DỰ ÁN INTELLECTUAL SHOWDOWN
BAN CHUYÊN MÔN

CONTEST LA GRANDE FINALE 2024
NĂM HỌC 2024 – 2025

HƯỚNG DẪN GIẢI

Bài 1. Thế Giới Âm Nhạc

Subtask 1: $n \leq 20$

Vét cạn:

- Vì $n \leq 20$ nên ta có thể thử hết tất cả 2^n cách chọn các phần tử. Với mỗi cách chọn có số lượng phần tử là lẻ, ta lấy tổng giá trị các phần tử đó và cập nhật đáp án.

Subtask 2: $\min(a_1, a_2, \ldots, a_n) = \max(a_1, a_2, \ldots, a_n)$

Tham lam:

- Điều kiện của subtask 2 cho ta biết giá trị của tất cả phần tử của mảng a đều bằng nhau: $a_1 = a_2 = \ldots = a_n = x$.
- Do đó ta sẽ tham lam:
  - Nếu $x > 0$: đáp án sẽ là $x \times (n - n\%2)$ (với % là phép chia lấy dư).
  - Nếu $x \leq 0$: đáp án sẽ bằng 0.

Subtask 3: $a_i \geq 0$ $(1 \leq i \leq n)$

Tham lam:

- Với điều kiện tất cả các phần tử trong mảng a đều không âm, đáp án sẽ là tổng của k phần tử lớn nhất trong của mảng a với $k = n$ nếu n lẻ, nếu n chẵn thì $k = n - 1$.

Subtask 4: $a_1 \geq a_2 \geq \ldots \geq a_n$

Tham lam:

- Với các phần tử đã được sắp xếp, ta gọi i là vị trí cuối cùng sao cho $a_i > 0$.
- Nếu i là số lẻ, đáp án sẽ là: $a_1 + a_2 + \ldots + a_i$.
- Nếu i là số chẵn, đán án sẽ là: $\max(a_1 + a_2 + \ldots + a_{i-1},\ a_1 + a_2 + \ldots + a_{i+1})$ nếu $i + 1 \leq n$, ngược lại đáp án sẽ là $a_1 + a_2 + \ldots + a_{i-1}$.

Subtask 5: Ràng buộc gốc

Cách 1: Sử dụng quy hoạch động:

- $f[i][0]$ là tổng phần tử được chọn lớn nhất khi chọn chẵn phần tử từ 1 đến i.
- $f[i][1]$ là tổng phần tử được chọn lớn nhất khi chọn lẻ phần tử từ 1 đến i.
- Ta có $f[1][0] = 0$ và $f[1][1] = a[1]$.
- Công thức truy hồi $(1 < i \leq n)$:

$$f[i][0] = \max(f[i-1][0], f[i-1][1] + a[i])$$

$$f[i][1] = \max(f[i-1][1], f[i-1][0] + a[i])$$

Cách 2: Tham lam.

- Đầu tiên ta sẽ chọn hết tất cả các phần tử có giá trị dương vào tập đáp án.
- Nếu số phần tử dương là số lẻ thì ta in tổng giá trị của tập đáp án.
- Nếu số phần tử dương là số chẵn thì ta có 2 trường hợp:
  - Trường hợp 1: bỏ phần tử dương bé nhất ra khỏi tập để tập đáp án hiện tại trở thành tập có lẻ phần tử.
  - Trường hợp 2: thêm một phần tử x ($x \leq 0$) lớn nhất vào tập để tập đáp án hiện tại trở thành tập có lẻ phần tử (nếu tồn tại ít nhất một phần tử $x \leq 0$).
- Ta thử hai trường hợp và in ra trường hợp mang lại kết quả lớn nhất.

Code mẫu (Cách 1):

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

const int MX = 1000005;
int n;
int a[MX];
int f[MX][2];

