MỤC LỤC ATHENA XVII

WA31. BÀI TẬP *Tên chương trình: EXERCISE.CPP* 3
WA32.METRO *Tên chương trình: METRO.CPP* 6
WA33. CÂN ĐIỆN TỬ *Tên chương trình: BALANCE.CPP* 10
WA34. THI TRẮC NGHIỆM *Tên chương trình: CHOICE.CPP* 14
WA35. ẢO THUẬT *Tên chương trình: JUGGLE.CPP* 17
WA37. SỐ ĐƯỜNG ĐI *Tên chương trình: ROUTES.CPP* 22
WA38. DỊCH CHUYỂN *Tên chương trình: SHIFT.CPP* 35
WA39. NÉN TÍCH *Tên chương trình: COMPRESS.CPP* 39
WA40. CHẴN – LẺ *Tên chương trình: ODD_EVEN.CPP* 42
WA41. BẤT BIẾN ĐẠI SỐ *Tên chương trình: TOPO.CPP* 44
WA42. MÃ CAESAR *Tên chương trình: CAESAR.CPP* 49
WA43. KHU CÔNG NGHIỆP *Tên chương trình: INDZONE.CPP* 52
WA44. ĐƯỜNG KÍNH *Tên chương trình: DIAMETER.CPP* 55
WA45. KHỐI LẬP PHƯƠNG *Tên chương trình: CUBES.CPP* 59
WA46. BÓ HOA *Tên chương trình: BOUQUET.CPP* 63
WA47. THẮNG CỐ *Tên chương trình: BROTH.CPP* 66
WA48. PHỦ BÓNG *Tên chương trình: OVERLAP.CPP* 70
WA49. TAM GIÁC CÂN *Tên chương trình: ISOSCELE.CPP* 75
WA50. ĐỊNH LÝ FERMA LỚN *Tên chương trình: FERMAT.CPP* 77
WB01. THỨ TỰ TỪ ĐIỂN *Tên chương trình: LEXICO.CPP* 90
WB03. NGÀY THÁNG *Tên chương trình: DATE.CPP* 96
WB04. TÍNH NHẨM *Tên chương trình: MENTAL.CPP* 99
WB05. DÃY SỐ NGUYÊN TỐ *Tên chương trình: PRIMES.CPP* 102
WB06. KHÓA TRUY CẬP *Tên chương trình: KEY.CPP* 106
WB07. NHÓM 3 *Tên chương trình: TRIPLE.CPP* 110
WB08. XML *Tên chương trình: XML.CPP* 113
WB09.ĐÁM CƯỚI *Tên chương trình: WEDDING.CPP* 116
WB10. BÓNG BÀN *Tên chương trình: PINGPONG.CPP* 120
WB11. CƠ SỞ DỮ LIỆU *Tên chương trình: DATABASE.CPP* 127

WB12. HIỆU SỐ *Tên chương trình: DIF_SQR.CPP* 129
WB13. PHẠT TỐC ĐỘ *Tên chương trình: PAYMENT.CPP* 132
WB15. TÍCH TRỌNG SỐ *Tên chương trình: MAXPROD.CPP* 136
WB16. QUY HOẠCH *Tên chương trình: LAYOUT.CPP* 139
WB17. ATM *Tên chương trình: ATM.CPP* 142
WB18. XÀ LAN *Tên chương trình: BARGE.CPP* 145
WB19. ĐƯỜNG THỬ NGHIỆM *Tên chương trình: SEGMENTS.CPP* 148
WB20. CHỤP ẢNH *Tên chương trình: PHOTO.CPP* 152
WB21. NHÓM 3 *Tên chương trình: TROIKA.CPP* 155
WB22. THỰC PHẨM *Tên chương trình: PRODUCTS.CPP* 158
WB23. CHUYỂN BƯU KIỆN *Tên chương trình: POSTROBOT.CPP* 161
WB24. BIỂU THỨC NGOẶC ĐÚNG *Tên chương trình: CBE.CPP* 168
WB25. KHOẢNG CÁCH *Tên chương trình: DISTANCES.CPP* 174
WB26. BÃO CÁT *Tên chương trình: SANDSTORM.CPP* 177
WB27. BÀI TẬP *Tên chương trình: EXERCISES.CPP* 179
WB28. DỮ LIỆU LỚN *Tên chương trình: BIGDATA.CPP* 181
WB29. VẮC XIN *Tên chương trình: VACCINE.CPP* 184
WB30. TỔNG FIBONACCI *Tên chương trình: FIBSUM.CPP* 189
WB31. CHẤT LƯỢNG *Tên chương trình: QUALITY.CPP* 193
WB32. ĐỘI TUYỂN *Tên chương trình: TEAM.CPP* 196
WB33. CỬA TRƯỢT *Tên chương trình: MOVEMENT.CPP* 199

WA31. BÀI TẬP

Tên chương trình: EXERCISE.CPP

Giáo sư Braun chuẩn bị bài tập về toán rời rạc cho sinh viên. Dự kiến đầu bài sẽ là "*Cho* n *số nguyên* $x_1, x_2, \ldots, x_n$ *và số nguyên dương* m *(*$1 \le m < 2^{30}$*). Yêu cầu tìm* n *số nguyên* $a_1, a_2, \ldots, a_n$ *,* a_i *nhận giá trị từ tập {*$-1, 0, 1$*},* $i = 1 \div n$ *sao cho* $\sum_{i=1}^{n} a_i \times x_i$ *chia hết cho* m".

Ông chuẩn bị sẵn đáp án là dãy số nguyên $a_1, a_2, \ldots, a_n$, $-1 \le a_i \le 1$ với mọi i và có ít nhất một số khác 0. Để dễ chấm bài ông tìm dãy số nguyên $x_1, x_2, \ldots, x_n$ và số nguyên m sao cho dãy số a_i, $i = 1 \div n$ đã chọn là đáp án duy nhất. Đáng tiếc, điều đó là không thể vì nếu $a_1, a_2, \ldots, a_n$ là nghiệm thì $-a_1, -a_2, \ldots, -a_n$ cũng là nghiệm thỏa mãn yêu cầu bài toán.

Cuối cùng giáo sư Braun đành chấp nhận tìm dãy số nguyên $x_1, x_2, \ldots, x_n$ và số nguyên m sao cho kết quả chỉ có thể là $a_1, a_2, \ldots, a_n$ hoặc $-a_1, -a_2, \ldots, -a_n$.

Hãy đưa ra m và dãy số nguyên $x_1, x_2, \ldots, x_n$. Trong trường hợp tồn tại nhiều dãy số thỏa mãn – đưa ra dãy số tùy chọn.

Dữ liệu: Vào từ file EXERCISE.INP:

- Dòng đầu tiên chứa số nguyên n ($1 \le n \le 30$),
- Dòng thứ 2 chứa n số nguyên $a_1, a_2, \ldots, a_n$, , $-1 \le a_i \le 1$ với mọi i .

Kết quả: Đưa ra file văn bản EXERCISE.OUT, dòng đầu tiên chứa số nguyên m, dòng thứ 2 chứa n số nguyên $x_1, x_2, \ldots, x_n$ xác định được.

Ví dụ:

EXERCISE.INP	EXERCISE.OUT
3 0 -1 1	6 1 -2 4

WA27 StP 20192710 E A XVI

Giải thuật: Xử lý bit.

Để tiện xử lý

Xét trường hợp $a_i \neq 0$ với mọi i. Đặt $x_i = a_i \times 2^i$, $i = 0 \div n-1$, ta có:

$$\sum_{i=0}^{n-1} a_i \times x_i = \sum_{i=0}^{n-1} 2^i = 2^n - 1.$$

Như vậy cần chọn $m = 2^n - 1$.

Trường hợp tồn tại $a_i = 0$:

Gọi z là số lượng phần tử $a_i = 0$.

Ví dụ, với $n = 10$ và $A = (0, 1, 1, 1, 1, 0, 0, 0, 1, 0) \rightarrow z = 5$.

Giữ nguyên trình tự bộ phận, đổi chỗ để các cặp (a_i, i) với $a_i \neq 0$ về cuối, các cặp có $a_i = 0$ về đầu.

Nếu (a_i, i) ở vị trí k trong dãy đã đổi chỗ thì $x_i = 2^k$.

Giá trị m cần chọn sẽ là $\sum_{i=0}^{n-1} |a_i| \times 2^i - 1$.

Độ phức tạp của giải thuật: O(*n*).

Chương trình

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

int main()
{
    int n;
    fi >> n;
    vector<int> a(n);
    int z = 0;
    for (int i = 0; i < n; i++)
        fi >> a[i], z += (a[i] == 0);
    fo << (((1 << (n - z)) - 1) << z) << '\n';
    int bz = 0;

    for (int i = 0; i < n; i++)
        if (a[i] == 0)
            fo << (1 << bz++) << ' ';
        else
            fo << (1 << (i - bz + z)) * a[i] << ' ';

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

Thành phố có 3 tuyến Metro, tuyến I có ***x*** ga đánh số từ 1 đến ***x***, tuyến II có ***y*** ga đánh số từ 1 đến ***y*** và tuyến III có ***z*** ga đánh số từ 1 đến ***z***.

Các tuyến giao nhau: từ ga ***a*** của tuyến I có thể chuyển sang ga ***b*** thuộc tuyến II và ngược lại, từ ga ***a+1*** của tuyến I có thể chuyển sang ga ***c*** của tuyến III và ngược lại, từ ga ***b+1*** của tuyến II có thể chuyển sang ga ***c+1*** của tuyến III và ngược lại. Thời gian đi từ một ga sang ga tương ứng ở tuyến khác là ***d*** giây.

Các tuyến xây dựng ở những thời kỳ khác nhau nên có tốc độ khác nhau. Thời gian đi từ một ga tới ga tiếp sau của tuyến I là ***t1***, ở tuyến II là ***t2*** và ở tuyến 3 là ***t3***.

Thời gian dừng ở mỗi ga cũng để hành khách ra vào là không đáng kể.

Alice ở gần ga ***i*** tuyến ***K***. Hôm nay cô có nhiệm vụ trực kỹ thuật cho buổi liên hoan văn nghệ chào mừng ngày thành lập Đoàn. Trường của Alice ở gần ga ***j*** trên tuyến ***L***. Vốn đã đi lại nhiều lần, cô dễ dàng tới được trường một cách nhanh nhất.

Hãy xác định thời gian ngắn nhất để Alice đi từ nhà tới trường.

Dữ liệu: Vào từ file METRO.INP:

- Dòng đầu tiên chứa 3 số nguyên ***x***, ***y*** và ***z*** ($2 \le x, y, z \le 10^9$),
- Dòng thứ 2 chứa 3 số nguyên ***a***, ***b*** và ***c*** ($1 \le a < x$, $1 \le b < y$, $1 \le c < z$),
- Dòng thứ 3 chứa 4 số nguyên ***t1***, t***2***, ***t3*** và ***d*** ($1 \le t1, t2, t3, d \le 10^9$),
- Dòng thứ 4 chứa 4 số nguyên ***i***, ***K***, ***j*** và ***L*** ($1 \le K, L \le 3$), đảm bảo tồn tại các ga đã nêu, nơi xuất phát và nơi đến không trùng nhau.

Kết quả: Đưa ra file văn bản METRO.OUT một số nguyên – thời gian ngắn nhất tìm được.

Ví dụ:

METRO. INP
4 4 4 2 2 2 1 10 1 1 1 1 3 4

METRO.OUT
5

WA31 StP 20192710 F A XVII

Giải thuật: Kỹ thuật bảng phương án.

Có 4 khả năng xảy ra khi đi từ **i** tới **j**

- Các ga **i** và **j** nằm trên cùng một tuyến (**K** = **L**):
  - Không chuyển tuyến,
  - Chuyển tuyến 3 lần,
- Các ga **i** và **j** nằm trên các tuyến khác nhau (**K** ≠ **L**):
  - Chuyển tuyến một lần,
  - Chuyển tuyến hai lần.

Xây dựng bảng **cross** đánh dấu những đường đi có thể ở các ga cho phép chuyển tuyến:

(-1,-1)	(a,b+1)	(a+1,c)
(b+1,a)	(-1,-1)	(b,c+1)
(c,a+1)	(c+1,b)	(-1,-1)

Dựa trên bảng chuyển tuyến tính độ dài các đường đi có thể và chọn đường đi nhanh nhất.

Tổ chức dữ liệu:

- Mảng `int64_t t[3]={t1,t2,t3}` – Ghi nhận thời gian đi đến ga tiếp theo ở mỗi tuyến,
- Mảng `int len[3]={x,y,z}` – Ghi nhận số lượng ga ở mỗi tuyến,
- Bảng `pair<int, int> croses[3][3]` – Ghi nhận cách chuyển tuyến.

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int64_t INF = (int64_t)2e18;
int main()
{
    int x, y, z;
    fi >> x >> y >> z;
    int a, b, c;
    fi >> a >> b >> c;
    int t1, t2, t3, d;
    fi >> t1 >> t2 >> t3 >> d;
    int k, i, l, j;
    fi >> k >> i >> l >> j;

    int len[3] = {x, y, z};
    int64_t t[3] = {t1, t2, t3};
    pair<int, int> croses[3][3] = {{{-1, -1}, {a, b + 1}, {a + 1, c}},
                                   {{b + 1, a}, {-1, -1}, {b, c + 1}},
                                   {{c, a + 1}, {c + 1, b}, {-1,-1}}};

    int64_t ans = INF;
    if (k == l)
    {
        //zero cross case
        ans = abs(i - j) * t[k - 1];
        //three cross case
        pair<int, int> others[3] = {{2, 3}, {1, 3}, {1, 2}};
        int next1 = others[k - 1].first;
        int next2 = others[k - 1].second;

        int from1 = croses[k - 1][next1 - 1].first;
        int to3 = croses[next2 - 1][l - 1].second;

        int64_t ans1 = 3ll * d + t[next1 - 1] + t[next2 - 1] +
                     abs(i - from1) * t[k - 1] + abs(j-to3) * t[l-1];

        swap(next1, next2);

        from1 = croses[k - 1][next1 - 1].first;
        to3 = croses[next2 - 1][l - 1].second;

        int64_t ans2 = 3ll * d + t[next1 - 1] + t[next2 - 1] +
                     abs(i - from1) * t[k - 1] + abs(j-to3)*t[l - 1];
        ans = min(ans, min(ans1, ans2));
    }
    else
```

```
    {
        // one cross case
        int from = croses[k - 1][l - 1].first;
        int to = croses[k - 1][l - 1].second;
        int64_t ans1 = abs(i - from) * t[k - 1]+abs(j-to)*t[l-1] + d;

        //two cross case
        int other = k ^ l;
        int from1 = croses[k - 1][other - 1].first;
        int to1 = croses[k - 1][other - 1].second;
        int from2 = croses[other - 1][l - 1].first;
        int to2 = croses[other - 1][l - 1].second;

        int64_t ans2 = abs(i - from1) * t[k - 1] +
                    abs(j - to2) * t[l - 1] + 2ll * d + t[other - 1];
        ans = min(ans1, ans2);
    }
    fo << ans << endl;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Ủy ban Quản lý Sở hữu trí tuệ nhận được đơn xin cấp bằng phát minh sáng chế cân đĩa điện tử. Cân có hai đĩa và bộ quả cân gồm n quả cân, quả thứ i có khối lượng a_i, $i = 1 \div n$. Vật cần cân được đặt trên một đĩa, các quả cân có thể bỏ ở cả hai đĩa.

Nếu khối lượng ở hai đĩa bằng nhau thì dĩ nhiên trên bảng điện tử của cân sẽ thông báo kết quả là bằng nhau. Nhưng chiếc cân này độc đáo ở chỗ có một số trường hợp khi khối lượng đặt ở hai đĩa khác nhau nhưng kết quả vẫn thông báo là bằng bởi vì hệ thống xử lý điện tử so sánh không phải là khối lượng ở trên hai đĩa mà là phần dư của khối lượng khi chia cho số nguyên m. Số m có thể nêu ở mỗi lần cân.

Để minh họa cho hoạt động của cân, theo số m mà Ủy ban nêu, tác giả xin cấp bằng phát minh cần đặt một số quả cân lên đĩa thứ nhất và một số quả cân trên đĩa thứ 2 để cân cho kết quả là bằng nhau.

Tác giả phải đặt ít nhất một quả cân lên đĩa. Hãy xác định những quả cân nào cần đặt lên đĩa thứ nhất và những quả cân nào cần đặt lên đĩa thứ hai.

Dữ liệu: Vào từ file BALANCE.INP:

- Dòng đầu tiên chứa 2 số nguyên n và m ($1 \le n \le 25$, $1 \le m \le 4\times10^7$),
- Dòng thứ 2 chứa n số nguyên $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$, $i = 1 \div n$).

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

- Dòng thứ nhất chứa số nguyên p – số lượng quả cân cần đặt lên đĩa thứ nhất,
- Dòng thứ 2 chứa p số nguyên xác định số thứ tự của các quả cân được đặt lên đĩa thứ nhất,
- Dòng thứ 3 chứa số nguyên q – số lượng quả cân cần đặt lên đĩa thứ thứ 2,
- Dòng thứ 2 chứa q số nguyên xác định số thứ tự của các quả cân được đặt lên đĩa thứ hai.

Nếu p hoặc q bằng 0 thì sẽ không có dòng tiếp theo xác định các quả cân cần đặt

Nếu không có cách đặt thì đưa ra số -1.

Ví dụ:

BALANCE.INP
4 14 1 3 7 10

BALANCE.OUT
1 4 2 2 3

Giải thuật: Kỹ thuật bảng phương án, Duyệt tổ hợp .

Duyệt các tổ hợp chọn bộ quả cân đặt lên đĩa, ghi nhận *khối lượng* mỗi bộ được chọn *theo mô đun* **m** cùng với thông tin về bộ quả cân.

Với giá trị nhận được từ mỗi tổ hợp, kiểm tra xem trước đó đã có tổ hợp với giá trị khối lượng nhận được như vậy hay chưa.

Nếu có rồi – lọc bỏ đi những quả cân chung trong 2 bộ có cùng khối lượng ghi nhận và đưa ra kết quả.

Cách duyệt tổ hợp:

- Dùng bản đồ **n** bít,
- Mỗi số nguyên **k** trong phạm vi từ 0 đến 2^n-1 xác định một tổ hợp chọn quả cân,
- Bít thứ **i** trong **k** bằng 1 tương ứng với việc quả cân thứ **i** được chọn.

Kết quả xử lý mỗi bộ quả cân là cặp giá trị (*Số_lượng_quả_cân*, *Khối_lượng*).

Tổ chức dữ liệu:

- Mảng `vector<int64_t> a(n)` – Lưu khối lượng các quả cân,
- Mảng `vector<pair<int, int>> fit(m, {0, 0})` – Lưu kết quả xử lý bộ quả cân.

Xử lý:

Duyệt tổ hợp:

```
for (int mask = 1; mask < (1 << n); mask++)
{
    int cnt = 0;
    int64_t sum = 0;
    for (int bit = 0; bit < n; bit++)
        if ((mask >> bit) & 1)
        {
            cnt++;
            sum += a[bit];
        }
    int mod = sum % m;
    . . . . . . .
    . . . . . . .
}
```

Xác định vị trí bít 1

Kiểm tra điều kiện tìm được kết quả và lọc các quả cân trùng:

Có bộ quả cân cùng khối lượng (mod m)

Lọc bít trùng

```
int mod = sum % m;
if (mod == 0 || fit[mod].first)
{
    int mask0,mask1,mask2;
    mask0 = mask & fit[mod].second;
    mask1 = mask ^ mask0;
    mask2 = fit[mod].second ^ mask0;
    print_masked(a, mask1);
    print_masked(a, mask2);
    cout<<"\nTime: "<<clock()/(double)1000<<" sec";
    return 0;
} else fit[mod] = {cnt, mask};
```

Dẫn xuất kết quả:

```
void print_masked(vector<int64_t> const &a, int mask)
{
    vector<int> res;
    for (int bit = 0; bit < n; bit++)
        if ((mask >> bit) & 1) res.push_back(bit + 1);
    cout << res.size() << '\n';
    for (int t : res) cout << t << ' ';
    cout << '\n';
}
```

Ghi nhận số thứ tự quả cân

Độ phức tạp của giải thuật: $O(n2^n)$.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
int n, m;
void print_masked(vector<int64_t> const &a, int mask)
{
    vector<int> res;
    for (int bit = 0; bit < n; bit++)
        if ((mask >> bit) & 1) res.push_back(bit + 1);
    cout << res.size() << '\n';
    for (int t : res) cout << t << ' ';
    cout << '\n';
}
int main()
{
    freopen("balance.inp", "r", stdin);
    freopen("balance.out", "w", stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    cin >> n >> m;
    vector<int64_t> a(n);
    for(int64_t &i:a) cin>>i;
    vector<pair<int, int>> fit(m, {0, 0});
    for (int mask = 1; mask < (1 << n); mask++)
    {
        int cnt = 0;
        int64_t sum = 0;
        for (int bit = 0; bit < n; bit++)
            if ((mask >> bit) & 1)
            {
                cnt++;
                sum += a[bit];
            }
        int mod = sum % m;
        if (mod == 0 || fit[mod].first)
        {
            int mask0,mask1,mask2;
            mask0 = mask & fit[mod].second;
            mask1 = mask ^ mask0;
            mask2 = fit[mod].second ^ mask0;
            print_masked(a, mask1);
            print_masked(a, mask2);
            cout<<"\nTime: "<<clock()/(double)1000<<" sec";
            return 0;
        } else fit[mod] = {cnt, mask};
    }
    cout << "-1\n";
    cout<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

WA34. THI TRẮC NGHIỆM

Tên chương trình: CHOICE.CPP

Kỳ thi cuối học kỳ được thực hiện theo hình thức trắc nghiệm. Hệ thống lần lượt đưa ra các câu hỏi và 4 đáp án ***A***, ***B***, ***C***, ***D***, trong đó chỉ có một đáp án đúng. Người thi phải chọn đáp án mà mình coi là đúng. Có ***n*** câu hỏi đã được đưa ra.

Lớp có ***m*** học sinh. Kết quả làm bài của mỗi học sinh là một xâu độ dài ***n*** chỉ chứa các ký tự trong tập {***A***, ***B***, ***C***, ***D***}.

Giáo viên phụ trách biết học sinh thường nhìn và copy kết quả của nhau. Ông quyết định lọc ra các cặp bài tương tự nhau để xem lại và ra quyết định cuối cùng. Hai bài được coi là tương tự nhau nếu hơn một nửa kết quả đúng của 2 bài là trùng nhau và hơn một nữa kết quả sai trùng nhau.

Hãy đưa ra số lượng cặp bài tương tự nhau và chỉ ra các cặp tương tự.

Dữ liệu: Vào từ file CHOICE.INP:

- Dòng đầu tiên chứa số nguyên ***n*** ($1 \le n \le 100$),
- Dòng thứ 2 chứa xâu độ dài ***n*** từ các ký tự thuộc tập {***A***, ***B***, ***C***, ***D***}, ký tự thứ ***i*** là đáp án đúng của câu hỏi ***i***, ***i*** = 1 ÷ ***n***,
- Dòng thứ 3 chứa số nguyên ***m*** ($1 \le m \le 100$),
- Dòng thứ ***j*** trong ***m*** dòng sau chứa xâu độ dài ***n*** từ các ký tự thuộc tập {***A***, ***B***, ***C***, ***D***}, ký tự thứ ***i*** là đáp án của thí sinh ***j*** cho câu hỏi ***i***, i = 1 ÷ ***n***.

Kết quả: Đưa ra file văn bản CHOICE.OUT, dòng đầu tiên chứa số nguyên ***k*** – số cặp bài tương tự nhau. Nếu ***k*** > 0 – mỗi dòng trong ***k*** dòng sau chứa cặp số nguyên xác định cặp bài tương tự nhau. Số thứ tự trong mỗi cặp là tùy chọn.

Ví dụ:

CHOICE. INP
6 ABCDAB 3 ABCCCC BBCDCC ACCDCC

CHOICE.OUT
3 1 2 1 3 2 3

Giải thuật: Duyệt vét cạn .

Tính số lượng kết quả đúng ở mỗi bài,

Duyệt tất cả các cặp bài:

- Đếm số câu trả trả lời giống nhau và đúng,
- Đếm số câu trả trả lời giống nhau và sai,
- Kiểm tra tiêu chuẩn "bài tương tự" và ghi nhận cặp bài đó nếu thỏa mãn điều kiện kiểm tra.

Tổ chức dữ liệu:

- Mảng `string tests[m]` – Ghi nhận kết quả mỗi học sinh,
- Mảng `int correct[m]` – Ghi nhận số kết quả đúng ở mỗi bài,
- Mảng `vector<pair<int, int> > similar` – Ghi nhận các cặp bài tương tự nhau.

Xử lý:

Thống kê kết quả đúng:

```
int correct[m];
for (int i = 0; i < m; i++)
{
    correct[i] = 0;

    for (int j = 0; j < n; j++)
        if (tests[i][j] == answers[j]) correct[i]++;
}
```

Duyệt kết quả các câu hỏi

Kiểm tra cặp bài:

Điều kiện "**x** lớn hơn một nữa **y**" được kiểm tra dưới dạng **2*x > y** .

```
vector<pair<int, int> > similar;
for (int i = 0; i < m; i++)
    for (int j = i + 1; j < m; j++)
    {
        int same_correct = 0, same_wrong = 0;
        for (int k = 0; k < n; k++)
            if (tests[i][k] == tests[j][k])
                if (tests[i][k] == answers[k]) same_correct++;
                    else same_wrong++;

        if (same_correct * 2 > max(correct[i], correct[j]) &&
            same_wrong * 2 > max(n - correct[i], n - correct[j]))
                similar.push_back({i, j});
    }
```

Tính số kết quả Đúng /Sai giống nhau

Độ phức tạp của giải thuật: $O(m^2 \times n)$

Chương trình

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

int main()
{
    freopen("choice.inp", "r", stdin);
    freopen("choice.out", "w", stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    int n;
    cin >> n;
    string answers;
    cin >> answers;
    int m;
    cin >> m;
    string tests[m];
    for (int i = 0; i < m; i++) cin >> tests[i];
    int correct[m];
    for (int i = 0; i < m; i++)
    {
        correct[i] = 0;
        for (int j = 0; j < n; j++)
            if (tests[i][j] == answers[j]) correct[i]++;
    }
    vector<pair<int, int> > similar;
    for (int i = 0; i < m; i++)
        for (int j = i + 1; j < m; j++)
        {
            int same_correct = 0, same_wrong = 0;
            for (int k = 0; k < n; k++)
                if (tests[i][k] == tests[j][k])
                    if (tests[i][k] == answers[k]) same_correct++;
                        else same_wrong++;

            if (same_correct * 2 > max(correct[i], correct[j]) &&
                same_wrong * 2 > max(n - correct[i], n - correct[j]))
                                similar.push_back({i, j});
        }

    cout << similar.size() << "\n";
    for (auto it : similar)
        cout << it.first + 1 << " " << it.second + 1 << "\n";

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

Alice đi xem xiếc và được mời lên sân khấu tham gia một tiết mục ảo thuật. Ảo thuật gia đề nghị Alice viết và giữ riêng cho mình một hoán vị $a_1, a_2, \ldots, a_n$ các số tự nhiên từ 1 đến n, với mỗi i từ 1 đến n Alice xét bộ 3 $\{a_i, a_{i+1}, a_{i+2}\}$, $a_{n+1} = a_1$, $a_{n+2} = a_2$, viết ra mẩu giấy con 3 số này theo trình tự tùy ý. Tất cả n mẩu giấy con này được xáo trộn và đưa cho ảo thuật gia.

Sau một thời gian chăm chú xem các mẩu giấy nhận được, ảo thuật gia ghi lên bảng hoán vị mà Alice đã viết, bắt đầu từ 1.

Hãy xác định hoán vị mà ảo thuật gia tìm được.

Dữ liệu: Vào từ file JUGGLE.INP:

- Dòng đầu tiên chứa số nguyên n ($3 \le n \le 2\times10^5$),
- Mỗi dòng trong n dòng sau chứa 3 số nguyên đã ghi trên một mẩu giấy.

Dữ liệu đảm bảo hợp lý với hoán vị đã nghĩ của Alice.

Kết quả: Đưa ra file văn bản JUGGLE.OUT trên một dòng hoán vị được ghi trên bảng.

Ví dụ:

JUGGLE. INP
6 3 4 1 5 1 6 5 4 2 2 4 3 2 5 6 6 1 3

JUGGLE.OUT
1 3 4 2 5 6

Giải thuật: Kỹ thuật tổ chức dữ liệu.

Mỗi *cặp số đứng liên tiếp* nhau trong hoán vị cần tìm sẽ *xuất hiện 2 lần* trong các bộ 3 đã cho.

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

Xét một bộ 3 số trong dữ liệu vào.

Để tiện lưu trữ và tìm kiếm: Sắp xếp các số trong bộ 3 theo thứ tự tăng dần.

Gọi kết quả sắp xếp là (**a, b, c**).

Có 3 cách tách nhóm 3 này thành một một cặp 2 số và số tiếp theo:

- ((**a,b**),**c**)
- ((**a,c**),**b**)
- ((**b,c**),**a**)

Khi cố định một cặp 2 số thì số thứ 3 đứng kề sẽ trở nên tiền định, đồng thời cũng xác định cặp số tiếp theo chứa số thứ 3.

Với mỗi cách tách đã nêu, tìm các số tiếp theo. Nếu kết quả tìm kiếm cho dãy **n** số thì đó là nghiệm bài toán. Vấn đề còn lại chỉ là xoay dãy số, đưa 1 về vị trí đầu.

Trường hợp riêng: Với **n** ≤ 4 – mọi hoán vị đều phù hợp!

Tổ chức dữ liệu:

- Mảng `vector<array<int, 3>> in(n)` – Lưu dữ liệu vào,
- Bản đồ `map<pair<int, int>, vector<int>> lists` – Lưu quan hệ cặp số và số thứ 3,
- Mảng `vector<int> cur` – Lưu kết quả cần tìm.

*Xử lý:*Lưu quan hệ cặp số và số thứ 3:

```
for (int i = 0; i < n; ++i)
{
    cin>>in[i][0]>>in[i][1]>>in[i][2];
    --in[i][0]; --in[i][1] ; --in[i][2] ;

    sort(in[i].begin(), in[i].end());
    lists[{in[i][0], in[i][1]}].push_back(in[i][2]);
    lists[{in[i][0], in[i][2]}].push_back(in[i][1]);
    lists[{in[i][1], in[i][2]}].push_back(in[i][0]);
}
```

Duyệt cặp số:

```
for (int st = 0; st < 3; ++st)
    {
        vector<int> cur;
        for (int i = 0; i < 3; ++i)
            cur.push_back(in[0][(st + i) % 3]);

        while (cur.size() < n)
        {
            int a = cur.back();
            int b = cur[cur.size() - 2];

            bool ok = false;
            for (int x: lists[{min(a,b), max(a,b)}])
                if (x != cur[cur.size() - 3])
                {
                    cur.push_back(x);
                    ok = true;
                    break;
                }

            if (not ok) break;
        }

        if (cur.size() == n)
            ..... // Đưa ra kết quả

    }
```

Xác định số đứng kề

Xác định các số tiếp theo

Gặp lại số đã xét

Dấu hiệu số mới

Đưa ra kết quả:

Tìm vị trí số 0 trong dãy

```
vector<int>::iterator it=find(cur.begin(),cur.end(),0);
rotate(cur.begin(),it,cur.end());
for (int elem: cur) cout << 1 + elem << ' ';
cout << '\n';
cout << "\nTime: "<<clock()/(double)1000<<" sec";
return 0;
```

Độ phức tạp của giải thuật: O(*n*).

