Bài 1. Lễ hội

Hạn chế thời gian: 1 second
Hạn chế bộ nhớ: 256 megabytes

Có một lễ hội được tổ chức ở trường THPT chuyên Nguyễn Chí Thanh, trong đó có một cuộc thi thu hút được rất nhiều khán giả, do đó ban tổ chức rất chú tâm vào cuộc thi này.

Cuộc thi là một cuộc chạy đua tiếp sức, trường có tất cả M lớp, mỗi lớp cử N bạn học sinh tham gia cuộc thi, học sinh thứ j học ở lớp thứ i mất $a_{i,j}$ giây để chạy hết một vòng (sau khi người chơi thứ i chạy xong thì người chơi thứ $i+1$ sẽ tới tiếp sức, nếu người chơi thứ N chạy xong thì người chơi thứ 1 sẽ vào tiếp sức).

Mỗi khi có tiếng trống đánh tại giây thứ t, những bạn cổ vũ cho lớp có xếp hạng đang đứng đầu cuộc đua sẽ hô hét rất to.

Tại mỗi lần trống đánh, hãy in ra chỉ số của lớp hét cực kì to đó để ban tổ chức cộng điểm hoạt động sôi nổi cho những lớp hô to đó.

Dữ liệu

- Dòng đầu tiên chứa 3 số nguyên dương M, N, Q lần lượt mô tả số lượng lớp, số lượng học sinh của mỗi lớp và số lượng truy vấn ($M, N, Q \leq 1000$).
- M dòng tiếp theo, dòng thứ i chứa N số nguyên dương $a_{i,1}, a_{i,2}, \ldots, a_{i,N}$ mô tả thời gian chạy hết một vòng của N bạn ở lớp i ($a_{i,j} \leq 10^9$).
- Q dòng cuối cùng, dòng thứ k gồm 1 số nguyên dương t_k mô tả truy vấn thứ k ($t_k \leq 10^{12}$).

Kết quả

- In ra Q dòng, dòng thứ i là chỉ số lớp hét to nhất tại thời điểm thứ t_i.

Ví dụ

lehoi.inp	lehoi.out
3 4 2 1 2 3 5 4 3 2 1 5 5 5 5 5 11	1 2

Giải thích

- Tại giây thứ 5, lớp thứ 1 đã chạy đến vòng thứ 3 (người chơi thứ 3 đang chạy được $\frac{2}{3}$ quãng đường). Trong khi đó lớp thứ 2 mới chạy đến vòng thứ 2 (người chơi thứ 2 đang chạy được $\frac{1}{3}$ quãng đường) và lớp thứ 3 mới chạy xong vòng thứ 1.

- Tại giây thứ 11, lớp thứ 1 chạy xong 4 vòng, lớp thứ 2 chạy đến vòng thứ 5 (người chơi 1 đang chạy được $\frac{1}{4}$ quãng đường), lớp thứ 3 đang chạy đến vòng thứ 3 (người chơi 3 chạy được $\frac{1}{5}$ quãng đường).

Lễ hội

Tutorial

Nhận thấy giới hạn số lượng truy vấn là 1000 và giới hạn số lớp là 1000, ta có thể đoán ra được độ phức tạp của thuật toán cần cài đặt là $O(M \times Q)$ trở lên.

Từ đó ta có thể nghĩ ra được hướng giải quyết là với mỗi truy vấn, ta sẽ đếm số vòng và độ dài người cuối cùng đã chạy được cho từng lớp. Với mỗi lớp ta sẽ tính số lượng vòng mà cả N người đã chạy được, rồi lấy t_k chia lấy dư cho tổng thời gian chạy của N người đó, rồi ta sử dụng tìm kiếm nhị phân để tìm vị trí người đang chạy.