signed main(){
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> a[i];
    memset(f, -0x3f, sizeof f);
    f[0][0] = 0;
    for(int i = 1; i <= n; i++){
        f[i][0] = max(f[i - 1][0], f[i - 1][1] + a[i]);
        f[i][1] = max(f[i - 1][1], f[i - 1][0] + a[i]);
    }
    cout << f[n][1];
    return 0;
}
```

Bài 2. Kỳ thi tranh tài

Subtask 1: $n \leq 20$

- Với subtask này, ta chỉ cần for qua mọi cách chọn (subsets).
- Độ phức tạp $O(2^n)$.

Subtask 2: Mọi phần tử trong mảng đều chia hết cho 3

- Ta sử dụng tham lam trong trường hợp này. Nhận thấy do mọi số đều chia hết cho 3 nên ta có thể chọn tất cả các phần tử có giá trị dương.
- Độ phức tạp $O(n)$.

Từ Subtask 3 trở đi, ta sẽ sử dụng:

- Định nghĩa hàm $f(x) = x\%3$.
- Hàm $fmax(a, b)$ sẽ được định nghĩa bằng phép gán $a = max(a, b)$.
- Định nghĩa $M = \sum_{i=1}^{k} a_{u_i} + a_{v_i}$.

Subtask 3: $n \leq 100$

- Định nghĩa $dp[i][j]$ là tổng lớn nhất tạo được đến vị trí i, và có $f(M) = j$.
- Tại vị trí i, ta xét toàn bộ các vị trí j với định nghĩa là các đoạn được chọn trước đó $\leq j$. Xét toàn bộ các k từ $j+2$ đến i, với định nghĩa là đoạn mới được chọn sẽ là $[k, i]$. Xét toàn bộ các m từ 0 tới 2, với định nghĩa là mod tạo được ở những đoạn $\leq j$. Khi này ta có công thức truy hồi như sau:

$$fmax(dp[i][f(a[i] + a[k] + m)], dp[j][m] + sum(k, i))$$

- Vì ta cần tính max của 1 tiền tố, cập nhật $fmax(dp[i][j], dp[i-1][j])$ (với $0 \leq j \leq 2$).
- Độ phức tạp: $O(n^3)$.

Subtask 4: $n \leq 1000$

- Dựa trên tư tưởng của subtask 3, ta nhận ra việc for qua các giá trị j như trên là không cần thiết. Tại vị trí i, xét toàn bộ các $j \leq i$ với định nghĩa $[j, i]$ là đoạn mới được chọn. Dễ thấy, ta sẽ lấy max với $dp[j-2][m]$ với định nghĩa m như subtask bên trên. Công thức truy hồi:

$$fmax(dp[i][f(a[i] + a[j] + m)], dp[j-2][m] + sum(j, i))$$

- Sau đó cập nhật lại $fmax(dp[i][j], dp[i-1][j]$ như subtask 3.
- Độ phức tạp $O(n^2)$.

Subtask 5: $n \leq 10^6$

- Định nghĩa $dp[i][j][0]$ là tổng lớn nhất tạo được tính tới vị trí i, giá trị $f(M)$ hiện tại là j, và đoạn gần nhất chọn **chưa** kết thúc (tức ở vị trí này vẫn tiếp tục thêm phần tử).
- Định nghĩa $dp[i][j][1]$ là tổng lớn nhất tạo được tính tới vị trí i, giá trị $f(M)$ hiện tại là j, và đoạn gần nhất chọn **đã** kết thúc (tức ở vị trí này không thêm phần tử).
- Ở mỗi vị trí i, ta sẽ duyệt qua các j có thể ($0 \leq j < 3$).
- Biết vị trí hiện tại là i, ta gọi $x = a_i$.
- Với trường hợp đoạn chưa kết thúc, ta có các trường hợp sau:
  - Tiếp tục thêm phần tử: $fmax(dp[i][j][0], dp[i-1][j][0] + x)$
  - Bắt đầu 1 đoạn mới:
    - Nếu ($i \leq 2$): $fmax(dp[i][f(x)][0], x)$
    - Nếu ($i > 2$): $fmax(dp[i][f(x+j)][0], dp[i-2][j][1] + x)$
- Với trường hợp đoạn đã kết thúc, ta có 2 trường hợp:
  - Đoạn gần nhất không phải vị trí hiện tại: $fmax(dp[i][j][1], dp[i-1][j][1])$
  - Đóng đoạn tại vị trí hiện tại: $fmax(dp[i][f(j+x)][1], dp[i][j][0])$
- Độ phức tạp: $O(n)$

Code mẫu:

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

const int N = 1e6 + 5, INF = 9999999999;
int dp[N][3][2], a[N];

void fmax(int &a, int b){
    a = max(a, b);
}

int f(int x){
    return (x + INF) % 3;
}

signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int n; cin >> n;
    for(int i = 1; i <= n; ++i)
        cin >> a[i];
    for(int i = 0; i <= n; ++i){
        for(int j = 0; j < 3; ++j)
            for(int k = 0; k < 2; ++k)
                dp[i][j][k] = -1e18;
        dp[i][0][1] = 0;
    }

    for(int i = 1; i <= n; ++i){
        int x = a[i];
        for(int j = 0; j < 3; ++j){
            fmax(dp[i][j][0], dp[i - 1][j][0] + x);
            if(i <= 2)
                fmax(dp[i][f(x)][0], x);
            else
                fmax(dp[i][f(x + j)][0], dp[i - 2][j][1] + x);
        }
        for(int j = 0; j < 3; ++j){
            fmax(dp[i][j][1], dp[i - 1][j][1]);
            fmax(dp[i][f(j + x)][1], dp[i][j][0]);
        }
    }
    cout << dp[n][0][1];
    return 0;
}
```

Bài 3. Ghép cây

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

Xét từng cặp đỉnh (i, j) trong cây A và B rồi tính đường kính của cây mới.
Độ phức tạp: $O((n \times m)^2)$.