Chương trình

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

int main()
{
    freopen("juggle.inp","r",stdin);
    freopen("juggle.out","w",stdout);
    std::iostream::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    int n;
    cin>>n;
    if (n <= 4)
    {
        for (int i = 1; i <= n; ++i)
            cout << i << " ";
        cout << "\n";
        return 0;
    }

    vector<array<int, 3>> in(n);
    map<pair<int, int>, vector<int>> lists;
    for (int i = 0; i < n; ++i)
    {
        cin>>in[i][0]>>in[i][1]>>in[i][2];
        --in[i][0]; --in[i][1] ; --in[i][2] ;

        sort(in[i].begin(), in[i].end());
        lists[{in[i][0], in[i][1]}].push_back(in[i][2]);
        lists[{in[i][0], in[i][2]}].push_back(in[i][1]);
        lists[{in[i][1], in[i][2]}].push_back(in[i][0]);
    }

    for (int st = 0; st < 3; ++st)
    {
        vector<int> cur;
        for (int i = 0; i < 3; ++i)
            cur.push_back(in[0][(st + i) % 3]);

        while (cur.size() < n)
        {
            int a = cur.back();
            int b = cur[cur.size() - 2];

            bool ok = false;
            for (int x: lists[{min(a,b), max(a,b)}])
                if (x != cur[cur.size() - 3])
                {
                    cur.push_back(x);
```

```
                    ok = true;
                    break;
                }

            if (not ok) break;
        }

        if (cur.size() == n)
        {
            vector<int>::iterator it=find(cur.begin(),cur.end(),0);
            rotate(cur.begin(),it,cur.end());
            for (int elem: cur) cout << 1 + elem << ' ';
            cout << '\n';
            cout << "\nTime: "<<clock()/(double)1000<<" sec";
            return 0;
        }
    }
    return 0;
}
```

Cho lưới ô vuông kích thước $n \times m$, ô (1, 1) ở góc dưới trái, ô (n,m) – trên phải. Có k ô chứa chướng ngại vật, ô thứ i ở tọa độ (x_i, y_i), $1 \le x_i \le n$, $1 \le y_i \le m$, $i = 1 \div k$, Không có chướng ngại vật ở ô (1, 1) và (n, m).

Rô bốt xuất phát từ ô (1, 1), ở mỗi bước được chuyển sang ô kề cạnh bên phải hoặc bên trên nếu ô tới không chứa chướng ngại vật.

Hãy xác định số lượng đường rô bốt có thể đi từ ô (1, 1) đến ô (n, m) và đưa ra số lượng theo mô đun p, trong đó p – số nguyên tố.

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

- Dòng đầu tiên chứa 4 số nguyên n, m, k và p ($1 \le n, m \le 10^5$, $0 \le k \le 100$, $2 \times max\{m,n\} < p < 2 \times 10^9$),
- Nếu $k > 0$, dòng thứ i trong k dòng tiếp theo chứa 2 số nguyên x_i, y_i ($1 \le x_i \le n$, $1 \le y_i \le m$).

Kết quả: Đưa ra file văn bản ROUTES.OUT một số nguyên không âm – số lượng đường tìm được theo mô đun p.

Ví dụ:

ROUTES.INP
5 6 3 101 2 2 3 5 4 2

ROUTES.OUT
25

Giải thuật: Nguyên lý bù trừ và quy hoạch động theo lát cắt .

Giải thuật I

Với **n** và **m** đủ nhỏ (**n** , **m** ≤ 1000) bài toán tìm số lượng đường đi dễ dàng giải quyết bằng *phương pháp quy hoạch động* với độ phức tạp O($n \times m$).

Gọi $\mathbf{dp}_{i,j}$ – số lượng đường đi hợp lệ từ ô (1,1) tới ô (**i**,**j**), $\mathcal{K}$ – tập các ô có vật cản.

Ban đầu $\mathbf{dp}_{i,j}$ = -1 nếu (**i**, **j**) ∈ $\mathcal{K}$.

Công thức lặp:

$$\mathbf{dp}_{i,j} = \begin{cases} 1 \text{ với } \mathbf{i} = 1 \text{ hoặc } \mathbf{j} = 1, (\mathbf{i}, \mathbf{j}) \notin \mathcal{K}, \\ 0 \text{ nếu } (\mathbf{i}, \mathbf{j}) \in \mathcal{K}, \\ (\mathbf{dp}_{i-1,j} + \mathbf{dp}_{i,j-1}) \; mod \; \mathbf{p} \text{ với } (\mathbf{i}, \mathbf{j}) \notin \mathcal{K}. \end{cases}$$

Tổng số đường đi cần tìm là giá trị $\mathbf{dp}_{n,m}$.

Chương trình I:

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

int n,m,k,p,x,y;
vector<vector<int>> dp;

int main()
{
    fi>>n>>m>>k>>p;
    dp.resize(n+1,vector<int> (m+1,0));

    for(int i=0; i<k; ++i)
    {
        fi>>x>>y;
        dp[x][y]=-1;
    }
    dp[0][1]=1;
    for(int i=1; i<=n; ++i)
```

```
        for(int j=1; j<=m; ++j)
            if(dp[i][j]==-1) dp[i][j]=0;
                    else dp[i][j]=(dp[i-1][j]+dp[i][j-1])%p;

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

Giải thuật II

Với **n**, **m** lớn hơn 10^3, việc sử dụng mảng **dp** 2 chiều đòi hỏi bộ nhớ sử dụng lớn, vượt quá khả năng phục vụ của hệ thống lập trình.

Dòng thứ **i** của bảng **dp** được tính dựa trên dòng **i-1**. Vì vậy chỉ cần giữ 2 dòng của bảng và dùng kỹ thuật con lắc để từ dòng cũ tính dòng mới.

Trong trường hợp này cần lưu các ô chứa vật cản và sắp xếp thứ tự từ điển tăng dần của các tọa độ.

Độ phức tạp của giải thuật vẫn là O(n×m), nhưng bộ nhớ sử dụng chỉ thuộc bậc O(m).

Chương trình II:

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
ifstream fi ("route.inp");
ofstream fo ("route.out");
typedef pair<int,int> pii;
int n,m,k,p,x,y,u=1,v=0,ib=0;
int main()
{
    fi>>n>>m>>k>>p;
    vector<vector<int>> dp(2,vector<int> (m+1,0));
    vector<pii>ob(k);                    // Lưu các ô cấm
    for(int i=0; i<k; ++i)
    {
        fi>>x>>y;
        ob[i]={x,y};
    }
    sort(ob.begin(),ob.end());
    ob.push_back({n+1,m+1});             // Phần tử hàng rào
    dp[0][1]=1;
```

```
    for(int i=1; i<=n; ++i)
    {
        u^=1; v^=1;                    // Tạo con lắc
        for(int j=1; j<=m; ++j)
            if(ob[ib]==make_pair(i,j))dp[v][j]=0,++ib;
                    else dp[v][j]=(dp[v][j-1]+dp[u][j])%p;
    }
    fo<<dp[v][m];
    fo<<"\nTime: "<<clock()/(double)1000<<" sec ";

}
```

Thời gian thực hiện chương trình lớn vì độ phức tạp của giải thuật là O($n \times m$).

Giải thuật III

Áp dụng nguyên lý bù trừ có thể xây dựng giải thuật có *độ phức tạp chỉ phụ thuộc vào* ***k***. Như vậy sẽ xây dựng cho phép làm việc với *bảng kích thước rất lớn* (**n**, **m** $\leq 10^9$). Tuy vậy dưới đây ta chỉ xét *chi tiết cách triển khai* giải thuật cho trường hợp **n, m** $\leq 10^6$.

Mảng tọa độ ô cấm cần lưu dưới dạng sắp xếp theo thứ tự từ điển.

Xét số lượng đường đi từ **A** tới **B**, kể cả đi qua ô cấm (nếu có). Gọi số ô cần chuyển sang phải là **x** và sau đó phải chuyển lên trên **y** ô. Khi đó số lượng đường đi từ **A** tới **B** sẽ là C_{x+y}^{x} .

Ở ví dụ trong hình bên phải số lượng đường đi sẽ là $C_8^5 = C_8^3 = \frac{8\times7\times6}{1\times2\times3} = 56$.

Áp dụng đúng sơ đồ lý thuyết ta có nghiệm là tổng số đường đi qua mọi ô (kể cả ô cấm) trừ đi số đường đi qua từng ô cấm, cộng với số đường đi qua 2 ô cấm , trừ số đường đi qua 3 ô cấm , . . .

Số đường đi qua 3 ô cấm bằng 0

126 – 135 + 34 = 25

Giải thuật có độ phức tạp O($2^k \times k$).

Giải thuật IV

Giải thuật III có *độ phức tạp hàm mũ* vì vậy chỉ có hiệu quả khi **k** đủ nhỏ. Để giảm độ phức tạp của giải thuật cần kết hợp giữa nguyên lý bù trừ và quy hoạch động. Việc kết hợp 2 phương pháp nói trên sẽ tạo ra *giải thuật độ phức tạp đa thức đối với **k*** và vì vậy có thể áp dụng một cách có hiệu quả với **k** bậc 10^2, không phụ thuộc vào **n** và **m**.

Để tiện xử lý, các dòng và cột được đánh số bắt đầu từ 0.

Xét $\mathcal{K}$ – tập các ô có vật cản, ô xuất phát (**0**, **0**) và ô đích (**n-1**, **m-1**). Tập sẽ có **k+2** phần tử, đánh số từ **0** đến **k+1**.

Sắp xếp các phần tử của $\mathcal{K}$ theo thứ tự tăng dần.

Gọi $\mathbf{dp}_{i,j}$ – số lượng đường đi từ ô **i** $\in \mathcal{K}$ tới ô **j** $\in \mathcal{K}$, không chứa ô nào khác thuộc $\mathcal{K}$.

Dựa vào tọa độ của các điểm **i** và **j**, như đã nêu ở trên, ta có thể dễ dàng tính được $\mathbf{rt}_{i,j}$ – tất cả các đường đi từ **i** tới **j**, kể các đường đi qua ô cấm.

$$\mathbf{rt}_{i,j} = \begin{cases} 0 \text{ nếu không có cách đi,} \\ C_{x+y}^{x} \text{ trong trường hợp ngược lại.} \end{cases}$$

Gọi **t** – ô cấm nằm giữa **i** và **j** trong $\mathcal{K}$ đã sắp xếp, **i** < **t** < **j**, có

$$\mathbf{dp}_{i,j} = \mathbf{rt}_{i,j} - \sum_{\forall t} dp_{i,t} \times rt_{t,j}$$

Nghiệm của bài toán là $\mathbf{dp}_{0,k+1}$.

Độ phức tạp của giải thuật: O(k^3).

Để giảm thời gian thực hiện cần tính sẵn bảng giá trị **i!** với **i** = 0, 1, 2, . . .

Xét *giải thuật IV_a* tính trực tiếp giá trị thực của $\mathbf{dp}_{i,j}$ với **n**, **m** đủ nhỏ, trên cơ sở đó – cải tiến để nhận được lời giải bài toán đã nêu.

Giải thuật IV_a

Tổ chức dữ liệu

- Mảng `int64_t ft[20]` – lưu các giai thừa, $\mathbf{ft}_i$ = **i!**,
- Mảng `int64_t dp[100][100]` – phục vụ sơ đồ quy hoạch động,
- Mảng `vector<pii> ob` – lưu các phần tử thuộc tập $\mathcal{K}$.

Xử lý

Tính hệ số nhị thức:

$C_{x+y}^{x} = C_{x+y}^{y}$

```
int64_t comb(int u, int v)
{
    int q=u+v;
    u=min(u,v);
    return ft[q]/ft[u]/ft[q-u];
}
```

$C_q^u = \dfrac{q!}{u! \times (q-u)!}$

Tính $\mathbf{rt}_{\mathbf{i},\mathbf{j}}$ – tất cả các đường đi từ **i** tới **j**, kể các đường đi qua ô cấm:

```
int all_r(int i, int j)
{
    x=ob[j].ff-ob[i].ff;
    y=ob[j].ss-ob[i].ss;

    if(y<0) return 0;

    if(x==0 || y==0) if(i+1==j) return 1;

    return comb(x,y);
}
```

Số bước sang phải và lên trên

Không có đường đi

Cùng hàng/cột và kề nhau

Tính $\mathbf{rt}_{\mathbf{t},\mathbf{j}}$ – tất cả các đường đi *từ* ***t*** *tới* ***j*** (kể các đường đi qua ô cấm):

```
for(int t=1;t<=k;++t)
    for(int i=0; i<=k-t; ++i)
    {
        int j=i+t;

        r=0;
        for(int it=i+1; it<j; ++it)
            r+=dp[i][it]*get_cb(i,j,it);

        dp[i][j]=all_r(i,j)-r;
    }
```

Các đường đi bị loại

Tính $\mathbf{dp}_{\mathbf{i},\mathbf{j}}$:

Trình tự tính:

$dp_{0,1}$, $dp_{1,2}$, $dp_{2,3}$ . . .

$dp_{0,2}$, $dp_{1,3}$, $dp_{2,4}$ . . .

$dp_{0,3}$, $dp_{1,4}$, $dp_{2,5}$ . . .

. . . . . .

Tính số đường đi cần loại bỏ

```
r=0;
for(int it=1; it<=k; ++it)
            r+=dp[0][it]*get_cb(0,k+1,it);

ans = comb(n-1,m-1)-r;
```

Tổng hợp kết quả:

```
int get_cb(int i, int j, int u)
{
    if((ob[u].ss>ob[j].ss)||(ob[u].ss<ob[i].ss))
                                    return 0;

    x=ob[j].ff-ob[u].ff;
    y=ob[j].ss-ob[u].ss;

    if(x==0 || y==0) return 1;

    return comb(x,y);
}
```

t nằm ngoài hình chữ nhật

Chương trình IV_a:

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
ifstream fi ("route.inp");
ofstream fo ("route.ou2");
typedef pair<int,int> pii;
int n,m,k,p,x,y,u=1,v=0,ib=0;
int64_t ft[20],dp[100][100];
vector<pii>ob;
int64_t ans,r;

int64_t comb(int u, int v)
{
    int q=u+v;
    u=min(u,v);
    return ft[q]/ft[u]/ft[q-u];
}

int get_cb(int i, int j, int u)
{
    if((ob[u].ss > ob[j].ss)||(ob[u].ss < ob[i].ss)) return 0;
    x=ob[j].ff-ob[u].ff;
    y=ob[j].ss-ob[u].ss;
    if(x==0 || y==0) return 1;
    return comb(x,y);
}

int all_r(int i, int j)
{
    x=ob[j].ff-ob[i].ff;
    y=ob[j].ss-ob[i].ss;
    if(y<0) return 0;
    if(x==0 || y==0) if(i+1==j) return 1;
    return comb(x,y);
}


int main()
{
    fi>>n>>m>>k>>p;
    ob.resize(k);
    ft[0]=1;
    for(int i=1; i<19; ++i) ft[i]=ft[i-1]*i;
    for(int i=0; i<k; ++i)
    {
        fi>>x>>y; --x; --y;
```

```
        ob[i]={x,y};
    }
    ob.push_back({0,0}); ob.push_back({n-1,m-1});
    sort(ob.begin(),ob.end());

    for(int t=1;t<=k;++t)
        for(int i=0; i<=k-t; ++i)
        {
            int j=i+t;
            r=0;
            for(int it=i+1; it<j; ++it)
                    r+=dp[i][it]*get_cb(i,j,it);
            dp[i][j]=all_r(i,j)-r;
        }
    r=0;
    for(int it=1; it<=k; ++it) r+=dp[0][it]*get_cb(0,k+1,it);
    ans = comb(n-1,m-1)-r;
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec ";
}
```

Giải thuật IV_b – Lời giải bài toán

Hệ số C_{x+y}^{x} của nhị thức Newton tăng rất nhanh: $C_{2n}^{n} \approx \frac{2^{2n}}{\sqrt{\pi n}}$

Giải thuật đòi hỏi phải tính nhiều C_{x+y}^{x} với các **x**, **y** khác nhau,

Để giảm độ phức tạp của giải thuật:

- Lưu trữ bảng giá trị **q!** *mod* **p** với **q** = 0, 1, 2, . . ., 2×10^5,
- Tính C_{x+y}^{x} theo công thức $C_{x+y}^{x} = \frac{(x+y)!}{x!\times y!}$ và thay việc thực hiện phép chia bằng cách tính nghịch đảo theo mô đun.

Dựa vào bảng giá trị giai thừa đã lưu, có

(x+y)! = **a** (*mod* **p**),

x! = **b** (*mod* **p**),

y! = **c** (*mod* **p**).

$\frac{(x+y)!}{x!\times y!}$ = **z** (*mod* **p**) → **a** = **z×b×c** → **a=z×r**, trong đó **r** = **b×c** .

Lưu ý, ở đây *tử số chia hết cho mẫu số*.

$a=z\times r \rightarrow z = a\times r^{p-2}$

Bằng sơ đồ *tính nhanh lũy thừa* ta dễ dàng tìm được **z**.

```
int64_t pw(int64_t u )
{
    int64_t res=1,tm=u, tp=p-2;
    while(tp)
    {
        if(tp&1)res=res*tm%p;
        tm=tm*tm%p;
        tp>>=1;
    }
    return res;
}

int64_t comb(int u, int v)
{
    int64_t a,b,c;
    int q=u+v;
    u=min(u,v);
    a=ft[q]; b= ft[u]*ft[q-u]%p;
    b=pw(b);
    return a*b%p;
}
```

Trong phần tính $dp_{i,j}$ và dẫn xuất kết quả: lấy mô đun theo **p** ở các đại lượng tính được.

Chương trình IV_b:

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
ifstream fi ("route.inp");
ofstream fo ("route.ou3");
typedef pair<int,int> pii;
int n,m,k,p,x,y,u=1,v=0,ib=0;
int64_t ft[2000001],dp[102][102];
vector<pii>ob;
int64_t ans,r;

int64_t pw(int64_t u )
{
    int64_t res=1,tm=u, tp=p-2;
    while(tp)
    {
        if(tp&1)res=res*tm%p;
        tm=tm*tm%p;
        tp>>=1;
    }
    return res;
}

int64_t comb(int u, int v)
{
    int64_t a,b,c;
    int q=u+v;
    u=min(u,v);
    a=ft[q]; b= ft[u]*ft[q-u]%p;
    b=pw(b);
    return a*b%p;
}

int get_cb(int i, int j, int u)
{
    if((ob[u].ss > ob[j].ss)||(ob[u].ss < ob[i].ss)) return 0;
    x=ob[j].ff-ob[u].ff;
    y=ob[j].ss-ob[u].ss;
    if(x==0 || y==0) return 1;
    return comb(x,y);
}

int all_r(int i, int j)
{
    x=ob[j].ff-ob[i].ff;
```

```
    y=ob[j].ss-ob[i].ss;
    if(y<0) return 0;
    if(x==0 || y==0) if(i+1==j) return 1;
    return comb(x,y);
}

int main()
{
    fi>>n>>m>>k>>p;
    ob.resize(k);
    ft[0]=1;
    for(int i=1; i<=2000000; ++i) ft[i]=ft[i-1]*i%p;
    for(int i=0; i<k; ++i)
    {
        fi>>x>>y; --x; --y;
        ob[i]={x,y};
    }
    ob.push_back({0,0}); ob.push_back({n-1,m-1});
    sort(ob.begin(),ob.end());

    for(int t=1;t<=k;++t)
        for(int i=0; i<=k-t; ++i)
        {
            int j=i+t;
            r=0;
            for(int it=i+1; it<j; ++it)
                r=(r+dp[i][it]*get_cb(i,j,it))%p;
            dp[i][j]=(all_r(i,j)-r)%p;
        }
    r=0;
    for(int it=1; it<=k; ++it)
              r=(r+dp[0][it]*get_cb(0,k+1,it))%p;
    ans = (comb(n-1,m-1)-r+p)%p;
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec ";
}
```

Để theo dõi tình hình thời tiết ngoài khơi người thả ***n*** phao nổi chứa các thiết bị đo đạc. Ban đầu các phao này được thả và neo dọc theo một đường thẳng.

Sau một thời gian, dưới tác động của gió và sóng, một số neo có thể bị chuyển dịch làm cho các phao có khả năng không còn nằm trên một đường thẳng nữa.

Số liệu tự động gửi về cho thấy phao thứ ***i*** đang ở tọa độ (x_i, y_i), ***i*** = 1 ÷ ***n***.

Hãy xác định có hay không việc chuyển dịch làm các phao không còn nằm trên một đường thẳng nữa và nếu có – chỉ ra 3 phao không ở trên cùng một đường thẳng.

Dữ liệu: Vào từ file SHIFT.INP:

- Dòng đầu tiên chứa số nguyên ***n*** ($3 \le n \le 10^5$),
- Dòng thứ ***i*** trong ***n*** dòng sau chứa 2 số nguyên x_i và y_i xác định tọa độ của phao ***i*** ($|x_i|$, $|y_i| \le 10^9$).

Không có 2 phao nào ở cùng một điểm trên mặt phẳng.

Kết quả: Đưa ra file văn bản SHIFT.OUT đưa ra thông báo ***Yes*** hoặc ***No***. Nếu có sự chuyển dịch làm các phao không còn thẳng hàng thì ở dòng tiếp theo đưa ra 3 số nguyên xác định theo tùy chọn 3 phao không thẳng hàng.

Ví dụ:

SHIFT.INP	SHIFT.OUT
5 1 2 0 0 3 6 4 8 4 4	Yes 3 2 5

Giải thuật: Cơ sở lập trình .

Cách I:

Gọi P_i là điểm tương ứng với vị trí của phao thứ i.

Xét các tam giác $P_1P_2P_i$, $i = 3 \div n$. Nếu các điểm nằm thẳng hàng thì các tam giác này đều có diện tích bằng 0 với mọi i.

Nếu tồn tại i để tam giác $P_1P_2P_i$ có diện tích khác 0 thì 3 điểm P_1, P_2 và P_i không thẳng hàng.

Công thức tính 2 lần diện tích (có dấu) của tam giác $P_1P_2P_i$:

$$x_1y_2-y_1x_2+x_2y_i-y_2x_i+x_iy_1-y_ix_1$$

Cách II:

Xác định phương trình đường thẳng $f(x,y) = ax+by+c$ đi qua 2 điểm P_1 và P_2, trong đó:

- $a = y_2-y_1$
- $b = -(x_2-x_1)$
- $c = y_1(x_2-x_1)-x_1(y_2-y_1)$

Tính $f(x_i,y_i)$, $i = 3 \div n$, nếu $f(x_i,y_i) \neq 0 \rightarrow$ 3 điểm P_1, P_2 và P_i không thẳng hàng.

Độ phức tạp của giải thuật: O(*n*).

Chương trình I

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("shift.inp");
ofstream fo ("shift.out");
int n, ir=0;
int64_t s,xa,ya,xb,yb,xc,yc;

int main()
{
    fi>>n;
    fi>>xa>>ya>>xb>>yb;
    s=xa*yb-xb*ya;
    for(int i=3; i<=n; ++i)
    {
        fi>>xc>>yc;
        if(s+xb*yc-xc*yb+xc*ya-xa*yc != 0)
        {
           ir=i; break;
        }
    }
    if(ir) fo<<"Yes\n1 2 "<<ir; else fo<<"No";
    fo<<"\nTime: "<<clock()/(double)1000<<" sec ";
}
```

Chương trình II

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("shift.inp");
ofstream fo ("shift.out");
int n, ir=0;
int64_t xa,ya,xb,yb,xc,yc,a,b,c;

int main()
{
    fi>>n;
    fi>>xa>>ya>>xb>>yb;
    a=yb-ya;
    b=xa-xb;
    c=ya*(xb-xa)-xa*(yb-ya);

    for(int i=3; i<=n; ++i)
    {
        fi>>xc>>yc;
        if(a*xc+b*yc+c != 0)
        {
           ir=i; break;
        }
    }
    if(ir) fo<<"Yes\n1 2 "<<ir; else fo<<"No";
    fo<<"\nTime: "<<clock()/(double)1000<<" sec ";
}
```

Các ngôn ngữ lập trình như Java, Python cho phép thực hiện các phép tính số học với số lượng các chữ số của mỗi toán hạng hay kết quả là không hạn chế. Nhưng cả khi đó việc xử lý một cách có hiệu quả các số rất lớn cũng không phải là một vấn đề đơn giản.

Để chứng minh cho điều này học sinh nhận được bài tập về nhà như sau: Cho 2 số nguyên dương ***a*** và ***b*** ($1 \le a \le b \le 10^{100\,000}$). Yêu cầu thực hiện các phép biến đổi:

1. Tính tích các số nguyên từ ***a*** đến ***b***, kể cả ***a*** và ***b***,
2. Tính tổng các chữ số trong kết quả nhận được,
3. Nếu tổng không nhỏ hơn 10 thì thực hiện lại bước 2, trong trường hợp ngược lại – đưa ra chữ số nhận được.

Hãy xác định chữ số cần đưa ra.

Dữ liệu: Vào từ file COMPRESS.INP:

- Dòng đầu tiên chứa số nguyên ***a***,
- Dòng thứ 2 chứa số nguyên ***b***.

Kết quả: Đưa ra file văn bản COMPRESS.OUT chữ số nhận được.

Ví dụ:

COMPRESS.INP
6 8

COMPRESS.OUT
3

WA39 Io20191109 D A XVII

Giải thuật: Nhận dạng tình huống, cộng số lớn.

Dễ dàng chứng minh với số nguyên **x** cho trước, quá trình lặp lại nhiều lần việc tính tổng các chữ số cho đến khi nhận được số có một chữ số (*sơ đồ nén*) tương đương với việc lấy phần dư của phép chia **x** cho 9. Nếu **x** *chia hết* cho 9 thì *kết quả cần tìm sẽ là 9*.

Ví dụ:

x = 446 → 4+4+6 = 14 → 1+4 = 5 **446%9 = 5**

x = 864 → 8+6+4 = 18 → 1+8 = 9 **864%9 = 0 → 9**

Phép **%** có tính chất phân phối nên chỉ cần một thừa số có x**%9 = 0**, tích sẽ bằng **0**.

Lưu trữ **a** và **b** dưới dạng *xâu*.

Gọi **na** và **nb** là độ dài các xâu **a** và **b**.

Nếu **na** < **nb**, kết quả lấy mô đun cho 9 sẽ bằng 0 (trong tích chứa thừa số dạng **99…9**).

Tính **ta** = **a%9** (theo sơ đồ nén), **tb** = **b%9**.

Nếu **ta** = 0 hoặc **tb** = 0 hoặc **tb** < **ta** sơ đồ nén tích các số sẽ cho kết quả 0.

Trường hợp **ta** ≤ **tb**:

m = 9 – **ta**,

Thực hiện cộng số lớn , tính **a = a+m**,

Nếu **a** < **b** : sơ đồ nén tích các số sẽ cho kết quả 0.

Trường hợp còn lại:

Tính **tg** =($\prod_{ta}^{tb} i$) % 9 .

Từ kết quả nhận được theo sơ đồ nén suy ra giá trị cần tìm.

Độ phức tạp của giải thuật: ≈O(1).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("compress.inp");
ofstream fo ("compress.out");
int na, nb, ta, tb, tg, ans, k, m;
string a,b;

int main()
{
    fi>>a>>b;
    na=a.size(); nb=b.size();
    ta=0; tb=0;
    if(na<nb) {fo<<'9'; return 0;}

    for(int i=0; i<na; ++i) ta+=a[i]-48, tb+=b[i]-48;
    ta%=9; tb%=9;

    if(ta==0 || tb==0 || tb<ta) {fo<<'9'; return 0;}

    m=9-ta;
    tg=m;
    for(int i=na-1; i>=0; --i)
    {
        a[i]+=tg;
        if(a[i]>57) a[i]-=10, tg=1; else {tg=0; break;}
    }
    if(tg==1 || a<b) {fo<<'9'; return 0;}

    tg=1;
    for(int i=ta; i<=tb; ++i) tg*=i;
    ans=0;
    //while(tg>0){ans+=tg%10; tg/=10;}
    ans= tg%9; if(ans==0) ans=9;
    fo<<ans;

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

Alice và Bob sống trong khu phố có các nhà từ đầu ngả tư này đến đầu ngã tư kia được đánh số liên tiếp từ ***a*** đến ***b***, bên phải là các nhà có số chẵn, bên trái – có số lẻ. Alice sống bên số nhà lẻ còn Bob – bên chẵn.

Để rèn luyện trí nhớ và kỹ năng tính nhẩm khi đi bộ trên phố Alice nhìn sang bên kia đường và tính tổng các số ghi biển số nhà bên đó.

Hôm nay, lúc đứng chờ xe buýt về nhà sau giờ học Alice nói cho Bob về môn thể thao trí tuệ của mình. Bob rất ngạc nhiên: "Mình cũng vậy và cứ tưởng chỉ có mình làm chuyện đó! Nhưng khác với bạn, mình nhìn và đếm tổng các số nhà bên lẻ". Hai bạn quyết định thi xem ai tính nhanh hơn bằng cách cho biết hiệu của tổng Alice tính được với tổng của Bob. Cả hai bạn gần như đồng thời nói được ngay kết quả.

Hãy xác định kết quả các bạn đã nói.

Dữ liệu: Vào từ file ODD_EVEN.INP, dòng thứ nhất chứa số nguyên ***a***, dòng thứ 2 chứa số nguyên ***b*** ($1 \le a < b \le 2\times10^9$).

Kết quả: Đưa ra file văn bản ODD_EVEN.OUT một số nguyên – hiệu tính được.

Ví dụ:

ODD_EVEN.INP
3 7

ODD_EVEN.OUT
-5

WA40 MosSch1120191119 B A XVII

Giải thuật: Cấp số cộng .

Các số nguyên chẵn và các số nguyên lẻ trong đoạn [**a**, **b**] tạo thành 2 cấp số cộng với công bội là 2.

Để tính tổng mỗi cấp số cộng trong trường hợp cụ thể này cần biết:

- Số hạng đầu tiên và số hạng cuối cùng của mỗi cấp số cộng,
- Số lượng phần tử trong cấp số.

Gọi **odd1** và **odd2** – Số lẻ đầu tiên và số lẻ cuối cùng trong [**a**, **b**],

Gọi **even1** và **even2** – Số chẵn đầu tiên và số chẵn cuối cùng trong [**a**, **b**].

Kết quả cần tìm sẽ là:

$$\mathbf{ans} = \frac{even2+even1}{2} \times \frac{even2-even1+2}{2} - \frac{odd2+odd1}{2} \times \frac{odd2-odd1+2}{2}$$

Độ phức tạp của giải thuật: O(1).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("Odd_Even.inp");
ofstream fo ("Odd_Even.out");
int64_t a,b,odd1,odd2,even1,even2,ans;

int main()
{
    fi>>a>>b;
    if(a&1)odd1=a, even1=(a+1);
        else odd1=a+1, even1=a;
    if(b&1)odd2=b, even2=(b-1);
        else odd2=b-1, even2=b;

    ans=((even2+even1)/2*((even2-even1)/2+1));
    ans-=((odd2+odd1)/2*((odd2-odd1)/2+1));
    fo<<ans;
}
```

Các số nguyên không âm theo cơ số B (B >1) tạo thành một nhóm có các tính chất tương tự nhau không phụ thuộc vào cơ số B. Vì vậy, nếu một vấn đề đã giải quyết được với cơ số B1 thì cũng có thể dễ dàng giả quyết ở cơ số B2. Việc bảo toàn các tính chất đó được gọi là bất biến đại số.

Trong Tin học tính bất biến hỗ trợ rất nhiều trong việc xác định giải thuật. Một bài toán đã giải quyết được ở cơ số 3 thì có thể dễ dàng triển khai để giải với cơ số B > 1 bất kỳ.

Để chứng minh cho điều đó bài tập về nhà cho cả lớp là cho 2 số nguyên dương ***x*** và ***y*** ở cơ số ***B***, ***x*** ≤ ***y*** và mỗi số có không quá 5×10^5 chữ số. Yêu cầu thực hiện các phép biến đổi:

1. Tính tích các số nguyên từ ***a*** đến ***b***, kể cả ***a*** và ***b***,
2. Tính tổng các chữ số trong kết quả nhận được,
3. Nếu tổng không nhỏ hơn ***B*** thì thực hiện lại bước 2, trong trường hợp ngược lại – đưa ra chữ số nhận được.

Hãy xác định chữ số cần đưa ra.

Mỗi người trong lớp nhận được một cơ số ***B*** khác nhau và 2 số ***x***, ***y*** ở cơ số tương ứng.

Alice nhận được cơ số ***B*** là 16 – cơ số Hexa. Các chữ số của cơ số Hexa được nêu ở bảng bên.

Dec	Hexa
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	a *hoặc* A
11	b *hoặc* B
12	c *hoặc* C
13	d *hoặc* D
14	e *hoặc* E
15	f *hoặc* F

Các chữ số Hexa

Hãy xác định chữ số Alice cần đưa ra. Nếu chữ số nhận được lớn hơn 9 thì đưa ra dưới dạng chữ cái hoa.

Dữ liệu: Vào từ file input chuẩn của hệ thống, dòng thứ nhất chứa số nguyên ***x***, dòng thứ 2 chứa số nguyên ***y*** (***x*** ≤ ***y***), mỗi số có không quá 5×10^5 chữ số, trong một số các chữ số lớn hơn 9 có thể ghi dưới dạng cả ký tự hoa lẫn ký tự hoa thường, ví dụ ***1bA*** hoặc ***1BA*** hay ***1Ba***.

Kết quả: Đưa ra file output chuẩn của hệ thống chữ số nhận được. Nếu chữ số nhận được lớn hơn 9 – đưa ra dưới dạng ký tự hoa.

Ví dụ:

INPUT
1bA 1Bd

OUTPUT
F

Giải thuật: Nhận dạng tình huống lô gic .

Xét số **x** chỉ có *một chữ số khác 0* trong cơ số **B**: $x = d \times B^k$, $k \geq 0$, $1 \leq d < B$.

Dễ dàng thấy rằng:

$$d = \begin{cases} X \bmod (B-1) \text{ nếu } x \bmod (B-1) > 0, \\ B-1 \text{ nếu } x \bmod (B-1) = 0, \end{cases}$$

Từ đây suy ra việc lấy tổng các chữ số của số **x** bất kỳ và lập lại cho đến khi còn một chữ số sẽ cho kết quả bằng tổng các chữ số của **x** theo mô đun **B-1** và thay nó bằng **B-1** nếu kết quả tính mô đun bằng 0.

Gọi $f_B(x)$ – hàm tính tổng các chữ số của **x** (theo cơ số **B**) theo quy tắc nêu trên

Không phụ thuộc vào cơ số, ta có:

$$f_B(x \times y) = f_B(f_B(x) \times f_B(y))$$

Ví dụ, với **x** = 11, **y** = 11

	$x \times y$	$f_B(x)$	$f_B(y)$	$f_B(f_B(x) \times f_B(y))$
Cơ số 2 :	1001	1	1	1
Cơ số 10:	121	2	2	4
Cơ số 16:	121	2	2	4

Với 2 số **x** và **y** ($x \leq y$) với $y = x+k$ ($k \geq 0$), gọi **ans** là kết quả cần tìm.

Ta có:

$$ans = f_B(x \times (x+1) \times (x+2) \times \ . \ . \ . \ \times (x+k))$$

$$= f_B(f_B(x) \times (f_B(x)+1) \times (f_B(x)+2) \times \ . \ . \ . \ \times (f_B(x)+k))$$

Nếu $k \geq B-1$ hoặc $f_B(x) > f_B(f_B(x)+k)$ thì một trong số các $f_B(f_B(x)+i)$ sẽ bằng **B-1**và **ans** sẽ bằng **B-1**.

Như vậy trong trường hợp xấu nhất, để dẫn xuất kết quả cần tính

$$f_B(f_B(x) \times (f_B(x)+1) \times (f_B(x)+2) \times \ . \ . \ . \ \times (f_B(x)+m)),$$

trong đó $0 \leq m < B-1$.

Tổ chức dữ liệu:

- Hai xâu **string a,b** – lưu trữ dữ liệu vào,
- Xâu **string pat="0123456789ABCDEF"** – Phục vụ chuẩn hóa dữ liệu và xác định giá trị của các chữ số Hexa.

Xử lý:

Chuẩn hóa dữ liệu vào:

