bits/stdc++.h>
#define NAME "olympiad."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef tuple<int,int,int> ti3;
int n,p[201],q[201],t=0,evl,k,u,v,id,gr;
float a,b,c,d;
vector<pair<int,int>>res;

int main()
{
    fi>>n;
    for(int i=0;i<n;++i) {fi>>p[i]; t+=p[i];}
    for(int i=0;i<n;++i) fi>>q[i];
    vector<ti3>rtab(t);
    for(int i=0;i<t;++i)
    {
        fi>>id>>gr>>a>>b>>c>>d;
        evl=int(a*100+b*100+c*100+d*100+0.005);
        rtab[i]=make_tuple(gr,evl,id);
    }
    sort(rtab.begin(),rtab.end(),
          [] (ti3 a,ti3 b) {return (get<0>(a)<get<0>(b) ||
                                (get<0>(a)==get<0>(b) && get<1>(a)>=get<1>(b)));}
    );
    u=0; v=p[0];
    for(int i=0;i<n;++i)
    {
        t=u+q[i]-1; k=0;
        int x=get<1>(rtab[t]);
        if(x==0) while(t>=u && get<1>(rtab[t])==0) --k,--t;
        else {++t; while(t<v && get<1>(rtab[t])>x) ++k,++t;}
        t=q[i]+k;
        for(int j=0;j<t;++j)
            res.push_back({get<1>(rtab[u+j]),get<2>(rtab[u+j])});
        u=v; v+=p[i+1];
    }
    sort(res.rbegin(),res.rend());
    fo<<res.size()<<'\n';
    for(auto &t:res)
        fo<<t.second<<' ' <<fixed<<setprecision(2)<<(float)t.first/100<<'\n';
    Times;
}
```

Chương trình II

```
#include<bits/stdc++.h>
#define NAME "olympiad."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
int n,p[201],q[201],t=0,evl,k,id,gr;
float a,b,c,d;
vector<pii>res;

