DỰ ÁN INTELLECTUAL SHOWDOWN
BAN CHUYÊN MÔN

PRECONTEST LA GRANDE FINALE 2024
NĂM HỌC 2024 – 2025

HƯỚNG DẪN GIẢI

Bài 1. Đêm Đếm Đom Đóm

Subtask 1: $x = y$

Vì $x = y$ nên $lcm(a, x) = y$ với mọi a là ước của x nên chỉ cần đếm số ước của x.
Độ phức tạp: $O(\sqrt{x})$.

Subtask 2: Không có ràng buộc gì thêm

Nhận xét: với số nguyên tố p thì ta có p_y, p_a, p_x lần lượt số số ước p mà y, a, x có thì $p_y = \max(p_a, p_x)$.
Gọi ans là đáp án đề bài.
Duyệt mọi số nguyên tố p rồi xét:

- Nếu $p_x > p_y$ thì $ans = 0$.
- Nếu $p_x = p_y$ thì p_a có thể là bất kỳ số nào từ 0 tới p_x nên nhân thêm vào ans 1 lượng $p_x + 1$.
- Nếu $p_x < p_y$ thì p_a phải bằng p_y nên ans không thay đổi.

Độ phức tạp: $O(\sqrt{x})$.

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

const long long Mx = 1e6;
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    long long x, y;
    cin >> x >> y;
    long long ans = 1;
    if (y % x != 0) {
        cout << 0;
        exit(0);
    }
    for (long long i = 2; i <= Mx; i++) {
        long long cntx = 0;
        while (x % i == 0) {
            cntx++;
            x /= i;
        }
        long long cnty = 0;
        while (y % i == 0) {
            cnty++;
            y /= i;
        }
        if (cntx == cnty) ans *= cntx + 1;
    }
    if (y > 1 && y == x) ans *= 2;
    cout << ans;
    return 0;
}
```

Bài 2. Cây bí ẩn

Đầu tiên, ta đưa ra nhận xét, mảng **time_in**[] và **time_out**[] của một cây hợp lệ sẽ thỏa các tính chất sau:

- Sau khi sắp xếp các đỉnh theo thứ tự **time_in**[] không giảm, **time_in**[i] $= i$ $(1 \leq i \leq n)$.
- **time_in**[i] $\leq$ **time_out**[i] $(1 \leq i \leq n)$.
- Nếu đỉnh j là tổ tiên của đỉnh i thì **time_in**[j] $\leq$ **time_in**[i] $\leq$ **time_out**[i] $\leq$ **time_out**[j]. (1)

Từ những quan sát trên, ta nhận thấy rằng ta sẽ sắp xếp các đỉnh theo thứ tự mảng **time_in**[] không giảm để thuận tiện cho việc tính toán.

Subtask 1: $n \leq 10$

- Với mỗi đỉnh i, ta sẽ thử chọn từng đỉnh j $(j < i)$ thỏa (1) và đặt j làm cha của i. Sau khi đã duyệt qua hết n đỉnh, vì các đỉnh đã được sắp xếp theo thứ tự nên ta chỉ cần duyệt dfs qua n đỉnh và kiểm tra xem thứ tự duyệt có đúng với mảng **time_in**[] và **time_out**[] hay không.
- Độ phức tạp: $O(n!)$

Subtask 2: $n \leq 100$

- Với mỗi đỉnh i, các đỉnh thuộc cây con của i gồm các đỉnh k thỏa (1). Ta nhận xét rằng các cây con của các con j trực tiếp của i sẽ chia mảng ra thành các "block". Cụ thể hơn, khi xét qua một con j của i, ta nhận xét rằng j sẽ là cha của mọi đỉnh k thỏa (1). Do đó, ta sẽ duyệt tiếp cây con của j thông qua đệ quy và nhảy tới phần tử tiếp theo chứ không xét lại cây con của j.
- Độ phức tạp: $O(n^2)$

Subtask 3: Không có ràng buộc gì thêm

- Tương tự như subtask 2, ta sẽ duyệt qua các đỉnh nằm trong cây con của i. Tuy nhiên, lần này, ta nhận xét chỉ cần duyệt qua những đỉnh có **time_in**[] và **time_out**[] nằm trong khoảng [**time_in**[i], **time_out**[i]] thay vì duyệt qua tất các các đỉnh. Vì thế, mỗi đỉnh chỉ được duyệt qua đúng một lần, giúp giảm độ phức tạp của thuật toán trở thành $O(n)$ cho thao tác duyệt.
- Độ phức tạp: $O(n \times \log n)$

Ngoài thuật toán tụi mình nêu trên, bài này cũng có nhiều cách tiếp cận khác, ví dụ như duyệt từ các phần tử lá đi lên, phần suy nghĩ và cài đặt xin nhường cho bạn đọc.

Code mẫu