```
fi>>a>>b;
na=a.size(); nb=b.size();
ta=0; tb=0;
for(auto &c:a)c=toupper(c);
for(auto &c:b)c=toupper(c);
```

Tính $f_{16}(a)$ và $f_{16}(b)$:

```
for(char c:a)
{
    k=pat.find(c);
    ta=ta+k;
}
ta%=15;
for(char c:b)
{
    k=pat.find(c);
    tb=tb+k;
}
tb%=15;
```

Nhận dạng các trường hợp có $f_B(f_B(x)+i)$ bằng 15:

```
if(na<nb || tb<ta || ta==0 || tb==0)
                    {fo<<'F'; return 0;}
  tg=0; tg=15-ta;
  for(int i=na-1; i>=0; --i)
  {
      k=pat.find(a[i]);
      t=k+tg; tg=t/16;
      a[i]=pat[t%16];
  }
  if(tg || a<=b) {fo<<'F'; return 0;}
```

Tính **a+=tg**

$\exists f_B(f_B(x)+i)=15$

Trường hợp chưa biết chắc chắn có **$f_B(f_B(x)+i)=15$** :

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

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("topo.inp");
ofstream fo ("topo.out");
int na, nb, t, ta, tb, tg, ans, k, m;
string pat="0123456789ABCDEF";
string a,b;

int main()
{
    fi>>a>>b;
    na=a.size(); nb=b.size(); ta=0; tb=0;
    for(auto &c:a)c=toupper(c);
    for(auto &c:b)c=toupper(c);

    for(char c:a)
    {
        k=pat.find(c);
        ta=ta+k;
    }
    ta%=15;
    for(char c:b)
    {
        k=pat.find(c);
        tb=tb+k;
    }
    tb%=15;

    if(na<nb || tb<ta ||ta==0 || tb==0) {fo<<'F'; return 0;}
    tg=0; tg=15-ta;
    for(int i=na-1; i>=0; --i)
    {
        k=pat.find(a[i]);
        t=k+tg; tg=t/16;
        a[i]=pat[t%16];
    }
    if(tg || a<=b) {fo<<'F'; return 0;}

    tg=1;
    for(int i=ta; i<=tb; ++i) tg*=i;
    ans=0;
    while(tg>0){ans+=tg&15; tg>>=4;}
    ans%=15; if(ans==0) ans=15;
    fo<<pat[ans];

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

Bob làm nhiệm vụ lên kế hoạch giao hàng ở một công ty bán hàng qua mạng. Việc lên kế hoạch sẽ dễ dàng hơn nhiều nếu tìm thấy một kế hoạch như thế trước đó.

Hôm nay cần chuyển ***n*** gói hàng cho khách. Bob xếp các gói hàng thành một dãy dài và ghi trên nhãn mỗi gói một chữ cái từ ***a*** đến ***z*** tùy thuộc vào tính chất của hàng, từ đó nhận được xâu đặc trưng ***s*** độ dài ***n***. Với trí nhớ tốt, Bob có cảm giác đã có một kế hoạch tương tự như vậy với xâu đặc trưng ***t*** cũng có độ dài ***n***. Nếu quả thật như thế thì từ ***s*** có thể nhận được ***t*** bằng cách thực hiện một lần hai phép biến đổi sau:

- Mang ***k*** gói hàng ở đầu dãy đặt xuống cuối dãy ($0 \le k < n$), giữ nguyên trình tự trong các gói hàng mang xuống cuối,
- Áp dụng cách mã hóa Ceasar với các ký tự ghi trên nhãn hàng, tức là thay vòng tròn mỗi ký tự bằng ký tự đứng trước nó ***d*** vị trí ($0 \le d < 26$), ví dụ, với ***d*** = 4 thì ***e*** chuyển thành ***a***, ***c*** chuyển thành ***y***.

Thà mất công kiểm tra một chút còn hơn đi xây dựng kế hoạch mới, Bob quyết định thực hiện biến đổi từ ***s*** sang ***t***.

Hãy xác định từ ***s*** Bob có nhận được xâu ***t*** hay không và nếu được thì cần thực hiện biến đổi với ***k*** nhỏ nhất và ***d*** là bao nhiêu.

Dữ liệu: Vào từ file CAESAR.INP:

- Dòng đầu tiên chứa số nguyên ***n*** ($1 \le n \le 2\times10^5$),
- Dòng thứ 2 chứa xâu ***s*** từ các ký tự la tinh thường độ dài ***n***,
- Dòng thứ 2 chứa xâu ***t*** từ các ký tự la tinh thường độ dài ***n***.

Kết quả: Đưa ra file văn bản CAESAR.OUT thông báo "***Impossible***" nếu không thể đưa ***s*** về ***t*** theo cách đã nêu và thông báo "***Succes***" nếu biến đổi được. Trong trường hợp biến đổi được – dòng thứ 2 đưa ra các giá trị ***k*** và ***d***.

Ví dụ:

CAESAR. INPUT	CAESAR. OUTPUT
5 abcde cdeab	Succes 3 0

WA42 Io20191123 C A XVII

Giải thuật: Xử lý dữ liệu vòng tròn và nhận dạng .

Để dễ dàng thực hiện việc đẩy vòng các ký tự sang trái cần tạo xâu mẫu **p**:

Ký tự la tinh thường **s[i]** khi đẩy vòng sang trái một vị trí sẽ có giá trị mới là

p[s[i]-97+25] .

Tạo xâu **s2** bằng cách gấp đôi xâu **s**: **s2 = s + s**.

Kiểm tra 26 lần sự tồn tại của **t** trong **s2** với các ký tự trong **s2** lần lượt được đẩy vòng sang trái một vị trí.

Nếu **t** không xuất hiện lần nào trong số các **s2** nhận được thì đưa ra thông báo "**Impossible**".

Nếu tồn tại cách đẩy vòng: Độ dài đoạn cần mang lên đầu là khoảng cách từ đầu **s2** tới **t**.

Độ phức tạp của giải thuật: O(*n*).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("42");
ofstream fo ("ceasar.out");
int n,d,k=-1;
string
s,s2,t,tm,p="abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz";

int main()
{
    fi>>n>>s>>t;
    s2=s+s; d=-1;
    for(int i=0; i<26; ++i)
    {
        k=s2.find(t);
        if(k!=string::npos) {d=i; break;}
        for(int j=0; j<2*n; ++j)s2[j]=p[s2[j]-72];    //97-25
    }

    if(d<0)
    {
        k=s2.find(t);
        if(k!=string::npos) d=25;
    }
    if(k)k=n-k;
    if(d>=0) fo<<"Succes\n"<<k<<' '<<d; else fo<<"Impossible";

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

Bước đầu hoạt động Khu công nghiệp có hình vuông diện tích 1. Sau một thời gian hoạt động có hiệu quả Khu công nghiệp mở rộng thêm một diện mới có hình vuông diện tích 1 kề cạnh với diện tích cũ, tạo thành hình chữ nhật kích thước 1×2. Lần mở rộng thứ ***i*** – bổ sung thêm một diện tích hình vuông cạnh bằng cạnh lớn của hình chữ nhật ở bước trước và toàn bộ diện tích luôn là một hình chữ nhật.

Hãy xác định diện tích của Khu công nghiệp sau lần thứ mở rộng thứ ***n*** và đưa ra theo mô dun 998244353.

Dữ liệu: Vào từ file input chuẩn của hệ thống, gồm một dòng chứa số nguyên ***n*** ($1 \le n \le 10^{18}$).

Kết quả: Đưa ra file output chuẩn của hệ thống diện tích tìm được theo mô đun 998244353.

Ví dụ:

INPUT	OUTPUT
4	40

WA43 A XVII

Giải thuật: Tính nhanh lũy thừa ma trận.

Hình khu công nghiệp ở các bước phát triển đầu tiên:

Dễ dàng thấy rằng sau bước mở rộng thứ **i**, Khu công nghiệp có hình chữ nhật với hai cạnh là các số Fibonacci F_i và F_{i+1}.

Như vậy, sau bước mở rộng thứ **n** diện tích **s** của khu công nghiệp sẽ là $F_n \times F_{n+1}$.

Với mọi **n** ta có

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

Việc tính F_n và F_{n+1} có thể dễ dàng thực hiện theo sơ đồ tính nhanh lũy thừa.

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "indzone."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int64_t p = 998244353;
vector<int64_t> x(4),y(4),z(4);
int64_t n,ans;

vector<int64_t> mult(vector<int64_t> a,vector<int64_t> b)
{
    vector<int64_t> t(4);
    t[0]=(a[0]*b[0]%p+a[1]*b[2]%p)%p;
    t[1]=(a[0]*b[1]%p+a[1]*b[3]%p)%p;
    t[2]=(a[2]*b[0]%p+a[3]*b[2]%p)%p;
    t[3]=(a[2]*b[1]%p+a[3]*b[3]%p)%p;
    return t;
}
int main()
{
    fi>>n;
    z[0]=z[3]=1; z[1]=z[2]=0;
    x[0]=x[1]=x[2]=1; x[3]=0;
    while(n>0)
    {
        if(n&1) z=mult(x,z);
        x=mult(x,x);
        n>>=1;
    }
    ans=z[0]*z[2]%p;

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

Xét cây **n** đỉnh, các đỉnh đánh số từ 1 đến **n**.

Mã Prüfer là ánh xạ đơn trị hai chiều một cây sang *dãy n-2 số nguyên*, mỗi số trong đoạn [1, **n**].

Việc xác định mã Prufer bao gồm **n-2** bước, ở mỗi bước: chọn lá có số nhỏ nhất, bổ sung nút cha của lá đó vào mã cần tìm và xóa nút lá đã chọn khỏi cây.

Khi cây chỉ còn 2 nút – giải thuật kết thúc. Bản thân 2 nút này không được ghi nhận (một cách tường minh) vào mã cần tìm.

Ví dụ, cây ở hình bên có mã Prufer là 5, **1**, **1**, **5**).

Đường kính của cây là đường đi dài nhất (tính theo số cạnh) nối 2 nút bất kỳ của cây.

Cho **k** và các số nguyên **p1**, **p2**, . . ., **pk** xác định mã Prufer của một cây.

Hãy xác định đường kính của cây.

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

- Dòng đầu tiên chứa số nguyên **k** ($1 \le k \le 10^6$),
- Dòng thứ 2 chứa **k** số nguyên $p_1, p_2, \ldots, p_k$ ($1 \le p_i \le k+1$, $i = 1 \div k$).

Kết quả: Đưa ra file output chuẩn của hệ thống một số nguyên – đường kính tìm được.

Ví dụ:

INPUT
4 5 1 1 5

OUTPUT
3

WA44 A XVII

Giải thuật: Khôi phục cây theo mã Prufer .

Tồn tại các giải thuật khôi phục cây với độ phức tạp O(*n*log*n*) và O(*n*).

Với ràng buộc **n** ≤ 10^6 cần sử dụng giải thuật độ phức tạp O(*n*) để đảm bảo thời gian thực hiện chương trình.

Số đỉnh của cây sẽ là **n+2**.

Tính chất của mã Prufer: *Số lần mỗi đỉnh tham gia vào mã Prufer bằng số bậc của nó trừ* 1. Điều này không khó hiểu vì đỉnh sẽ bị loại bỏ khi có bậc bằng 1. Điều này cũng đúng với cả 2 đỉnh còn lại của cây!

Từ đây có thể xác định lá đầu tiên bị loại trong cây: đó là *lá có số nhỏ nhất* và *đỉnh cha* của nó là *số đầu tiên trong mã Prufer*.

Ghi nhận cạnh này và giảm bậc ở các đỉnh tương ứng,

Lặp lại các bước xử lý này cho đến khi còn 2 đỉnh bậc 1 không được ghi nhận trong mã đang xét, bổ sung cạnh nối 2 đỉnh này vào kết quả.

Để tìm được lá có số nhỏ nhất *không nhất thiết phải sử dụng cấu trúc dữ liệu hỗ trợ tìm min*.

Sử dụng con trỏ **ptr** chỉ tới nút cần xử lý.

Đồng thời với việc loại bỏ nút lá: cập nhật khoảng cách từ nút cha tới nút lá xa nhất đã xác định.

Khi còn lại 2 nút: đường kính của cây bằng *tổng khoảng cách đã xác định* của 2 nút đó tới nút lá xa nhất của mình và *cộng* 1.

Tổ chức dữ liệu:

- Mảng `vector<int> pruf_code(n)` – Lưu dữ liệu vào xác định mã Prufer,
- Mảng `vector<int> degree (n, 1)` – Lưu bậc các đỉnh của cây,
- Mảng `vector<int> dist` – Lưu khoảng cách một nút tới nút lá xa nhất,
- Con trỏ **leaf** chỉ tới nút lá tiếp theo cần xử lý (sau khi đã loại bỏ khỏi cây một số nút theo quá trình xử lý).

Lưu ý: Trong chương trình còn sử dụng mảng `vector<pair<int,int>> result` lưu cấu trúc cây, *phục vụ mục đích kiểm tra kép kết quả* cần tìm.

Xử lý:

Tính bậc các nút:

```
n +=2;
vector<int> degree (n, 1);
for (int i=0; i<n-2; ++i)
    ++degree[prufer_code[i]];
```

Tính chất mã Prufer

Tìm nút lá xuất phát:

```
int ptr = 0;
while (ptr < n && degree[ptr] != 1) ++ptr;
int leaf = ptr;
```

Nút lá có bậc bằng 1

Tìm đường kính:

```
for (int i=0; i<n-2; ++i)
{
    int v = prufer_code[i];
    //result.push_back (make_pair (leaf, v));
    dist[v]=max(dist[v],dist[leaf]+1);
    --degree[leaf];

    if (--degree[v] == 1 && v < ptr)
        leaf = v;
    else
    {
        ++ptr;
        while (ptr < n && degree[ptr] != 1) ++ptr;
        leaf = ptr;
    }
}
//for (int v=0; v<n-1; ++v) if (degree[v] == 1)
//result.push_back ({v, n-1});
res=dist[n-1]+dist[leaf]+1;
```

Cập nhật khoảng cách tới nút lá

Tìm nút lá tiếp theo

Xác định kết quả

Độ phức tạp của giải thuật: O(*n*).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("diameter.inp");
ofstream fo ("diameter.out");
int n, a, res;
vector<int> dist;
vector<pair<int,int>> prufer_decode_linear (const vector<int> & prufer_code)
{
    n +=2;
    vector<int> degree (n, 1);
    for (int i=0; i<n-2; ++i)
        ++degree[prufer_code[i]];
    int ptr = 0;
    while (ptr < n && degree[ptr] != 1) ++ptr;
    int leaf = ptr;
    vector < pair<int,int> > result;
    for (int i=0; i<n-2; ++i)
    {
        int v = prufer_code[i];
        result.push_back (make_pair (leaf, v));
        dist[v]=max(dist[v],dist[leaf]+1);
        --degree[leaf];
        if (--degree[v] == 1 && v < ptr)
            leaf = v;
        else
        {
            ++ptr;
            while (ptr < n && degree[ptr] != 1) ++ptr;
            leaf = ptr;
        }
    }
    for (int v=0; v<n-1; ++v)
        if (degree[v] == 1)
    result.push_back ({v, n-1});
    res=dist[n-1]+dist[leaf]+1;

    fo<<res;
    return result;
}
int main()
{
    fi>>n;
    vector<int> pruf_code(n);
    for(int i=0; i<n; ++i)
    {
        fi>>a;
        pruf_code[i]=--a;
    }
    vector<pair<int, int>> ans;
    dist.assign(n+2,0);
    ans=prufer_decode_linear(pruf_code);
}
```

WA45. KHỐI LẬP PHƯƠNG

Tên chương trình: CUBES.CPP

Procon là cuộc thi trang bị trí tuệ nhân tạo đơn giản cho robot. Với các nhiệm vụ đặt ra các đội phải lập trình, trang bị cho robot để nó có thể thực hiện các nhiệm vụ một cách độc lập, không có sự can thiệp tiếp theo của con người.

Một trong số các bài thi năm nay là lắp ráp số nguyên. Mỗi robot được giao ***n*** khối lập phương. Trên mỗi mặt của khối lập phương có in một chữ số trong phạm vi từ 0 đến 9. Như vậy mỗi khối lập phương chứa 6 chữ số. Các chữ số này có thể giống nhau.

Robot phải lấy các khối lập phương, xoay và đặt cạnh nhau sao cho mặt trên của chúng thể hiện một số nguyên không bắt đầu bằng chữ số 0. Các số cần tạo ra lần lượt là 1, 2, 3, . . . Mỗi khối lập phương có thể tham gia vào việc tạo các số khác nhau. Quá trình tạo số sẽ dừng lại khi gặp số nguyên đầu tiên không thể lắp ráp từ các khối đã cho.

Hãy xác định số nguyên làm dừng quá trình tạo số.

Dữ liệu: Vào từ file CUBES.INP:

- Dòng đầu tiên chứa số nguyên ***n*** ($1 \le n \le 10^5$),
- Dòng thứ ***i*** trong ***n*** dòng chứa xâu 6 ký tự số trong tập các chữ số hệ 10 xác định các chữ số ghi trên khối lập phương thứ ***i***.

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

Ví dụ:

CUBES.INP
3 123456 789012 345678

CUBES.OUT
90

WA45 VKOI 20191130 A XVII

Giải thuật: Kỹ thuật tổ chức dữ liệu, Bảng phương án.

Số nguyên tiếp theo nhỏ nhất không thể xây dựng phụ thuộc vào tần số xuất hiện các chữ số.

Nếu các chữ số từ 1 đến 9 xuất hiện **k** lần, chữ số 0 xuất hiện **k-1** lần và tồn tại bộ các khối lập phương khác nhau chứa chúng, ta có thể xây dựng mọi số có **k** chữ số (tức là từ 1 đến 10^{k-1}).

Gọi **d** là *chữ số nhỏ nhất khác 0* có *tần số xuất hiện nhỏ nhất* và **k** – tần số xuất hiện của nó ở các khối khác nhau.

Nếu tần số xuất hiện chữ số 0 là **k-1** thì số nhỏ nhất cần tìm sẽ là 10^{k-1} (*Trường hợp **a***).

Trường hợp tần số xuất hiện chữ số **0** lớn hơn hoặc bằng **k**:

- Nếu bộ các khối lập phương chứa **d** và **0** trùng nhau thì số nhỏ nhất cần tìm sẽ là $d \times 10^{k-1}$ (*Trường hợp **b***).
- Trong trường hợp ngược lại – số cần tìm sẽ có dạng

$$\underbrace{dddddd...dd}_{k+1 \text{ chữ số}} \qquad (\textit{Trường hợp } \mathbf{c}).$$

Tổ chức dữ liệu:

- Mảng `vector<int> cnt(10)` – `ctn[i]` lưu số khối lập phương chữ số i,
- Mảng hai chiều `vector<vector<int>> cubes(10)` – Lưu số của các khối lập phương chứa mỗi chữ số.

Xử lý:

Xử lý một khối lập phương:

```
fi >> s;
vector<int> cur;
for (char c : s) cur.push_back(c - '0');

sort(cur.begin(), cur.end());
cur.erase(unique(cur.begin(), cur.end()), cur.end());

for (int d : cur)
{
        cnt[d]++;
        cubes[d].push_back(i);
}
```

Ghi nhận chữ số

Lọc bỏ chữ số giống nhau

Ghi nhận tần số

Ghi nhận khối lập phương

Dẫn xuất kết quả:

```
string ans;
char c;
int k = min(cnt[0] + 1, *min_element(cnt.begin() + 1, cnt.end()));

if (cnt[0] == k - 1) ans = '1'+ string(k - 1, '0');

    else
        for (int i = 1; i < 10; ++i)
            if (cnt[i] == k)
            {
                c = char(48+i);
                if (cubes[0] == cubes[i]) ans = c+string(k, '0');
                    else ans = string(k + 1, c);
                break;
            }
```

Xác định tần số min

Trường hợp a

Trường hợp b

Trường hợp c

Độ phức tạp của giải thuật: O(nlogn).

Chương trình

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

int main()
{
    int n;
    fi >> n;
    vector<int> cnt(10);
    vector<vector<int>> cubes(10);
    for (int i = 0; i < n; ++i)
    {
        string s;
        fi >> s;
        vector<int> cur;
        for (char c : s) cur.push_back(c - '0');
        sort(cur.begin(), cur.end());
        cur.erase(unique(cur.begin(), cur.end()), cur.end());
        for (int d : cur)
        {
            cnt[d]++;
            cubes[d].push_back(i);
        }
    }
    string ans;
    char c;
    int k = min(cnt[0] + 1, *min_element(cnt.begin() + 1, cnt.end()));
    if (cnt[0] == k - 1) ans = '1'+ string(k - 1, '0');
        else
            for (int i = 1; i < 10; ++i)
                if (cnt[i] == k)
                {
                    c = char(48+i);
                    if (cubes[0] == cubes[i]) ans = c+string(k, '0');
                        else ans = string(k + 1, c);
                    break;
                }
  fo<<ans;
  fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Bob được lớp cử đi mua hoa tặng Alice nhân ngày sinh nhật.

Quầy bán hoa trưng bày n loại hoa. Tại quầy loại thứ i có a_i bông, $i = 1 \div n$. Bob biết Alice thích số lẻ hơn số chẵn vì vậy quyết định mua một bó hoa gồm số lẻ loại hoa và mỗi loại – có số lẻ bông.

Hãy xác định bó hoa Bob sẽ có số lượng tối đa bao nhiêu bông.

Dữ liệu: Vào từ file BOUQUET.INP:

- Dòng đầu tiên chứa số nguyên n ($1 \le n \le 10^5$),
- Dòng thứ 2 chứa n số nguyên $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^3$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản BOUQUET.OUT một số nguyên – số lượng tối đa các bông hoa được mua.

Ví dụ:

BOUQUET.INP
3 3 5 10

BOUQUET.OUT
17

WA46 VKOI 20191130 A XVII

Giải thuật: Cơ sở lập trình.

Số loại hoa sẽ mua là **n** nếu **n** lẻ và bằng **n**-1 trong trường hợp ngược lại,

Nếu **n** chẵn – không mua loại hoa có số lượng nhỏ nhất,

Với loại hoa **i** được mua: số lượng bông sẽ mua là a_i nếu a_i lẻ và là a_i-1 trong trường hợp ngược lại.

Độ phức tạp của giải thuật: O(*n*).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "bouquet."
using namespace std;
int n, ans=0,bmn=1001;

int main()
{
    freopen(NAME"inp","r",stdin);
    freopen(NAME"out","w",stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    cin>>n;
    vector<int> b(n);
    for(int &i:b) cin>>i;
    if(n&1) for(int i=0; i<n; ++i) ans+=(b[i]&1)? b[i]:b[i]-1;
        else
        {
            for(int i=0; i<n; ++i) bmn=min(bmn,b[i]);
            for(int i=0; i<n; ++i) ans+=(b[i]&1)? b[i]:b[i]-1;
            if(bmn&1 == 0)bmn--;
            ans-=bmn;
        }
    cout<<ans;
    cout<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

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

int main()
{
```

```
    int n;
    fi>>n;
    vector<int> a(n);
    for(int &i:a) fi>>i;
    int ans = *min_element(a.begin(),a.end())*((n&1) - 1);
    for(int i:a) ans += i - (1 - (i&1));
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}


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

int main()
{
    int n, b = 1010;
    fi>>n;
    vector<int> a(n);
    for(int &i:a)
    {
        fi>>i;
        if(i<b) b = i;
    }
    int ans = b*((n&1) - 1);
    for(int i:a) ans += i - (1 - (i&1));
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}


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

int main()
{
    int n, a, ans=0, b = 1010;
    fi>>n;
    for(int i=0; i<n; ++i)
    {
        fi>>a;
        if(a<b) b = a;
        ans +=  a - (1 - (a&1));
```

```
    }
    ans += b*((n&1) - 1);
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

WA47. THẮNG CỐ

Tên chương trình: BROTH.CPP

Sắp sửa khai giảng năm học mới và các bạn trường nội trú bắt đầu trở lại trường. Ngày đầu tiên lớp 11 Toán – Tin có ***m*** bạn tới. Các bạn rất vui khi được gặp lại nhau và còn vui hơn khi Cô giáo chủ nhiệm thông báo buổi tối sẽ chiêu đãi một nồi thắng cố. Cô có ***n*** loại gia vị khác nhau có thể sử dụng khi nấu và dĩ nhiên có thể cho hoặc không cho một số gia vị nào đó và cũng có thể không dùng gia vị nào cả. Các gia vị được đánh số từ 1 đến ***n***.

Đã chủ nhiệm lớp một năm nên cô biết rõ sở thích của từng bạn. Mỗi bạn thích thắng cố phải có một số lại gia vị hoặc không có một số loại gia vị nào đó, các gia vị khác – có hay không có cũng được. Việc đáp ứng đúng mọi sở thích của mỗi bạn nói chung là không thể. Tuy vậy các bạn trẻ của chúng ta dễ tính và sẽ rất hài lòng nếu có ít nhất một sở thích của mình được cô giáo lưu ý đáp ứng.

	Ớt	*Hoa hồi*	*Tiêu xanh*	*Gừng nướng*
Bạn I	**Có**		**Không**	
Bạn II		**Có**		**Có**
Bạn III		**Không**		

Ví dụ, Cô giáo có 4 loại gia vị là Ớt, Hoa hồi, Tiêu xanh và Gừng nướng. Có 3 bạn trở lại trường trong ngày đầu tiên với các sở thích nêu ở bảng bên. Cô giáo có thể nấu một nồi thắng cố không có Hoa hồi, nhưng có Ớt, Gừng nướng và Cô sẽ nói với bạn I: " *Có Ớt, em xem đã đủ cay chưa*?", với bạn II: "*Cô có cho Gừng Lạng Sơn đấy*" còn với bạn III: "*Cô không cho Hoa hồi đâu*".

Có rất nhiều cách nấu khác nhau để mỗi bạn có ít nhất một sở thích được đáp ứng. Hai cách nấu gọi là khác nhau nếu tồn tại một gia vị có ở cách nấu này và không có ở cách nấu kia.

Ở ví dụ trên có 3 cách nấu: Chỉ cho Gừng nướng, cho Gừng nướng và Ớt, cho Gừng nướng, Tiêu xanh và Ớt.

Hãy xác định Cô giáo có bao nhiêu cách nấu khác nhau. Số lượng cách nấu tìm được có thể rất lớn vì vậy chỉ cần đưa ra số dư của số lượng cách nấu chia cho **998244353**.

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

- Dòng đầu tiên chứa 2 số nguyên n và m ($1 \le n \le 10^3$, $1 \le m \le 20$),
- Dòng thứ i trong m dòng sau mô tả sở thích bạn thứ i gồm số nguyên k_i – số lượng gia vị muốn có hoặc không có, sau đó là k_i số nguyên $a_{i,1}, a_{i,2}, \ldots, a_{i,k_i}$, $1 \le |a_{i,j}| \le n$, $a_{i,j} > 0$ – muốn có gia vị $a_{i,j}$, $a_{i,j} < 0$ – không muốn có gia vị $a_{i,j}$. Các gia vị nêu trong một dòng đều khác nhau.

Các số trên một dòng ghi cách nhau một dấu cách.

Kết quả: Đưa ra file văn bản BROTH.OUT một số nguyên – số dư của số lượng cách nấu chia cho **998244353**.

Ví dụ:

BROTH.INP
4 3 2 1 -3 2 2 4 1 -2

BROTH.OUT
3

WA47 A XVII

Giải thuật: Nguyên lý bù trừ, Xử lý bít .

Sử dụng hai bản đồ kiểu **bitset** cho mỗi học sinh để đánh dấu các gia vị thích và không thích.

Áp dụng nguyên lý bù trừ trên cơ sở các bản đồ đánh dấu.

Độ phức tạp của giải thuật: $O(2^m \times n/32)$. VX 27

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("broth.inp");
ofstream fo ("broth.out");
const int md = 998244353;

const int N = 1010;
const int M = 22;

bitset<N> b[M][2];
int ans;
bitset<N> c[M][2];
```

```
int n, m;
int p2[N];

void dfs(int v, int w)
{
  if (v == m)
  {
    if ((c[v][0] & c[v][1]).count() != 0) return;
    int cnt = (c[v][0] | c[v][1]).count();
    if (w == 1) ans = (ans + p2[n - cnt]) % md;
        else ans = (ans - p2[n - cnt] + md) % md;
    return;
  }
  c[v + 1][0] = c[v][0];
  c[v + 1][1] = c[v][1];
  dfs(v + 1, w);
  c[v + 1][0] |= b[v][0];
  c[v + 1][1] |= b[v][1];
  dfs(v + 1, -w);
}

int main()
{
  fi >> n >> m;
  for (int i = 0; i < m; i++)
  {
    int k;
    fi >> k;
    for (int j = 0; j < k; j++)
    {
      int spice;
      fi >> spice;
      int flag;
      if (spice > 0) flag = 0;
        else
        {
            spice = -spice;
            flag = 1;
        }
      b[i][flag].set(spice - 1);
    }
  }
  p2[0] = 1;
  for (int i = 0; i < n; i++)
        p2[i + 1] = (p2[i]<<1) % md;