int main()
{
    fi>>n;
    for(int i=0;i<n;++i) {fi>>p[i]; t+=p[i];}
    for(int i=0;i<n;++i) fi>>q[i];
    vector<pii>rtab[n];
    for(int i=0;i<t;++i)
    {
        fi>>id>>gr>>a>>b>>c>>d; --gr;
        evl=int(a*100+b*100+c*100+d*100+0.005);
        rtab[gr].push_back({evl,id});
    }
    for(int i=0;i<n;++i)
    {
        sort(rtab[i].rbegin(),rtab[i].rend());
        k=rtab[i][q[i]-1].first; if(k==0) k=1;
        for(int j=0;j<p[i];++j)
            if(rtab[i][j].first>=k) res.push_back(rtab[i][j]);
    }
    sort(res.rbegin(),res.rend());
    fo<<res.size()<<'\n';
    for(auto &t:res)
        fo<<t.second<<' '<<fixed<<setprecision(2)<<(float)t.first/100<<'\n';
    Times;
}
```



Alice và Bob ngồi cạnh nhau trong cuộc họp tổng kết công tác sáu tháng đầu năm của Công ty. Nội dung báo cáo không có gì hấp dẫn. Hai người quyết định chơi một trò chơi đơn giản để chống buồn ngủ. Alice viết một số nguyên dương không quá 4 chữ số trong hệ 10 và chuyển cho Bob. Nếu số nhận được chưa đủ 4 chữ số thì Bob thêm các số 0 vào vị trí tùy ý để nhận được số có 4 chữ số (có thể bắt đầu bằng 0), sau đó tăng một chữ số tùy chọn lên 1, 2 hoặc 3. Dĩ nhiên chữ số sau khi thay đổi phải không vượt quá 9. Kết quả được chuyển lại cho Alice. Alice tăng một chữ số tùy chọn lên 1, 2 hoặc 3 và chuyển lại cho Bob. Quá trình trên tiếp diễn cho đến khi nhận được số 9999. Ai đến lượt mình đi phải ghi số 9999 là thua.

Hãy xác định với số ban đầu đã viết Alice có thắng được hay không nếu cả 2 người biết cách đi tối ưu.

Dữ liệu: Vào từ file văn bản G9999.INP:

- ✚ Dòng đầu tiên chứa một số nguyên n – số lượng tests ($1 \leq n \leq 100$),
- ✚ Mỗi dòng trong n dòng tiếp theo chứa một số nguyên dương không quá 4 chữ số.

Kết quả: Đưa ra file văn bản G9999.OUT xâu độ dài n chỉ chứa các ký tự trong tập $\{Y, N\}$, ký tự Y ở vị trí i cho biết Alice thắng ở test thứ i , ký tự N nói lên rằng Alice thua ở test này ($i = 1 \div n$).

Ví dụ:

G9999.INP	G9999.OUT
<pre>4 9989 999 9899 5999</pre>	<pre>YYYYN</pre>



Giải thuật: Kỹ thuật bảng phương án.

Trò chơi nêu trong đầu bài thuộc loại *đối kháng* và *tài nguyên hạn chế*: hai người chơi và nếu một trong 2 người phạm sai làm trong cách chọn nước đi thì sẽ bị đối phương tận dụng để chuyển bại thành thắng, trò chơi chắc chắn kết thúc vì số nước đi có thể sẽ giảm dần trong quá trình chơi.

Nếu xét mỗi số như một ô trong bảng 4 chiều $b[10][10][10][10]$ bài toán có thể phát biểu lại như sau: từ ô $b[p][q][u][v]$ hai người lần lượt đi, mỗi người, khi đến lượt mình – chuyển tới ô mới theo một trục nào đó với tọa độ của trục đó lớn hơn và độ lớn không vượt quá 3. Không được phép đi ra khỏi bảng. Ai đến lượt mình phải đi vào ô $b_{9,9,9,9}$ là thua.

Vị trí (i, j, k, l) gọi là *vị trí thắng* nếu xuất phát từ đó, với cách đi tối ưu có thể buộc đối phương phải đi vào ô $(9, 9, 9, 9)$, trong trường hợp ngược lại – vị trí được gọi là *thua*. Mỗi ô chỉ có thể thuộc một trong 2 loại: thắng hoặc thua.

Ô $(9, 9, 9, 9)$ là vị trí thắng bởi vì đối phương đã tới đó trước và đã thua!

Đánh dấu các vị trí thua là **1** và vị trí thắng là **0**.

Một ô sẽ được đánh dấu là thua nếu với mọi khả năng thực hiện một bước đi, từ đó không thể tới một ô mới là ô thua.

Bảng phương án được xây dựng từ vị trí $(9, 9, 9, 9)$ lùi dần về $(0, 0, 0, 0)$.

Vị trí (i, j, k, l) sẽ là vị trí thắng nếu từ đó có cách đi tới vị trí thua, tức là ít nhất một trong số các ô

$$\begin{array}{lll} (i+1, j, k, l) & (i+2, j, k, l) & (i+3, j, k, l) \\ (i, j+1, k, l) & (i, j+2, k, l) & (i, j+3, k, l) \\ (i, j, k+1, l) & (i, j, k+2, l) & (i, j, k+3, l) \\ (i, j, k, l+1) & (i, j, k, l+2) & (i, j, k, l+3) \end{array}$$

phải là vị trí thua.

Để thuận tiện tính toán cần mở rộng bảng thêm 3 ô về phía cuối theo mỗi chiều, các ô mở rộng chứa giá trị 0.

Giá trị ô xuất phát trong bảng sẽ cho biết người thắng trong ván đó.

Tổ chức dữ liệu:

Mảng `int b[13][13][13][13]={0}` để ghi nhận bảng phương án. Có thể dùng mảng loại `bool` hoặc `int8_t` để tiết kiệm bộ nhớ.

Độ phức tạp của giải thuật: O(n).

Chương trình

```
#include<bits/stdc++.h>
#define NAME "g999."
#define Times fo<<"\nTime: "<<clock() / (double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int b[13][13][13][13]={0},t1,ls;
int n;
string s,ans="";
int main()
{
    for(int i=9;i>=0;--i)
        for(int j=9;j>=0;--j)
            for(int k=9;k>=0;--k)
                for(int l=9;l>=0;--l)
                {
                    t1=0;if(i+j+k+l==36)continue;
                    for(int iii=1;iii<4;++iii)t1+=b[i+iii][j][k][l];
                    for(int iii=1;iii<4;++iii)t1+=b[i][j+iii][k][l];
                    for(int iii=1;iii<4;++iii)t1+=b[i][j][k+iii][l];
                    for(int iii=1;iii<4;++iii)t1+=b[i][j][k][l+iii];
                    b[i][j][k][l]=t1==0;
                }
    fi>>n;
    for(int i=0;i<n;++i)
    {
        fi>>s;ls=s.size();
        if(ls<4) for(int i=0;i<4-ls;++i)s+='0';
        t1=b[s[0]-48][s[1]-48][s[2]-48][s[3]-48];
        if(t1)ans+='Y'; else ans+='N';
    }
    fo<<ans;
    Times;
}
```