```cpp
#include <bits/stdc++.h>
#define pii pair<int, int>

using namespace std;
const int N = 5e5 + 5, MOD = 1e9 + 7;

int n;
array<int, 3> a[N];
vector<int> adj[N];
bool flag;

void dfs(int i, int pre) {
    int u = a[i][2];
    int nxt = a[i][0] + 1;
    for (; nxt <= a[i][1]; nxt = a[nxt][1] + 1) {
        int v = a[nxt][2];
        if (a[nxt][1] > a[i][1]) {
```

```
            flag = false;
            return;
        }
        adj[v].push_back(u);
        adj[u].push_back(v);
        if (a[nxt][0] != a[nxt][1]) dfs(nxt, i);
    }
}
void solve() {
    cin >> n;
    flag = true;
    for (int i = 1; i <= n; i++) {
        for (int j = 0; j < 2; j++) cin >> a[i][j];
        a[i][2] = i;
    }

    if (a[1][0] != 1 && a[1][1] != n) flag = false;
    sort(a + 1, a + n + 1);
    for (int i = 1; i <= n; i++)
        if (a[i][0] != i || a[i][1] < a[i][0]) flag = false;
    if (flag) dfs(1, 1);

    if (!flag) {
        cout << -1 << endl;
    } else {
        for (int i = 1; i <= n; i++) {
            sort(adj[i].begin(), adj[i].end());
            cout << adj[i].size() << endl;
            for (int x : adj[i]) cout << x << " ";
            cout << endl;
        }
    }
}
signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    solve();
    return 0;
}
```

Bài 3. Rót nước

Subtask 1: $n, m \leq 5 \times 10^3$

Mô phỏng các bước của bài toán: Chạy qua từng phần tử của mảng b và duyệt tính tổng đáp án với các phần tử của mảng a.
Độ phức tạp: $O(n \times m)$.

Subtask 2: $a_i, b_j \leq 10^5$ $(1 \leq i \leq n,\ 1 \leq j \leq m)$

- Nhận xét quan trọng: $\log(n) \approx 1/1 + 1/2 + \ldots + 1/n$ (còn được gọi là Harmonic series).
- Với MỖI phần tử b_i, ta có thể chạy qua các bội j của b_i (với j có dạng $j = b_i \times k,\ k \geq 0$).
- Với mỗi bội như vậy, tìm các phần tử trong mảng a có giá trị trong khoảng $[b_i \times k, b_i \times (k+1)]$, và mỗi phần tử x trong khoảng giá trị như thế sẽ đóng góp một lượng giá trị là $x - b_i \times k$ vào biến tổng.
- Thao tác trên có thể sử dụng chặt nhị phân trên mảng a sau khi được sắp xếp tăng dần.
- Ta chọn b_i có tổng bé nhất, đó chính là đáp án bài toán.

Độ phức tạp: $O(n \times \log \max_b \times \log n)$.

Code mẫu

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

const int N = 5e5, M = 5e6;

int n, m;
int a[N + 5], b[N + 5];
long long s[N + 5];

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

    for (int i = 1; i <= m; i++) {
        cin >> b[i];
    }

    sort(a + 1, a + 1 + n);
    sort(b + 1, b + 1 + m);

    for (int i = 1; i <= n; i++) s[i] = s[i - 1] + a[i];

    long long res = 1e18;
    for (int j = 1; j <= m; j++) {
        if (b[j] == b[j - 1]) continue;
        long long val = 0;
        for (int k = 0; k < a[n]; k += b[j]) {
            int l = lower_bound(a + 1, a + 1 + n, k) - a;
            int r = lower_bound(a + 1, a + 1 + n, k + b[j]) - a - 1;
            if (l == n + 1) break;

            val += (s[r] - s[l - 1]) - (1LL * k * (r - l + 1));
        }

        res = min(res, val);
    }

    cout << res;
}
```

Subtask 3: $1 \leq n \leq 2 \times 10^5, 1 \leq m \leq 2 \times 10^5$

Cải tiến từ Subtask 2, ta có thể loại bỏ việc chặt nhị phân bằng cách sử dụng mảng cộng dồn trên **giá trị** thay vì trên **mảng** a. Độc giả có thể đọc đoạn code mẫu dưới đây để hiểu cách cài đặt bài toán.