Solution

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

using ll = long long;
#define fi first
#define se second
const int MAX = 1e3 + 7;

int m, n, q;
int a[MAX][MAX];

ll f[MAX][MAX];

struct Phanso {
    ll a, b;
    Phanso(ll tu = 1, ll mau = 1) {
        a = tu, b = mau;

        ll gcd = __gcd(a, b);
        a /= gcd;
        b /= gcd;
    }

    bool operator < (const Phanso &x) const {
        return a * x.b < x.a * b;
    }
};

pair<ll, Phanso> dodai[MAX];
```

```
int process(ll t) {
    for (int i = 1; i <= m; i ++) {
        ll d = n * (t / f[i][n]);
        ll cur_t = t % f[i][n];
        int u = upper_bound(f[i] + 1, f[i] + 1 + n, cur_t) - f[i] + 1;

        d += u - 1;
        cur_t -= f[i][u - 1];

        dodai[i].fi = d;
        dodai[i].se = Phanso(cur_t, a[i][u]);
    }

    pair<ll, Phanso> dmax = dodai[1];
    int ans = 1;

    for (int i = 2; i <= n; i ++) if (dmax < dodai[i]) {
        dmax = dodai[i];
        ans = i;
    }
    return ans;
}

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

    cin >> m >> n >> q;
    for (int i = 1; i <= m; i ++) {
        for (int j = 1; j <= n; j ++) cin >> a[i][j];
    }

    for (int i = 1; i <= m; i ++) {
        for (int j = 1; j <= n; j ++) f[i][j] = a[i][j] + f[i][j - 1];
    }