  ans = 0;
  dfs(0, 1);
  fo<<ans<<'\n';
  fo<<"\nTime: "<<clock()/(double)1000<<" sec ";
}
```

Có hai bức tường mỏng hình chữ nhật cùng độ cao. Chân của bức tường thứ nhất là đoạn thẳng AB có tọa độ các điểm đầu và cuối tương ứng là (x_a, y_a) và (x_b, y_b). Chân của bức tường thứ hai là đoạn thẳng CD có tọa độ các điểm đầu và cuối tương ứng là (x_c, y_c) và (x_d, y_d). Các bức tường không có điểm chung.

Một nguồn sáng chiếu vào bức tường thứ nhất theo hướng $\vec{v}$ = (v_x, v_y)tạo thành một bóng râm hình hộp chữ nhật kéo sau bức tường.

Nói bức tường thứ nhất phủ bóng lên bức tường thứ hai nếu bóng râm đè lên ít nhất một điểm của bức tường thứ hai.

Hãy xác định bức tường thứ nhất có phủ bóng lên bức tường thứ hai hay không và đưa ra thông báo **Yes** hoặc **No** tương ứng.

Dữ liệu: Vào từ file OVERLAP.INP:

- Dòng đầu tiên chứa số nguyên n – số lượng tests cần kiểm tra ($1 \le n \le 50\,000$),
- Mỗi dòng trong n dòng thiếp theo chứa thông tin về một test, bao gồm 10 số nguyên x_a, y_a, x_b, y_b, x_c, y_c, x_d, y_d ,v_x, v_y, các số có giá trị tuyệt đối không vượt quá 10^6, vevc tơ v khác 0.

Kết quả: Đưa ra file văn bản OVERLAP.OUT, kết quả mỗi test là **Yes** hoặc **No** và được đưa ra trên một dòng.

Ví dụ:

OVERLAP.INP
2 0 2 1 1 2 2 3 1 1 1 0 2 1 1 2 2 3 1 -1 -1

OVERLAP.OUT
Yes No

WA48_VOKO20191130 B_A XVII

Giải thuật: Phương trình đường thẳng.

Vấn đề phải xét ở đây là kiểm tra *một tia* có *cắt đoạn thẳng* cho trước hay không.

Đoạn AB phủ bóng lên đoạn CD theo hướng $\vec{v}$ khi thỏa mãn một trong 2 điều kiện sau:

- Tia theo hướng $\vec{v}$ từ A hoặc B có điểm chung với đoạn CD,
- Tia theo hướng $\overleftarrow{v}$ từ C hoặc D có điểm chung với đoạn AB.

Trường hợp **CD** *không đồng phương với* $\vec{v}$

Trường hợp **CD** *đồng phương với* $\vec{v}$

Cơ sở của lời giải là hàm **f(u,v,p,q)** kiểm tra tia $\vec{v}$ xuất phát từ điểm **U** có cắt đoạn thẳng **PQ** hay không. Hàm **f** cho giá trị **true** nếu tia cắt đoạn thẳng và cho giá trị **false** trong trường hợp nhược lại.

Việc kiểm tra được thực hiện với 4 trường hợp:

- **U = A, P = C, Q = D** và **V** = $\vec{v}$
- **U = B, P = C, Q = D** và **V** = $\vec{v}$
- **U = C, P = A, Q = B** và **V** = $\overleftarrow{v}$
- **U = D, P = A, Q = B** và **V** = $\overleftarrow{v}$

Xây dựng hàm **f(u,v,p,q)**:

Ký hiệu tọa độ các điểm cần xét tương ứng là (u_x, u_y), (p_x, p_y), (q_x, q_y) và phương (v_x, v_y).

Phương trình đường thẳng đi qua điểm **U** theo phương **V**:

$$\frac{x - u_x}{v_x} = \frac{y - u_y}{v_y}$$

$$x \times v_y - y \times v_x + u_y \times v_x - u_x \times v_y = 0$$

Khoảng cách có dấu *chưa chuẩn hóa* từ **P** và **Q** tới đường thẳng trên sẽ là:

$$t1 = p_x \times v_y - p_y \times v_x + u_y \times v_x - u_x \times v_y$$

$$t2 = q_x \times v_y - q_y \times v_x + u_y \times v_x - u_x \times v_y$$

Nếu **t1** và **t2** khác 0 và có cùng dấu – tia đang xét không cắt đoạn thẳng.

Nếu **t1** = 0 và **t2** = 0 ta có một trong hai trường hợp:

Xét phương trình của tia xuất phát từ **U** theo hướng $\vec{v}$:

$x = u_x + t*v_x$,

$y = u_y + t*v_y$, (*Phương trình dưới dạng tham số*)

$t \in [0,\infty]$.

Giải phương trình để xác định **t** cho tọa độ của tia ứng với điểm **P**:

$p_x = u_x + t*v_x \rightarrow t*v_x = p_x - u_x$

$p_y = u_y + t*v_y \rightarrow t*v_y = p_y - u_y$

Cần xét cả 2 phương trình vì v_x hoặc v_y có thể bằng 0.

Tồn tại nghiệm **t** không âm khi hệ số ở vế trái và giá trị ở vế phải cùng dấu.

Trường hợp đường thẳng đi qua **U**, có phương **v** cắt **PQ**:

Phương trình đường thẳng chứa đoạn **PQ**:

$$\frac{x - p_x}{q_x - p_x} = \frac{y - p_y}{q_y - p_y}$$

$$x \times (q_y - p_y) - y \times (q_x - p_x) + p_y \times (q_x - p_x) - p_x \times (q_y - p_y) = 0$$

Phương trình xác định giá trị **t** tương ứng với giao điểm của 2 đường:

$$(u_x + t*v_x)*(q_y - p_y) - (u_y + t*v_y)*(q_x - p_x) + p_y*(q_x - p_x) - p_x*(q_y - p_y) = 0$$

$$t*(v_x*(q_y - p_y) - v_y*(q_x - p_x)) = (u_y - p_y)*(q_x - p_x) - (u_x - p_x)*(q_y - p_y)$$

$$\underbrace{t*(v_x*(q_y - p_y) - v_y*(q_x - p_x))}_{t3} = \underbrace{(u_y - p_y)*(q_x - p_x) - (u_x - p_x)*(q_y - p_y)}_{t4}$$

Đoạn **PQ** cắt tia đang xét nếu phương trình trên có nghiệm **t** không âm, tức là **t3** và **t4** phải cùng dấu.

Từ kết quả gọi hàm **f** 4 lần với các tham số đã nêu ở trên ta có thrre xác định AB có phủ bóng lên CD theo hướng $\vec{v}$ hay không.

Tổ chức dữ liệu: Tọa độ mỗi điểm đã cho có thể lưu dưới dạng cặp dữ liệu **pair<int,int>**.

Độ phức tạp của giải thuật: O(*n*).

Chương trình

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
ifstream fi ("overlap.inp");
ofstream fo ("overlap.out");
typedef pair<int64_t,int64_t> pii;
int n;
pii a1, a2, b1, b2, vv, vr;

bool f(pii u, pii v, pii p, pii q)
{
    int64_t t1,t2,t3,t4,t33,t44;
    t3=u.ss*v.ff-u.ff*v.ss;
    t1=v.ss*p.ff-v.ff*p.ss+t3;
    t2=v.ss*q.ff-v.ff*q.ss+t3;
    if((t1>0 && t2>0) || (t1<0 && t2<0)) return false;

    if(t1==0 && t2==0)
      if((v.ff>=0 && p.ff-u.ff>=0 && v.ss>=0 && p.ss-u.ss>=0) ||
            (v.ff<=0 && p.ff-u.ff<=0 && v.ss<=0 && p.ss-u.ss<=0))return true;
                else return false;

    t3=v.ff*(q.ss-p.ss)-v.ss*(q.ff-p.ff);
    t4=-(u.ff-p.ff)*(q.ss-p.ss)+(u.ss-p.ss)*(q.ff-p.ff);

     if(t3>=0) t33=1; else t33=-1; if(t4>=0) t44=1; else t44=-1;
     if(t4==0 ||(t33*t44>0)) return true; else return false;
}

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i)
    {
       fi>>a1.ff>>a1.ss>>a2.ff>>a2.ss>>b1.ff>>b1.ss>>
              b2.ff>>b2.ss>>vv.ff>>vv.ss;
        vr.ff=-vv.ff; vr.ss=-vv.ss;
        if(f(a1,vv,b1,b2) || f(a2,vv,b1,b2) ||
           f(b1,vr,a1,a2) || f(b2,vr,a1,a2))
               fo<<"Yes\n"; else fo<<"No\n";
    }
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

WA49. TAM GIÁC CÂN

Tên chương trình: ISOSCELE.CPP

Tam giác cân là tam giác có 2 cạnh bằng nhau. Tam giác đều là trường hợp riêng của tam giác cân.

Cho đa giác đều ***n*** đỉnh. Hãy xác định số tam giác cân có đỉnh đồng thời là đỉnh của đa giác đều.

Ví dụ, với ***n*** = 5 ta có 10 tam giác cân.

Dữ liệu: Vào từ file ISOSCELE.INP gồm một dòng chứa số nguyên ***n*** ($3 \le n \le 10^9$).

Kết quả: Đưa ra file văn bản ISOSCELE.OUT một số nguyên – số tam giác cân xác định được.

Ví dụ:

ISOSCELE.INP
5

ISOSCELE.OUT
10

WA49 Voko 20191130 A XVII

Giải thuật: Thống kê đơn giản .

Để có tam giác cân, từ một đỉnh xuất phát ta phải nối với 2 đỉnh bên phải và bên trái cách đều **k** đỉnh so với đỉnh xuất phát.

Từ một đỉnh xuất phát ta có (**n**-1)/2 cách chọn cặp đỉnh còn lại.

Có tất cả **n** cách chọn đỉnh xuất phát.

Như vậy, số tam giác cân có thể nhận được là (**n**-1)/2*__n__.

Tuy vậy khi **n** chia hết cho 3 sẽ có **n**/3 tam giác đều, mỗi tam giác đều xuất hiện 3 lần trùng nhau, vì vậy cần phải bớt đi 2*__n__/3 số lần tính lặp với các tam giác đều.

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int main()
{
    int64_t n;
    fi >> n;
    int64_t ans = n * ((n - 1) / 2);
    if (n % 3 == 0)
        ans -= 2 * n / 3;
    fo << ans << endl;
}
```

Ferma năm 1637 đã viết bên lề một cuốn sách: "Phương trình $a^n+b^n=c^n$ không có nghiệm nguyên dương a, b, c với $n \geq 3$, nhưng ở đây không đủ chỗ để trình bày chứng minh".

Mãi đến năm 1995 nhà toán học Anh W. Andrew mới chứng minh được điều này và với công trình đó ông đã được tặng Giải thưởng toán học Abel năm 2016.

Trước đó người ta đã dùng máy tính để so sánh các giá trị a^n+b^n và c^n với a, b, c, n nguyên dương đủ lớn, $n \geq 3$..

Để tránh trùng lặp cũng như bỏ sót các trường hợp cần kiểm tra, người ta xét bộ số (a, b, c, n) theo thứ tự tăng dần của *max*(*a*,*b*,*c*,*n*), với các bộ số có max giống nhau – xét theo thứ tự từ điển tăng dần. Ví dụ, sau (1, 1, 1, 3) là (1, 1, 2, 3), sau (3, 3, 3, 3) là (1, 1, 1, 4). Bộ số có thứ tự từ điển nhỏ nhất là (1, 1, 1, 3) và được đánh số là 1.

Với mỗi bộ số (a, b, c, n) , kết quả được đưa ra dưới dạng **a^n+b^n>c^n** hoặc **a^n+b^n<c^n**.

Hãy đưa ra kết quả kiểm tra các bộ số có số thứ tự từ **lf** đến **rt**.

Dữ liệu: Vào từ file FERMAT.INP gồm một dòng chứa 2 số nguyên ***lf*** và ***rt*** ($1 \leq lf \leq rt \leq 10^{12}$, $rt-lf \leq 10^4$)

Kết quả: Đưa ra file văn bản FERMAT.OUT, kết quả của mỗi bộ số đưa ra trên một dòng, theo thứ tự từ điển của các bộ số.

Ví dụ:

FERMAT.INP
1 4

FERMAT.OUT
1^3+1^3>1^3
1^3+1^3<2^3
1^3+1^3<3^3
1^3+2^3>1^3

WA250 Voko20191130 C A XVII

Giải thuật: Thứ tự từ điển và tính nhanh lũy thừa .

Các bộ 4 số (**a**, **b**, **c**, **n**) có thể phân nhóm theo giá trị lớn nhất của các số trong nhóm.

Nhóm đầu tiên có giá trị *max* là 3, nhóm thứ 2 có giá trị *max* là 4, . . .

Gọi số lớn nhất là đại diện của nhóm.

Các bộ cùng nhóm đứng liên tiếp nhau trong dãy đã sắp xếp theo quy tắc nêu trong đầu bài.

Các vấn đề cần giải quyết:

- Tìm số đại diện của nhóm đầu tiên trong khoảng cần xử lý,
- Xác định nhóm đầu tiên cần xử lý,
- Xác định nhóm tiếp sau theo thứ tự từ điển,
- Tính, so sánh các lũy thừa và đưa ra kết quả.

a) Tìm số đại diện của nhóm đầu tiên trong khoảng cần xử lý:

Gọi **m** là số đại diện cần tìm.

Xác định số phần tử trong nhóm:

Tạm gác sự tồn tại bắt buộc của phần tử đại diện:

- Có **m** khả năng lựa chọn **a** (từ 1 đến **m**),
- Tương tự như vậy, có **m** khả năng lựa chọn **b** và **m** khả năng lựa chọn **c**,
- Với **n**: **m−2** khả năng lựa chọn: từ **3** đến **m**.

Tổng cộng số lượng nhóm sẽ là **m×m×m×(m−2)**.

Trong số nhóm nói trên, có những *nhóm không chứa phần tử đại diện*:

- ✓ Có **m−1** cách chọn **a** nhỏ hơn **m**,
- ✓ Tương tự, có **m−1** cách chọn **b** và **m−1** cách chọn **c** không nhận giá trị **m**,
- ✓ Có **m−3** khả năng chọn **n ≠ m**.

Tổng cộng có **(m−1)×(m−1)×(m−1)×(m−3)**.

Như vậy số phần tử trong nhóm nhận **m** là đại diện sẽ là

$$P_m = m \times m \times m \times (m-2) - (m-1) \times (m-1) \times (m-1) \times (m-3)$$

Lần lượt trừ **lf** cho P_m, **m** = 3, 4, 5, . . .chừng nào **lf** còn lớn hơn P_m ta sẽ xác định được số đại diện **m** của nhóm đầu tiên cần xử lý.

Tổ chức dữ liệu:

Mảng `int x[4]` – Lưu các giá trị **a**, **b**, **c**, **n**.

Xử lý:

a) Tìm đại diện của nhóm đầu tiên:

```
m=3; m4=m*m*m*(m-2)-(m-1)*(m-1)*(m-1)*(m-3);
while(lf>m4)
{
    lf-=m4; ++m;
    m4=m*m*m*(m-2)-(m-1)*(m-1)*(m-1)*(m-3);
}
```

b) Xác định bộ số đầu tiên cần xử lý:

Để thuận tiện xử lý, thứ tự các bộ số trong nhóm được tính bắt đầu từ 0.

Việc xác định bộ số được thực hiện theo nguyên tắc *cục bộ hóa dần* phạm vi tìm kiếm, đầu tiên theo **a**, sau đó – theo **b** và cuối cùng – theo **c**. Giá trị **n** được tự động xác định dựa theo *kết quả quá trình cục bộ hóa*.

Mỗi số **a** cố định trong bộ (**a**, **b**, **c**, **n**) sẽ gắn với mọi khả năng có thể của bộ 3 số (**b**, **c**, **n**).

Dễ dàng thấy được trong nhóm cùng đại diện **m**, bộ 4 số (**a**, **b**, **c**, **n**) với **a** cố định xuất hiện **m×m×(m−2)−(m−1)×(m−1)×(m−3)** lần.

Ký hiệu **m4= m×m×(m−2)−(m−1)×(m−1)×(m−3)** , **lf** – thứ tự từ điển của bộ số cần tìm.

a = u = *min*(m,lf/m4+1)

Ta có:

Gọi mỗi bộ 4 số (**a**, **b**, **c**, **n**) là một phần tử. Phần tử đầu tiên của nhóm con (**u**, **b**, **c**, **n**) có thứ tự từ điển là **p=1f-(u-1)×m4)**.

Gọi **kb** là kích thước của nhóm các phần tử có **a** = **u** .

*Trong số các **b**, **c**, **n** còn lại phải có đại lượng bằng **m***

$$kb = \begin{cases} m\times(m-2)-(m-1)\times(m-3) & \text{nếu } u < m, \\ m\times(m-2) & \text{nếu } u = m. \end{cases}$$

Lập luận tương tự như trên ta có:

$$\boxed{b = v = \min(m, p/kb+1)}$$

Phần tử đầu tiên của nhóm con (**u**, **v**, **c**, **n**) có thứ tự từ điển là **q = p-(v-1)×kb)**.

Kích thước **kc** của nhóm các phần tử có **a** = **u** và **b** = **v** sẽ là:

$$kc = \begin{cases} 1 & \text{nếu } u = m \text{ hoặc } v = m, \\ m-2 & \text{trong trường hợp ngược lại.} \end{cases}$$

*Trong hai số **c**, **n** còn lại phải có đại lượng bằng **m***

c nhận giá trị $\boxed{w = \min(m, q/kc+1)}$

Khoảng cách còn lại tới phần tử cần tìm là **r = q-(v-1)×kc)**.

Giá trị **n** được xác định theo công thức:

$$n = \begin{cases} r+3 & \text{nếu } u = m \text{ hoặc } v = m \text{ hoặc } v = m, \\ m & \text{trong trường hợp ngược lại.} \end{cases}$$

Lưu ý: Khi lập trình, giá trị **`lf`** sẽ được thu lùi dần về các mốc địa chỉ đã nêu ở trên.

Hàm xác định bộ dữ liệu xuất phát:

```
void get_first()
{
    int a,b,c,d,t;

    m4=m*m*(m-2)-(m-1)*(m-1)*(m-3);
    t=0;a=lf/m4+1; if(a>m) t=a-m,a=m; lf=lf%m4+t*m4;
    if(a==m)m4=m*(m-2);else m4=m*(m-2)-(m-1)*(m-3);

    t=0; b=lf/m4+1; if(b>m) t=b-m,b=m; lf=lf%m4+t*m4;
    if(a==m || b==m) m4=m-2; else m4=1;

    t=0; c=lf/m4+1; if(c>m) t=c-m,c=m; lf=lf%m4+t*m4;

    d=3+lf; if(a<m && b<m && c<m) d=m;

    x[0]=a; x[1]=b; x[2]=c; x[3]=d;
}
```

Tính a

Tính b

Tính c

Tính n

*c) Tìm bộ số (**a**, **b**, **c**, **n**) có thứ tự từ điển tiếp theo:*

```
void get_next()
{
    if(x[3]<m) {++x[3]; return;}

    if(x[2]<m){++x[2];
      {if(x[0]!=m &&x[1]!=m&&x[2]!=m)x[3]=m; else x[3]=3; return;}}
    if(x[1]<m)
    {
        ++x[1]; x[2]=1;
        if(x[0]<m&&x[1]<m) x[3]=m; else x[3]=3;
        return;
    }
    if(x[0]<m)
    {
       ++x[0];x[1]=x[2]=1;
       if(x[0]<m) x[3]=m; else x[3]=3;
       return;
    }
    x[0]=x[1]=x[2]=1; x[3]=++m;
}
```

d) Dẫn xuất kết quả:

Để giảm thiểu việc phải làm việc với số lớn khi tính $\mathbf{x^n}$ cần phân lập một số trường hợp riêng cho phép rút ngay ra kết luận cần tìm:

- Nếu $max(\mathbf{a}, \mathbf{b}) \geq \mathbf{c}$ ta có ngay kết quả $\mathbf{a^n+b^n > c^n}$,
- $\mathbf{a^n+b^n} \leq \mathbf{2\times}(max(\mathbf{a},\mathbf{b}))^{\mathbf{n}} \rightarrow$ nếu $\mathbf{2\times}(max(\mathbf{a},\mathbf{b}))^{\mathbf{n}} < \mathbf{c^n}$ ta có:

$$\log(\mathbf{2\times}(max(\mathbf{a},\mathbf{b}))^{\mathbf{n}}) < \log\mathbf{c^n}$$

$$\log\mathbf{2}+\mathbf{n\times}\log(max(\mathbf{a},\mathbf{b})) < \mathbf{n\times}\log\mathbf{c}$$

$$\mathbf{n\times}(\log\mathbf{c}- \log(max(\mathbf{a},\mathbf{b}))) > \log\mathbf{2}$$

Kết quả sẽ là $\mathbf{a^n+b^n < c^n}$,

Các trường hợp còn lại: Cần xây dựng hàm tính nhanh lũy thừa, xác định kết quả $\mathbf{x^n}$ theo kiểu `long double`. Kiểu dữ liệu này cho phép biểu diễn chính xác 20 chữ số bậc cao nhất, vừa đủ để so sánh trong phạm vi miền xác định các tham số của đầu bài.

Hàm tính nhanh lũy thừa:

```
long double poww(long double a, int n)
{
    long double ans = 1;
    while (n)
    {
        if (n & 1)
            ans = ans * a;
        a = a * a;
        n /= 2;
    }
    return ans;
}
```

Hàm dẫn xuất kết quả:

```
void print(int a, int b, int c, int n)
{
    if (max(a, b) >= c)
        fo<<a<<'^'<<n<<'+'<<b<<'^'<<n<<'>'<<c<<'^'<<n<<'\n';
    else if (n * (log(c) - log(max(a, b)) > log(2)))
        fo<<a<<'^'<<n<<'+'<<b<<'^'<<n<<'<'<<c<<'^'<<n<<'\n';
    else if (poww(a, n) + poww(b, n) < poww(c, n))
        fo<<a<<'^'<<n<<'+'<<b<<'^'<<n<<'<'<<c<<'^'<<n<<'\n';
    else
        fo<<a<<'^'<<n<<'+'<<b<<'^'<<n<<'>'<<c<<'^'<<n<<'\n';
}
```

Độ phức tạp của giải thuật: O(*k*), *k = rt-lf*.

Chương trình

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

typedef int64_t ll;
int64_t lf,rt,m, m4, m41;
int n, ix,t, x[4];
char c;

long double poww(long double a, int n)
{
    long double ans = 1;
    while (n)
    {
        if (n & 1)
            ans = ans * a;
        a = a * a;
        n /= 2;
    }
    return ans;
}

void print(int a, int b, int c, int n)
{
    if (max(a, b) >= c)
        fo<<a<<'^'<<n<<'+'<<b<<'^'<<n<<'>'<<c<<'^'<<n<<'\n';
    else if (n * (log(c) - log(max(a, b)) > log(2)))
        fo<<a<<'^'<<n<<'+'<<b<<'^'<<n<<'<'<<c<<'^'<<n<<'\n';
    else if (poww(a, n) + poww(b, n) < poww(c, n))
        fo<<a<<'^'<<n<<'+'<<b<<'^'<<n<<'<'<<c<<'^'<<n<<'\n';
    else
        fo<<a<<'^'<<n<<'+'<<b<<'^'<<n<<'>'<<c<<'^'<<n<<'\n';
}
    //Get first tuple
void get_first()
{
    int a,b,c,d,t;
    m4=m*m*(m-2)-(m-1)*(m-1)*(m-3);
    t=0;a=lf/m4+1; if(a>m) t=a-m,a=m; lf=lf%m4+t*m4;
    if(a==m)m4=m*(m-2);else m4=m*(m-2)-(m-1)*(m-3);
    t=0; b=lf/m4+1; if(b>m) t=b-m,b=m; lf=lf%m4+t*m4;
    if(a==m || b==m) m4=m-2; else m4=1;
    t=0; c=lf/m4+1; if(c>m) t=c-m,c=m; lf=lf%m4+t*m4;
    d=3+lf; if(a<m && b<m && c<m) d=m;
    x[0]=a; x[1]=b; x[2]=c; x[3]=d;
}

   //auto get_next = [&]()
void get_next()
{
    if(x[3]<m) {++x[3]; return;}

    if(x[2]<m){++x[2];
```

```
      {if(x[0]!=m &&x[1]!=m&&x[2]!=m)x[3]=m; else x[3]=3; return;}}
    if(x[1]<m)
    {
        ++x[1]; x[2]=1;
        if(x[0]<m&&x[1]<m) x[3]=m; else x[3]=3;
        return;
    }
    if(x[0]<m)
    {
       ++x[0];x[1]=x[2]=1;
       if(x[0]<m) x[3]=m; else x[3]=3;
       return;
    }
    x[0]=x[1]=x[2]=1; x[3]=++m;
}

int main()
{
    fi>>lf>>rt;
    n=rt-lf+1;
    m=3; m4=m*m*m*(m-2)-(m-1)*(m-1)*(m-1)*(m-3);
    while(lf>m4)
    {
        lf-=m4; ++m;
        m4=m*m*m*(m-2)-(m-1)*(m-1)*(m-1)*(m-3);
    }
    --lf;
    get_first();

    for (int i = 0; i < n; i++)
    {
        print(x[0], x[1], x[2], x[3]);
        get_next();
    }

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

Chương trình II

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

typedef int64_t ll;
int64_t lf,rt,m, m4, k;
int n, ix;
int a,b,c,d,t;

long double poww(long double a, int n)
{
    long double ans = 1;
    while (n)
    {
        if (n & 1)
            ans = ans * a;
        a = a * a;
        n /= 2;
    }
    return ans;
}

void print()
{
    if (max(a, b) >= c)
        fo<<a<<'^'<<d<<'+'<<b<<'^'<<d<<'>'<<c<<'^'<<d<<'\n';
    else if (n * (log(c) - log(max(a, b)) > log(2)))
        fo<<a<<'^'<<d<<'+'<<b<<'^'<<d<<'<'<<c<<'^'<<d<<'\n';
    else if (poww(a, d) + poww(b, d) < poww(c, d))
        fo<<a<<'^'<<d<<'+'<<b<<'^'<<d<<'<'<<c<<'^'<<d<<'\n';
    else
        fo<<a<<'^'<<d<<'+'<<b<<'^'<<d<<'>'<<c<<'^'<<d<<'\n';
}
    //Get first tuple
void get_first()
{

    m4=m*m*(m-2)-(m-1)*(m-1)*(m-3);
    t=0;a=k/m4+1; if(a>m) t=a-m,a=m; k=k%m4+t*m4;
    if(a==m)m4=m*(m-2);else m4=m*(m-2)-(m-1)*(m-3);
    t=0; b=k/m4+1; if(b>m) t=b-m,b=m; k=k%m4+t*m4;
    if(a==m || b==m) m4=m-2; else m4=1;
    t=0; c=k/m4+1; if(c>m) t=c-m,c=m; k=k%m4+t*m4;
    d=3+k; if(a<m && b<m && c<m) d=m;
}

void get_m()
{
    m=3; m4=m*m*m*(m-2)-(m-1)*(m-1)*(m-1)*(m-3);
    while(k>m4)
    {
        k-=m4; ++m;
        m4=m*m*m*(m-2)-(m-1)*(m-1)*(m-1)*(m-3);
```

```
    }
}

int main()
{
    fi>>lf>>rt;
    for (int64_t i = lf; i <= rt; i++)
    {
        k=i; get_m(); --k;
        get_first();
        print();
    }

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

Chương trình III

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

typedef int64_t ll;
int64_t lf,rt,m, m4, k;
int n, ix;
int a,b,c,d,t;

long double poww(long double a, int p)
{
    long double ans = 1;
    while (p)
    {
        if (p & 1)
            ans = ans * a;
        a = a * a;
        p /= 2;
    }
    return ans;
}

void print()
{
    if (max(a, b) >= c)
        fo<<a<<'^'<<d<<'+'<<b<<'^'<<d<<'>'<<c<<'^'<<d<<'\n';
    else if (n * (log(c) - log(max(a, b)) > log(2)))
        fo<<a<<'^'<<d<<'+'<<b<<'^'<<d<<'<'<<c<<'^'<<d<<'\n';
    else if (poww(a, d) + poww(b, d) < poww(c, d))
        fo<<a<<'^'<<d<<'+'<<b<<'^'<<d<<'<'<<c<<'^'<<d<<'\n';
    else
        fo<<a<<'^'<<d<<'+'<<b<<'^'<<d<<'>'<<c<<'^'<<d<<'\n';
}
    //Get first tuple
void get_first()
{

    m4=m*m*(m-2)-(m-1)*(m-1)*(m-3);
    t=0;a=k/m4+1; if(a>m) t=a-m,a=m; k=k%m4+t*m4;
    if(a==m)m4=m*(m-2);else m4=m*(m-2)-(m-1)*(m-3);
    t=0; b=k/m4+1; if(b>m) t=b-m,b=m; k=k%m4+t*m4;
    if(a==m || b==m) m4=m-2; else m4=1;
    t=0; c=k/m4+1; if(c>m) t=c-m,c=m; k=k%m4+t*m4;
    d=3+k; if(a<m && b<m && c<m) d=m;
}

void get_m()
{
    m=3; m4=m*m*m*(m-2)-(m-1)*(m-1)*(m-1)*(m-3);
    while(lf>m4)
    {
        lf-=m4; ++m;
        m4=m*m*m*(m-2)-(m-1)*(m-1)*(m-1)*(m-3);
```

```
    }
}

int main()
{
    fi>>lf>>rt;
    n=rt-lf+1;
    get_m(); k=--lf;
    for (int i=0; i<n; i++)
    {
        get_first();
        print();
        if(a+b+c+d==4*m) ++m, lf=0; else ++lf;
        k=lf;
    }
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Ferma năm 1637 đã viết bên lề một cuốn sách: "Phương trình $a^n+b^n=c^n$ không có nghiệm nguyên dương a, b, c với $n \geq 3$, nhưng ở đây không đủ chỗ để trình bày chứng minh".

Mãi đến năm 1995 nhà toán học Anh W. Andrew mới chứng minh được điều này và với công trình đó ông đã được tặng Giải thưởng toán học Abel năm 2016.

Trước đó người ta đã dùng máy tính để so sánh các giá trị a^n+b^n và c^n với a, b, c, n nguyên dương đủ lớn, $n \geq 3$..

Để tránh trùng lặp cũng như bỏ sót các trường hợp cần kiểm tra, người ta xét bộ số (a, b, c, n) theo thứ tự tăng dần của *max*(*a*,*b*,*c*,*n*), với các bộ số có max giống nhau – xét theo thứ tự từ điển tăng dần. Ví dụ, sau (1, 1, 1, 3) là (1, 1, 2, 3), sau (3, 3, 3, 3) là (1, 1, 1, 4). Bộ số có thứ tự từ điển nhỏ nhất là (1, 1, 1, 3) và được đánh số là 1.

Với mỗi số nguyên k hãy đưa ra bộ số (a, b, c, n) có thứ tự từ điển là k.

Dữ liệu: Vào từ file LEXICO.INP:

- Dòng đầu tiên chứa số nguyên n – số bộ dữ liệu cần tìm ($1 \leq n \leq 10^5$),
- Mỗi dòng trong n dòng sau chứa một số nguyên k ($1 \leq k \leq 10^{15}$),

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

Ví dụ:

LEXICO.INP
4 35 4 28 32768

LEXICO.OUT
1 2 3 4 1 2 1 3 1 1 1 4 14 14 1 10

WB01 A XVII

Giải thuật: Thứ tự từ điển và tính nhanh lũy thừa .

Các bộ 4 số (**a**, **b**, **c**, **n**) có thể phân nhóm theo giá trị lớn nhất của các số trong nhóm.

Nhóm đầu tiên có giá trị *max* là 3, nhóm thứ 2 có giá trị *max* là 4, . . .

Gọi số lớn nhất là đại diện của nhóm.

Các bộ cùng nhóm đứng liên tiếp nhau trong dãy đã sắp xếp theo quy tắc nêu trong đầu bài.

Các vấn đề cần giải quyết:

- Tìm số đại diện của nhóm đầu tiên trong khoảng cần xử lý,
- Xác định nhóm đầu tiên cần xử lý,

a) Tìm số đại diện của nhóm đầu tiên trong khoảng cần xử lý:

Gọi **m** là số đại diện cần tìm.

Xác định số phần tử trong nhóm:

Tạm gác sự tồn tại bắt buộc của phần tử đại diện:

- Có **m** khả năng lựa chọn **a** (từ 1 đến **m**),
- Tương tự như vậy, có **m** khả năng lựa chọn **b** và **m** khả năng lựa chọn **c**,
- Với **n**: **m−2** khả năng lựa chọn: từ **3** đến **m**.

Tổng cộng số lượng nhóm sẽ là **m×m×m×(m−2)**.

Trong số nhóm nói trên, có những *nhóm không chứa phần tử đại diện*:

- ✓ Có **m−1** cách chọn **a** nhỏ hơn **m**,
- ✓ Tương tự, có **m−1** cách chọn **b** và **m−1** cách chọn **c** không nhận giá trị **m**,
- ✓ Có **m−3** khả năng chọn **n ≠ m**.

Tổng cộng có **(m−1)×(m−1)×(m−1)×(m−3)**.

Như vậy số phần tử trong nhóm nhận **m** là đại diện sẽ là

$$P_m = m \times m \times m \times (m-2) - (m-1) \times (m-1) \times (m-1) \times (m-3)$$

Lần lượt trừ **lf** cho P_m, **m** = 3, 4, 5, . . .chừng nào **lf** còn lớn hơn P_m ta sẽ xác định được số đại diện **m** của nhóm đầu tiên cần xử lý.

Tổ chức dữ liệu:

Mảng `int x[4]` – Lưu các giá trị **a**, **b**, **c**, **n**.

Xử lý:

a) Tìm đại diện của nhóm đầu tiên:

```
m=3; m4=m*m*m*(m-2)-(m-1)*(m-1)*(m-1)*(m-3);
while(lf>m4)
{
    lf-=m4; ++m;
    m4=m*m*m*(m-2)-(m-1)*(m-1)*(m-1)*(m-3);
}
```

b) Xác định bộ số đầu tiên cần xử lý:

Để thuận tiện xử lý, thứ tự các bộ số trong nhóm được tính bắt đầu từ 0.

Việc xác định bộ số được thực hiện theo nguyên tắc *cục bộ hóa dần* phạm vi tìm kiếm, đầu tiên theo **a**, sau đó – theo **b** và cuối cùng – theo **c**. Giá trị **n** được tự động xác định dựa theo *kết quả quá trình cục bộ hóa*.

Mỗi số **a** cố định trong bộ (**a**, **b**, **c**, **n**) sẽ gắn với mọi khả năng có thể của bộ 3 số (**b**, **c**, **n**).

Dễ dàng thấy được trong nhóm cùng đại diện **m**, bộ 4 số (**a**, **b**, **c**, **n**) với **a** cố định xuất hiện **m×m×(m−2)−(m−1)×(m−1)×(m−3)** lần.

Ký hiệu **m4= m×m×(m−2)−(m−1)×(m−1)×(m−3)** , **lf** – thứ tự từ điển của bộ số cần tìm.

Ta có: **a = u = *min*(m,lf/m4+1)**

Gọi mỗi bộ 4 số (**a**, **b**, **c**, **n**) là một phần tử. Phần tử đầu tiên của nhóm con (**u**, **b**, **c**, **n**) có thứ tự từ điển là **p=lf-(u-1)×m4)**.

Gọi **kb** là kích thước của nhóm các phần tử có **a** = **u** .

*Trong số các **b**, **c**, **n** còn lại phải có đại lượng bằng **m***

$$kb = \begin{cases} m\times(m-2)-(m-1)\times(m-3) & \text{nếu } u < m, \\ m\times(m-2) & \text{nếu } u = m. \end{cases}$$

Lập luận tương tự như trên ta có:

$$\boxed{b = v = \min(m, p/kb+1)}$$

Phần tử đầu tiên của nhóm con (**u**, **v**, **c**, **n**) có thứ tự từ điển là **q = p-(v-1)×kb)**.

Kích thước **kc** của nhóm các phần tử có **a** = **u** và **b** = **v** sẽ là:

$$kc = \begin{cases} 1 & \text{nếu } u = m \text{ hoặc } v = m, \\ m-2 & \text{trong trường hợp ngược lại.} \end{cases}$$

*Trong hai số **c**, **n** còn lại phải có đại lượng bằng **m***

c nhận giá trị $\boxed{w = \min(m, q/kc+1)}$

Khoảng cách còn lại tới phần tử cần tìm là **r = q-(v-1)×kc)**.

Giá trị **n** được xác định theo công thức:

$$n = \begin{cases} r+3 & \text{nếu } u = m \text{ hoặc } v = m \text{ hoặc } v = m, \\ m & \text{trong trường hợp ngược lại.} \end{cases}$$

Lưu ý: Khi lập trình, giá trị **lf** sẽ được thu lùi dần về các mốc địa chỉ đã nêu ở trên.