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

- Thay vì phải tính lại đường kính của cây mới mỗi lần thì ta có thể tìm đường kính dài nhất mà chứa cạnh (i, j) rồi so sánh với 2 đường kính ban đầu để tìm đường kính của cây mới.
- Gọi da_i và db_i lần lượt là khoảng cách xa nhất từ đỉnh i trong cây A và B tới 1 đỉnh.
- Gọi D là đường kính dài nhất của cây A và B.

- Gọi ans là đáp án đề bài.
- Xét từng cặp đỉnh (i, j) trong cây A và B rồi xét:
  - Nếu $da_i + db_j + 1 \geq D$ thì ans += $da_i + db_j + 1$.
  - Nếu $da_i + db_j + 1 < D$ thì ans += D.

Độ phức tạp: $O(n \times m)$.

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

- Trong subtask 2: Với mỗi đỉnh trong cây A thì ta phải phải cần $O(m)$ để đi qua hết đỉnh trong cây B.
- Nhận xét: với mỗi đỉnh i trong cây A, tất cả đỉnh j trong cây B sẽ được chia thành **một trong hai** trường hợp được nêu ở subtask 2. Như vậy, ta chỉ cần tìm ra công thức tính nhanh cho mỗi trường hợp, thay vì xét riêng từng đỉnh trong m đỉnh.
- Nhận thấy điều kiện để chia hai trường hợp là điều kiện về quan hệ so sánh (*lớn hơn, bé hơn*), ta có thể sử dụng ý tưởng chặt nhị phân, với mục đích tìm ra "ranh giới" giữa các đỉnh rơi vào trường hợp một và các đỉnh rơi vào trường hợp hai.
- Đầu tiên sort lại mảng db, gọi sb_i là tổng từ db_i tới db_m.
- Xét từng đỉnh i trong cây A, gọi j là chỉ số bé nhất sao cho $da_i + db_j + 1 \geq D$, ta tìm được công thức: ans += $D \times (j-1) + sb_j + da_i \times (m-j+1)) + m - j + 1$. Phần giải thích công thức sẽ được để lại như một bài tập cho độc giả.

Độ phức tạp: $O(n \times log(m))$.

Code mẫu:

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

using namespace std;
const int N = 2e5 + 5;
ll dis[N];
vector<ll> adj[N];
ll n, m, da, db;
ll fi[N], se[N], out[N], best[N], pref[N];

void dfs1(ll u, ll par) {
   for (auto v: adj[u]) {
      if (v == par) continue;
      dfs1(v, u);
      if (fi[v] + 1 > fi[u]) {
         se[u] = fi[u];
         fi[u] = fi[v] + 1;
      } else if (fi[v] + 1 > se[u]) {
         se[u] = fi[v] + 1;
      }
   }
}

void dfs2(ll u, ll par) {
   for (auto v: adj[u]) {
      if (v == par) continue;
      if (fi[u] == fi[v] + 1) {
         out[v] = max(se[u] + 1, out[u] + 1);
      } else {
         out[v] = max(fi[u] + 1, out[u] + 1);
      }
      dfs2(v, u);
```

```
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i < n; i++) {
        ll a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    for (int i = 1; i < m; i++) {
        ll a, b;
        cin >> a >> b;
        a += n;
        b += n;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    dfs1(1, 0);
    dfs1(n + 1, 0);
    dfs2(1, 0);
    dfs2(n + 1, 0);
    for (int i = 1; i <= n + m; i++) {
        best[i] = max(fi[i], out[i]);
    }
    for (int i = 1; i <= n; i++) {
        da = max(da, best[i]);
    }
    for (int i = 1; i <= m; i++) {
        db = max(db, best[i + n]);
    }
    sort(best + 1, best + n + 1);
    for (int i = 1; i <= n; i++) {
        pref[i] = pref[i - 1] + best[i];
    }
    ll ans = 0;
    for (int i = 1; i <= m; i++) {
        ll lo = 1, hi = n, pos = 0;
        while (lo <= hi) {
            ll mid = (lo + hi) / 2;
            if (best[mid] + best[i + n] + 1 <= max(da, db)) {
                pos = mid;
                lo = mid + 1;
            } else {
                hi = mid - 1;
            }
        }
        ans += pos * max(da, db) + (n - pos) * best[i + n] + pref[n] - pref[pos] + n - pos;
    }
    cout << ans;
}
```

Bài 4. AND Table

Subtask 1: $1 \leq n, m \leq 50$

Duyệt qua tất cả hình chữ nhật con, để tính tổng AND trong $O(1)$, ta duy trì mảng calc$[i][j][u][v]$ là tổng AND của hình chữ nhật con có góc trên trái là ô (i, j) và góc dưới phải là ô (u, v). Ta có công thức quy hoạch động:

$$calc[i][j][u][v] = calc[i][j][u-1][v] \ \& \ calc[i][j][u][v-1] \ \& \ a[u][v]$$

Độ phức tạp: $O(n^2 \times m^2)$.