Code mẫu

```cpp
#include<bits/stdc++.h>
#define int long long
#define all(x) x.begin(), x.end()

using namespace std;
const int maxn = 5e6 + 5;
int n, m;
int a[maxn], pf[maxn][2];
vector<int> b;

signed main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
```

```
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> a[i],
        pf[a[i]][0] += a[i],
        pf[a[i]][1] += 1;
    for (int i = 1, x; i <= m; i++)
        cin >> x,
        b.emplace_back(x);
    sort(all(b));
    b.resize(unique(all(b)) - b.begin());
    for (int i = 1; i < maxn; i++)
        for (int j = 0; j < 2; j++)
            pf[i][j] += pf[i - 1][j];
    int res = 1e18;
    for (int val : b)
    {
        int cur = 0;
        for (int k = 0;; k++)
        {
            int l = k * val, r = min(maxn - 1, (k + 1) * val - 1);
            if (l >= maxn) break;
            int sum = pf[r][0] - pf[l][0], cnt = pf[r][1] - pf[l][1];
            cur += sum - l * cnt;
        }
        res = min(res, cur);
    }
    cout << res;
}
```

Bài 4. Traffics

Subtask 1: $R = 0$

Thuật toán: *Dijkstra*
Với $R = 0$ thì đồ thị không khác gì đồ thị vô hướng có trọng số bình thường, gọi $d[i]$ là đường đi ngắn nhất từ 1 đến i, với mỗi x_i đáp án sẽ là $d[x_i] \times 2$.

Độ phức tạp: $O((n+m) \times \log n)$

Subtask 2: $1 \leq k \leq 10$

Thuật toán: *Dijkstra*

- Vì k bé nên với subtask này với mỗi cô người yêu ta sẽ thực hiện 2 lần dijkstra: để tính thời gian đi từ 1 đến x_i và tính thời gian đi từ x_i về 1.
- Ta nhận xét rằng chỉ cần dijkstra tìm đường đi nhanh nhất là được, nếu gặp đèn đỏ thì cộng thời gian chờ đèn đỏ vào thời gian di chuyển.
- Xét riêng từng trường hợp x (x là nhà người yêu), gọi $d1[i]$ là thời gian ngắn nhất để đi từ 1 đến i khi, khi ở đỉnh i, ta kiểm tra xem tại thời điểm đó đang là đèn xanh hay đèn đỏ, nếu là đèn xanh thì có thể đi tiếp, ngược lại thì phải cộng thêm thời gian chờ đèn đỏ. Sau đó, gọi $d2[i]$ là thời gian ngắn nhất để đi từ x đến i, cách tính $d2$ tương tự như cách tính $d1$.
- Ban đầu memset hai mảng $d1$ $d2$ dương vô cùng. Gán $d1[1] = 0$, tính $d1$ xong, gán $d2[x] = d1[x]$ sau đó tính $d2$. Kết quả bài toán là $d2[1]$.

Độ phức tạp: $O((n+m) \times \log n \times k)$

Subtask 3: Không có ràng buộc gì thêm

Thuật toán: *Dijkstra*

Đầu tiên ta thực hiện *Dijkstra* để tìm thời gian ngắn nhất để đi từ 1 đến mọi đỉnh còn lại.
Kế tiếp, ta có thể thấy, tín hiệu đèn tại một đỉnh phụ thuộc vào thời gian hiện tại, và chỉ có tối đa $B + R$ thời điểm làm ảnh hưởng đến tín hiệu đèn. Ví dụ: G = 4, R = 5.

thời điểm	0	1	2	3	4	5	6	7	8	9
t đếm ngược	4	3	2	1	5	4	3	2	1	4
màu	xanh	xanh	xanh	xanh	đỏ	đỏ	đỏ	đỏ	đỏ	xanh

Do đó ta gọi $f[i][t]$ là thời gian ngắn nhất để đi từ 1 đến i và $f[i][t]\%(B + R) = t$.