    while (q --) {
        ll t; cin >> t;
        cout << process(t) << '\n';
    }
}
```

Bài 2. Magicka

Hạn chế thời gian: 1 seconda
Hạn chế bộ nhớ: 256 megabytes

Bạn vừa bị công ti sa thải sau đợt khủng hoảng kinh tế vừa rồi, quá buồn nên bạn đi giải sầu tại quán bar. Khi đi về bạn băng qua đường thì bị xe tải mất lái (sau đó như thế nào thì bạn cũng biết rời đấy). Bạn tưởng đã ngỏm nhưng không, bạn đã được isekai qua một thế giới nơi tràn đầy phép thuật.

Ở thế giới này có N loại **phép thuật sơ cấp** với nhau được đánh số từ 1 tới N, từ những phép thuật sơ cấp đó ta có thể kết hợp với nhau để tạo ra các phép thuật cấp cao, người ta định nghĩa **phép thuật cấp** K là phép thuật được tạo ra bởi K phép thuật sơ cấp (các phép thuật sơ cấp không cần phải khác nhau, và việc thay đổi thứ tự kết hợp thì đều tạo ra chung một phép thuật).

Tuy nhiên không phải kết hợp theo bất kỳ cách nào được, một vài cặp phép thuật sơ cấp sau khi kết hợp sẽ triệt tiêu lẫn nhau và biến mất, các cặp phép thuật đó được gọi là **cặp phép xung khắc**. Ngoài ra có một số phép thuật sơ cấp ngoại lệ hoàn toàn mới được tạo bởi 2 phép thuật sơ cấp khác nhau, tức là thay vì 2 phép thuật sơ cấp đó kết hợp với nhau tạo thành một phép thuật cấp 2 thì lại tạo thành **phép thuật sơ cấp ngoại lệ** (viết tắt là **PTSCNL**), 2 phép đó được gọi là **cặp phép tương sinh**, mỗi phép thuật sơ cấp tạo ra PTSCNL đó được gọi là **phép hợp thành của PTSCNL**. PTSCNL này mang các đặc tính xung khắc của 2 phép thuật tạo ra nó, tức là khi kết hợp PTSCNL này với một phép thuật sơ cấp xung khắc với ít nhất một trong hai phép hợp thành của PTSCNL thì đồng thời cả 2 phép đó bị tan biến (đảm bảo rằng mỗi một phép thuật sơ cấp là phép hợp thành của nhiều nhất 1 PTSCNL).

Sau khi bạn học xong trường *Pháp sư Phổ thông* và học lên *Pháp sư hoàng gia*, để tốt nghiệp được thì bạn phải đóng góp một bài nghiên cứu, và bài nghiên cứu mà bạn chọn là Sự đa dạng của phép thuật. Nhưng rồi bạn đang gặp phải phần cực kì đau đầu mà trước khi isekai bạn đã học rất kém, đó chính là môn toán. Vấn đề được đặt ra là có tất cả bao nhiêu phép thuật cấp K có thể được tạo ra?

Dữ liệu

- Dòng đầu tiên gồm 4 số nguyên dương N, K, P, Q lần lượt là số lượng phép thuật sơ cấp, cấp bậc của phép thuật cần đếm, số cặp phép xung khắc, số cặp phép tương sinh ($N + Q, K \leq 20; P + Q \leq \frac{N \times (N-1)}{2}$).
- P dòng tiếp theo, mỗi dòng gồm thứ tự của 2 cặp phép xung khắc (Đảm bảo rằng không có 2 cặp phép nào trùng nhau).
- Q dòng cuối cùng, mỗi dòng gồm thứ tự của 2 cặp phép tương sinh (Đảm bảo rằng không có 2 cặp phép nào trùng nhau, và không có cặp phép tương sinh nào trùng với cặp phép xung khắc).

Kết quả

- In ra một số nguyên không âm duy nhất là số lượng phép cấp K khác nhau (vì đáp án có thể rất lớn nên chỉ cần in ra phần dư sau khi chia cho $10^9 + 7$).

Ví dụ

magicka.inp	magicka.out
4 3 2 1 1 2 3 4 1 4	15

Giải thích

Có 15 phép thuật cấp 3 có thể được tạo ra:

- 1 1 1
- 2 2 2
- 3 3 3
- 4 4 4
- 1 1 3
- 1 3 3
- 2 3 3
- 2 2 3
- 2 4 4
- 2 2 4
- (1 4) (1 4) (1 4)
- 1 (1 4) (1 4)
- 1 1 (1 4)
- 4 (1 4) (1 4)
- 4 4 (1 4)

Hạn chế

- **Subtask 1 (**20% **số điểm):** $P = Q = 0$.
- **Subtask 2 (**30% **số điểm):** $Q = 0$.
- **Subtask 3 (**50% **số điểm):** Không có giới hạn gì thêm.

Magicka

Tutorial

Bài này thuộc thể loại **DP Bitmask**, đặc biệt ở bài này là có thêm các cặp xung khắc và tương sinh. Cặp xung khắc thì không có gì cần xử lý, tuy nhiên cặp tương sinh thì phức tạp hơn.

Cách làm ở bài này là tạo ra một phép mới có các đặc trưng giống 2 phép của các cặp tương sinh, sau đó coi cặp tương sinh đó là cặp xung khắc (2 cặp này sẽ không thể tồn tại cùng nhau nữa bởi vì chúng sẽ ngay lập tức tạo thành PTSCNL).

Sau khi cài xong thì chỉ cần áp dụng DP BITMASK như mọi bài khác vào là xong thôi.

Solution

```cpp
#include <bits/stdc++.h>
using namespace std;
using ii = pair<int, int>;
#define BIT(n) (1 << (n))
#define GETBIT(mask, i) (((mask) >> (i)) & 1)

const int MAX = 21;
const int MOD = 1e9 + 7;

int n, k, p, q;
vector<int> a[MAX];
vector<int> adj[MAX];

int dp[BIT(MAX)][MAX];

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

    cin >> n >> k >> p >> q;
    for (int i = 1; i <= p; i ++) {
        int u, v; cin >> u >> v;
        if (u > v) swap(u, v);

        a[v].push_back(u);

        adj[u].push_back(v);
        adj[v].push_back(u);
```

```
    }
    for (int i = 1; i <= q; i ++) {
        int u, v; cin >> u >> v;
        int id = i + n;

        for (int x : adj[u]) a[id].push_back(x);
        for (int x : adj[v]) a[id].push_back(x);

        if (u > v) swap(u, v);
        a[v].push_back(u);
    }
    n += q;

    dp[0][0] = 1;