```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())

int a[100][100], calc[100][100], n, m, L, R;

bool ok (int i, int j, int u, int v) {
    int area = (u - i + 1) * (v - j + 1);
    return L <= area && area <= R;
}

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

    cin >> n >> m >> L >> R;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) cin >> a[i][j];

    int ans = 0;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            calc[i][j] = a[i][j];
            for (int u = i + 1; u <= n; u++)
                calc[u][j] = (calc[u - 1][j] & a[u][j]);
            for (int v = j + 1; v <= m; v++)
                calc[i][v] = (calc[i][v - 1] & a[i][v]);

            for (int u = i + 1; u <= n; u++)
                for (int v = j + 1; v <= m; v++)
                    calc[u][v] = ((calc[u - 1][v] & calc[u][v - 1]) & a[u][v]);

            for (int u = i; u <= n; u++)
                for (int v = j; v <= m; v++)
                    if (ok(i, j, u, v)) ans = max(ans, calc[u][v]);
        }
    }
    cout << ans;

    return 0;
}
```

Subtask 2: $1 \leq n, m \leq 300$

Nhận xét: Khi đã có một hình nhữ nhật con có diện tích $\geq L$ rồi thì việc mở rộng nó chắc chắn không cải thiện được đáp án.

Duyệt mọi cặp dòng (R_1, R_2), với mỗi lần như vậy tính dãy $p_1, p_2, \cdots, p_m$ với p_j là tổng AND của các phần tử trên cột j, dòng R_1 đến R_2, tức là:

$$p_j = a_{R_1,j} \& a_{R_1+1,j} \& \cdots \& a_{R_2,j}$$

Tính $sz_{\min}$ là giá trị nhỏ nhất sao cho $(R_2 - R_1 + 1) \times sz_{\min} \geq L$. Nếu $(R_2 - R_1 + 1) \times sz_{\min} \leq R$ thì tính tổng AND của mọi đoạn con có độ dài $sz_{\min}$ của p, có thể dùng sparse table.
Độ phức tạp: $O(n^2 \times m \log m)$.

```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())

int a[1010][1010], spt[1010][11], n, m, L, R;

int query (int l, int r) {
    int p = 31 - __builtin_clz(r - l + 1);
    return (spt[l][p] & spt[r - (1 << p) + 1][p]);
}

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

    cin >> n >> m >> L >> R;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) cin >> a[i][j];

    int ans = 0;
    for (int r1 = 1; r1 <= n; r1++) {
        vector<int> p(m + 1, (1 << 30) - 1);
        for (int r2 = r1; r2 <= n; r2++) {
            for (int j = 1; j <= m; j++) p[j] = (p[j] & a[r2][j]);
            int sz_min = L / (r2 - r1 + 1);
            if (L % (r2 - r1 + 1)) sz_min++;
            if ((r2 - r1 + 1) * sz_min > R) continue;

            for (int j = 1; j <= m; j++) spt[j][0] = p[j];
            for (int s = 1; (1 << s) <= m; s++) {
                for (int j = 1; j + (1 << s) - 1 <= m; j++) {
                    int pp = s - 1;
                    spt[j][s] = (spt[j][pp] & spt[j + (1 << pp)][pp]);
                }
            }

            for (int j = 1; j + sz_min - 1 <= m; j++)
                ans = max(ans, query(j, j + sz_min - 1));
        }
    }
    cout << ans;

    return 0;
}
```

Subtask 3: $1 \leq n, m \leq 1000$ và $R = n \times m$

Lưu ý, ở subtask này chúng ta chỉ quan tâm cận dưới L, không cần quan tâm cận trên R.

Gọi hàm $check(mask)$ là hàm tính xem có tồn tại một cách chọn ra hình chữ nhật con sao cho tổng AND là một tập cha của $mask$ (tức là $AND \& mask = mask$).

Nhận xét 1: Nếu $check(mask)$ thỏa thì $check(submask)$ cũng thỏa với $submask \& mask = submask$.

Nhận xét 2: Rõ ràng $check(mask)$ không phải là một hàm đơn điệu, nhưng ta vẫn có thể chặt nhị phân theo bit trên hàm này, dựa vào ý tưởng tham lam đó là ta sẽ cố gắng bật bit lớn nhất của $mask$ lên. Nếu có thể, việc bật bit lớn nhất lên chắc chắn sẽ tốt hơn tất cả các trường hợp có thể xảy ra tiếp theo nếu ta không bật bit đó.

Tóm lại, ta có thể chặt nhị phân theo cách như sau:

```
int ans = 0;
for (int i = 29; i >= 0; i--)
    if (check(ans | (1 << i))) ans |= (1 << i);
```

Nhận xét 3: Để tính hàm $check(mask)$, ta nhận xét rằng một tập các số có tổng AND là tập cha của $mask$ khi mọi phần tử trong tập đó là tập cha của $mask$.

Ta tạo bảng b có cùng kích thước với a sao cho $b_{i,j} = 1$ nếu $a_{i,j}$ là tập cha của $mask$, ngược lại thì $b_{i,j}$. Nhiệm vụ còn lại là tìm hình chữ nhật con lớn nhất trong bảng b, như vậy bài toán quy về bài Hình chữ nhật 0 1 trên VNOJ.

Độ phức tạp: $O(n \times m \times \alpha(m))$ hoặc $O(mn)$ tùy cách cài đặt.