Vì ta không thể tìm thời gian di chuyển ngắn nhất từ từng đỉnh x_i về 1 nên ta sẽ phải tính thời gian bằng cách đi từ 1 với mọi thời điểm $0 \le t \le B + R - 1$.

Với cách tính thời gian ngược (trụ đèn tại các đỉnh cũng sẽ chạy ngược).
Cuối cùng, với mỗi đỉnh x_i, $ans = \min(d[x_i] + f[x_i][t] + g(d[x_i], t))$ với $g(t1, t2)$ là giá trị bé nhất sao cho $(t1 + g(t1, t2))\%(B + R) = t2$.

Độ phức tạp: $O((n \times (B + R) + m) \times \log(n \times (B + R)))$.

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

using namespace std;
typedef pair<int, int> pii;
typedef tuple<int, int, int> tpiii;
const int INF = 1e18;
const int MX = 100005;
int n, m, k, B, R, BR;
int x[MX];
int l[MX], r[MX];
int d[MX];
int f[MX][60];
vector<pii> G[MX];

void nhap() {
    cin >> n >> m >> k >> B >> R;
    for (int i = 1; i <= k; i++) cin >> x[i];
    for (int i = 1; i <= m; i++) {
        int u, v, c;
        cin >> u >> v >> c;
        G[u].push_back({v, c});
        G[v].push_back({u, c});
    }
    for (int i = 1; i <= n; i++) {
        cin >> l[i];
        r[i] = l[i] + R - 1;
    }
}

int ttb(int x, int id) {  /// time to green
    int d = x % BR;
    return d >= l[id] && d <= r[id] ? r[id] - d + 1 : 0;
}

void dijkstra() {
    priority_queue<pii, vector<pii>, greater<pii>> Q;
    for (int i = 0; i <= n; i++) d[i] = INF;
    d[1] = 0;
    Q.push({d[1], 1});
```

```
    while (Q.size()) {
        int du, u;
        tie(du, u) = Q.top();
        Q.pop();
        if (du != d[u]) continue;
        for (pii &tmp : G[u]) {
            int v, c;
            tie(v, c) = tmp;
            c = c + d[u] + ttb(d[u], u);
            if (d[v] > c) {
                d[v] = c;
                Q.push({d[v], v});
            }
        }
    }
}

void dijkstra2() {
    priority_queue<tpiii, vector<tpiii>, greater<tpiii>> Q;
    memset(f, 0x3f, sizeof f);
    for (int i = 0; i < B + R; i++) {
        f[1][i] = 0;
        Q.push({0, i, 1});
    }
    while (Q.size()) {
        int du, t, u;
        tie(du, t, u) = Q.top();
        Q.pop();
        if (du != f[u][t]) continue;

        for (pii &tmp : G[u]) {
            int v, c;
            tie(v, c) = tmp;
            int nt = (t - c) % BR;
            if (nt < 0) nt += BR;
            if (ttb(nt, v) == 0 && f[v][nt] > du + c) {
                f[v][nt] = du + c;
                Q.push({du + c, nt, v});
            }
        }

        int nt = t - 1;
        if (nt < 0) nt += BR;
        if (f[u][nt] > du + 1) {
            f[u][nt] = du + 1;
            Q.push({du + 1, nt, u});
        }
    }
}

void solve() {
    BR = B + R;
    dijkstra();
    dijkstra2();
    for (int i = 1; i <= k; i++) {
        int x = ::x[i];
        int ans = 1e18;
        for (int j = 0; j < BR; j++) {
            int c = (j - d[x]) % BR;
            if (c < 0) c += BR;
            ans = min(ans, f[x][j] + c);
        }
        cout << ans + d[x] << ' ';
    }
}

int32_t main() {
```

```
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    nhap();
    solve();