Hàm xác định bộ dữ liệu xuất phát:

```
void get_first()
{
    int a,b,c,d,t;

    m4=m*m*(m-2)-(m-1)*(m-1)*(m-3);
    t=0;a=lf/m4+1; if(a>m) t=a-m,a=m; lf=lf%m4+t*m4;
    if(a==m)m4=m*(m-2);else m4=m*(m-2)-(m-1)*(m-3);

    t=0; b=lf/m4+1; if(b>m) t=b-m,b=m; lf=lf%m4+t*m4;
    if(a==m || b==m) m4=m-2; else m4=1;

    t=0; c=lf/m4+1; if(c>m) t=c-m,c=m; lf=lf%m4+t*m4;

    d=3+lf; if(a<m && b<m && c<m) d=m;

    x[0]=a; x[1]=b; x[2]=c; x[3]=d;
}
```

Tính a

Tính b

Tính c

Tính n

Độ phức tạp của giải thuật: O(nlog(*max(k)*)).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("lexico.inp");
ofstream fo ("lexico.out");
int64_t n,k,m,m4;

void get_first()
{
    int a,b,c,d,t;
    m4=m*m*(m-2)-(m-1)*(m-1)*(m-3);
    t=0;a=k/m4+1; if(a>m) t=a-m,a=m; k=k%m4+t*m4;
    if(a==m)m4=m*(m-2);else m4=m*(m-2)-(m-1)*(m-3);
    t=0; b=k/m4+1; if(b>m) t=b-m,b=m; k=k%m4+t*m4;
    if(a==m || b==m) m4=m-2; else m4=1;
    t=0; c=k/m4+1; if(c>m) t=c-m,c=m; k=k%m4+t*m4;
    d=3+k; if(a<m && b<m && c<m) d=m;
    fo<<a<<' '<<b<<' '<<c<<' '<<d<<'\n';
}

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i)
    {
        fi>>k;
        m=3; m4=m*m*m*(m-2)-(m-1)*(m-1)*(m-1)*(m-3);
        while(k>m4)
        {
            k-=m4; ++m;
            m4=m*m*m*(m-2)-(m-1)*(m-1)*(m-1)*(m-3);
        }
        --k;
        get_first();
    }

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

Có một số chuẩn ghi ngày tháng khác nhau. Châu Âu ghi ngày tháng theo chuẩn "***dd.mm.yyyy***", chuẩn của nước Mỹ là "***mm/dd/yyyy***", trong đó ***dd*** là ngày, ***mm*** – tháng, ***yyyy*** – năm.

Thông tin tập hợp về trung tâm lưu trữ chứa ngày tháng theo các chuẩn khác nhau, hơn thế nữa có khi ngày và tháng chỉ có một chữ số, còn năm – có thể không đủ 4 chữ số.

Nhiệm vụ của mô đun chuẩn hóa dữ liệu là tìm tất cả số liệu về ngày tháng và sau đó, với mỗi ngày tháng tìm được đưa ra dạng chuẩn châu Âu và dạng chuẩn của Mỹ để mô đun lưu trữ lấy ra lưu trữ ở dạng cần thiết theo tùy chọn.

Có tất cả ***n*** ngày tháng được lọc ra. Với mỗi ngày tháng hãy đưa ra dạng chuẩn theo các kiểu đã nêu.

Dữ liệu: Vào từ file DATE.INP:

- Dòng đầu tiên chứa số nguyên ***n*** ($1 \le n \le 200\,000$),
- Mỗi dòng trong ***n*** dòng sau chứa thông tin chưa chuẩn hóa về một ngày tháng theo một trong hai dạng chuẩn đã nêu.

Kết quả: Đưa ra file văn bản DATE.OUT, với mỗi ngày tháng – đưa ra trên một dòng, cách nhau một dấu cách dạng chuẩn hóa châu Âu và dạng chuẩn của nước Mỹ.

Ví dụ:

DATE.INP
4 11.12.2000 1.2.1 20.10.2100 1/29/3000

DATE.OUT
11.12.2000 12/11/2000 01.02.0001 02/01/0001 20.10.2100 10/20/2100 29.01.3000 01/29/3000

Giải thuật: Nhận dạng và xử lý xâu .

Để đơn giản, gọi chuẩn ghi ngày tháng cúa châu Âu là dạng chuẩn I, chuẩn ghi ngày tháng nước Mỹ - chuẩn II, thông tin chưa chuẩn hóa – gọi tương ứng là dạng I và dạng II.

Trước hết cần xác định thông tin nhận được ở dạng nào.

Nếu trong xâu cần chuẩn hóa có ký tự '**.**' – thông tin ở dạng I, trong trường hợp ngược lại – ở dạng II. Các ký tự này chia xâu cần xử lý thành 3 trường **a**, **b** và **c**:

a.b.c *hoặc* **a/b/c**

Chuẩn hóa **a** và **b**: nếu độ dài bằng 1 thì thêm ký tự **0** ở đầu,

Chuẩn hóa **c**: Nếu độ dài nhỏ hơn 4 thì thêm các ký **0** ở đầu để có đủ 4 ký tự.Việc chuẩn hóa **c** có thể được thực hiện dễ dàng nếu sử dụng dữ liệu hỗ trợ.

Đưa về trường hợp **a** chứa ngày, **b** chứa tháng.

Từ các trường đã chuẩn hóa – dẫn xuất ngày tháng theo các chuẩn.

Độ phức tạp của giải thuật: O(*n*).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("date.inp");
ofstream fo ("date.out");
int n,k;
string s,res1,res2,a,b,c,p[3]={"0","00","000"};
char d;

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i)
    {
        fi>>s; a=b=c="";
        if(s.find('.')!=string:: npos) d='.'; else d='/';
        k=0;
        a=s[k++];if(s[k]!=d)a+=s[k++]; ++k;
        b=s[k++];if(s[k]!=d)b+=s[k++]; ++k;
        c=s.substr(k,4);
        if(d=='/') swap(a,b);
        if(a.size()==1)a='0'+a;
        if(b.size()==1)b='0'+b;
        k=c.size();
        if(k<4)c=p[3-k]+c;
        res1=a+'.'+b+'.'+c;
        res2=b+'/'+a+'/'+c;
        fo<<res1<<' '<<res2<<'\n';
    }
}
```

Trí tuệ là một là một thuộc tính quan trọng của con người. Người thắng cử trong cuộc thi sắc đẹp phải là người đẹp và có khả năng ứng xử tốt. Người thắng trong cuộc thi tính nhẩm phải là người có trí nhớ tốt, khả năng tính toán nhanh và phải có kiến thức toán học đủ sâu.

Ứng cử viên cho chức vô địch tính nhanh năm nay nhận được n cặp số (a_i, b_i), mỗi số có không quá 18 chữ số thập phân, $0 < a_i \le b_i$, $i = 1 \div n$. Với mỗi số nguyên trong khoảng [a_i, b_i] phải thay thế nó bằng tổng các chữ số của số đó. Nếu số nhận được có nhiều hơn một chữ số thì lại thay thế thay theo cách đã nêu cho đến khi nhận được số có một chữ số. Tính tổng các số nhận được từ các số trong khoảng.

Thí sinh đã trả lời đúng tất cả!

Hãy xác định chữ số được nêu với mỗi khoảng đã cho.

Dữ liệu: Vào từ file MENTAL.INP:

- Dòng đầu tiên chứa số nguyên n ($1 \le n \le 100$),
- Dòng thứ i trong n dòng tiếp theo chứa 2 số nguyên a_i và b_i.

Kết quả: Đưa ra file văn bản MENTAL.OUT các kết quả nhận được, mỗi kết quả trên một dòng.

Ví dụ:

MENTAL.INP
3 2 6 8 12 2000 2020

MENTAL.OUT
20 23 99

WB03 Cr_1 201911019 A A XVII

Giải thuật: Tổng tiền tố .

Dễ dàng chứng minh với số nguyên **x** cho trước, quá trình lặp lại nhiều lần việc tính tổng các chữ số cho đến khi nhận được số có một chữ số tương đương với việc lấy phần dư của phép chia **x** cho 9. Nếu **x** *chia hết* cho 9 thì *kết quả cần tìm sẽ là 9*.

Ví dụ:

x = 446 → 4+4+6 = 14 → 1+4 = 5 **446%9 = 5**

x = 864 → 8+6+4 = 18 → 1+8 = 9 **864%9 = 0 → 9**

Thay các số nguyên trong khoảng **[a, b]** bằng kết quả xử lý đã cho ta được dãy số có chu trình lặp độ dài 9 và một phần không đầy đủ của chu trình ở cuối.

Số chu trình đầy đủ sẽ là **(b-a+1)/9**.

Tính chu kỳ không đầy đủ: Sử dụng mảng tổng tiền tố các giá trị trong một chu trình.

Hai trường hợp:

Tính trực tiếp từ mảng tổng tiền tố của chu kỳ

Tính phần bù

Độ phức tạp của giải thuật: O(*n*).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("mental.inp");
ofstream fo ("mental.out");
int n;
int64_t a,b,t,da,db,df,f[10] = {0,1,3,6,10,15,21,28,36,45};

int main()
{
    fi>>n;
    for(int i=0; i<n; ++i)
    {
        fi>>a>>b;
        da=a%9; if(da==0) da=9;
        db=b%9; if(db==0) db=9;
        t=(b-a+1)/9;
        if(da<=db) df=f[db]-f[da-1]; else df=45-f[da-1]+f[db];
        fo<<t*45+df<<'\n';
    }
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Cho 2 số nguyên tố ***a*** và ***b***, ***a*** ≠ ***b***.

Hãy chèn vào giữa **a** và **b** các số nguyên tố, sao cho giá trị tuyệt đối hiệu hai số đứng cạnh nhau trong dãy cũng là một số nguyên tố.

Ví dụ với ***a*** = 2 và ***b*** =7 ta có thể có dãy số thỏa mãn điều kiện đã nêu: {**2, 5, 7**}: |**2-5**| = **3**, |**5-7**| = **2**. Số lượng số cần chèn có thể là 0.

Dữ liệu: Vào từ file PRIMES.INP gồm một dòng chứa 2 số nguyên tố ***a*** và ***b***, 2 ≤ ***a***, ***b*** ≤ 10^{14}, ***a*** ≠ ***b***.

Kết quả: Đưa ra file văn bản PRIMES.OUT, dòng đầu tiên chứa số nguyên ***n*** – số lượng số trong dãy kết quả, dòng thứ 2 chứa ***n*** số trong dãy, số đầu tiên của dãy là ***a*** và số cuối cùng là ***b***. Nhếu không tồn tại cách tạo dãy kết quả thì đưa ra số **-1**. Trong trường hợp tồn tại nhiều dãy thỏa mãn – đưa ra dãy tùy chọn.

Ví dụ:

PRIMES.INP
2 17

PRIMES.OUT
3 2 19 17

WB05 Cr1 20191019 B A XVII

Giải thuật: Kỹ thuật bảng phương án .

Liệt kê các trường hợp cần kiểm tra và cách đưa ra kết quả tương ứng.

Nếu **b-a** là nguyên tố thì dãy cần đưa ra là **a b**

Nếu **b-a** là hợp số:

Hai số nguyên tố **x** và **y** có |**x-y**| cũng là nguyên tố *khi và chỉ khi* **|x-y| = 2**.

Gọi số *có thể* chèn vào *sau* **a** là **a2**, số *có thể* chèn vào *trước* **b** là **b2**.

Điều kiện cần:

- Nếu **a%3 = 1** thì **a+2** là hợp số → **a2** chỉ có thể là **a-2**, trong trường hợp ngược lại **a2 = a+2** .
- Nếu **b%3 = 1** thì **b+2** là hợp số → **b2** chỉ có thể là **b-2**, trong trường hợp ngược lại **b2 = b+2** .

Điều kiện đủ:

- Nếu **a2** – nguyên tố, **b2** – nguyên tố, **a2** > **a**, **b2** > **b** thì dãy cần đưa ra là

  a a2 2 b2 b

- Nếu **a2** – nguyên tố, **b2** – hợp số và **|a2-b|** – nguyên tố: dãy cần đưa ra là

  a a2 b

- Nếu **a2** – hợp số, **b2** – nguyên tố và **|b2-a|** - nguyên tố: dãy cần đưa ra là

  a b2 b

- Nếu **a2** và **b2** đều là nguyên tố và **|a2-b2| = 2** thì dãy cần đưa ra là

  a a2 b2 b

- Nếu **a2** và **b2** đều là nguyên tố và **a2 = b2** thì dãy cần đưa ra là

  a a2 b

- Nếu **a2** và **b2** đều là nguyên tố và **a2 < a**, **b2 < b** thì dãy cần đưa ra là

  a 2 b

Trong trường hợp còn lại: Đưa ra **-1**.

Kiểm tra số **x** là nguyên tố:

```
bool check_p(int64_t x)
{
    if(x==2) return 1;
    if((x&1)==0 || x<=1) return 0;
    for(int64_t i=3; i*i<x; i+=2)
        if(x%i==0) return 0;
    return 1;
}
```

Kiểm tra các số lẻ

Trường hợp đặc biệt

Độ phức tạp của giải thuật: $O(b^{0.5})$.

Chương trình

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

int64_t a,b,a0,a2,b0,b2,t,c;

bool check_p(int64_t x)
{
    if(x==2) return 1;
    if((x&1)==0 || x<=1) return 0;
    for(int64_t i=3; i*i<x; i+=2)
        if(x%i==0) return 0;
    return 1;
}

void print(int v)
{
    switch(v)
    {
        case 1: fo<<"2\n"<<a<<' '<<b; exit(0);
        case 2: fo<<"5\n"<<a<<' '<<a2<<" 2 "<<b2<<' '<<b;; exit(0);
        case 3: fo<<"3\n"<<a<<' '<<a2<<' '<<b; exit(0);
        case 4: fo<<"3\n"<<a<<' '<<b2<<' '<<b;; exit(0);
        case 5: fo<<"4\n"<<a<<' '<<a2<<' '<<b2<<' '<<b; exit(0);
        case 6: fo<<"3\n"<<a<<" 2 "<<b; exit(0);
        case 7: fo<<"-1"; exit(0);
    }
}
int main()
{
    fi>>a>>b;
    if(check_p(abs(a-b)))  print(1);

    if(a%3==1) a2=a-2, c=0; else a2 = a+2, c=2;
    if(b%3==1) b2=b-2; else b2 = b+2, ++c;
    bool fla =check_p(a2), flb=check_p(b2);

    if(fla && flb && c==3 ) print(2);

    if(fla && !flb && check_p(abs(a2-b)) ) print(3);

    if(!fla && flb && check_p(abs(b2-a)) ) print(4);

    if(fla && flb && (abs(a2-b2)==2) ) print(5);

    if(fla && flb && a2==b2) print(3);

    if(fla && flb && c==0) print(6);
    print(7);
}
```

Để phân loại quyền tiếp cận tài liệu mật trong kho lưu trữ mỗi nhân viên được cấp phát một khóa truy nhập riêng. Server tạo mật khẩu theo quy tắc sau: Xuất từ xâu ***s*** thực hiện một số lần các phép xử lý:

- Xóa ký tự đầu tiên của xâu,
- Xóa ký tự thứ hai của xâu,
- Xóa ký tự cuối cùng của xâu,
- Xóa ký tự trước ký tự cuối cùng của xâu.

Mỗi người sẽ nhận được một số nguyên ***k***. Khóa truy nhập cấp phát sẽ là xâu thứ tự từ điển nhỏ nhất độ dài ***k*** nhận được từ ***s*** bằng các phép biến đổi trên.

Hãy xác định khóa truy nhập.

Dữ liệu: Vào từ file KEY.INP:

- Dòng đầu tiên chứa xâu ***s*** chỉ gồm các ký tự la tinh thường, độ dài không quá 5×10^5,
- Dòng thứ 2 chứa số nguyên dương ***k***, ***k*** không vượt quá độ dài xâu ***s***.

Kết quả: Đưa ra file văn bản KEY.OUT khóa truy nhập tìm được.

Ví dụ:

KEY. INP
abacaaba **6**

KEY.OUT
aacaaa

WB06 Io20191123A B A XVII

Giải thuật: Lớp tương đương và mảng hậu tố .

Nhận xét:

- Việc thực hiện **p** lần xóa ký tự thứ 2 sau đó xóa ký tự đầu tương đương với việc xóa **p+1** lần ký tự đầu,
- Việc thực hiện **q** lần xóa ký tự trước cuối sau đó xóa ký tự cuối tương đương với việc xóa **q+1** lần ký tự cuối.

Như vậy xâu kết quả bao gồm 3 thành phần:

Gọi **s** là xâu dữ liệu vào, **`n = s.size()`**.

Phân biệt 3 trường hợp:

k = 1: Kết quả là ký tự nhỏ nhất trong s,

k = 2: Kết quả là xâu 2 ký tự **lf** và **rt**, trong đó **`s[lf]`** là ký tự nhỏ nhất trong đoạn **`s[0..n-2]`** và **`s[rt]`** là ký tự nhỏ nhất trong đoạn **`s[lf+1..n-1]`**.

k > 2:

- Xác định **lf**: **`s[lf]`** là ký tự nhỏ nhất *đầu tiên* thuộc **`s[0..n-k]`**,
- Tìm xâu con **sm** độ dài **k-2** có thứ tự từ điển nhỏ nhất trong **`s[lf+1..n-2]`**, đó là **k-2** ký tự đầu của hậu tố có thứ tự từ điển nhỏ nhất có độ dài không nhỏ hơn **k-2** của **`s[lf+1..n-2]`**,
- Xác định **rt**: **`s[rt]`** là ký tự nhỏ nhất sau đoạn tìm được ở phần trên.

Để tìm được **sm** cần trích từ **s** đoạn **sf** chứa **sm** và áp dụng giải thuật sắp xếp hậu tố của **sf** theo *lớp tương đương* với bảng chữ cái 256 ký tự.

Lưu ý: *Độ dài của **sf** có thể nhỏ hơn* 256.

Chương trình

```
#include<bits/stdc++.h>
using namespace std;
ifstream fi ("key.inp");
ofstream fo ("key.out");
int n,k,m,p,icb,ice;
int alphabet = 256; // alphabet length
vector<int> res;
string s,sf,sm,ans;
char cb,ce;

vector<int> suffix_array(string s)
{
    s.push_back('#');
    int n = s.size();
    vector<int> c(n), new_c(n), suf(n), new_suf(n), starts(max(n, alphabet));
    for (int i = 0; i < n; i++)
    {
        suf[i] = i;
        c[i] = s[i];
        starts[s[i] + 1]++;
    }
    long long q = 1ll*max(n, alphabet);
    int sum = 0;
    for (auto &v : starts) sum += v, v = sum;
    for (int l = 0; l < n; l = (l ? 2 * l : 1))
    {
        for (auto v : suf)
        {
            int pos = (v - l + n) % n;
            new_suf[starts[c[pos]]++] = pos;
        }
        int type = -1;
        long long last = -1;
        for (int i = 0; i < n; i++)
        {
            int v = new_suf[i];
            long long t = (q + c[(v + l) % n]) * c[v];
            if (last != t)
            {
                last = t;
                starts[++type] = i;
            }
            new_c[v] = type;
        }
        swap(c, new_c);
        swap(suf, new_suf);
    }
    return suf;
}

int main()
{
    fi>>s>>k;
    if(k==1){fo<<*min_element(s.begin(),s.end()); return 0;}
```

```
    if(k==2)
    {
        auto c = min_element(s.begin(), s.end() - 1);
        fo<<*c << *min_element(c + 1, s.end());
        return 0;
    }

    n = s.size(); cb = 'z'+1;
    for(int i=n-k; i>=0; --i) if(s[i]<=cb) cb=s[i], icb=i;
    sf=s.substr(icb+1,n-icb-2);  m = sf.size();
    k-=2;
    res = suffix_array(sf);

    for(int i=1; i<=m; ++i)
        if(res[i]+k<=m)
        {
            sm=sf.substr(res[i],k);
            p = res[i]+k; break;
        }
    ce = 'z'+1;
    for(int i=p+icb+1; i<n; ++i)if(s[i]<ce) ce=s[i];
    fo<<cb<<sm<<ce;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Cho dãy số nguyên không âm $A = (a_1, a_2, \ldots, a_n)$.

Hãy tính

$$\sum_{1 \le i < j < k \le n} a_i \times a_j \times a_k$$

Và đưa ra theo mô đun 10^9+7.

Dữ liệu: Vào từ file TRIPLE.INP:

- Dòng đầu tiên chứa số nguyên n ($3 \le n \le 10^6$),
- Dòng thứ 2 chứa n số nguyên $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^6$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản TRIPLE.OUT tổng tính được theo mô đun 10^9+7.

Ví dụ:

TRIPLE. INP
4 0 5 6 7

TRIPLE.OUT
210

WB07 Io20191109 J A A XVII

Giải thuật: Tổng tiền tố.

Xét các nhóm 3 (**i**, **j**, **k**) trong tổng cần tìm và cố định **j**.

Với **j** cố định tổng cần tính sẽ chứa tất cả các cặp tích a_i và a_k với **i** < **j** và **k** > **j**.

Như vậy tổng cần tính sẽ có dạng

$$\sum_{j=1}^{n}(a_j \times \sum_{i<j,k>j} a_i \times a_k)$$

Vì **i** và **k** không phụ thuộc lẫn nhau nên tổng trên có thể viết dưới dạng:

$$\sum_{j=1}^{n}(a_j \times (\sum_{i=1}^{j-1} a_i) \times (\sum_{k=j+1}^{n} a_k)$$

Gọi **B** = (b_0, b_1, . . . , b_n) – vector tổng tiền tố của **A**.

Khi đó tổng cần tìm có dạng:

$$\sum_{j=1}^{n} a_j \times b_{j-1} \times (b_n - b_j)$$

Độ phức tạp của giải thuật: O(*n*).

Tổ chức dữ liệu:

- Mảng `vector<int> a(n+1)` – Lưu các giá trị a_i,
- Mảng `vector<int64_t>b(n+1)` – Lưu tổng tiền tố của **A**.

Chương trình

```
#include<bits/stdc++.h>
using namespace std;
ifstream fi ("triple.inp");
ofstream fo ("triple.out");
int n;
const int p = 1000000007;

int main()
{
    fi>>n;
    vector<int> a(n+1);
    vector<int64_t>b(n+1);
    b[0]=0; a[0]=0;
    for(int i=1; i<=n; ++i)
    {
        fi>>a[i];
        b[i]= (b[i-1]+a[i])%p;
    }
    int64_t ans=0,t;
    for(int i =1; i<=n; ++i)
    {
        t=b[i-1]*(b[n]-b[i])%p;
        ans = (ans+t*a[i])%p;
    }

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

Trong Trại hè Tin học ngoài phần thi và bồi dưỡng kiến thức còn có các hoạt động giao lưu và giải trí. Năm nay, sau khi thi xong các đội sẽ tham gia tìm cuốn "*Ngôn ngữ lập trình Python*" được cất dấu ở một nơi nào đó. Trên đường đi các đội sẽ nhận các hướng dẫn tiếp theo nếu hiểu được các gợi ý cung cấp qua Wi-fi.

Đội của Alice đã nhanh chóng vượt qua mọi trở ngại và chỉ còn tìm khóa để hệ thống gửi về cuốn sách đang tìm. Đội nhận được xâu ký tự chỉ gồm các ký tự la tinh thường và các ký tự trong tập {**<**, **>**, **/**}. Trên máy của đội xuất hiện văn bản gợi ý:

<note> <to></to> <from></from> <heading></heading> <body></body> </note>

Sau một lúc quan sát và thảo luận mọi người hiểu từ xâu nhận được, bằng cách đổi chỗ các ký tự phải tạo ra xâu có tính chất như ví dụ đã nêu, cụ thể là với **s** là xâu không rỗng, nếu coi **<s>** là ngoặc mở và **</s>** là ngoặc đóng thì ta phải có một biểu thức ngoặc đúng.

Có thể có nhiều cách tạo biểu thức ngoặc đúng, chỉ cần đưa ra một biểu thức đúng tùy chọn. Cũng có thể từ xâu đã cho không thể tạo biểu thức ngoặc đúng, khi đó trả lời là **Impossible**.

Hãy đưa ra câu trả lời đúng để nhận phần thưởng.

Dữ liệu: Vào từ file XML.INP gồm một dòng chứa xâu độ dài không quá 10^5, chứa các ký tự như đã nêu ở trên.

Kết quả: Đưa ra file văn bản XML.OUT xâu kết quả hoặc thông báo **Impossible**.

Ví dụ:

XML.INP	XML.OUT
te<ste>st/<t>	<estt></estt>

Giải thuật: Tổ chức dữ liệu, Thống kê tần số .

Để tạo ra biểu thức đúng cần có:

- Số lượng ký tự **<** phải bằng số lượng ký tự **>**,
- Số lượng ký tự **<** phải gấp đôi số lượng ký tự **/**,
- Mỗi ký tự chữ cái phải có số lượng chẵn,
- Để đảm bảo có **s** khác rỗng trong **<s>** số lượng ký tự chữ cái phải không ít hơn số lượng ký tự **<**.

Nếu không thỏa mãn một trong số các điều kiện trên câu trả lời sẽ là **Impossible**.

Có thể dùng cấu trúc **map<char,int>** để tích lũy số liệu thống kê.

Dẫn xuất biểu thức đúng:

Đơn giản nhất là đưa liên tiếp các cặp biểu thức mở và đóng, tức là các cặp dạng **<s></s>** , các cặp đầu tiên chỉ chứa một ký tự, ở cặp cuối cùng – sử dụng các ký tự còn lại để tạo xâu **s**.

Tổ chức dữ liệu:

- Xâu **s** – Lưu dữ liệu vào,
- Bản đồ `map<char, int> cnt` – Lưu tần số xuất hiện các ký tự ,
- Xâu **lets** – Lưu các ký tự để xây dựng xâu kết quả, mỗi ký tự chỉ lưu với số lượng bằng nửa số lần xuất hiện trong **s**.

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

Lưu ý: Việc lấy và loại bỏ ký tự ở cuối xâu nhanh hơn lấy và loại bỏ ký tự ở đầu xâu.

Chương trình

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
ifstream fi ("XML.inp");
ofstream fo ("XML.out");

int main()
{
    string s;
    fi>>s;
    map<char, int> cnt;
    for (char c : s) cnt[c]++;
    int br = cnt['/'];
    if (br*2 != cnt['<']||br*2 != cnt['>'] || br*2>s.size()-br*5)
    {
        fo << "Impossible\n";
        return 0;
    }
    string lets = "";
    for (auto p : cnt)
    {
        if ('a' <= p.ff && p.ff <= 'z')
        {
            if (p.ss % 2)
            {
                fo << "Impossible\n";
                return 0;
            }
            for (int i = 0; i < p.ss / 2; ++i) lets += p.ff;
        }
    }
    reverse(lets.begin(),lets.end());
    for (int i = 0; i < br; ++i)
    {
        string cur;
        if (i < br - 1)
        {
            cur += lets.back();
            lets.pop_back();
        } else cur = lets;
        fo << "<" << cur << "></" << cur << ">";
    }
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Alice là người điều hành chung cho đám cưới của người bạn thân. Ban đầu có n người tới dự. Theo thói quen của người tổ chức, cô quan sát và đánh giá độ giao tiếp của từng người. Khách được đánh số từ 1 đến n, người thứ i có độ giao tiếp là a_i.

Trong quá trình diễn ra đám cưới có 3 loại sự kiện mà Alice đặc biệt quan tâm:

1 – Xuất hiện thêm người khách thứ j với độ giao tiếp v_j, người khách này sẽ được đánh số là $n+j$, j = 1, 2, . . .

2 – Người khách có số p_j ra về,

3 – Một ca sỹ lên biểu diễn với độ truyền cảm e_j. Sau khi thưởng thức tiết mục văn nghệ, độ giao tiếp của mọi người thay đổi. Nếu một người đang có độ giao tiếp b thì sau tiết mục văn nghệ, độ giao tiếp của họ sẽ là b^e_j (phép tính ***XOR***).

Alice rất quan tâm giữ không khí vui vẻ tự nhiên của tiệc cưới nên cứ sau mỗi sự kiện cô lại nhẩm tính tổng độ giao tiếp của những người đang có mặt.

Hãy xác định các kết quả Alice tính được.

Dữ liệu: Vào từ file WEDDING.INP:

- Dòng đầu tiên chứa 2 số nguyên n và q, trong đó q là số sự kiện ($1 \le n, q \le 10^5$),
- Dòng thứ 2 chứa n số nguyên $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$, $i = 1 \div n$),
- Mỗi dòng trong q dòng sau chưa 2 số nguyên mô tả sự kiện theo quy cách c v, trong đó c nhận giá trị từ 1 đến 3 tương ứng với các sự kiện đã nêu ở trên. Với c = 2, đảm bảo khách thứ v hiện đang có mặt.

Kết quả: Đưa ra file văn bản WEDDING.OUT đưa ra các tổng độ giao tiếp sau mỗi sự kiện, mỗi giá trị trên một dòng.

Ví dụ:

WEDDING. INP
6 5
2 3 9 5 6 6
1 3
3 5
2 2
3 2
2 7

WEDDING.OUT
34
37
31
27
23

Giải thuật: Xử lý bít.

Để quản lý tổng độ giao tiếp chỉ cần quản lý các bít 1 của các số tham gia. Như vậy, thay vì làm việc với **n** (hoặc hơn nữa) số chỉ cần quản lý 31 giá trị ghi nhận tần số xuất hiện các bít 1 ở các vị trí 0, 1, . . ., 29, 30.

Xử lý sự kiện:

Loại 3: Sau buổi biểu diễn với độ truyền cảm **e**, ở những vị trí mà **e** có giá trị 1 trong biểu diễn nhị phân giá trị ở vị trí tương ứng của mọi a_i sẽ bị đảo. Ở vị trí này tổng số lượng bít 1 mới sẽ bằng tổng số lượng bít 0 cũ. *Tổng số lượng bít 1 và 0 ở mỗi vị trí luôn bằng số người có mặt*.

Cần lưu trữ lịch sử đảo bít bằng biến kiểu int và cập nhật lịch sử khi có buổi biểu diễn.

Dựa vào bảng tần số tính lại kết quả cần đưa ra.

Loại 2: Cập nhật lại kết quả cần đưa ra dựa theo độ giao tiếp của người rời đi và lịch sử đảo bít.

Loại 1: Cập nhật lại bảng quản lý bít và số lượng người hiện có mặt. Tổng độ giao tiếp tăng một lượng bằng độ giao tiếp của người mới tới. Gắn lịch sử đảo bít với người mới.

Tổ chức dữ liệu:

- Mảng `vector<int> a` – Lưu độ giao tiếp,
- Mảng `vector<int> iv` – Lưu lịch sử đảo bít,
- Biến `int invert` – Lưu lịch sử đảo bít,
- Biến `int advent` – Lưu số người có mặt.

Độ phức tạp của giải thuật: O(n+q).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("wedding.inp");
ofstream fo ("wedding.out");
int n,m,q,t,c,v,advent,b[31]={0};
int64_t ans=0;

void add_b(int x)
{
    for(int i=0; i<31; ++i)
    {
        t = (x>>i)&1;
        b[i]+=t;
    }
}

void sub_b(int x)
{
    for(int i=0; i<31; ++i)
    {
        t = (x>>i)&1;
        b[i]-=t;
    }
}


void upd_b(int x)
{
    for(int i=0; i<31; ++i)
    {
        t = (x>>i)&1;
        if(t) b[i]=advent-b[i];
    }
    ans=0;
    for(int i=0; i<31; ++i)ans+= (int64_t)(1ll<<i)*b[i];
}

int main()
{
    fi>>n>>q;
    vector<int> a(n+1,0),iv(n+1,0);
    for(int i=1; i<=n; ++i)
    {
        fi>>a[i]; ans +=a[i];
        add_b(a[i]);
    }
    advent=n;
    int invert=0;
    for(int i=0; i<q; ++i)
```

```
    {
        fi>>c>>v;
        switch (c)
        {
            case 1: add_b(v); a.push_back(v);iv.push_back(invert);
                    ans+=v; ++advent; fo<<ans<<'\n'; break;
            case 2: m=a[v]^invert^iv[v]; sub_b(m); --advent;ans-=m;
                    fo<<ans<<'\n'; break;
            case 3: upd_b(v); invert^=v; fo<<ans<<'\n'; break;
        }
    }
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Hai bạn trong đội tuyển bóng bàn của nhà trường luyện tập chuẩn bị cho Hội khỏe Phù Đổng. Tỷ số ban đầu là **0:0**. Điểm số của người chơi tăng thêm 1 khi thắng. Vì là luyện tập nên các bạn chơi liên tục cho đến khi hết giờ và tỷ số cuối cùng là ***a:b***.

Alice được phân công ngồi ghi biên bản tỷ số trong quá trình chơi. Rất buồn chán vì không mấy hứng thú với bóng bàn, bên cạnh tỷ số Alice ghi thêm ước số chung lớn nhất của 2 số nhận được và nhẩm tính tổng các ước số chung lớn nhất nhận được.

Ví dụ

Tỷ số	*gcd*	*Tổng gcd*
0:0	0	0
1:0	1	1
2:0	2	3
2:1	1	4
2:2	2	6
2:3	1	7

Sau buổi tập Alice thở phào nhẹ nhõm và, theo thói quen chuyên môn – tự hỏi không biết tổng nhận được có giá trị nhỏ nhất là bao nhiêu nếu tỷ số cuối cùng là ***a:b*** ?

Hãy xác định giá trị tổng nhỏ nhất có thể nhận được.

Dữ liệu: Vào từ file PINGPONG.INP gồm một dòng chứa 2 số ***a*** và ***b*** ($0 \le a, b \le 10^9$).

Kết quả: Đưa ra file văn bản PINGPONG.OUT một số nguyên – tổng nhỏ nhất có thể.

Ví dụ:

PINGPONG.INP
10 10

PINGPONG.OUT
31

Giải thuật: Quy hoạch động cập nhật trễ .