```
#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())

int a[1010][1010], n, m, S;

bool check (int mask) {
    vector<vector<bool>> b(n + 1, vector<bool>(m + 1));
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            b[i][j] = ((a[i][j] & mask) == mask ? 1 : 0);

    int best = 0;

    vector<int> h(m + 2);
    h[0] = h[m + 1] = -1;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) h[j] = (b[i][j] ? h[j] + 1 : 0);
        vector<int> lb(m + 1), rb(m + 1);

        stack<int> st; st.push(0);
        for (int j = 1; j <= m; j++) {
            while (st.size() && h[st.top()] >= h[j]) st.pop();
            lb[j] = st.top() + 1;
            st.push(j);
        }

        while (st.size()) st.pop(); st.push(m + 1);
        for (int j = m; j >= 1; j--) {
            while (st.size() && h[st.top()] >= h[j]) st.pop();
            rb[j] = st.top() - 1;
            st.push(j);
        }

        for (int j = 1; j <= m; j++)
```

```
            best = max(best, h[j] * (rb[j] - lb[j] + 1));
    }
    return best >= S;
}

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

    int R; cin >> n >> m >> S >> R;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) cin >> a[i][j];

    int ans = 0;
    for (int i = 29; i >= 0; i--)
        if (check(ans | (1 << i))) ans |= (1 << i);
    cout << ans;

    return 0;
}
```

Subtask 4: $1 \leq n, m \leq 1000$ và $0 \leq a_{i,j} \leq 1$

Ở subtask này, đáp án chỉ có thể là 0 hoặc 1. Do đó ta tạo một hàm kiểm tra xem có tồn tại một hình chữ nhật con trong bảng a sao cho:

- Diện tích nằm trong khoảng $[L; R]$.
- Chỉ chứa toàn số 1.

Để viết hàm kiểm tra, ta duyệt qua từng dòng i và đặt là biên dưới của các hình chữ nhật đang được duyệt và duy trì một mảng h có độ dài m với h_j là độ cao lớn nhất sao cho tất cả các giá trị ở cột j, từ dòng $i - h_j + 1$ đến dòng i đều có giá trị là 1.

Ta sẽ thử mọi độ cao của một hình chữ nhật (từ 1 đến n), với một độ cao H bất kì, tìm đoạn dài nhất các vị trí liên tiếp thỏa $h_j \geq H$, gọi là $sz_{\max}$. Như vậy, ta đã tìm được các hình chữ nhật $H \times 1, H \times 2, H \times 3, \cdots, H \times sz_{\max}$. Nếu có một hình chữ nhật nào có diện tích nằm trong khoảng $[L; R]$ thì ta đã tìm được một hình chữ nhật con thỏa.

Để duy trì các đoạn các vị trí liên tiếp thỏa $h_j \geq H$, ta duyệt H từ n trở về 1 và sử dụng cấu trúc dữ liệu (CTDL) Disjoint Set Union (DSU), có thể sử dụng kỹ thuật khác để có đô phức tạp $O(1)$.

Độ phức tạp: $O(n(n+m) \times \alpha(m))$ hoặc $O(n(n+m))$ tùy cách cài đặt.

```
#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())

int a[1010][1010], n, m, L, R;

struct DSU {
    vector<int> lab, weight;
    int best;

```

```
    DSU (int sz) : lab(sz + 1, -1), weight(sz + 1), best(0) {}

    int get (int u) {
        if (lab[u] < 0) return u;
        return lab[u] = get(lab[u]);
    }

    void incr (int u) {
        weight[u]++, best = max(best, weight[u]);
    }

    int getWeight (int u) {
        return weight[get(u)];
    }

    int getBest() {
        return best;
    }

    void unite (int a, int b) {
        a = get(a), b = get(b);
        if (a == b) return;
        if (-lab[a] < -lab[b]) swap(a, b);
        weight[a] += weight[b], lab[a] += lab[b], lab[b] = a;
        best = max(best, weight[a]);
    }
};

bool check() {
    vector<int> h(m + 1);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) h[j] = (a[i][j] ? h[j] + 1 : 0);

        vector<vector<int>> pos(n + 1);
        for (int j = 1; j <= m; j++)
            pos[h[j]].push_back(j);