    return 0;
}
```

Bài 5. Cắt dãy

Subtask 1: $2 \leq k, n \leq 12$

- Thử hết tất cả $\binom{n}{k}$ cách cắt dãy a và $\binom{n}{k}$ cách cắt dãy b và chọn cách tốt nhất.
- Độ phức tạp: $O(2^n + \binom{n}{k}^2 \times n)$

Subtask 2: $1 \leq k \leq 10$

- Do $k = 2$ nên dãy a và b sẽ được cắt thành 2 phần tiến tố và hậu tố. Bài toán trở thành biểu thức:

$$\max_{1 \leq i,j < n} |pre_a[i] - pre_b[j]| + |suf_a[i+1] - suf_b[j+1]|$$

- Với pre_a, pre_b là mảng tổng tiền tố của a, b và suf_a, suf_b là mảng tổng hậu tố của a, b.
- Độ phức tạp: $O(n^2)$

Subtask 3: Không có ràng buộc gì thêm

- Nhận xét: $|a - b| = \max(a - b, b - a)$
- Điều này đồng nghĩa với việc, để cực đại hóa tổng của các trị tuyệt đối, với $|a - b|$, ta chỉ cần lấy một cái mang dấu cộng và một cái mang dấu trừ. Với cách làm này, nếu một phương án nào đó vi phạm quy tắc tính trị tuyệt đối thì sẽ tự động bị hàm max loại ra.
- Mỗi trị tuyệt đối có 2 cách xét dấu, như vậy ta có 2^k cách xét dấu trị tuyệt đối. Mỗi cách xét dấu như vậy sẽ được biểu diễn bằng một dãy bit. Với bit thứ j bằng 1 nghĩa là a_j mang dấu cộng và b_j mang dấu trừ, ngược lại là a_j mang dấu trừ và b_j mang dấu cộng.
- Bài toán lúc này được đơn giản hóa thành bài toán cắt dãy a thành k đoạn sao cho các phần tử của đoạn thứ j bị đảo dấu nếu bit thứ j của $mask$ bằng 0 và cực đại hóa tổng dãy a. Làm tương tự với dãy b, nhưng với dãy bit ngược lại.
- Dễ thấy, ta có thể giải bài toán trên bằng Quy hoạch động:

$$dp_b[i][j] = \begin{cases} 0 & i = 0 \text{ và } j = 0 \\ -\infty & (i > 0 \text{ và } j = 0) \text{ hoặc } (i = 0 \text{ và } j > 0) \\ \max(dp_a[i-1][j], dp_a[i-1][j-1]) + a[i] & \text{bit thứ j của mask là 1} \\ \max(dp_a[i-1][j], dp_a[i-1][j-1]) - a[i] & \text{bit thứ j của mask là 0} \end{cases}$$

- Xây dựng mảng dp_b theo ý tưởng tương tự.
- Cuối cùng, đáp án là $dp_a[n][k] + dp_b[n][k]$.
- Độ phức tạp $O(2^k \times nk)$.

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

typedef long long ll;
typedef long double ld;
typedef pair<ll,ll> pl;
typedef pair<int,int> pii;
typedef tuple<int,int,int> tt;

#define all(a) a.begin(), a.end()
#define filter(a) a.erase(unique(all(a)), a.end())

const int mn = 1e4 + 4;
ll dp_a[mn][20], dp_b[mn][20], a[mn], b[mn];

bool getCur (int mask, int pos) {
    return mask & (1 << pos);
}

ll add (ll a, ll b) {
    if (min(a, b) == LLONG_MIN) return LLONG_MIN;
    return a + b;
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, k; cin >> n >> k;
    for (int i = 1; i <= n; i++) cin >> a[i];
    for (int i = 1; i <= n; i++) cin >> b[i];

    ll ans = LLONG_MIN;
    for (int msk = 0; msk < (1 << k); msk++) {
        int mask = msk << 1; // turn to 1-indexed
        for (int i = 0; i <= n; i++)
            for (int j = 0; j <= k; j++)
                dp_a[i][j] = dp_b[i][j] = LLONG_MIN;

        dp_a[0][0] = dp_b[0][0] = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= k; j++) {
                ll cur_a = (getCur(mask, j) ? a[i] : -a[i]);
                ll cur_b = (getCur(mask, j) ? -b[i] : b[i]);

                dp_a[i][j] = add(max(dp_a[i - 1][j], dp_a[i - 1][j - 1]), cur_a);
                dp_b[i][j] = add(max(dp_b[i - 1][j], dp_b[i - 1][j - 1]), cur_b);
            }
        }

        ans = max(ans, add(dp_a[n][k], dp_b[n][k]));
    }
    cout << ans;

    return 0;
}
```

— HẾT —