    for (int u = 1; u <= n; u ++) {
       for (int mask = BIT(u - 1); mask < BIT(u); mask ++) {
            if (GETBIT(mask, u)) break;

            bool ok = true;
            for (int v : a[u]) if (GETBIT(mask, v - 1)) {
                ok = false;
                break;
            }
            if (!ok) continue;

            for (int d = 1; d <= k; d ++) {
                for (int old_d = 0; old_d < d; old_d ++) {
                    (dp[mask][d] += dp[mask ^ BIT(u - 1)][old_d]) %= MOD;
               }
            }
        }
    }

    int ans(0);
    for (int mask = 1; mask < BIT(n); mask ++) (ans += dp[mask][k]) %= MOD;
    cout << ans;
}
```

Bài 3. Kéo rank

Hạn chế thời gian: 1 second
Hạn chế bộ nhớ: 256 megabytes

Bạn đang đạt mức rank thách đấu tại một tựa game online nọ, có 4 người bạn của bạn trình thì kém mà cứ thích rank cao, do đó 4 người bạn của bạn đều muốn bạn kéo rank, biết rằng ban đầu cả 4 người bạn của bạn đều ở mức rank 0 điểm.

Bạn cảm thấy khó xử, bởi một trận đấu bạn chỉ có thể kéo 2 người bạn lên mà thôi. Thấy thế, bụt hiện lên và nói: "bây giờ ta sẽ tiên đoán trước cho con tại trận đấu thứ i: a_i, b_i, c_i, d_i lần lượt là số điểm mà con kéo người bạn A, B, C, D nếu con chọn họ để kéo rank cho trận đấu đó.

Vì để cho công bằng, bạn muốn tìm cách chọn sao cho kết quả cuối cùng là tất cả các bạn đó đều có cùng số điểm.

Dữ liệu

- Dòng đầu tiên gồm số nguyên dương N là số lượng trận đấu mà bạn chơi ($N \leq 16$)
- 4 Dòng tiếp theo, mỗi dòng gồm N số nguyên dương lần lượt mô tả điểm nhận được nếu họ được chọn ($a_i, b_i, c_i, d_i \leq 10^5$).

Kết quả

Nếu có cách chọn sao cho cả 4 người đều cùng số điểm, hãy in ra cách chọn, ngược lại hãy in ra "Imposible"

-

Ví dụ

keorank.inp	keorank.out
3 1 3 9 2 2 1 2 2 4 2 1 4	AB AB CD

Giải thích

Trận đàu chọn A và B, trận thứ 2 chọn A và B, trận thứ 3 chọn C và D.

Hạn chế

- Có 40% số test có $N \leq 8$.

Kéo rank

Tutorial

Dễ dàng nhận thấy tại bài này sẽ áp dụng **meet in the middle**, tức là chia thành 2 tập với mỗi tập có kích thước là $\frac{n}{2}$ phần tử. Mỗi phần tử được lưu bởi 3 trạng thái là $ab = a - b$, $bc = b - c$, $cd = c - d$. Lý do là bởi gọi a_l, b_l, c_l, d_l lần lượt là tổng số điểm tại tập đầu tiên, a_r, b_r, c_r, d_r lần lượt là tổng số điểm tại tập thứ hai. khi đó đề bài sẽ yêu cầu $a_l + a_r = b_l + b_r = c_l + c_r + d_l + d_r$.

Thay đổi một chút, ta sẽ có được $a_l - b_l = b_r - a_r$, $b_l - c_l = c_r - b_r$, $c_l - d_l = d_r - c_r$, $c_l - d_l = d_r - c_r$. Khi này ta có thể tách các số của phần bên trái ra một phần tử (ab, bc, cd) và bên phải cũng tương tự. Ta sẽ đệ quy để tạo ra tất cả các trường hợp cho mỗi tập, độ phức tạp ở đây là $O(C_4^2 \times \frac{N}{2})$. Sau đó ta sẽ tìm kiếm xem có 2 phần tử giống nhau ở 2 tập khác nhau bằng cách sử dụng set.

Solution

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1679616 + 7; // 6^8 + 7
int n;