        DSU dsu(m + 1);
        for (int high = n; high >= 1; high--) {
            for (int p : pos[high]) {
                dsu.incr(p);
                if (p > 1 && dsu.getWeight(p - 1)) dsu.unite(p - 1, p);
                if (p < m && dsu.getWeight(p + 1)) dsu.unite(p, p + 1);
            }

            int sz_max = min(dsu.getBest(), R / high);
            if (L <= sz_max * high) return 1;
        }
    }
    return 0;
}

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

    cin >> n >> m >> L >> R;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) cin >> a[i][j];

    cout << check();

    return 0;
}
```

Subtask 5: $1 \leq n, m \leq 1000$

Cuối cùng, ta dùng nhận xét ở subtask 3 để chặt nhị phân trên hàm *check* của subtask 4 để ra được thuật toán chuẩn cho bài toán.
Độ phức tạp: $O(n(n+m)\log a_i \times \alpha(m))$ hoặc $O(n(n+m)\log a_i)$ tùy cách cài đặt.

```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())

int a[1010][1010], n, m, L, R;

struct DSU {
    vector<int> lab, weight;
    int best;

    DSU (int sz) : lab(sz + 1, -1), weight(sz + 1), best(0) {}

    int get (int u) {
        if (lab[u] < 0) return u;
        return lab[u] = get(lab[u]);
    }

    void incr (int u) {
        weight[u]++, best = max(best, weight[u]);
    }

    int getWeight (int u) {
        return weight[get(u)];
    }

    int getBest() {
        return best;
    }

    void unite (int a, int b) {
        a = get(a), b = get(b);
        if (a == b) return;
        if (-lab[a] < -lab[b]) swap(a, b);
        weight[a] += weight[b], lab[a] += lab[b], lab[b] = a;
        best = max(best, weight[a]);
    }
};

bool ok (int mask) {
    vector<vector<bool>> b(n + 1, vector<bool>(m + 1));
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            b[i][j] = ((a[i][j] & mask) == mask ? 1 : 0);

    vector<int> h(m + 1);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) h[j] = (b[i][j] ? h[j] + 1 : 0);

        vector<vector<int>> pos(n + 1);
        for (int j = 1; j <= m; j++)
            pos[h[j]].push_back(j);

        DSU dsu(m + 1);
```

```
        for (int high = n; high >= 1; high--) {
            for (int p : pos[high]) {
                dsu.incr(p);
                if (p > 1 && dsu.getWeight(p - 1)) dsu.unite(p - 1, p);
                if (p < m && dsu.getWeight(p + 1)) dsu.unite(p, p + 1);
            }

            int sz_max = min(dsu.getBest(), R / high);
            if (L <= sz_max * high) return 1;
        }
    }
    return 0;
}

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

    cin >> n >> m >> L >> R;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) cin >> a[i][j];

    int ans = 0;
    for (int i = 29; i >= 0; i--)
        if (ok(ans | (1 << i))) ans |= (1 << i);
    cout << ans;