Chuẩn hóa dữ liệu: Đưa về trường hợp **a** ≥ **b** để đơn giản hóa các sơ đồ cần xét,

Xét các trường hợp riêng:

- **b = 0**:

Dãy tỷ số	*ƯCLN*
(0,0)	0
(1,0)	1
(2,0)	2
. . .	. . .
(i,0)	i
. . .	. . .
(a,0)	a

Tổng GCD = 1 + 2 + . . . + (**a**-1) + **a**

$$= \frac{a \times (a+1)}{2}$$

- **b > 0** và **a** – nguyên tố:
  - Trường hợp **b < a**:

Dãy tỷ số	*ƯCLN*	
(0,0)	0	
(1,0)	1	1
(1,1)	1	
(2,1)	1	a
. . .	. . .	
(a,1)	1	
(a,2)	1	
. . .	. . .	b-1
(a,b)	1	

Tổng GCD = **a + b**

  - Trường hợp **b = a**:

Dãy tỷ số	*ƯCLN*	
(0,0)	0	
(1,0)	1	1
(1,1)	1	
(2,1)	1	a
. . .	. . .	
(a,1)	1	
(a,2)	1	
. . .	. . .	a-1
(a,a-1)	1	
(a,a)	a	a

Tổng GCD = **a + a + (a-1)**

- Trường hợp **a** – hợp số:

Gọi **p** là số nguyên tố lớn nhất nhỏ hơn **a**.

- Nếu **b < p** và từ **p** đến **a** không có số nào chia hết cho **a** thì kết quả cần tìm sẽ là **a+b**.

Dấu hiệu của điều kiện trên là **p!=b && p/b == a/b**.

Biên bản được ghi theo trình tự:

(**0,0**) → (**1,0**) → (**1,1**) → . . . → (**p,1**) → . . . → (**p,b**) → . . . →(**a,b**).

- Trường hợp ngược lại:

Áp dụng sơ đồ quy hoạch động cập nhật trễ:

Gọi $dp_{a,b}$ là tổng GCD nhỏ nhất có thể đạt được khi ghi tỷ số từ (**0,0**) đến (**a,b**).

$$dp_{a,b} = gcd(a,b) + min(dp_{a,b-1}, dp_{a-1,b})$$

Việc tính toán có thể thực hiện theo sơ đồ đệ quy. Quá trình đệ quy sẽ dừng khi gặp một trong số các trường hợp đã xét ở trên.

Do khoảng cách từ p đến a thông thường không vượt quá log*a* nên *độ phức tạp của giải thuật* sẽ là O(*b*log*a*).

Việc kiểm tra số nguyên dương **x** có phải là nguyên tố hay không cần được thực hiện dựa theo định lý Fecma nhỏ:

$$y^{x-1} \equiv 1 \ mod \ x$$

với mọi **y** nguyên dương.

Trên thực tế, chỉ cần kiểm tra với *vài giá trị* **y** *chọn ngẫu nhiên*.

Mô đun **mt19937 rnd** trong C++ cung cấp số ngẫu nhiên kiểu **int**.

y^{x-1} *mod* **x** có thể tính với độ phức tạp O(log*x*) theo sơ đồ nâng nhanh lũy thừa.

Để ngắt sớm quá trình đệ quy khi tính $\mathbf{dp_{x,y}}$ cần tổ chức bản đồ lưu ánh xạ (**u**,**v**) → **w**. Thay vì lưu trữ khóa dưới cặp giá trị nguyên có thể dùng một số 64 bít để lưu trữ cặp số nguyên.

Tổ chức dữ liệu:

- Mảng `map<int64_t,int64_t> cache` – Lưu giá trị $\mathbf{dp_{x,y}}$ theo cặp số (**x**,**y**),
- Mảng `unordered_map<int,char> prime_cache` – Đánh dấu số nguyên tố.

Xử lý:

Kiểm tra: Số nguyên **n** có phải là nguyên tố hay không?

Tính nhanh lũy thừa

```
int fpow(int a, int n, int mod)
{
    int rs = 1;
    while (n)
    {
        if (n % 2) rs = (rs * int64_t(a)) % mod;
        a = (a * int64_t(a)) % mod;
        n /= 2;
    }
    return rs;
}

bool is_prime(int n)
{
    if (n <= 1)
        return false;

    if (n % 2 == 0) return (n == 2);

    static std::mt19937 rnd;
    static std::unordered_map<int, char> prime_cache;

    char &result = prime_cache[n];
    if (result != 0)
        return result - 1;

    for (int i = 0; i < 2; ++i)
        if (fpow(1 + rnd() % (n - 1), n - 1, n) != 1)
        {
            result = 1;
            return false;
        }
    result = 2;
    return true;
}
```

n đã được kiểm tra trước đó

Dựa theo định lý Fermat nhỏ

Tìm số nguyên tố lớn nhất nhỏ hơn **n**:

```
int prime_below(int n)
{
    while (not is_prime(n)) --n;
    return n;
}
```

Chương trình

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

```
ifstream fi ("pingpong.inp");
ofstream fo ("pingpong.out");

map<int64_t, int64_t> cache;

int64_t &cget(int a, int b)
{
    return cache[int64_t(a) << int64_t(31) | b];
}

int fpow(int a, int n, int mod)
{
    int rs = 1;
    while (n)
    {
        if (n % 2) rs = (rs * int64_t(a)) % mod;
        a = (a * int64_t(a)) % mod;
        n /= 2;
    }
    return rs;
}

bool is_prime(int n)
{
    if (n <= 1)
        return false;

    if (n % 2 == 0) return (n == 2);

    static std::mt19937 rnd;
    static std::unordered_map<int, char> prime_cache;

    char &result = prime_cache[n];
    if (result != 0)
        return result - 1;

    for (int i = 0; i < 2; ++i)
        if (fpow(1 + rnd() % (n - 1), n - 1, n) != 1)
        {
            result = 1;
            return false;
        }
    result = 2;
    return true;
}

int prime_below(int n)
{
    while (not is_prime(n)) --n;
    return n;
}

int64_t get(int a, int b)
{
    if (not (a >= b))
        swap(a, b);
    if (b == 0) return a * int64_t(a + 1) / 2;
```

```
    int64_t &result = cget(a, b);
    if (result != 0)
        return result;

    if (is_prime(a))
        return result = a + int64_t(b) + int64_t(a == b ? a-1 : 0);
    if (is_prime(b))
    {
        int p = prime_below(a);
        if (p != b && p / b == a / b)
            return result = a + (int64_t)b;
    }
    return result = __gcd(a, b) + min(get(a, b-1), get(a-1, b));
}

int main()
{
    int a, b;
    fi >> a >> b;
    fo << get(a, b) << "\n";

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

Alice có nhiệm vụ tổ chức lưu trữ dữ liệu cho một cơ sở dữ liệu lớn, kích thước ***a*** GB. Dữ liệu được chia thành các khối kích thước bằng nhau, mỗi khối 1 GB. Trung tâm có ***n*** máy giống nhau, các máy đánh số từ 1 đến ***n*** và kết nối vòng tròn: Máy 1 nối với máy 2, máy 2 nối với máy 3, . . ., máy ***n*** nối với máy 1. Theo yêu cầu của giải thuật xử lý, mỗi khối dữ liệu phải được lưu trữ trên ***b*** máy nối liên tiếp nhau. Ví dụ, với ***n*** = 5 và ***b*** = 3, một khối có thể được lưu trữ trên các máy {2, 3, 4} hoặc {5, 1, 2} nhưng không được lưu trên các máy {1, 3, 4}.

Để đảm bảo không có máy nào phải lưu trữ quá nhiều dữ liệu, cần phải phân bổ việc lưu trữ, sao cho số lượng khối ở máy phải lưu trữ nhiều dữ liệu hơn cả là nhỏ nhất.

Hãy xác định số lượng khối ở máy phải lưu trữ nhiều dữ liệu nhất.

Dữ liệu: Vào từ file DATABASE.INP:

- Dòng đầu tiên chứa số nguyên ***n*** ($3 \le n \le 2\times10^9$),
- Dòng thứ 2 chứa số nguyên ***a*** ($1 \le a \le 2\times10^9$),
- Dòng thứ 3 chứa số nguyên ***b*** ($1 \le b \le n$).

Kết quả: Đưa ra file văn bản DATABASE.OUT một số nguyên – số lượng nhỏ nhất các khối ở máy lưu trữ nhiều dữ liệu nhất.

Ví dụ:

DATABASE. INP
3 4 2

DATABASE.OUT
3

WB11 MZO20191109 A A XVII

Giải thuật: Cơ sở lập trình .

Số lượng khối dữ liệu lưu trữ trên các máy là **a×b**.

Số lượng này cần phân phối ở các máy để chênh lệch ở máy chứa nhiều khối với máy chứa ít khối nhất không quá 1.

Như vậy kết quả cần tìm sẽ là $\left\lceil \frac{a \times b}{n} \right\rceil$.

Độ phức tạp của giải thuật:O(1).

Chương trình

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

int main()
{
    int64_t n,a,b;
    fi>>n>>a>>b;
    fo<<(a*b+n-1)/n;
}
```

Phương trình Diophantine là phương trình tìm nghiệm nguyên và có nhiều ứng dụng trong thực tế. Việc tìm nghiệm nguyên khó hơn nhiều so với với việc tìm nghiệm thực.

Alice tham gia vào dự án xây dựng thư viện giải các phương trình Diophantine. Nhiệm vụ của cô là xây dựng mô đun tìm 2 số tự nhiên ***x*** và ***y***, mỗi số không vượt quá 2^{62}-1, thỏa mãn điều kiện $x^2-y^2=n$, trong đó ***n*** là số nguyên không âm.

Không phải với mọi ***n*** phương trình trên đều có nghiệm. Nếu phương trình vô nghiệm, Alice đưa ra thông báo ***No***, trong trường hợp ngược lại – đưa ra thông báo ***Yes*** và ở dòng tiếp theo là 2 số ***x***, ***y*** tìm được. Phương trình có thể có nhiều nghiệm và Alice chỉ đưa ra một trong số đó.

Hãy xác định kết quả mà Alice có thể đưa ra với ***n*** cho trước.

Dữ liệu: Vào từ file DIF_SQR.INP gồm một dòng chứa số nguyên ***n*** ($0 \le n \le 2^{60}$).

Kết quả: Đưa ra file văn bản DIF_SQR.OUT thông tin Alice xác định được.

Ví dụ:

DIF_SQR.INP
3

DIF_SQR.OUT
Yes 2 1

WB12 R_Reg20200116 A A XVII

Giải thuật: Kỹ thuật bảng phương án.

Dễ dàng chứng minh được nếu **n** bằng 1, 2 hoặc 4 – phương trình vô nghiệm.

Với **n** = 0: Nghiệm của phương trình có thể là: **x** = 1 và **y** = 1.

Từ $x^2-y^2 = n$ có $(x-y)\times(x+y) = n$.

Nếu **x** và **y** không cùng chẵn hay cùng lẻ, vế trái của phương trình trên nhận giá trị lẻ.

Nếu **x** và **y** *cùng chẵn* hoặc *cùng lẻ*: *vế trái của phương trình chia hết cho* 4.

Vậy nếu n *chẵn* và *không chia hết cho 4* – phương trình *vô nghiệm*.

a) Trường hợp n lẻ

Xét hệ phương trình:

$$\begin{cases} x-y = 1 \\ x+y = n \end{cases} \Rightarrow \begin{cases} 2x = n+1 \\ x+y = n \end{cases} \Rightarrow \begin{cases} x = (n+1)/2 \\ y = (n-1)/2 \end{cases}$$

b) Trường hợp n chia hết cho 4

Xét hệ phương trình:

$$\begin{cases} x-y = 2 \\ x+y = n/2 \end{cases} \Rightarrow \begin{cases} 2x = n/2+2 \\ x+y = n/2 \end{cases} \Rightarrow \begin{cases} x = (n+4)/4 \\ y = (n-4)/4 \end{cases}$$

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int main()
{
    int64_t n,x,y;
    fi>>n;
    if(n==0){fo<<"Yes\n1 1"; return 0;}
    if(n < 3 || n == 4 ||(n%2 == 0 && n%4 != 0))
        { fo<<"No"; return 0;}
    if(n&1) x = (n+1)>>1, y = (n-1)>>1;
        else x = (n+4)>>2, y = (n-4)>>2;
    fo<<"Yes\n"<<x<<' '<<y;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Lái xe vượt quá tốc độ cho phép là nguyên nhân của nhiều tai nạn giao thông nghiêm trọng. Việc đặt các camera đo tốc độ cũng đã hạn chế một phần lỗi vượt quá tốc độ cho phép. Tuy vậy một số người đã đối phó bằng cách giảm tốc độ nơi có camera và sau đó lại phóng nhanh. Để kiểm soát tốc độ chung trên toàn tuyến đường người ta đặt camera ghi nhận thời điểm xe vào tuyến và thời điểm khi xe rời tuyến, dựa và thời gian đi trên toàn tuyến để xác định mức phạt.

Xét tuyến đường có n đoạn, đoạn thứ i có độ dài l_i và tốc độ tối đa cho phép là v_i, $i = 1 \div n$.

Gọi e là độ lớn tối đa vượt tốc độ cho phép trên tuyến, tức là giá trị lớn nhất của hiệu tốc độ đi và tốc độ cho phép ở mỗi đoạn. Nếu $e > 0$, lái xe sẽ bị phạt theo các mức như sau:

- $0 < e \le a_1$ – mức phạt là f_1 đồng,
- $a_1 < e \le a_2$ – mức phạt là f_2 đồng,
- $a_2 < e \le a_3$ – mức phạt là f_3 đồng,
- . . . . . .
- $a_{m-2} < e \le a_{m-1}$ – mức phạt là f_{m-1} đồng,
- $a_{m-1} < e$ – mức phạt là f_m đồng.

Hiện tại đang có số liệu chưa xử lý của q xe, xe thứ i vào tuyến ở thời điểm s_i và ra khỏi tuyến ở thời điểm t_i, $i = 1 \div q$.

Với mỗi xe hãy xác định mức phạt tối đa chắc chắn đúng về lỗi tốc độ.

Dữ liệu: Vào từ file PAYMENT.INP:

- Dòng đầu tiên chứa số nguyên n ($1 \le n \le 10$),
- Dòng thứ 2 chứa n số nguyên $v_1, v_2, \ldots, v_n$ ($1 \le v_i \le 10^9$, $i = 1 \div n$),
- Dòng thứ 3 chứa n số nguyên $l_1, l_2, \ldots, l_n$ ($1 \le l_i \le 10^9$, $i = 1 \div n$),
- Dòng thứ 4 chứa số nguyên m ($1 \le m \le 10^5$),
- Dòng thứ 5 chứa $m-1$ số nguyên tăng dần $a_1, a_2, \ldots, a_{m-1}$ ($1 \le a_i \le 10^9$, $i = 1 \div m-1$), nếu $m = 1$ thì dòng này rỗng,
- Dòng thứ 6 chứa m số nguyên tăng dần $f_1, f_2, \ldots, f_n$ ($1 \le f_i \le 10^9$, $i = 1 \div m$),
- Dòng thứ 7 chứa số nguyên q ($1 \le q \le 10^5$),
- Dòng thứ i trong q dòng sau chứa 2 số nguyên s_i và t_i ($1 \le s_i < t_i \le 10^9$).

Kết quả: Đưa ra file văn bản PAYMENT.OUT q số nguyên, mỗi số trên một dòng – số tiền phạt mỗi xe phải nộp. Dữ liệu đảm bảo, nếu s_i và t_i thay đổi không quá 10^{-5}, số tiền nộp phạt không thay đổi.

Ví dụ:

PAYMENT. INP	PAYMENT.OUT
3 10 20 30 400 500 600 6 1 5 10 12 16 100 300 600 800 1000 1500 3 10 100 20 70 45 100	0 800 600

WB13 R_Reg20200116 B A XVI

Giải thuật: Tìm kiếm nhị phân .

Giả thiết ở mỗi đoạn đường xe chạy vượt quá tốc độ cho phép một lượng không quá **d**, tức là ở đoạn **i** xe chạy với tốc độ không quá v_i+d. Khi đó thời gian đi hết đoạn đường thứ **i** là không quá $l_i/(v_i+d)$. Thời gian đi hết toàn tuyến không vượt quá $\sum_{i=1}^{n} \frac{l_i}{v_i+d}$.

Như vậy, có thể khẳng định việc vượt tốc độ so với mức cho phép một lượng không quá d khi

$$s + \sum_{i=1}^{n} \frac{l_i}{v_i+d} \geq t .$$

Việc xác định **d** nhỏ nhất có thể thực hiện bằng tìm kiếm nhị phân.

Với các ràng buộc đã nêu có thể tính thời gian đi hết tuyến bằng các phép tính số thực.

Tổ chức dữ liệu:

- Mảng `vector<int> v(n)` – Lưu tốc độ tối đa cho phép trên mỗi đoạn đường,
- Mảng `vector<int> l(n)` – Lưu độ dài mỗi đoạn đường,
- Mảng `vector<int> a` – Lưu các mốc tính mức phạt,
- Mảng `vector<int> f(m)` – Lưu giá trị phạt ở mỗi mức.

Độ phức tạp của giải thuật: O(*nq*log*m*).

Chương trình

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

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

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

    int m;
   fi >> m;
    vector<int> a;
    for (int i = 0; i < m - 1; i++)
    {
        int x;
        fi >> x;
        a.push_back(x);
    }
    vector<int> f(m);
    for (int i = 0; i < m; i++) fi >> f[i];

    int q;
    fi >> q;
    while (q--)
    {
        int s, t;
        fi >> s >> t;
        t = t - s;
        int bl = 0;
        int br = m - 2;
        int res = -1;
        while (bl <= br)
        {
            int mid = (bl + br) / 2;
            long double cur = 0;
            for (int i = 0; i < n; i++)
                cur += 1.0 * l[i] / (1.0 * (v[i] + a[mid]));
            if (cur > t)
            {
```

```
                res = mid;
                bl = mid + 1;
            } else br = mid - 1;
        }
        if (res == -1)
        {
            long double cur = 0;
            for (int i = 0; i < n; i++)
                cur += 1. * l[i] / (1. * (v[i]));
            if (cur <= t) fo << "0\n";
               else fo << f[0] << '\n';
        }
         else fo << f[res + 1] << '\n';
    }
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

WB15. TÍCH TRỌNG SỐ

Tên chương trình: MAXPROD.CPP

Cho mảng số $A = (a_1, a_2, \ldots, a_n)$. Trọng số của A là tổng các phần tử của nó.

Cần tách A thành 2 dãy khác rỗng $(a_1, a_2, \ldots, a_i)$ và $(a_{i+1}, a_{i+2}, \ldots, a_n)$ sao cho tích các trọng số của 2 dãy này là lớn nhất.

Hãy chỉ ra vị trí phần tử cuối cùng của dãy thứ nhất.

Dữ liệu: Vào từ file MAXPROD.INP:

- Dòng đầu tiên chứa số nguyên n ($2 \le n \le 2\times10^5$),
- Dòng thứ 2 chứa n số nguyên $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản MAXPROD.OUT một số nguyên – vị trí phần tử cuối cùng của dãy thứ nhất.

Ví dụ:

MAXPROD. INP
3 1 2 3

MAXPROD.OUT
2

WB15 R_Reg20200116 E A XVI

Giải thuật: Tổng tiền tố .

Việc tính trọng số có thể dễ dàng thực hiện bằng nhiều cách khác nhau, ví dụ thông qua việc chuẩn bị tổng tiền tố.

Gọi **i** là vị trí phần tử cuối của dãy thứ nhất, t_i – trọng số của dãy thứ nhất, **sa** – trọng số của dãy **A**.

Ta phải tìm **i** để cực đại hóa tích $t_i \times (sa - t_i)$.

Giá trị của tích có thể vượt quá khả năng biểu diễn số nguyên 64 bít. Các ngôn ngữ Java, Python cho phép làm việc với số nguyên độ dài tùy ý, nhưng cũng sẽ mất nhiều thời gian tính toán.

Tích cần xét sẽ đạt giá trị cực đại khi giá trị tuyệt đối hiệu 2 thừa số là nhỏ nhất.

Đặt **s** = **sa/2**. Gọi **f** là mảng chứa tổng tiền tố của **A**, **k** là chỉ số nhỏ nhất thỏa mãn điều kiện $f_k \geq s$.

Nghiệm cần tìm sẽ là **k** hoặc **k-1** phụ thuộc vào việc **s** gần f_k hay gần f_{k-1} hơn.

Việc tìm **k** có thực hiện bằng hàm **lower_bound**.

Tổ chức dữ liệu:

Mảng `vector<int64_t> f(n+1)` – Lưu tổng tiền tố của **A**.

Độ phức tạp của giải thuật: O(*n*).

Chương trình

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

int main()
{
    fi>>n;
    vector<int64_t> f(n+1);
    f[0]=0;
    for(int i=0; i<n; ++i)
    {
        fi>>t;
        f[i+1]=f[i]+t;
    }
    s=f[n]/2;
    ans=lower_bound(f.begin(),f.end(),s)-f.begin();
    if(f[ans]-s > s-f[ans-1]) --ans;
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Người ta dự định xây dựng một phòng nghiên cứu sản xuất vắc xin tổng hợp chống đồng thời nhiều bệnh. Khu đất đặt phòng nghiên cứu phải có hình chữ nhật kích thước ***a×b***, phòng nghiên cứu cũng có hình chữ nhật kích thước ***c×d***. Các giá trị ***a***, ***b***, ***c***, ***d*** chưa được xác định cụ thể, nhưng việc lựa chọn giá trị phải đáp ứng các yêu cầu sau:

- Độ dài các cạnh ***a***, ***b***, ***c***, ***d*** phải nguyên (tính theo đơn vị độ dài),
- Để đảm bảo an toàn tuyệt đối, các thiết bị lọc và khử trùng môi trường đòi hỏi ***a ≠ x*** và ***b ≠ x***,
- Khu đất sẽ có tường bao quanh, vì vậy phòng nghiên cứu phải nằm gọn trong khu đất, tức là ***a>c***, ***b>d***,
- Vùng trống trong khu đất phải có diện tích đúng bằng ***n***, tức là ***a×b−c×d=n***.

Để xây dựng dự án tiền khả thi chỉ mới xác định chính xác các giá trị ***n*** và ***x***. Người ta cần biết có thể có bao nhiêu cách chọn bộ giá trị ***a***, ***b***, ***c***, ***d*** thỏa mãn các yêu cầu đã nêu.

Với ***n*** và ***x*** cho trước, hãy xác định số cách chọn bộ giá trị ***a***, ***b***, ***c***, ***d*** .

Dữ liệu: Vào từ file LAYOUT.INP gồm một dòng chứa 2 số nguyên ***n*** và ***x*** ($1 \le n \le 3\,000$, $0 \le x \le 3\,000$).

Kết quả: Đưa ra file văn bản LAYOUT.OUT một số nguyên – số cách chọn xác định được.

Ví dụ:

LAYOUT.INP
5 3

LAYOUT.OUT
2

WB15 R_reg 20200117 F A XVII

Giải thuật: Lọc trạng thái và đếm cấu hình .

Nhận xét:

- Nếu **a** > **n** ta có **a×b-c×d** > **a×b-a×d** = **a×(b-d)** > **n** (vì **b** > **d**), như vậy không tồn tại bộ giá trị cần tìm thỏa mãn các điều kiện đã nêu,
- Tương tự như vậy với **b** > **n**,
- Như vậy ta chỉ cần tìm cấu hình với **a** ≤ **n** và **b** ≤ **n**,
- Có 4 tham số và một phương trình, vì vậy chỉ cần duyệt tìm 3 tham số, từ đó kiểm tra và tính tham số còn lại,
- Việc duyệt độc lập các tham số sẽ dẫn đến phải xét nhiều tổ hợp không thỏa mãn điều kiện đã cho với các tham số.

Cách duyệt tối ưu: Duyệt tính phần để trống.

Thay vì duyệt **a** sẽ duyệt theo **a-c**, thay duyệt **b** bằng duyệt **b-d**.

Việc duyệt **b-d** sẽ dừng khi **(a-c)×(b-d)** ≥ **n**.

Như vậy số lần duyệt sẽ là $\frac{n}{1}+\frac{n}{2}+\frac{n}{3}+...+\frac{n}{n} = n\times(1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{n})=$ $n\log n$.

Với **c**: duyệt từ 1 trở đi, chừng nào còn có

$$\mathbf{left = n-(a-c)\times(b-d)-c\times(b-d)} > 0$$

Số lần duyệt sẽ còn được rút gọn nhiều hơn nữa.

Với mỗi **c** giá trị **left** phải chia hết cho **a-c**, khi đó sẽ tồn tại **d**.

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

Chương trình

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

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

  int ans = 0;
  for (int a_c = 1; a_c < n; ++a_c)
    for (int b_d = 1; a_c * b_d < n; ++b_d)
    {
      auto rest = n - a_c * b_d;
      for (int c = 1; c * b_d < rest; ++c)
      {
        auto left = rest - c * b_d;
        if (left % a_c) continue;
        auto d = left / a_c;
        if (c + a_c == x || d + b_d == x) continue;
        ans++;
      }
    }
  fo << ans << '\n';

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

Hệ thống tiền trong một nước có các tờ bạc với mệnh giá $a_1, a_2, \ldots, a_n$, trong đó $a_1 = 1$, $a_i > a_{i-1}$, $i = 2, 3, \ldots, n$. Máy rút tiền xác định các tờ bạc cần đưa ra theo giải thuật tham lam: Để trả số tiền x máy sẽ chọn tờ a_i có mệnh giá cao nhất nhỏ hơn hoặc bằng x, đưa ra tờ bạc đó và lặp lại với số tiền $x-a_i$. Do $a_1 = 1$ nên máy bao giờ cũng đưa ra đủ số tiền cần rút.

Ngân hàng phát hành q loại thẻ, loại thẻ thứ j có khả năng rút tối đa một lần là b_j, $j = 1 \div q$.

Với mỗi loại thẻ hãy chỉ ra số tiền cần rút để nhận được nhiều tờ bạc nhất.

Dữ liệu: Vào từ file ATM.INP:

- Dòng đầu tiên chứa số nguyên n ($1 \le n \le 2\times10^5$),
- Dòng thứ 2 chứa n số nguyên tăng dần $a_1, a_2, \ldots, a_n$ ($1 = a_1 < a_2 < \ldots < a_n \le 10^{18}$),
- Dòng thứ 3 chứa số nguyên q ($1 \le q \le 2\times10^5$),
- Dòng thứ j trong q dòng sau chứa số nguyên b_j ($1 \le b_j \le 10^{18}$).

Kết quả: Đưa ra file văn bản ATM.OUT với mỗi loại thẻ đưa ra trên một dòng số tiền cần rút để nhận được nhiều tờ bạc nhất và số lượng tờ bạc tương ứng. Nếu có nhiều cách rút thỏa mãn yêu cầu cần tìm – đưa ra cách rút tùy chọn.

Ví dụ:

ATM. INP	ATM.OUT
4 1 5 10 50 3 2 8 50	2 2 8 4 49 9

WB15 R_Reg20200116 E A XVII

Giải thuật: Quy hoạch động.

Xét số tiền cần rút **x** thỏa mãn điều kiện $a_i \le x < a_{i+1}$. Với **x** cần dùng một hoặc nhiều tờ mệnh giá a_i.

Ký hiệu:

- mx_i – số tờ mệnh giá a_i cần dùng để chi trả khi số tiền cần rút là **x** lớn nhất thỏa mãn điều kiện đã nêu,
- $start_i$ – tổng số tiền đã trả *với nhiều nhất các tờ tiền* mệnh giá nhỏ hơn a_i,
- $pref_i$ – số tờ tiền mệnh giá nhỏ hơn a_i đã dùng.

Ta có:

- $start_0 = pref_0 = 0$,
- $mx_i = (a_{i+1}-1 - start_i)/a_i$,
- $start_{i+1} = start_i + mx_i*a_i$,
- $pref_{i+1} = pref_i + mx_i$.

Tính số tiền không vượt quá **x** và số tờ bạc nhiều nhất cần chi trả:

- Xác định $i = max\{i \mid start_j \le x\}$: Các mệnh giá cần sử dụng,
- Số tiền cần thanh toán bằng các tờ bạc mệnh giá **ai**:

$$y = (x-start_i) - (x-start_i)\%a_i$$

- Số tiền cần phải thanh toán bằng nhiều tờ bạc nhất là $start_i+y$,
- Số tờ bạc cần sử dụng: $pref_i + y/a_i$.

Độ phức tạp của giải thuật: $max\{O(n), O(q*\log n)\}$

Chương trình

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

int main()
{
  int n;
  fi >> n;
  vector<int64_t> a(n);
  for (auto& x : a) fi>>x;
  vector<int64_t> mx(n - 1),start(n),pref(n);
  start[0]=pref[0]=0;
  for (int i = 0; i + 1 < n; ++i)
  {
    mx[i] = (a[i + 1] - 1 - start[i]) / a[i];
    start[i + 1] = start[i] + mx[i] * a[i];
    pref[i + 1] = pref[i] + mx[i];
  }

  auto solve = [&](int64_t x)
  {
    auto i = prev(upper_bound(start.begin(), start.end(), x)) -
start.begin();
    auto coins = pref[i];
    x -= start[i];
    x -= (x % a[i]);
    auto new_x = start[i] + x;
    coins += x / a[i];
    return make_pair(new_x, coins);
  };

  int q;
  fi >> q;
  while (q--)
  {
    int64_t x;
    fi >> x;
    auto res = solve(x);
    fo << res.first << ' ' << res.second << '\n';
  }
   fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Để xây dựng một tượng đài người ta khai thác được 3 tảng đá nguyên khối hình hộp chữ nhật, tảng thứ i có kích thước đáy là $a_i \times b_i$, i = 1, 2, 3. Các tảng đá được vận chuyển về nơi xây dựng bằng đường thủy.

Xà lan chuyên chở phải được cải tạo lại để làm thành một khoang hình chữ nhật chứa đồng thời cả 3 tảng đá. Các tảng đá được đặt để cạnh đáy song song với cạnh của khoang. Các tảng đá không được đè lên nhau, nhưng có thể có cạnh đáy hoặc đỉnh tiếp xúc nhau.

Hãy xác định diện tích nhỏ nhất của khoang đáp ứng yêu cầu chuyên chở.

Dữ liệu: Vào từ file văn bản BARGE.INP gồm 6 dòng, mỗi dòng chứa một số nguyên lần lượt xác định các số a_1, b_1, a_2, b_2, a_3 và b_3 ($0 < a_i, b_i \le 10^4$, i = 1, 2, 3).

Kết quả: Đưa ra file văn bản BARGE.OUT một số nguyên – diện tích nhỏ nhất của khoang chứa.

Ví dụ:

BARGE. INP
4 10 5 11 12 3

BARGE.OUT
144

WB15 Io20200314 A A XVII

Giải thuật: Duyệt vét cạn .

Các tảng đá có thể xoay 90^0 khi đặt vào xà lan.

Tồn tại hai cách bố trí đá trên xà lan:

Diện tích tương đương

Ở cách thứ I diện tích nhỏ nhất của khoang chứa sẽ là:

$$max(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)\times(\mathbf{a}_1+\mathbf{a}_2+\mathbf{a}_3)$$

Ở cách thứ II diện tích nhỏ nhất của khoang chứa sẽ là:

$$max(\mathbf{a}_1+\mathbf{a}_3, \mathbf{a}_2)\times max(\mathbf{b}_1+\mathbf{b}_2, \mathbf{b}_3)$$

Vấn đề còn lại chỉ là duyệt các khả năng xoay đá, đổi vai trò các tảng, tính diện tích khoang trong hai cách bố trí và chọn diện tích nhỏ nhất.

Số lượng trường hợp cần xét sẽ là 8×6×2 = 96.

Tổ chức dữ liệu:

Để thuận tiện cho việc tổ chức duyệt, kích thước đáy mỗi tảng đá được lưu trữ dưới dạng cặp dữ liệu nguyên và toàn bộ dữ liệu vào được lưu trữ dưới dạng véc tơ 3 phần tử, mỗi phần tử - một cặp giá trị nguyên.

Độ phức tạp: O(1).

Chương trình

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
typedef pair<int64_t, int64_t> pll;
ifstream fi ("barge.inp");
ofstream fo ("barge.out");

int main()
{
    vector<pll> a(3);
    for (int i = 0; i < 3; i++)
        fi >> a[i].ff >> a[i].ss;

    int64_t ans = LLONG_MAX;
    auto relax = [&](int64_t value, int64_t m = 1)
    { ans = min(ans, value * m);};

    auto solve = [&](vector<pll> a)
    {
        relax(max({a[0].ss,a[1].ss,a[2].ss})*
                        (a[0].ff+a[1].ff+a[2].ff));
        if (a[0].ff >= a[1].ff || a[2].ss <= a[0].ss)
            relax(max(a[0].ss + a[1].ss, a[2].ss),
                          max(a[0].ff + a[2].ff, a[1].ff));
    };
    sort(a.begin(), a.end());
    do
    {
        for (int mask = 0; mask < (1 << 3); mask++)
        {
            auto cur = a;
            for (int i = 0; i < 3; i++)
                if (mask & (1 << i))
                    swap(cur[i].ff, cur[i].ss);
            solve(cur);
        }
    } while (next_permutation(a.begin(), a.end()));
    fo << ans << endl;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Đoạn đường thử nghiệm các phương tiện giao thông mới được chia thành các đoạn bằng nhau, các đoạn được đánh số từ 1 trở đi.

Hiện nay có 2 phương tiện mới đang được thử nghiệm: Tàu đệm từ và Xe lửa trong đường ống chân không (Tàu chân không).

Ban đầu các đoạn từ ***a*** đến ***b*** (kể cả ***a*** và ***b***, $1 \le a \le b$) được thiết kế để thử tàu đệm từ, còn các đoạn từ ***c*** tới ***d*** (kể các ***c*** và ***d***, $c \le d$) – thử tàu chân không. Hai tuyến thử nghiệm không có đoạn chung, tức là $b < c$.