int a[4][20];

int leftlen(0), rightlen(0);
struct DeltaPoint {
    int a, b, c; // a - b, b - c, c - d
    DeltaPoint(int a = 0, int b = 0, int c = 0) : a(a), b(b), c(c) {}

    DeltaPoint Neg() {
        return DeltaPoint(-a, -b, -c);
    }

    bool operator < (const DeltaPoint &x) const {
        if (a != x.a) return a < x.a;
        if (b != x.b) return b < x.b;
        return c < x.c;
    }
    bool operator == (const DeltaPoint &x) const {
        return a == x.a && b == x.b && c == x.c;
    }
} leftpoint[MAX], rightpoint[MAX];
```

```
DeltaPoint ansneed;
bool getans = false;
bool ok = false;

void process(int &d, DeltaPoint point[], int l, int r, DeltaPoint x =
↪  DeltaPoint(0, 0, 0)) {
    if (r < l) {
        if (!getans) point[++d] = x;
        if (getans && x == ansneed) {
            ok = true;
        }
        return;
    }

    for (int i = 1; i <= 6; i ++) {
        DeltaPoint newx = x;
        string res;

        if (i == 1) { // a b
            newx.a += a[0][r] - a[1][r];
            newx.b += a[1][r];
            res = "AB";
        }
        if (i == 2) { // a c
            newx.a += a[0][r];
            newx.b -= a[2][r];
            newx.c += a[2][r];
            res = "AC";
        }
        if (i == 3) { // a d
            newx.a += a[0][r];
            newx.c -= a[3][r];
            res = "AD";
        }
        if (i == 4) { // b c
            newx.a -= a[1][r];
            newx.b += a[1][r] - a[2][r];
            newx.c == a[2][r];
            res = "BC";
        }
        if (i == 5) { // b d
            newx.a -= a[1][r];
            newx.b += a[1][r];
            newx.c -= a[3][r];

            res = "BD";
```

```
        }
        if (i == 6) { // c d
            newx.b -= a[2][r];
            newx.c += a[2][r] - a[3][r];
            res = "CD";
        }

        process(d, point, l, r - 1, newx);
        if (ok) {
            cout << res << '\n';
            return;
        }
    }
}



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

    cin >> n;
    for (int i = 0; i < 4; i ++) {
        for (int j = 1; j <= n; j ++) {
            cin >> a[i][j];
        }
    }

    process(leftlen, leftpoint, 1, n / 2);
    process(rightlen, rightpoint, n / 2 + 1, n);

    set<DeltaPoint> rpoint;

    for (int i = 1; i <= rightlen; i ++) {
        DeltaPoint x = rightpoint[i];
        x = x.Neg();
        rpoint.insert(x);
    }

    bool haveans = false;
    for (int i = 1; i <= leftlen; i ++) {
        if (rpoint.count(leftpoint[i])) {
            ansneed = leftpoint[i];
            haveans = true;
            break;
```

```
        }
    }

    if (!haveans) {
        cout << "Imposible" << '\n';
        return 0;
    }

    getans = true;
    ok = false;
    process(leftlen, leftpoint, 1, n / 2);

    ansneed = ansneed.Neg();
    ok = false;
    process(rightlen, rightpoint, n / 2 + 1, n);

    return 0;
}
```