    return 0;
}
```

Bài 5. Đại Hòa Nhạc

Subtask 1: $1 \leq n, c, q \leq 5000$

Thực hiện mô phỏng. Vì mỗi sóng âm chỉ đi $O(n)$ bước và có tối đa $O(q)$ sóng âm được sinh ra, độ phức tạp là $O(nq)$.

Subtask 2: $c = 1$

Nhận xét: Nếu cho sóng âm di chuyển theo một thứ tự ưu tiên: sóng âm gần gốc hơn di chuyển trước và sóng âm xa gốc hơn đi sau, thì khi một sóng âm đi vào một đỉnh đã đầy âm lượng thì sóng âm này sẽ không còn đóng góp gì thêm trong tương lai nữa (không thay đổi thêm âm lượng của bất kỳ đỉnh nào nữa) → Có thể xóa (dừng theo dõi) sóng âm này.

Do đó ta thực hiện mô phỏng: mỗi đỉnh chỉ có thể có tối đa c sóng âm đi vào do đó độ phức tạp là $O((nc + q) \times \log n)$, $\log n$ đến từ việc phải lưu thông tin tổng âm lượng trong cây con gốc u bằng một CTDL đơn giản (Ví dụ như Fenwick Tree) để trả lời truy vấn *count* u.

Subtask 3: $c = 10^5$

Notations:

- $begin(v)$ và $end(v)$ là thứ tự vào và thoát đỉnh v trong trình tự DFS của cây (độc giả có thể tìm hiểu về *Euler Tour*).
- $depth(v)$ là độ sâu của đỉnh v trong cây, tăng dần từ gốc đến lá.
- $wave\_starting\_position$ là vị trí (đỉnh) của sóng âm tại thời điểm *clap*.
- $wave\_current\_position$ là vị trí (đỉnh) của sóng âm tại thời điểm hiện tại.

Xét một truy vấn *count* u:

- Các sóng âm có ảnh hưởng âm lượng lên cây con gốc u này phải xuất phát trong cây con gốc u: $begin(u) \leq begin(wave\_starting\_position) < end(u)$. Do đó chỉ cần quan tâm tới các sóng âm trong khoảng này.
- Ta sẽ muốn phân tách sóng âm ra làm 2 nhóm: sóng âm vẫn còn trong cây con và sóng âm đã đi ra khỏi cây con.
  - Sóng âm vẫn còn trong cây con: $depth(wave\_current\_position) \geq depth(u)$. Với mỗi sóng âm này, nó sẽ đóng góp vào đáp án một lượng:
  $$depth(wave\_starting\_position) - depth(wave\_current\_position)$$
  - Sóng âm đã ra khỏi cây con: $depth(wave\_current\_position) < depth(u)$. Với mỗi sóng âm này, nó sẽ đóng góp vào đáp án một lượng:
  $$depth(wave\_starting\_position) - depth(u)$$
- Qua đó ta thấy cần phải theo dõi sóng âm qua 2 chiều là $begin(wave\_starting\_position)$ và $depth(wave\_current\_position)$ để có thể xem xét và phân loại các sóng âm giúp trả lời truy vấn. Và khi trả lời truy vấn *count* thì ta sẽ cần số lượng sóng, tổng $depth(wave\_starting\_position)$, tổng $depth(wave\_current\_position)$ của nhóm sóng âm mục tiêu → Ta cần một CTDL 2 chiều, ở đây tác giả sử dụng **Fenwick2D (BIT2D)** nén.

Nếu bạn tinh ý, bạn sẽ để ý $depth(wave\_current\_position)$ bị biến động bởi phép *travel*, để có thể giải quyết vấn đề này và tiếp tục duy trì được sóng này trong CTDL thì có thể sử dụng kỹ thuật tịnh tiến ngay từ khi sự kiện *clap* xảy ra: lưu trong CTDL bằng một *depth* ảo là $depth(wave\_starting\_position) + sum_d$, và tịnh tiến mọi sóng lại một lượng $-sum_d$ khi truy vấn *count* xảy ra. (sum_d là tổng của các d trong các phép *travel* cho tới sự kiện hiện tại).

Độ phức tạp là $O(n + q \times \log n \times \log(n \times q))$.

Subtask 4: Không có giới hạn gì thêm

Kết hợp nhận xét ở subtask 2 và CTDL ở subtask 3.
Ta sẽ muốn chia sóng âm làm 2 loại: sóng âm đã dừng đóng góp trong tương lai, và sóng âm vẫn tiếp tục đóng góp.

- Đối với các sóng âm tiếp tục đóng góp, sẽ là hoàn toàn ổn nếu ta lưu chúng trong CTDL của subtask 3 để tính toán tác động của loại sóng âm này (bởi những tác động của những sóng âm này không bị giới hạn bởi giới hạn âm lượng).
- Đối với các sóng âm đã dừng đóng góp: những tác động còn sót (trước khi xóa) là **tĩnh, những sóng âm này không còn di chuyển nên việc sử dụng kỹ thuật tịnh tiến ở subtask 3 sẽ là không ổn → Ta nên xóa nó khỏi CTDL của subtask 3 và lưu chúng vào một CTDL riêng biệt (có thể tiếp tục dùng *Fenwick2D* hoặc một cấu trúc khác như *Segment Tree* trên cây).**
- **Khi trả lời truy vấn *count*, đáp án sẽ là tổng của tác động của 2 loại sóng âm trên.**

Do đó, với mỗi sóng âm, ta nên tìm được thời điểm và vị trí mà sóng âm đó dừng đóng góp trong tương lai. Và khi tới thời điểm đó thì thực hiện chuyển đổi CTDL cho sóng âm này.
Độ phức tạp là $O(n + q \times \log n \times \log(n \times q))$.

Code mẫu:

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

using namespace std;

const int N = 1e5 + 10;

int n, C, q, m;
vector <int> adj[N];
int par[20][N];
int in[N], out[N], cnt = 0;
int d[N];
int hold[N];


void dfs(int x, int p, int dep = 0) {
    par[0][x] = p;
    d[x] = dep;
    in[x] = ++cnt;
    for (int i = 1; i < 20; i++)
        par[i][x] = par[i-1][par[i-1][x]];
    for (int i : adj[x])
        if (i != p)
            dfs(i, x, dep + 1), hold[x] = i;
    out[x] = cnt;