Sau giai đoạn thử nghiệm thứ I người ta thấy có thể giữ nguyên hoặc rút ngắn đoạn đường thử nghiệm tàu đệm từ. Cần giữ nguyên hoặc kéo dài đoạn đường thử nghiệm tàu chân không. Để giảm chi phí thiết kế lại, mỗi đường thử nghiệm vẫn chứa một số nguyên các đoạn, đường mới thử nghiệm tàu đệm từ phải là các đoạn liên tiếp nằm gọn trong đường cũ, đường mới thử nghiệm tàu chân không phải là các đoạn liên tiếp bao đường cũ, hai tuyến thử nghiệm không có đoạn chung và tổng số đoạn đường dành cho hai tuyến thử nghiệm vẫn như cũ.

Hãy xác định có bao nhiêu cách chọn các tuyến cho giai đoạn thử nghiệm tiếp theo. Hai cách chọn gọi là khác nhau nếu tồn tại ít nhất một đoạn thuộc một tuyến đường ở cách này và không thuộc tuyến đường đó ở cách khác.

Dữ liệu: Vào từ file văn bản SEGMENTS.INP chứa các số nguyên ***a***, ***b***, ***c***, ***d***, mỗi số trên một dòng, $1 \le a \le b < c \le d \le 10^5$.

Kết quả: Đưa ra file văn bản SEGMENTS.OUT một số nguyên – số cách chọn cho giai đoạn tiếp sau.

Ví dụ:

SEGMENTS. INP
1 2 4 5

SEGMENTS.OUT
5

WB19 Io20200314 B A XVII

Giải thuật: Phân tích mô hình toán học.

Để đơn giản hóa việc dẫn xuất các công thức cần xét ta tịnh tiến mốc đánh số để có **a** = 0: **b-=a**, **c-=a**, **d-=a**.

Mỗi cách lựa chọn hợp lệ 2 tuyến đường thử nghiệm mới là một cấu hình,

Việc đếm các cấu hình được thực hiện theo nguyên lý "*Cá kể đầu, rau kể mớ*".

Số lượng cấu hình mới phụ thuộc vào các tham số:

- Số cách chọn độ dài tuyến đường thử nghiệm Tàu đệm từ (Tuyến I),
- Vị trí bắt đầu của Tuyến I,
- Vị trí bắt đầu có thể của tuyến đường thử nghiệm Tàu chân không (Tuyến II).

Gọi độ dài mới của Tuyến I là **len1**.

len1 nhận giá trị từ **1** đến **b+1**.

Với mỗi đoạn độ dài **len1** có **Δ = b+1-len1** cách chọn. Độ dài tuyến thứ II cần tăng thêm một lượng là **Δ**. Việc kéo dài tuyến II có thể thực hiện bằng cách tăng **d** hoặc giảm **c** hay đồng thời tăng **d** và giảm **c**. Cấu hình tuyến II hoàn toàn xác định theo điểm đầu **c** của tuyến.

Như vậy, có bao nhiêu cách chọn **c** cho tuyến II mới sẽ có bấy nhiêu cấu hình khác nhau ứng với mỗi **len1**.

Đặt **L = c-Δ**.

Có 2 trường hợp xảy ra:

- **L > b**:

Khi đó số cấu hình cần tìm sẽ tăng một lượng là (**Δ+1**)× (**Δ+1**).

- **L ≤ b**:

Với mỗi điểm đầu mới của tuyến II trong khoảng từ **c** tới **b** có **Δ+1** cách chọn tuyến đầu độ dài **len1**.

Với mỗi điểm đầu **b-t** của tuyến II mới, **t** = 0, 1, . . ., **b-(c-Δ)**, có **Δ-t** cách chọn tuyến I độ dài **len1**. Tổng số cấu hình mới sẽ là một cấp số cộng **b-(c-Δ)+1** số hạng. Dễ dàng dẫn xuất số lượng cấu hình khác nhau trong trường hợp này.

Kết quả cần tìm sẽ là tổng các cấu hình có thể chọn với mọi **len1** trong phạm vi đã nêu ở trên.

Độ phức tạp giải thuật: O(*n*), trong đó *n* = **b-a+1**.

Nhận xét:

Với giải thuật đã xét, giá trị d không tham gia vào việc dẫn xuất kết quả,

Có thể triển khai công thức xác định kết quả cần tìm, loại bỏ tham số len1 và có giải thuật độ phức tạp O(1), nhưng việc này đòi hỏi quá nhiều thời gian và công sức xử lý một cách không cần thiết trước khi lập trình!

Chương trình

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

int main()
{
    ll a, b, c, d;
    fi>>a>>b>>c>>d;
    b-=a; c-=a; d-=a;
    ll ans = 0;
    for (ll len1 = b + 1; len1 > 0; --len1)
    {
        ll delta = (b + 1) - len1;
        ll L = c-delta;
        if (L > b) ans +=  (delta + 1) * (delta + 1);
            else
            {
                ans += (c - b) * (delta + 1);
                ll last = b - L + 1;
                ans += (2*delta  - last + 1) * last / 2;
            }
    }
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Người ta khai quật được n bức tượng cổ, bức tượng thứ i có độ cao a_i, $i = 1 \div n$. Trước khi chuyển về viện bảo tàng, các bức tượng được chụp hình để làm tư liệu phục chế khi cần thiết. Để tránh phải lưu trữ quá nhiều, nhưng các ảnh cũng phải đủ rõ ràng, Trưởng nhóm yêu cầu:

Trên mỗi bức ảnh có không quá 3 tượng,

- Nếu ảnh chụp đồng thời 3 tượng thì chênh lệch độ cao giữa 2 tượng bất kỳ trong đó không quá 10,
- Nếu ảnh chụp đồng thời 2 tượng thì chênh lệch độ cao giữa 2 tượng là không quá 20,
- Bức tượng được chụp một mình có thể có độ cao bất kỳ.

Hãy xác định số lượng ảnh ít nhất cần chụp.

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

- Dòng đầu tiên chứa một số nguyên n ($1 \le n \le 1000$),
- Dòng thứ 2 chứa n số nguyên $a_1, a_2, \ldots, a_n$ ($100 \le a_i \le 1000$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản PHOTO.OUT một số nguyên – số lượng ảnh ít nhất cần chụp.

Ví dụ:

PHOTO. INP
6 100 210 250 255 220 260

PHOTO.OUT
3

WB20 Io202000404 B A XVII

Giải thuật: Quy hoạch động.

Sắp xếp các a_i theo thứ tự tăng dần.

Gọi x_i là số tượng trong số từ 0 đến i-1 chụp chung với tượng thứ i, x_i có thể nhận các giá trị 0, 1 hoặc 2, $x_0 = 0$,

Gọi y_i – số lượng ảnh ít nhất chụp các tượng từ 0 đến i, $y_0 = 1$.

Công thức xác định x_i, y_i, $i = 1 \div n-1$:

- Nếu $i > 1$, $a_i - a_{i-2} \le 10$ và $x_i < 2 \rightarrow x_i = x_{i-1}+1$, $y_i = y_{i-1}$,
- Nếu $a_i - a_{i-1} \le 20$ và $x_{i-1} = 0 \rightarrow x_i = 1$, $y_i = y_{i-1}$,
- Trong các trường hợp còn lại: $x_i = 0$, $y_i = y_{i-1}+1$.

Kết quả cần tìm: y_{n-1}.

Tổ chức dữ liệu:

- Mảng `vector<int>a(n)` – Lưu dữ liệu vào,
- Các mảng `vector<int> x(n),y(n)` – phục vụ sơ đồ quy hoạch động.

Độ phức tạp: O(*n*lg*n*).

Chương trình

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

int main()
{
    int n;
    fi>>n;
    vector<int>a(n),x(n),y(n);
    for(int &i:a) fi>>i;
    sort(a.begin(),a.end());
    x[0]=0; y[0]=1;
    for(int i=1; i<n; ++i)
    {
        if(i>1 && a[i]-a[i-2]<=10 && x[i-1]<2)
        {
            x[i]=x[i-1]+1; y[i]=y[i-1]; continue;
        }
        if(a[i]-a[i-1]<=20 && x[i-1]==0)
        {
            x[i]=1; y[i]=y[i-1];continue;
        }
        x[i]=0; y[i]=y[i-1]+1;
    }

    fo<<y[n-1];
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Cho số nguyên dương ***n***. Các số nguyên ***a***, ***b***, ***c*** tạo thành nhóm 3 kỳ diệu nếu thỏa mãn các điều kiện:

- $1 \le a < b < c \le n$,
- ***a×b***, ***a×c*** và ***b×c*** đều là các số chính phương.

Ví dụ, với ***n*** = 10 các số 1, 4, 9 tạo thành một nhóm 3 kỳ diệu.

Với ***n*** đã cho hãy xác định số lượng các nhóm 3 kỳ diệu.

Dữ liệu: Vào từ file văn bản TROIKA.INP gồm một dòng chứa số nguyên ***n*** ($1 \le n \le 2\times10^5$).

Kết quả: Đưa ra file văn bản TROIKA.OUT một số nguyên – số lượng các nhóm 3 kỳ diệu tính được.

Ví dụ:

TROIKA.INP
20

TROIKA.OUT
5

WB20 Io202000404 B A XVII

Giải thuật: Cơ sở lập trình .

Nhận xét:

- Nếu **a**, **b**, **c** là 3 số chính phương khác nhau từng đôi một và không vượt quá **n** thì chúng tạo thành một nhóm 3 kỳ diệu,
- Nếu (**a**, **b**, **c**) là một nhóm 3 kỳ diệu thì (**k×a**, **k×b**, **k×c**) cũng là một nhóm 3 kỳ diệu với **k** nguyên dương và **k×c** ≤ **n**.

Với **k** = 1, số lượng các số chính phương không vượt quá **n** là **q** = $\lfloor\sqrt{n}\rfloor$.

Với **k** > 1 và **k** *không phải là bội của số chính phương,* số lượng các số tham gia tạo nhóm 3 kỳ diệu sẽ là **q** = ⌊**sqrt(n/k)** ⌋.

Với mỗi **q** nhận được, tổ hợp chập 3 của **q** sẽ là số lượng nhóm 3 kỳ diệu.

Tổng các tổ hợp chập 3 với mọi **q** > 2 nhận được sẽ là kết quả cần tìm.

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

Chương trình

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

int main()
{
    fi>>n;
    k=1; p=2; s.insert(p*p);
    while(n/k>=9)
    {
        flg=true;
        if(k==p*p){++k; ++p; s.insert(p*p);continue;}
        for(int t:s)if(k%t==0){++k; flg=false; break;}
        if(!flg) continue;
        q=floor(sqrt(n/k))+1;
        if(q*q>n/k)--q;
        ans+=(int64_t)q*(q-1)*(q-2)/6;
        ++k;
    }
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Các nhà khoa học đang lên dự án đảm bảo thực phẩm cho chuyến thám hiểm sao Hỏa. Theo dự kiến kho dự trữ sẽ có ***n*** loại thực phẩm, đánh số từ 1 đến ***n***, loại thực phẩm thứ ***i*** sẽ có k_i hộp khẩu phần và giữ được t_i ngày, ***i*** = 1 ÷ ***n***. Hết thời hạn lưu trữ, nếu còn, thực phẩm đó vẫn phải vất bỏ.

Đoàn thám hiểm sẽ có ***c*** người. Mỗi ngày người ta chọn và ăn ***c*** hộp loại tùy chọn. Trong ngày những người trong đoàn có thể ăn các món giống nhau hoặc khác nhau.

Để đánh giá độ dư thừa trong phương án chuẩn bị thực phẩm người ta cần biết nhiều nhất có bao nhiêu loại thực phẩm đã được ăn hết và đó là những loại nào.

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

- Dòng đầu tiên chứa 2 số nguyên ***n*** và ***c*** ($1 \le n \le 2\times10^5$, $1 \le c \le 10^9$),
- Dòng thứ ***i*** trong ***n*** dòng sau chứa 2 số nguyên t_i và k_i ($1 \le t_i \le 10^9$, $1 \le k_i \le 10^{18}$).

Kết quả: Đưa ra file văn bản PRODUCTS.OUT, dòng đầu tiên chứa một số nguyên xác số lượng nhiều nhất các loại thực phẩm được ăn hết, dòng thứ 2 chứa các số nguyên xác định những thực phẩm được ăn hết, trình tự đưa ra trong danh sách là tùy chọn.

Ví dụ:

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

WB21 Io20200314 C A XVII

Giải thuật: Kỹ thuật tổ chức dữ liệu.

Sắp xếp các cặp dữ liệu (t_i, k_i) theo chiều giảm dần của t_i và xét các phần tử của dãy đã sắp xếp theo trình tự từ 1 đến **n**. Khi đó các hộp khẩu phần còn lại của các loại thực phẩm đã xét đều có dự trữ thời gian bảo quản *không ít hơn* so với loại thực phẩm đang xét. Như vậy, với những loại thực phẩm đứng trước loại đang xét cần ưu tiên xử lý loại có số lượng còn lại ít nhất.

Để nhanh chóng tìm được loại thực phẩm cần ưu tiên xử lý có thể dùng cấu trúc dữ liệu tập hợp.

Do có yêu cầu dẫn xuất loại thực phẩm được dùng hết nên phần tử của dãy được sắp xếp cần phải chứa cả số thứ tự của loại thực phẩm tương ứng, nói cách khác, ta phải làm việc với nhóm dữ liệu (t_i, k_i, **i**).

Tương tự như trên, trong tập hợp phục vụ tìm loại thực phẩm cần ưu tiên xử lý, ngoài số lượng hộp còn lại, cần lưu trữ thêm loại thực phẩm.

Việc tính số hộp khẩu phần sử dụng dựa trên cơ sở là trong khoảng thời gian từ t_i đến t_{i+1} người ta sử dụng hết $c \times (t_i - t_{i+1})$ hộp.

Tổ chức dữ liệu:

- Mảng `vector<tuple<int, ll, int>> v` – Lưu trữ thông tin về dữ liệu vào,
- Tập hợp `set<pair<ll, int>> q` – Phục vụ tìm loại thức phẩm có số lượng khẩu phần còn lại ít nhất trong số các loại đã xét qua,
- Mảng `vector<int> ans` – Lưu kết quả cần dẫn xuất.

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

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("products.inp");
ofstream fo ("products.out");
typedef long long ll;
int n, t;
ll c, k;
vector<tuple<int, ll, int>> v;

int main()
{
    fi >> n >> c;
    for (int i = 0; i < n; i++)
    {
        fi >> t >> k;
        v.push_back(make_tuple(t, k, i));
    }
    sort(v.rbegin(), v.rend());
    v.push_back(make_tuple(0,0,0));
    set<pair<ll, int>> q;
    vector<int> ans;
    for (int i = 0; i < n; i++)
    {
        q.insert(make_pair(get<1>(v[i]), get<2>(v[i])));
        ll cur = get<0>(v[i]) -  get<0>(v[i + 1]);
        cur *= c;
        while (cur > 0 && !q.empty())
        {
            if (q.begin()->first <= cur)
            {
                cur -= q.begin()->first;
                ans.push_back(q.begin()->second);
                q.erase(q.begin());
            }
            else
            {
                pair<ll, int> t = *q.begin();
                q.erase(t);
                q.insert(make_pair(t.first - cur, t.second));
                cur = 0;
            }
        }
    }
    fo << ans.size() << "\n";
    for (int i : ans) fo << i + 1 << " ";
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Có ***n*** thị trấn nối với nhau bởi ***n-1*** đường hai chiều, đảm bảo có đường đi giữa 2 thị trấn bất kỳ. Mỗi thị trấn có một trạm bưu điện. Các bưu cục được đánh số từ 1 đến ***n***, bưu cục trung tâm có số là 1. Hai thị trấn có đường nối trực tiếp được gọi là liền kề. Mỗi thị trấn có không quá 12 thị trấn liền kề.

Hiện đang có ***m*** yêu cầu chuyển, yêu cầu thứ ***i*** được xác định bởi 2 số nguyên a_i và b_i – cần chuyển bưu kiện từ bưu cục a_i tới bưu cục b_i, $i = 1 \div m$.

Thư từ và bưu kiện được chuyển giữa các bưu cục bằng robot đặc biệt, thiết kế riêng cho ngành bưu điện. Hoạt động của robot được lập trình theo các quy tắc sau:

- Xuất phát từ bưu cục trung tâm robot lần lượt đi đến các bưu cục khác trong mạng, trong quá trình di chuyển có thể mang theo số lượng không hạn chế thư từ và bưu kiện,
- Khi lần đầu đến một thị trấn nào đó, robot sẽ ghé qua bưu cục, để lại thư từ và hàng hóa đang mang theo cần chuyển tới bưu cục này và tiếp nhận mọi thư từ, bưu kiện cần chuyển đi từ bưu cục này,
- Nếu có các thị trấn liền kề chưa tới robot sẽ đi tiếp tới một trong số đó,
- Trường hợp mọi thị trấn liền kề đã được tới: Nếu robot không ở bưu cục trung tâm thì robot sẽ chuyển tới thị trấn gần thị trấn có bưu cục trung tâm nhất, nếu robot đang ở thị trấn có bưu cục trung tâm – kết thúc chuyến đi.

Lưu ý là robot chỉ ghé vào mỗi bưu cục đúng một lần, khi tới thị trấn tương ứng lần đầu.
Hãy xác định số lượng đường đi khác nhau có thể thực hiện để chuyển hết thư từ và bưu kiện tới địa chỉ yêu cầu và đưa ra theo mô đun 10^9+7. Hai đường đi gọi là khác nhau nếu tồn tại ít nhất một cặp thị trấn có trình tự thăm khác nhau trong 2 đường.

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

- Dòng đầu tiên chứa một số nguyên ***n*** và ***m*** ($2 \le n \le 10^5$, $1 \le m \le 3\times10^5$),
- Mỗi dòng trong ***n-1*** dòng sau chứa 2 số nguyên ***u*** và ***v*** xác định hai thị trấn liền kề, $1 \le u, v \le n$, $u \ne v$,
- Dòng thứ ***i*** trong ***m*** dòng tiếp theo chứa 2 số nguyên a_i và b_i $1 \le a_i, b_i \le n$, $a_i \ne b_i$.

Kết quả: Đưa ra file văn bản POSTROBOT.OUT một số nguyên – số lượng đường đi khác nhau theo mô đun 10^9+7.

Ví dụ:

POSTROBOT. INP
6 2 1 2 1 3 1 4 2 5 2 6 5 3 4 6

POSTROBOT.OUT
5

Giải thuật: Các giải thuật về đồ thị và quy hoạch động.

Điều kiện đủ để số đường đi bằng 0: Tồn tại **i** để b_i là nút cha của a_i. Trong trường hợp này, với mọi cách đi, đỉnh b_i được thăm trước a_i vì vậy không thể lấy bưu kiện từ a_i để chuyển tới b_i.

Kiểm tra điều kiện đủ:

- Gọi **k** là số nguyên nhỏ nhất thỏa mãn điều kiện $2^k \geq n$,
- Xác định độ sâu $depth_i$ của nút **i** thuộc cây,
- Gọi việc di chuyển từ một nút về nút cha trực tiếp của nó là một bước, với mỗi nút ghi nhận nút tới sau 1, 2, 4, . . ., 2^k bước,
- Từ bảng ghi nhận này dễ dàng với độ phức tạp O(log*n*) xác định **a** có phải là cha của **b** hay không trong cặp đỉnh (**a**, **b**) của cây.

Tính số lượng đường đi:

- Với mỗi nút đánh số các nút con trực tiếp của nó,
- Xét cặp đỉnh (**a**, **b**) xác định yêu cầu chuyển bưu kiện từ **a** tới **b**,
- Gọi **c** – cha chung gần nhất của **a** và **b**, **a′** – nút con trực tiếp của **c** trên đường đi từ **c** tới **a**, **b′** – nút con trực tiếp của **c** trên đường đi từ **c** tới **b**,
- Ghi nhận cặp chỉ số của **a′** và **b′** vào danh sách tương ứng với **c**,
- Với mỗi **c** và cặp chỉ số: tính số lượng đường đi có thể,
- Kết quả cần tìm là tích các giá trị nhận được ở bước trên.

Tính số đường đi từ mỗi nút:

Gọi deg_i là bậc (số đỉnh kề) của nút **i**,

Nếu **i** là nút gốc của cây thì số đường đi khác nhau (chưa tính ràng buộc) sẽ là $2^{deg[i]}$,

Trường hợp **i** không phải là nút gốc của cây thì số đường đi khác nhau (chưa tính ràng buộc) sẽ là $2^{deg[i]-1}$,

Mỗi đường đi tương ứng với một số nhị phân **j** có deg_i hoặc deg_i-1 bít,

Các bít 1 của **j** đánh dấu những đỉnh mà đường đi đã qua,

Gọi d_j là số lượng cách duyệt khác nhau để tạo ra đường đi **j**,

Với mỗi bít **k** được đánh dấu trong **j** d_j bằng tổng các đường đi chưa qua đỉnh thứ **k** và **k** không thuộc cặp đỉnh (**a′**, **b′**),

d_j với **j** có tất cả các bít đều bằng 1 là số cách lựa chọn đường đi hợp lệ qua đỉnh **i** đang xét,

Tích (theo mô đun 10^9+7)các cách lựa chọn đường đi hợp lệ qua mỗi đỉnh của cây là kết quả cần tìm.

Tổ chức dữ liệu:

- Mảng `vector<vector<int>> graph(n)` – lưu danh sách đỉnh kề của cây,
- Mảng `vector<vector<int>> binup(bp, vector<int>(n))` – phục vụ tìm kiếm nhị phân đỉnh cha chung gần nhất,
- Mảng `vector<int> depth(n)` – xác định độ sâu mỗi đỉnh,
- Mảng `vector<int> ind_child(n)` – lưu chỉ số danh sách đỉnh con của mỗi đỉnh,
- Mảng `vector<vector<pii>> here(n)` – lưu cặp chỉ số đỉnh ứng với từng đường đi bắt buộc để chuyển bưu kiện,
- Mảng `vector<int> matr(12)` – ghi nhận đỉnh xác định các đường đi bắt buộc từ mỗi đỉnh,
- Mảng `vector<int> d(1 << 12)` – phục vụ quy hoạch động tính số lượng đường đi có thể chọn.

Xử lý:

Ghi nhận cấu trúc cây:

```
int n, m;
fi >> n >> m;
vector<vector<int>> graph(n);
for (int i = 1; i < n; ++i)
{
    int a, b;
    fi >> a >> b;
    --a; --b;
    graph[a].push_back(b);
    graph[b].push_back(a);
}
```

Đánh số đỉnh từ 0

Ghi nhận danh sách đỉnh kề

Hàm xác định các tham số phục vụ tìm kiếm:

Hàm được xây dựng bên trong chương trình chính để có thể sử dụng các biến khai báo cục bộ trước đó, vòng tránh việc truyền nhiều tham số dùng trong sơ đồ đệ quy.

Khai báo hàm

Đỉnh duyệt

Đỉnh cha

Độ sâu

```
function<void(int, int, int)> dfs = [&](int v, int p, int d)
{
    depth[v] = d;
    binup[0][v] = p;
    for (int i = 1; i < bp; ++i)
        binup[i][v] = binup[i - 1][binup[i - 1][v]];

    int cnt = 0;
    for (int i = 0; i < graph[v].size(); ++i)
        if (graph[v][i] != p)
        {
            ind_child[graph[v][i]] = cnt++;
            dfs(graph[v][i], v, d + 1);
        }
};
```

Tạo danh sách đỉnh con

Kiểm tra điều kiện đủ để không có đường đi:

Chuẩn hóa trường hợp xử lý

Tìm cha chung gần nhất

```
fi >> a >> b;
--a; --b;
int ma = a, mb = b;
bool sw = false;
if (depth[a] > depth[b])
{
    sw = true;
    swap(a, b);
}
for (int i = bp - 1; i >= 0; --i)
    if (depth[b] - (1 << i) >= depth[a])
        b = binup[i][b];
if (a == b)
{
    if (ma == a) continue;
    fo << "0\n";
    return 0;
```

Ghi nhận cặp đỉnh xác định trình tự duyệt bắt buộc:

```
for (int i = bp - 1; i >= 0; --i)
    if (binup[i][a] != binup[i][b])
    {
        a = binup[i][a];
        b = binup[i][b];
    }
if (sw) swap(a, b);
int tmp = binup[0][a];
here[tmp].push_back({ind_child[a], ind_child[b]});
```

Lùi về nút cha chung

Ghi nhận cặp trình tự bắt buộc

Sơ đồ quy hoạch động tính số đường đi:

```
for (int i = 0; i < n; ++i)
{
    int deg = graph[i].size();
    if (i) --deg;
    fill(matr.begin(), matr.begin() + deg, 0);
    for (pii e : here[i]) matr[e.ff] |= 1 << e.ss;
    fill(d.begin(), d.begin() + (1 << deg), 0);
    d[0] = 1;
    for (int j = 1; j < 1 << deg; ++j)
        for (int k = 0; k < deg; ++k)
            if (j & (1 << k) && (matr[k] & j) == 0)
                d[j]=((ll)d[j]+d[j^(1 << k)])%MOD;
    ans = ((ll)ans* d[(1 << deg) - 1])%MOD;
}
```

Ghi nhận cặp mã bít cố định

Điều kiện tự do lựa chọn đường

Số lượng đường có thể chọn

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

Chương trình

```
#include <bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
ifstream fi ("postrobot.inp");
ofstream fo ("postrobot.out");
const int MOD = 1000000007;
typedef int64_t ll;
typedef pair<int, int> pii;
int main()
{
    int n, m;
    fi >> n >> m;
    vector<vector<int>> graph(n);
    for (int i = 1; i < n; ++i)
    {
        int a, b;
        fi >> a >> b;
        --a; --b;
        graph[a].push_back(b);
        graph[b].push_back(a);
    }
    int bp = 0;
    while ((1 << bp) < n) bp++;
    vector<vector<int>> binup(bp, vector<int>(n));
    vector<int> depth(n);
    vector<int> ind_child(n);
    function<void(int, int, int)> dfs = [&](int v, int p, int d)
    {
        depth[v] = d;
        binup[0][v] = p;
        for (int i = 1; i < bp; ++i)
                binup[i][v] = binup[i - 1][binup[i - 1][v]];

        int cnt = 0;
        for (int i = 0; i < graph[v].size(); ++i)
            if (graph[v][i] != p)
            {
                ind_child[graph[v][i]] = cnt++;
                dfs(graph[v][i], v, d + 1);
            }
    };
    dfs(0, 0, 0);
    vector<vector<pii>> here(n);
    for (int q = 0; q < m; ++q)
    {
```

```
        int a, b;
        fi >> a >> b;
        --a; --b;
        int ma = a, mb = b;
        bool sw = false;
        if (depth[a] > depth[b])
           { sw = true;   swap(a, b); }
        for (int i = bp - 1; i >= 0; --i)
            if (depth[b] - (1 << i) >= depth[a])
                b = binup[i][b];
        if (a == b)
        {
            if (ma == a) continue;
            fo << "0\n";
            return 0;
        }
        for (int i = bp - 1; i >= 0; --i)
            if (binup[i][a] != binup[i][b])
            {
                a = binup[i][a];
                b = binup[i][b];
            }
        if (sw) swap(a, b);
        int tmp = binup[0][a];
        here[tmp].push_back({ind_child[a], ind_child[b]});
    }
    int ans = 1;
    const int MAXD = 12;
    vector<int> matr(MAXD);
    vector<int> d(1 << MAXD);
    for (int i = 0; i < n; ++i)
    {
        int deg = graph[i].size();
        if (i) --deg;
        fill(matr.begin(), matr.begin() + deg, 0);
        for (pii e : here[i]) matr[e.ff] |= 1 << e.ss;
        fill(d.begin(), d.begin() + (1 << deg), 0);
        d[0] = 1;
        for (int j = 1; j < 1 << deg; ++j)
            for (int k = 0; k < deg; ++k)
                if (j & (1 << k) && (matr[k] & j) == 0)
                        d[j]=((ll)d[j]+d[j^(1 << k)])%MOD;
        ans = ((ll)ans* d[(1 << deg) - 1])%MOD;
    }
    fo << ans << "\n";
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Xét xâu chỉ chứa các ký tự ngoặc tròn **(** và **)**. Xâu tạo thành một biểu thức ngoặc đúng nếu thỏa mãn điều kiện định nghĩa biểu thức ngoặc đúng:

- Xâu rỗng là một biểu thức ngoặc đúng,
- Nếu ***A*** là biểu thức ngoặc đúng thì ***(A)*** cũng là một biểu thức ngoặc đúng,
- Nếu ***A*** và ***B*** là các biểu thức ngoặc đúng thì ***AB*** cũng là một biểu thức ngoặc đúng.

Xét biểu thức ngoặc đúng độ dài 2×***n***. Biểu thức sẽ chứa ***n*** ngoặc mở và ***n*** ngoặc đóng. Mỗi ngoặc mở sẽ tương ứng với đúng một ngoặc đóng mà xâu con các ký tự giữa 2 ngoặc này là một biểu thức ngoặc đúng, 2 ngoặc này tạo thành cặp ngoặc liên hợp. Số ký tự nằm giữa 2 ngoặc của một cặp ngoặc liên hợp được gọi là khoảng cách của cặp ngoặc đó.

Cho số nguyên ***n*** và khoảng cách của các cặp ngoặc liên hợp trong biểu thức ngoặc.

Nếu có thể xác định biểu thức ngoặc đúng có khoảng cách các cặp ngoặc liên hợp thỏa mãn dữ liệu đã cho thì đưa ra thông báo ***Yes*** và ở dòng tiếp theo – một biểu thức ngoặc đúng tùy chọn thỏa mãn điều kiện đã nêu.

Nếu không tồn tại biểu thức ngoặc đúng thỏa mãn điều kiện đã nêu – đưa ra thông báo ***No***.

Dữ liệu: Vào từ file văn bản CBE.INP:

- Dòng đầu tiên chứa một số nguyên ***n*** ($1 \le n \le 20$),
- Dòng tứ 2 chứa ***n*** số nguyên $a_1, a_2, \ldots, a_n$ – khoảng cách của các cặp ngoặc liên hợp ($0 \le a_i \le 2\times n$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản CBE.OUT các thông tin xác định được theo quy cách đã nêu.

Ví dụ:

CBE. INP	CBE.OUT
5 0 0 0 2 6	Yes ()(()())

Giải thuật: Tổ chức dữ liệu, duyệt tổ hợp.

Nhận xét:

- Mỗi cặp ngoặc liên hợp sẽ tạo ra một biểu thức ngoặc đúng ,
- Biết khoảng cách của cặp ngoặc liên hợp ta có thể dễ dàng tính số cặp ngoặc tham gia vào biểu thức đúng tương ứng với cặp ngoặc đó,
- Số cặp ngoặc tham gia vào biểu thức đúng tương ứng với một cặp ngoặc liên hợp được gọi là *độ dài biểu thức*,
- Biểu thức ngoặc đúng độ dài lớn hơn 1 phải *chứa ít nhất một biểu thức ngoặc đúng độ dài 1*,
- Khoảng cách của một cặp ngoặc liên hợp phải là một số chẵn, trong trường hợp ngược lại – bài toán vô nghiệm.

Từ dữ liệu đã cho ta có thể tính *số lượng biểu thức độ dài* **i**, **i** = 1 ÷ **n**.

Gọi a_i – số lượng biểu thức độ dài **i**, **i** = 1 ÷ **n**.

a_1 biểu thức ngoặc đúng độ dài 1 là *cơ sở xuất phát* để xác định biểu thức cần tìm, mỗi biểu thức cơ sở có dạng **()**.

Lần lượt duyệt a_i với **i** = 2 ÷**n**.

Với mỗi $a_i > 0$:

- Tìm a_i tập con *không giao nhau từng đôi một* từ *cơ sở hiện có*, mỗi tập con cho tổng độ dài các phần tử bằng **i-1**,
- Gọi **s** – tổng các biểu thức xác định bởi một tập con, kết nạp biểu thức **'('+s+')'** độ dài **i** vào cơ sở,
- Xóa các biểu thức đã sử dụng trong cơ sở.

Nếu với một $a_i > 0$ nào đó không tìm được đủ a_i biểu thức ngoặc đúng độ dài **i** – bài toán vô nghiệm.

Trong trường hợp có nghiệm: Biểu thức cần tìm là tổng các biểu thức trong cơ sở cuối cùng.

Tìm tập con cho tổng độ dài các phần tử là **k** từ cơ sở có **m** phần tử: duyệt các tổ hợp chập 1, chập 2, . . ., chập **m** các phần tử của cơ sở. Số lượng tổ hợp cần kiểm tra không vượt quá 2^m.

Tổ chức dữ liệu:

- Mảng `vector<int> a(22,0)` – Lưu số lượng biểu thức đúng theo độ dài của nó,
- Mảng `vector<int> b(22)` – Lưu độ dài các phần tử trong cơ sở,
- Mảng `vector<string> sb(22)` – Lưu các biểu thức ngoặc trong cơ sở,
- Mảng `vector<int> b0(22)` – Đánh dấu các phần tử đã được sử dụng của cơ sở.

Độ phức tạp của giải thuật: $O(n2^n)$.

Xử lý:

Thống kê biểu thức theo độ dài:

```
fi>>n;
for(int i=1; i<=n; ++i)
{
    fi>>t;
    k=(t+2)/2;
    if((t&1) || (k>n)){ fo<<"No"; return 0;}
    ++a[k];
}
```

Số cặp ngoặc lớn hơn n

Khoảng cách lẻ

Xây dựng cơ sở xuất phát:

```
nb=a[1];
for(int i=1; i<=nb; ++i)
{
    b[i]=1; sb[i]= "()";
}
```

Độ dài

Biểu thức

Xác định biểu thức cơ sở mới theo tổ hợp thứ **y**:

```
int calc_sum(int y)
{
    int r = 0, p = nb+1, q;
    b_sum="";
    fill(b0.begin(),b0.end(),0);
    for(int i=0; i<nb; ++i)
    {
        q = nb-i;
        if(y&(1<<i))
        {
            r+=b[q];
            b_sum+=sb[q];
            b0[q]=1;
        }
    }
    return r;
}
```

Xóa bảng đánh dấu

Kiểm tra bít i được bật

Ưu tiên chọn độ dài lớn

Cập nhật bảng cơ sở:

```
void get_base(int x)
{
    --a[x+1];
    for(int i=1; i<(1<<nb); ++i)
    {
        t = calc_sum(i);
        flag = t==x;
        if(flag) break;
    }
    if(!flag) return;

    for(int i=1; i<=nb; ++i) if(b0[i]) b[i]=0;

    b[++nb]=x+1; sb[nb]='('+b_sum+')';
    k = 0;
    for(int i=1; i<=nb; ++i)
    {
        if(b[i]==0)continue;
        b[++k]=b[i]; sb[k]=sb[i];
    }
    nb = k;
    return;
}
```

Độ dài biểu thức cơ sở cần tìm

Duyệt tổ hợp thứ i

Bổ sung vào cơ sở

Lọc biểu thức đã sử dụng

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("cbe.inp");
ofstream fo {"cbe.out"};
int n, sum, t, k, nb;
bool flag;
string b_sum,ans;
vector<int> a(22,0);
vector<int> b(22),b0(22);
vector<string> sb(22);

int calc_sum(int y)
{
    int r = 0, p = nb+1, q;
    b_sum="";
    fill(b0.begin(),b0.end(),0);
    for(int i=0; i<nb; ++i)
    {
        q = nb-i;
        if(y&(1<<i))
        {
            r+=b[q];
            b_sum+=sb[q];
            b0[q]=1;
        }
    }
    return r;
}

void get_base(int x)
{
    --a[x+1];
    for(int i=1; i<(1<<nb); ++i)
    {
        t = calc_sum(i);
        flag = t==x;
        if(flag) break;
    }
    if(!flag) return;

    for(int i=1; i<=nb; ++i) if(b0[i]) b[i]=0;

    b[++nb]=x+1; sb[nb]='('+b_sum+')';
    k = 0;
    for(int i=1; i<=nb; ++i)
```

```
    {
        if(b[i]==0)continue;
        b[++k]=b[i]; sb[k]=sb[i];
    }
    nb = k;
    return;
}

int main()
{
    fi>>n;
    for(int i=1; i<=n; ++i)
    {
        fi>>t;
        k=(t+2)/2;
        if((t&1) || (k>n)){ fo<<"No"; return 0;}
        ++a[k];
    }
    nb=a[1];
    for(int i=1; i<=nb; ++i)
    {
        b[i]=1; sb[i]= "()";
    }
    for(int i=2; i<=n; ++i)
    {
        if(a[i]==0) continue;
        while(a[i]>0) get_base(i-1);
        if(!flag){ fo<<"No"; return 0;}
    }
    fo<<"Yes\n";
    if(nb==1) ans=sb[1];
        else for(int i=1; i<=nb; ++i) ans+=sb[i];
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Xét xâu chỉ chứa các ký tự ngoặc tròn **(** và **)**. Xâu tạo thành một biểu thức ngoặc đúng nếu thỏa mãn điều kiện định nghĩa biểu thức ngoặc đúng:

- Xâu rỗng là một biểu thức ngoặc đúng,
- Nếu ***A*** là biểu thức ngoặc đúng thì ***(A)*** cũng là một biểu thức ngoặc đúng,
- Nếu ***A*** và ***B*** là các biểu thức ngoặc đúng thì ***AB*** cũng là một biểu thức ngoặc đúng.

Biểu thức ngoặc đúng độ dài 2×***n*** sẽ chứa ***n*** ngoặc mở và ***n*** ngoặc đóng. Mỗi ngoặc mở sẽ tương ứng với đúng một ngoặc đóng mà xâu con các ký tự giữa 2 ngoặc này là một biểu thức ngoặc đúng, 2 ngoặc này tạo thành cặp ngoặc liên hợp. Số ký tự nằm giữa 2 ngoặc của một cặp ngoặc liên hợp được gọi là khoảng cách của cặp ngoặc đó.

Cho xâu ***a*** chỉ chứa các ký tự ngoặc tròn **(** và **)**. Hãy xác định và đưa ra khoảng cách của các cặp ngoặc liên hợp.

Dữ liệu: Vào từ file văn bản DISTANCES.INP gồm một dòng chứa xâu ***a*** độ dài không quá 10^6.

Kết quả: Đưa ra file văn bản DISTANCES.OUT. Nếu xâu ***a*** không là biểu thức ngoặc đúng thì đưa ra số **-1**, trong trường hợp ngược lại đưa ra một số nguyên – số cặp ngoặc liên hợp và ở dòng tiếp theo – các số nguyên xác định khoảng cách của các cặp ngoặc liên hợp, các giá trị được đưa ra theo thứ tự tăng dần của khoảng cách.

Ví dụ:

DISTANCES.INP
(((()(())())))

DISTANCES.OUT
6 0 0 0 2 8 10

Giải thuật: Ứng dụng các cấu trúc dữ liệu cơ sở .

Một biểu thức ngoặc đúng khi và chỉ khi:

- Số dấu ngoặc mở bằng số dấu ngoặc đóng,
- Khi duyệt từ trái sang phải số lượng dấu ngoặc mở gặp trong quá trình duyệt không ít hơn số số lần gặp dấu ngoặc đóng.

Như vậy, dễ dàng kiểm tra biểu thức ngoặc đúng hay sai.

Phát hiện cặp ngoặc liên hợp và tính khoảng cách giữa chúng:

- Lưu vị trí trong xâu của dấu ngoặc mở vào stack,
- Khi gặp dấu ngoặc đóng: Lấy ra khỏi stack vị trí ngoặc mở và tính khoảng cách,
- Lưu khoảng cách vào bảng tần số các giá trị tính được.

Số lượng tần số khác nhau và giá trị lớn nhất của khoảng cách không vược quá độ dài xâu đã cho.

Đưa ra kết quả:

- Đưa ra các số có tần số khác 0 theo thứ tự tăng dần,
- Mỗi số được đưa ra với số lần bằng tần số xuất hiện.

Độ phức tạp của giải thuật: O(n).

Chương trình

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

int main()
{
    string a;
    fi>>a;
    int n = a.size();
    int p, b = 0;
    stack<int> s;
    vector<int> ans(n+1,0);
    for(int i=0; i<n;++i)
    {
        if(a[i]=='(') {++b; s.push(i);}
            else
            {
                --b;
                if(b<0){fo<<"-1"; return 0;}
                p = s.top(); s.pop();
                ++ans[i-p-1];
            }
    }
    if(b>0){fo<<"-1"; return 0;}
    fo<<n/2<<'\n';
    for(int i=0; i<n; ++i)
    {
        p=ans[i];
        while(p>0){fo<<i<<' '; --p;}
    }

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

Để thu hút khách du lịch một nước vùng Vịnh xây dựng tổ hợp khách sạn độc đáo gồm nhiều nhà mái bằng liên tiếp nhau thành một dãy dài, tòa nhà đầu tiên có độ cao 1, tòa nhà thứ 2 có độ cao 1+2, tòa nhà thứ 3 có độ cao 1+2+3, . . ., tòa nhà thứ ***k*** có độ cao $\sum_{i=1}^{k} i$. Không nhìn thấy tòa nhà cuối cùng trong dãy nhưng hướng dẫn viên du lịch cho biết chiều cao của tòa nhà cuối cùng không vượt quá 10^9.

Một trận bão cát dữ dội đã làm tối đen bầu trời và phủ cát lên mọi vật. Lớp cát phủ trên nóc mỗi tòa nhà có độ dày như nhau. Kết quả đo đạc độ cao của 2 tòa nhà liên tiếp nhau cho thấy độ cao mới của chúng (kể cả lớp cát phủ) là ***a*** và ***b***.

Hãy xác định độ dày lớp cát phủ trên mái nhà.

Dữ liệu: Vào từ file văn bản SANDSTORM.INP gồm một dòng chứa 2 số nguyên ***a*** và ***b***, ***a*** < ***b*** $\leq 2\times10^9$.

Kết quả: Đưa ra file văn bản SANDSTORM.OUT một số nguyên – độ dày lớp cát phủ trên mái nhà.

Ví dụ:

SANDSTORM.INP	SANDSTORM.OUT
11 16	1

WB26 Azr 2020 B A XVII

Giải thuật: Cơ sở lập trình.

Giá trị tuyệt đối độ cao 2 tòa nhà liên tiếp nhau sẽ là số thứ tự của tòa nhà cao hơn.

Từ đó dễ dàng tính độ cao ban đầu của tòa nhà và xác định độ dày lớp cát trên mái nhà.

Độ phức tạp giải thuật: O(1).

Chương trình

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

int main()
{
    int a, b,k,ans;
    fi>>a>>b;
    k = b - a;
    ans = b - k*(k+1)/2;
    fo<<ans;
}
```

Để tổ chức thi trắc nghiệm cần có rất nhiều câu hỏi tương tự nhau, chỉ khác giá trị các đại lượng trong đó và vì vậy – sẽ có kết quả khác nhau. Tin học là công cụ hỗ trợ tuyệt vời phục vụ soạn đề: chỉ cần lập trình một lần, cho chạy với các tham số khác nhau sẽ có một loạt đề cùng độ khó.

Bộ môn Toán đang cần đưa vào đề câu hỏi về phương trình chứa giá trị tuyệt đối, cụ thể là với 2 số nguyên ***a*** và ***b*** khác nhau, hãy tìm ***x*** nguyên thỏa mãn điều kiện **|*a*-*x*| = |*b*-*x*|**. Vấn đề không khó, nhưng nếu cần vài trăm bộ giá trị ***a*** và ***b*** thì việc tìm các nghiệm cũng mất khá nhiều thời gian và cũng có thể xảy ra nhầm lẫn nữa!

Để an toàn, Bộ môn Toán đề nghị Bộ môn Tin hỗ trợ với yêu cầu cụ thể là cho ***n*** cặp số nguyên ***a*** và ***b*** (***a*** ≠ ***b***), với mỗi cặp cho biết ***x*** tương ứng hoặc thông báo vô nghiệm.

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

- Dòng đầu tiên chứa một số nguyên ***n*** ($1 \le n \le 100$),
- Mỗi dòng trong ***n*** dòng sau chứa 2 số nguyên ***a*** và ***b***, ***a*** ≠ ***b***, $-10^6 \le a, b \le 10^6$.

Kết quả: Đưa ra file văn bản EXERCISES.OUT với mỗi cặp số đã cho đưa ra trên một dòng số nguyên ***x*** tương ứng hay thông báo ***No*** nếu không tồn tại ***x***.

Ví dụ:

EXERCISES. INP
2 -3 9 10 21

EXERCISES.OUT
3 No

Giải thuật: Cơ sở lập trình .

Độ phức tạp của giải thuật: O(*n*)

Chương trình

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

int main()
{
    int n, a, b, x;
    fi>>n;
    for(int i =0; i<n; ++i)
    {
        fi>>a>>b;
        x = a+b;
        if(x&1) fo<<"No\n";
            else fo<< x/2<<'\n';
    }
```

}

WB28. DỮ LIỆU LỚN

Tên chương trình: BIGDATA.CPP

Thời đại bùng nổ thông tin đã làm thay đổi cuộc sống của mỗi người, thay đổi cả cách lưu trữ và xử lý thông tin.

Xét việc lưu trữ và tìm kiếm dữ liệu trong mảng số nguyên. Thông tin về một mảng số nguyên được thu thập và lưu trữ dưới dạngmột mảng rỗng và n cặp (a_i, b_i) cho biết số b_i xuất hiện thêm a_i lần (trước đó có thể chưa có hoặc đã có phần tử với giá trị b_i), $i = 1 \div n$.

Có q truy vấn, truy vấn thứ j yêu cầu đưa ra phần tử thứ k_j trong mảng đã sắp xếp theo không giảm. Các phần tử của mảng được đánh số bắt đầu từ 1.

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

- Dòng đầu tiên chứa một số nguyên n ($1 \le n \le 10^5$),
- Dòng thứ i trong n dòng sau chứa 2 số nguyên a_i và b_i ($1 \le a_i, b_i \le 10^5$),
- Dòng tiếp theo chứa số nguyên q ($1 \le q \le 10^4$),
- Dòng cuối cùng chứa q số nguyên dương $k_1, k_2, \ldots, k_q$. Dữ liệu đảm bảo tồn tại các phần tử cần tìm.

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

Ví dụ:

BIGDATA. INP	BIGDATA.OUT
3 2 8 3 5 5 9 3 3 4 10	5 8 9

WB28 Azr2020 D A XVII

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

Dữ liệu về mảng được lưu trữ dưới dạng bảng tần số các phần tử,

Từ bảng tần số xây dựng mảng tổng tiền tố các tần số và mảng giá trị các phần tử khác nhau tương ứng của mảng dữ liệu ban đầu.

Việc xác định phần tử thứ **k** của dữ liệu được thực hiện bằng tìm kiếm nhị phân trên mảng tổng tiền tố.

Tổ chức dữ liệu:

- Mảng `vector<int64_t> f(u,0)` – Lưu tần số các giá trị của mảng dữ liệu,
- Mảng `vector<int64_t>prf` – Lưu tổng tiền tố các tần số,
- Mảng `vector<int> v` – Lưu các giá trị khác nhau của mảng ban đầu.

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

Xử lý:

- Việc tìm kiếm nhị phân được thực hiện bằng hàm **`upper_bound`** trên mảng tiền tố **`prf`**,
- Cần lưu ý trường hợp khi phần tử cần tìm nằm trên biên của 2 khối với giá trị mảng khác nhau,
- Lưu ý kiểu dữ liệu các biến trung gian.

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("bigdata.inp");
ofstream fo ("bigdata.out");
const int u=(int)1e5+1;
int main()
{
    int n, a, b, q, m;
    int64_t k;
    fi>>n;
    vector<int64_t> f(u,0);
    for(int i =0; i<n; ++i)
    {
        fi>>a>>b;
        f[b]+=a;
    }
    vector<int64_t>prf;
    vector<int> v;
    prf.push_back(0);
    v.push_back(0);
    for(int i=1; i<u; ++i)
        if(f[i])
        {
            prf.emplace_back(f[i]);
            v.emplace_back(i);
        }

    k = v.size();
    for(int i=1; i<k; ++i) prf[i]+=prf[i-1];
    fi>>q;
    vector<int64_t>::iterator p;
    for(int i=0; i<q; ++i)
    {
        fi>>k;
        p = upper_bound(prf.begin(), prf.end(),k);
        m = p - prf.begin();
        if(*(p-1)==k) fo<<v[m-1]<<'\n';
            else fo<<v[m]<<'\n';
    }
   fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Để chuẩn bị sản xuất đại trà vắc xin chống một loại đại dịch đang hoành hành trên thế giới người ta cần có một trại nuôi ngựa lấy huyết thanh và một phòng thí nghiệm sản xuất vắc xin. Hai cơ sở này phải tách rời, nhưng ở càng gần nhau càng tốt.

Có n địa điểm có thể có thể đặt các cơ sở đó. Địa điểm thứ i được xác định bởi điểm có tọa độ (x_i, y_i), i = 1 ÷ n. Khoảng cách ***d*** giữa 2 điểm ***i*** và ***j*** được tính theo công thức ***d*** = $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ (*Khoảng cách Euclide*).

Hãy xác định khoảng cách ngắn nhất giữa 2 điểm trong số các điểm đã cho và chỉ ra một cặp điểm có khoảng cách ngắn nhất.

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

- Dòng đầu tiên chứa một số nguyên ***n*** ($2 \le n \le 10^5$),
- Dòng thứ ***i*** trong ***n*** dòng sau chứa 2 số nguyên x_i và y_i ($|x_i|, |y_i| \le 10^9$).

Kết quả: Đưa ra file văn bản VACCINE.OUT, dòng đầu tiên là bình phương khoảng cách ngắn nhất tìm được dòng thứ 2 chứa 2 số nguyên là số thứ tự của một cặp điểm có khoảng cách nhỏ nhất.

Ví dụ:

VACCINE.INP
3 1 1 3 1 1 2

VACCINE .OUT
4 1 3

WB29 Azr2020 E A XVII

Giải thuật: Chia để trị .

Sơ đồ xử lý là chia tập các điểm thành 2 phần, giải bài toán tương tự ở mỗi phần,

Xác định một dãi đệm kết nối 2 phần, giải bài toán tương tự ở lớp đệm và tổng hợp kết quả từ các kết quả nhận được ở ba phần đã nêu.

Để phục vụ xác định điểm có thể chọn, mỗi điểm tương ứng với cụm 3 đại lượng nguyên (x_i, y_i, i).

Việc phân chia tập ban đầu thành 2 phần không quá khó khăn:

- Sắp xếp các điểm đã cho theo trình tự tăng dần của **x**,
- Lấy các điểm ở nửa đầu của dãy cho vào tập **A1**, phần còn lại – vào tập **A2**.

Giả thiết ta đã giải được bài toán ở các tập **A1** và **A2** với khoảng cách nhỏ nhất tìm được tương ứng là **d1** và **d2**.

Gọi **d** = min{**d1**, **d2**}.

Rõ ràng tập các điểm **B** ở dải đệm chỉ chứa các điểm thỏa mãn điều kiện khoảng cách tới đường phân chia nhỏ hơn **d**

$$|x_v - x_B| < d,$$

trong đó x_b – tọa độ xác định đường phân chia.

Tập **C** các điểm cần xét trong dải đệm còn hẹp hơn, với mỗi điểm p_i trong **B** chỉ cần xét các điểm có chênh lệch theo **y** tới p_i không quá **d**. Gọi tập điểm cần xét với điểm p_i là $C(p_i)$. $C(p_i)$ dễ dàng xác định được nếu các điểm trong **B** được sắp xếp theo giá trị tọa độ **y**.

Xử lý đệ quy:

Hàm **`rec(int l, int r)`** tìm khoảng cách nhỏ nhất trong số các điểm từ **`l`** đến **`r`** của dãy đã sắp xếp:

- Nếu **`r-l`** là đủ bé – tiến hành vét cạn các cặp điểm, sắp xếp dữ liệu trong khoảng theo y,
- Trong trường hợp ngược lại:
  - Chia đôi khoảng và lần lượt gọi rec với các khoảng nhận được,
  - Hợp nhất khoảng và sắp xếp theo y,
  - Xử lý dãy đệm.

Lưu ý: dùng mảng dữ liệu trung gian kiểu static để tránh việc phải phân nhiều lần, điều có thể dẫn đến hiện tượng thiếu bộ nhớ làm việc.

Tổ chức dữ liệu:

Mảng `vector<tiii> a` – Lưu tọa độ các điểm từ dữ liệu vào,

Mảng `static tiii t[MAXN]` – Phục vụ lưu dữ liệu hợp nhất các vùng đã khảo sát.

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

Chương trình

```
#include <bits/stdc++.h>
#define NAME "Vaccine."
using namespace std;
typedef pair<double,double> pdd;
typedef tuple<int,int,int>tiii;
typedef int64_t ll;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int MAXN=100001;
int n;
vector<tiii> a;

inline bool cmp_y (const tiii & a, const tiii & b) { return
get<1>(a) < get<1>(b); }
 ll mindist;
 int ansa, ansb;
 inline void upd_ans (const tiii & a, const tiii & b)
     {ll dist = 1ll*(get<0>(a)-get<0>(b))*(get<0>(a)-get<0>(b))
             + 1ll*(get<1>(a)-get<1>(b))*(get<1>(a)-get<1>(b)) ;
      if(dist<mindist)
          {mindist=dist;ansa= get<2>(a);ansb=get<2>(b); }
     };

 void rec (int l, int r)
 {if (r - l <= 3)
  {
      for (int i=l; i<=r; ++i)
      for (int j=i+1; j<=r; ++j)
      upd_ans (a[i], a[j]);
      sort (a.begin()+l, a.begin()+r+1, &cmp_y);
      return;
  }
   int m = (l + r) >> 1;
   int midx = get<0>(a[m]);
   rec (l, m),  rec (m+1, r);
   static tiii t[MAXN];
   merge (a.begin()+l, a.begin()+m+1,
          a.begin()+m+1, a.begin()+r+1, t);

   copy (t, t+r-l+1, a.begin()+l);
   int tsz = 0;
   for (int i=l; i<=r; ++i)
    if (abs (get<0>(a[i]) - midx) < mindist)
    {
        for (int j=tsz-1;
```

```
            j>=0 && get<1>(a[i]) - get<1>(t[j]) < mindist; --j)
                                    upd_ans (a[i], t[j]);
        t[tsz++] = a[i];
    }
}

int main()
{
    fi>>n;
    int x,y,k;
    for(int i=0;i<n;++i)
    {
        fi>>x>>y;
        a.push_back(make_tuple(x,y,i));
    }
    sort(a.begin(),a.end());
    mindist=1ll*1e18;
    rec(0,n-1);
    fo<<mindist<<'\n'<<ansa+1<<' '<<ansb+1;

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

Xét dãy số Fibonacci:

- $F_0 = F_1 = 1$,
- $F_n = F_{n-1} + F_{n-2}$, $n \geq 2$.

Với số nguyên a và b cho trước, hãy tính $s = \sum_{i=a}^{b} F_i$ và đưa ra theo mô đun 10^9.

Dữ liệu: Vào từ file văn bản FIBSUM.INP gồm một số dòng, mỗi dòng chứa 2 số nguyên a và b ($0 \leq a \leq b \leq 10^9$).

Kết quả: Đưa ra file văn bản FIBSUM.OUT các kết quả tính được dưới dạng số nguyên, mỗi số trên một dòng.

Ví dụ:

FIBSUM.INP	FIBSUM.OUT
1 1 3 5 10 1000	1 16 496035733

WB30 Azr ADA 2020 A A XVII

Giải thuật: Tính nhanh lũy thừa.

Với mọi **n** ta có

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

Việc tính F_n và F_{n+1} có thể dễ dàng thực hiện theo sơ đồ tính nhanh lũy thừa.

Xuất từ ma trận ban đầu

$$x = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Ta sẽ tính x^2, x^4, x^8, . . . và tích lũy kết quả vào ma trận **z** kích thước 2×2

Để tăng tốc độ xử lý các ma trận **x** và **z** được lưu trữ dưới dạng một chiều:

$$x = \begin{bmatrix} x_0 & x_1 \\ x_2 & x_3 \end{bmatrix} \qquad z = \begin{bmatrix} z_0 & z_1 \\ z_2 & z_3 \end{bmatrix}$$

Xét $s = F_a + F_{a+1} + \ . \ . \ . + F_{b-1} + F_b$

Với **b-a+1** chẵn: nhóm từng cặp 2 số hàng từ cuối về đầu, ta được:

$$s = \underbrace{F_a + F_{a+1}}_{F_{a+2}} + \ . \ . \ . + \underbrace{F_{b-3} + F_{b-2}}_{F_{b-1}} + \underbrace{F_{b-1} + F_b}_{F_{b+1}}$$

Cộng thêm vào 2 vế đẳng thức giá trị F_{a+1}, ta có:

$$S + F_{a+1} = \underbrace{F_a + F_{a+1}}_{F_{a+2}} + \ . \ . \ . + \underbrace{F_{b-3} + F_{b-2}}_{F_{b-1}} + \underbrace{F_{b-1} + F_b}_{F_{b+1}}$$

F_{a+1} F_{a+2} F_{a+3} . . . F_{b-2} F_{b-1} F_b F_{b+1} F_{b+2}

Như vậy: $\boxed{s = F_{b+2} - F_{a+1}}$

Dễ dàng chứng minh công thức này đúng với cả trường hợp **b-a+1** là lẻ.

Tuy công thức vẫn đúng trong trường hợp **a = b**, nhưng để giảm thời gian tính toán nên phân biệt riêng trường hợp này.

Cần lưu ý tổ chức sơ đồ dẫn xuất kết quả không phụ thuộc vào dấu của $F_{b+2}-F_{a+1}$ (vì giá trị của mỗi toán hạng được tính theo mô đun 10^9).

Quá trình xử lý kết thúc khi file input rỗng. Với hệ thống lập trình C++, nếu sau dòng dữ liệu cuối cùng có dấu xuống dòng, chương trình sẽ đọc tiếp một dòng nữa và *không làm thay đổi giá trị các biến ở lần đọc trước*! Do đó, cần tổ chức nhận dạng trường hợp gặp dòng rỗng.

Độ phức tạp của giải thuật:

- Với mỗi cặp dữ liệu độ phức tạp xử lý là O(log*b*),
- Gọi **q** – số lượng các cặp dữ liệu cầ xử lý, **m** = *max* các giá trị **b**,
- Độ phức tạp toàn chương trình: O(*q*log*m*).

Chương trình

```
#include <bits/stdc++.h>
using namespace std;
ifstream fi ("fibsum.inp");
ofstream fo ("fibsum.out");
const int p=(int)1e9;
vector<int64_t> x(4),y(4),z(4);

vector<int64_t> mult(vector<int64_t> a,vector<int64_t> b)
{
    vector<int64_t> t(4);
    t[0]=(a[0]*b[0]%p+a[1]*b[2]%p)%p;
    t[1]=(a[0]*b[1]%p+a[1]*b[3]%p)%p;
    t[2]=(a[2]*b[0]%p+a[3]*b[2]%p)%p;
    t[3]=(a[2]*b[1]%p+a[3]*b[3]%p)%p;
    return t;
}
int get_fib(int k)
{
    z[0]=z[3]=1; z[1]=z[2]=0;
    x[0]=x[1]=x[2]=1; x[3]=0;
    while(k>0)
    {
        if(k&1) z=mult(x,z);
        x=mult(x,x);
        k>>=1;
    }
    return z[0];
}
int main()
{
    int a, b;
    int64_t ans;
    while(!fi.eof())
    {
        a=-1;
        fi>>a>>b;
        if(a==-1) break;
        if(a==b) ans = (a==0)? 1:get_fib(a);
            else ans = (get_fib(b+2) + p - get_fib(a+1))%p;
        fo<<ans<<'\n';
    }
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Một video clip đưa lên mạng trong vài ngày đầu được rất nhiều người truy cập nhưng sau đó số người xem giảm hẳn đi không phải là một clip hay. Ngược lại có những clip ban đầu chưa thu hút nhiều sự chú ý của công chúng nhưng càng ngày càng có thêm nhiều người truy cập là một clip có chất lượng cao. Dĩ nhiên, đến một lúc nào đó số lượt truy nhập sẽ giảm dần về 0.

Số liệu quan sát một vdeo clip trong n ngày cho thấy ngày thứ i có a_i lượt truy nhập, $i = 1 \div n$. Dãy con dài nhất có dạng **1**, **2**, **3**, . . ., **x-1**, **x**, **x-1**, . . ., **2**, **1** trích ra được từ (a_1, a_2, . . ., a_n) được dùng để đánh giá clip và chất lượng của nó được ghi nhận là x. Nếu không tìm được dãy con nào có dạng trên, chất lượng của clip sẽ là 0.

Cho n và dãy số a_1, a_2, . . ., a_n của một clip. Hãy xác định chất lượng của clip này.

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

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

Kết quả: Đưa ra file văn bản QUALITY.OUT một số nguyên – chất lượng của video clip.

Ví dụ:

QUALITY.INP
5 1 10 2 3 1

QUALITY .OUT
2

WB31 Azr2019 B A XVII

Giải thuật: Kỹ thuật 2 con trỏ .

Các con trỏ **p** và **q** chỉ tới phần tử cần tìm,

Ban đầu: **p = 0**, **q = n-1**,

Duyệt lần lượt **k** = 1, 2, 3, . . .

Với mỗi k:

- Tìm **p ≤ i ≤ q** thỏa mãn a_i = **k**, gán **p = i**,
- Tìm **p ≤ j ≤ q** thỏa mãn a_j = **k**, gán **q = j**.

Dẫn xuất nghiệm:

Độ phức tạp của giải thuật: O(*n*).

Chương trình

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

int main()
{
    int n,k,p,q;
    fi>>n;
    vector<int> a(n);
    for(int &i:a) fi>>i;
    p=0; q=n-1; k=0;
    while(p<q)
    {
        ++k;
        while(p<=q && a[p]!=k) ++p;
        while(q>=p && a[q]!=k)--q;
    }
    int ans = (a[p]==a[q])? a[p]:k-1;
    fo<<ans;

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

Để thành lập đội tuyển thi Tin học đồng đội người ta chỉ định một sinh viên có chỉ số năng lực chuyên môn là ***m*** làm đội trưởng và giao cho sinh viên đó nhiệm vụ chọn thêm 2 người nữa trong số ***n*** người đã qua vòng sơ loại để có một đội đi thi đấu, người thứ ***i*** trong số đó có chỉ số năng lực chuyên môn là a_i, $i = 1 \div n$.

Theo kinh nghiệm của mình Đội trưởng biết đội tuyển sẽ hoạt động hiệu quả nhất nếu tổng có chỉ số năng lực chuyên môn của cả đội là lớn nhất và tổng có chỉ số năng lực chuyên môn của 2 người được chọn không vượt quá ***m***.

Hãy đưa ra tổng chỉ số năng lực chuyên môn của cả đội hoặc số ***-1*** nếu không thể thành lập đội theo tiêu chí tối ưu đã nêu.

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

- Dòng đầu tiên chứa 2 số nguyên ***n*** và ***m*** ($2 \le n \le 10^5$, $0 \le m \le 10^9$),
- Dòng thứ 2 chứa ***n*** số nguyên $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$, $i = 1 \div n$).

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

Ví dụ:

TEAM.INP
4 116 31 52 73 84

TEAM .OUT
115

WB32 Azr2019 C A XVII

Giải thuật: Tìm kiếm nhị phân.

Nhập các giá trị a_i và sắp xếp theo thứ tự tăng dần,

Với mỗi **i** (**i** = 0, 1, 2, . . .) tìm a_j (**j** > **i**) lớn nhất thỏa mãn điều kiện

$$a_i + a_j \le m,$$

Với mỗi a_j tìm được: cập nhật kết quả *max* cần đưa ra.

Việc tìm kiếm được thực hiện với sự hỗ trợ của hàm **upper_bound**,

Để tránh việc phân tích các trường hợp ngoại lệ: thêm vào cuối mảng phần tử hàng rào với giá trị đủ lớn,

Quá trình tìm kiếm sẽ dừng khi $a_i > m - a_i$ (dãy đã được sắp xếp).

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

Chương trình

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

int main()
{
    int n, m, t;
    fi>>n>>m;
    vector<int> a(n);
    vector<int>::iterator p;
    for(int &i:a) fi>>i;
    sort(a.begin(), a.end());
    a.push_back(a[n-1]+m);
    int ans = -1;
    for(int i=0; i<n-1; ++i)
    {
        t = m - a[i];
        if(t<a[i]) break;
        p = upper_bound(a.begin()+i+1,a.end(),t);
        --p;
        if(p-a.begin()-1<=i || a[i]+*p>m) break;
        if(a[i]+*p > ans) ans = a[i]+*p;
    }
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Alice học ngành thiết kế mỹ thuật. Nhiệm vụ đầu tiên của cô là thiết kế cánh cửa trượt cho một tủ tường.

Mặt tủ có hình chữ nhật kích thước 2×***n***, có 2 cánh cửa hình chữ nhật, cánh ngắn kích thước 1×***a*** lắp ở dưới, cánh dài kích thước 1×***b*** lắp ở trên, có 2 thanh trượt để đẩy các cánh cửa chuyển động sang phải hoặc trái. Phạm vi chuyển động của cánh của dài là suốt toàn bộ chiều dài cửa tủ, cho đến khi một đầu chạm tường. Hai đầu của cánh cửa dài có các mấu giữ không cho cánh cửa ngắn trượt ra ngoài, vì vậy phạm vi chuyển động của cánh cửa ngắn là đoạn nằm trong phạm vi giữa 2 mấu giữ của cửa dài. Chỉ có thể đẩy riêng từng cánh cửa.

Một vị trí các cánh cửa

Vị trí ban đầu của các cánh cửa

Vị trí cuối cùng của các cánh cửa

Ban đầu các cánh cửa nằm ở sát mép trái của cửa. Để kiểm tra độ trơn chuyển động cần đẩy các cánh cửa về vị trí sát với mép phải của cửa.

Hãy xác định số thao tác đẩy ít nhất cần thực hiện.

Dữ liệu: Vào từ file văn bản MOVEMENT.INP gồm một dòng chứa 3 số nguyên ***a***, ***b*** và ***n*** ($1 \le a < b \le n \le 10^7$).

Kết quả: Đưa ra file văn bản MOVEMENT.OUT một số nguyên – số thao tác đẩy ít nhất cần thực hiện.

Ví dụ:

MOVEMENT.INP
1 3 6

MOVEMENT.OUT
5

Ví dụ về một cách đẩy

WB33 NWRRC 2019 A A XVII

Giải thuật: Cơ sở lập trình .

Khi di chuyển, cần đẩy cánh cửa sang phải nhiều nhất có thể.

Một lần di chuyển, mỗi cánh cửa lần lượt đi được một quảng đường tối đa là b-a,

Ban đầu cần chuyển cánh cửa ngắn sang phải tối đa có thể,

Sau đó lặp lại nhiều lần thao tác:

- Di chuyển cánh cửa dài,
- Di chuyển cánh cửa ngắn.

Số lần di chuyển cánh cửa dài là $2\times\left\lceil\frac{n-b}{b-a}\right\rceil+1$.

Độ phức tạp của giải thuật: O(1).

Chương trình

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

int main()
{
  int a, b, n;
  fi >> a >> b >> n;
  fo<< 1 + 2 * ((n - b + (b - a) - 1) / (b - a));
}
```