Bài 4. Ước chung lớn nhất của dãy

Hạn chế thời gian: 1 second
Hạn chế bộ nhớ: 256 megabytes

Bạn được cho một mảng a độ dài n. Có q truy vấn, mỗi truy vấn có 2 số u và val, sau truy vấn đó a_u được tăng thêm gấp val lần, tức là $a_u = a_u * val$. Sau mỗi truy vấn bạn cần in ra một số là ước chung lớn nhất của tất cả các phần tử trong mảng a.

Vì đáp án có thể rất lớn nên bạn chỉ cần in ra phần dư cho $10^9 + 7$.

Dữ liệu

- Dòng đầu tiên gồm 2 số nguyên dương n, q $(1 \leq n, q \leq 10^5)$.
- Dòng thứ hai chứa n số nguyên dương $a_1, a_2, \ldots, a_n$ $(1 \leq a_i \leq 2 \times 10^5)$
- q dòng tiếp theo, mỗi dòng chứa 2 số nguyên dương u và val $(1 \leq u \leq n; 1 \leq val \leq 2 \times 10^5)$.

Kết quả

- Gồm q dòng, mỗi dòng in ra đáp án cho mỗi truy vấn.

Ví dụ

gcd.inp	gcd.out
4 3 1 6 8 12 1 12 2 3 3 3	2 2 6

Hạn chế

- **Subtask 1 (30% điểm):** $n, q, a_i \leq 1000$.
- **Subtask 2 (70% điểm):** Không có giới hạn gì thêm.

Ước chung lớn nhất của dãy

Tutorial

Lưu ý rằng sau mỗi truy vấn, kết quả không giảm đi, do đó chúng ta có thể giải quyết bài toán cho từng ước số nguyên tố một cách độc lập.

Với mỗi số trong mảng, ta duy trì số lần xuất hiện của tất cả các ước số nguyên tố của nó (bạn có thể sử dụng map để triển khai). Với mỗi ước số nguyên tố, ta ghi vào **multiset** số lần nó xuất hiện trong tất cả các số của mảng, đồng thời tránh thêm các giá trị **null**.

Ban đầu, $ans = 1$. Hãy hiểu cách một ước số nguyên tố p được đưa vào kết quả. Nếu kích thước của **multiset** của nó không bằng n, thì ans không thay đổi. Ngược lại, $ans = (ans \times p^x)\%mod$, trong đó x là số nhỏ nhất trong **multiset**.

Do ans không giảm, ta có thể tránh tính toán lại hoàn toàn mà chỉ cần cập nhật các ước số bị thay đổi (vì giá trị nhỏ nhất trong **multiset** cũng không giảm, nên ta chỉ cần nhân kết quả hiện tại với số mới).

Để xử lý một truy vấn, ta cần phân tích thừa số nguyên tố của x (ví dụ, dùng Sàng Eratosthenes), sau đó thêm các ước số vào map của phần tử thứ i và cập nhật **multiset** tương ứng.

Mỗi truy vấn có độ phức tạp bằng số lượng ước số nguyên tố nhân với thời gian thực hiện thao tác trên map và multiset, tức là $O(\log x)$.

Solution

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

int const maxn = 2e5 + 5, max_val = 2e5 + 5;
ll mod = 1e9 + 7, ans = 1;
int nxt[max_val], n;
multiset <int> cnt[max_val];
map <int, int> cnt_divisor[maxn];

void add(int i, int x) {
    while (x != 1) {
        int div = nxt[x], add = 0;
        while (nxt[x] == div) add++, x = x / nxt[x];

        int lst = cnt_divisor[i][div];
        cnt_divisor[i][div] += add;
```

```
        int lst_min = 0;
        if ((int)cnt[div].size() == n) {
            lst_min = (*cnt[div].begin());
        }
        if (lst != 0) {
            cnt[div].erase(cnt[div].find(lst));
        }
        cnt[div].insert(lst + add);
        if ((int)cnt[div].size() == n) {
            for (int j = lst_min + 1; j <= (*cnt[div].begin()); ++j) {
                ans = ans * (ll)div % mod;
            }
        }
    }
}

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

    int q, l, x;
    cin >> n >> q;

    for (int i = 2; i < maxn; ++i) {
        if (nxt[i] == 0) {
            nxt[i] = i;
            if (i > 10000) continue;
            for (int j = i * i; j < maxn; j += i) {
                if (nxt[j] == 0) nxt[j] = i;
            }
        }
    }

    for (int i = 1; i <= n; ++i) {
        cin >> x;
        add(i, x);
    }

    for (int i = 1; i <= q; ++i) {
        cin >> l >> x;
        add(l, x);
        cout << ans << '\n';
    }
```

```
    return 0;
}
```