}

int find_par(int x, int k) {
    for (int i = 19; i >= 0; i--) {
        if ((1 << i) <= k) {
            k -= (1 << i);
            x = par[i][x];
        }
    }
    return x;
}

array<int, 3> operator+ (array<int, 3> x, array<int, 3> y) {
    return {x[0] + y[0], x[1] + y[1], x[2] + y[2]};
}
array<int, 3> operator- (array<int, 3> x, array<int, 3> y) {
    return {x[0] - y[0], x[1] - y[1], x[2] - y[2]};
}

struct Fenwick2D {
    int n;
    long long m;
    vector<vector<array<int, 3>>> bit;
    vector<vector<long long>> f;
    Fenwick2D(int n, long long m) : n(n), m(m), bit(n + 1), f(n + 1) {}
    void fakeAdd(int i0, long long j0) {
        for (int i = i0 + 1; i <= n; i += i & -i) {
            for (long long j = j0 + 1; j <= m; j += j & -j) {
                f[i].push_back(j);
            }
        }
    }
    void work() {
        for (int i = 1; i <= n; i++) {
            f[i].push_back(0);
            sort(f[i].begin(), f[i].end());
            f[i].resize(unique(f[i].begin(), f[i].end()) - f[i].begin());
            bit[i].resize(f[i].size(), array<int, 3>({0, 0, 0}));
        }
    }
```

```
    void add(int i0, long long j0, array<int, 3> x) {
        for (int i = i0 + 1; i <= n; i += i & -i) {
            for (int j = lower_bound(f[i].begin(), f[i].end(), j0 + 1) - f[i].begin(); j < (int
    )f[i].size(); j += j & -j) {
                bit[i][j] = bit[i][j] + x;
            }
        }
    }
    array<int, 3> sum(int i0, long long j0) { // [i, j0)
        array<int, 3> res = array<int, 3>({0, 0, 0});
        for (int i = i0; i > 0; i -= i & -i) {
            for (int j = (int)(upper_bound(f[i].begin(), f[i].end(), j0) - f[i].begin()) - 1; j
     > 0; j -= j & -j) {
                res = res + bit[i][j];
            }
        }
        return res;
    }
    array<int, 3> sum(int x1, int y1, int x2, int y2) { // [x1, x2), [y1, y2)
        return sum(x2, y2) - sum(x1, y2) - sum(x2, y1) + sum(x1, y1);
    }
};

int lazy[N];

multiset <array<int, 3>> nodes[N];

void dfs2(int x, int p) {
    for (int i : adj[x])
    if (i != p) {
        dfs2(i, x);
        if (nodes[x].size() < nodes[i].size())
            swap(nodes[x], nodes[i]);
        for (auto j : nodes[i])
            nodes[x].insert(j);
    }
    while (nodes[x].size() > C) {
        auto j = *nodes[x].rbegin();
        nodes[x].erase(nodes[x].find(j));
        int id = j[1];
        lazy[id] = hold[x];
    }

}


signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n >> C >> q;
    for (int i = 1; i < n; i++) {
        int u, v; cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    fill_n(lazy, q+1, 1);
    dfs(1, 0);
    vector <pair<string,int>> qu(q);
    int m = 2e10;
    Fenwick2D f = Fenwick2D (n + 10, m);

    for (int i = 0, T = 0; i < q; i++) {
        cin >> qu[i].first >> qu[i].second;
        if (qu[i].first == "travel")
            T += qu[i].second;
        else if (qu[i].first == "clap") {
            int u = qu[i].second;
```

```
            if (u != 1) nodes[par[0][u]].insert({d[u] + T, i});
        }
    }

    dfs2(1, 1);
    for (int i = 0, T = 0; i < q; i++) {
        if (qu[i].first == "travel")
            T += qu[i].second;
        else if (qu[i].first == "clap") {
            int u = qu[i].second;
            int v = lazy[i];
            f.fakeAdd(in[u],(d[u] + T));
            if (v != 0)
                f.fakeAdd(in[v], (d[v] + T + d[u] - d[v]));
        }
    }
    f.work();

    multiset<array<int, 4>> ms;
    for (int i = 0, T = 0; i < q; i++) {
        if (qu[i].first == "travel") {
            T += qu[i].second;
            while (!ms.empty() && (*ms.begin())[0] <= T) {
                auto j = *ms.begin();
                ms.erase(ms.find(j));
                int v = j[1];
                int T = j[0];
                f.add(in[v], (d[v] + T), array<int, 3>({-1, T, -d[v]}));
            }
        }
        else if (qu[i].first == "clap") {
            int u = qu[i].second;
            int v = lazy[i];
            f.add(in[u], (d[u] + T), array<int, 3>({1, -T, d[u]}));
            if (v != 0) {
                int T2 = T + d[u] - d[v];
                ms.insert({T2, v});
            }
        }
        else {
            int u = qu[i].second;
            int L = in[u]; int R = out[u]; int R2 = m - (d[u] + T);
            array<int, 3> val1 = f.sum(L, d[u] + T, R + 1, m);
            array<int, 3> val2 = f.sum(L, 0, R + 1, d[u] + T);
            int ans = val1[1] + val2[2] - d[u] * val2[0] + T * val1[0];
            cout << ans << endl;
        }
    }
}
```

— HẾT —