Mục lục

1 Bài A - IT

1.1 Hướng dẫn

Bài này ta cần thực hiên 2 thao tác là tăng giá trị của phần tử ở 1 ví trị trong mảng và truy vấn giá trị lớn nhất. Nên ta cần xây dựng 1 cây segment tree có chức năng tăng giá trị của 1 phần tử ở 1 vị trí và giá trị của node trên segment tree sẽ lưu phần tử lớn nhất trong đoạn mà node đó quản lý.

1.2 Code mẫu

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

using namespace std;

const int MAX_N = 1e5 + 5;
const long long inf = 1e18;

int n, q;
long long ST[4 * MAX_N]; // mang segment tree

void update(int id, int l, int r, int pos, int k) {
    if (pos > r || pos < l) {
        return;
    }

    if (l == r) {
        ST[id] += k;
        return;
    }

    int mid = (l + r) / 2;
    update(id * 2, l, mid, pos, k);
    update(id * 2 + 1, mid + 1, r, pos, k);

    ST[id] = max(ST[id * 2], ST[id * 2 + 1]);
}

long long query(int id, int l, int r, int u, int v) {
    if (u > r || v < l) {
        return -inf;
    }

    if (u <= l && r <= v) {
        return ST[id];
    }
```

```
    int mid = (l + r) / 2;
    return max(query(id * 2, l, mid, u, v), query(id * 2 + 1, mid + 1, r, u, v));
}

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

    cin >> n >> q;
    while (q--) {
        int type;
        cin >> type;

        if (type == 0) {
            int pos, k;
            cin >> pos >> k;
            update(1, 1, n, pos, k);
        }
        else {
            int x, y;
            cin >> x >> y;
            cout << query(1, 1, n, x, y) << '\n';
        }
    }

    return 0;
}
```

Đọc code đầy đủ hơn với chú thích ở *đây*

2 Bài B - Nklineup

2.1 Hướng dẫn

Bài này ta cần lấy ra được số lớn nhất và số nhỏ nhất trong đoạn từ A đến B, vậy nên ta sẽ cài 2 cây segment tree, một cây để lấy max trong đoạn, cây còn lại để lấy min trong đoạn.

2.2 Code mẫu

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

using namespace std;

const int MAX_N = 2e5 + 5;
const int inf = 1e9;

int n, q;
int a[MAX_N];
int ST_max[4 * MAX_N], ST_min[4 * MAX_N];

void build(int id, int l, int r) {
    if (l == r) {
        ST_min[id] = ST_max[id] = a[l];
        return;
    }

    int mid = (l + r) / 2;
    build(id * 2, l, mid);
    build(id * 2 + 1, mid + 1, r);

    ST_min[id] = min(ST_min[id * 2], ST_min[id * 2 + 1]);
    ST_max[id] = max(ST_max[id * 2], ST_max[id * 2 + 1]);
}

int getMin(int id, int l, int r, int u, int v) {
    if (u > r || v < l) {
        return inf;
    }

    if (u <= l && r <= v) {
        return ST_min[id];
    }

    int mid = (l + r) / 2;
    return min(getMin(id * 2, l, mid, u, v), getMin(id * 2 + 1, mid + 1, r, u, v));
}

int getMax(int id, int l, int r, int u, int v) {
```

```
    if (u > r || v < l) {
        return -inf;
    }

    if (u <= l && r <= v) {
        return ST_max[id];
    }

    int mid = (l + r) / 2;
    return max(getMax(id * 2, l, mid, u, v), getMax(id * 2 + 1, mid + 1, r, u, v));
}

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

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

    build(1, 1, n);

    while (q--) {
        int A, B;
        cin >> A >> B;
        cout << getMax(1, 1, n, A, B) - getMin(1, 1, n, A, B) << '\n';
    }

    return 0;
}
```

Đọc code đầy đủ hơn với chú thích ở *đây*

3 Bài C - LAZY

3.1 Hướng dẫn

Bài này có cách cài không cần sử dụng segment tree lazy, nhưng ở đây sẽ hướng dẫn cách cài lazy (có thể đọc thêm ở *đây*). Tương tự với bài A, mỗi node của segment tree ta sẽ lưu lại giá trị lớn nhất của đoạn mà nó quản lý.

3.2 Code mẫu

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

using namespace std;

const int MAX_N = 1e5 + 5;
const long long inf = 1e18;

int n, m;
long long ST[4 * MAX_N], lazy[4 * MAX_N];

void down(int id) {
    long long k = lazy[id];
    if (k) {
        lazy[id] = 0;
        lazy[id * 2] += k;
        ST[id * 2] += k;
        lazy[id * 2 + 1] += k;
        ST[id * 2 + 1] += k;
    }
}

void update(int id, int l, int r, int u, int v, int val) {
    if (u > r || v < l) {
        return;
    }

    if (u <= l && r <= v) {
        ST[id] += val;
        lazy[id] += val;
        return;
    }

    down(id);

    int mid = (l + r) / 2;
    update(id * 2, l, mid, u, v, val);
    update(id * 2 + 1, mid + 1, r, u, v, val);
```

```
    ST[id] = max(ST[id * 2], ST[id * 2 + 1]);
}

long long get(int id, int l, int r, int u, int v) {
    if (u > r || v < l) {
        return -inf;
    }

    if (u <= l && r <= v) {
        return ST[id];
    }

    down(id);

    int mid = (l + r) / 2;
    return max(get(id * 2, l, mid, u, v), get(id * 2 + 1, mid + 1, r, u, v));
}

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

    cin >> n >> m;
    while (m--) {
        int type;
        cin >> type;

        if (type == 0) {
            int x, y, k;
            cin >> x >> y >> k;
            update(1, 1, n, x, y, k);
        }
        else {
            int x, y;
            cin >> x >> y;
            cout << get(1, 1, n, x, y) << '\n';
        }
    }

    return 0;
}
```

Đọc code đầy đủ hơn với chú thích ở *đây*

4 Bài D - LIS

4.1 Hướng dẫn

Bài này ta có thuật toán O(n^2) là gọi dp_i = độ dài dãy con dài nhất mà số kết thúc là số a_i, công thức của ta là $dp_i = max(dp_j + 1)$ với mọi $1 \le j < i$ sao cho $a_j \le a_i$.
Ta có thể cải tiến độ phức tạp của thuật toán bằng cách tạo mảng f_k sẽ lưu lại giá trị lớn của dp_j sao cho a_j = k với mọi $j < i$, bây giờ công thức tính sẽ là $dp_i = max(f_j) + 1$ với mọi $1 \le j \le a_i$. sau đó lại cập nhật $f_{a_i} = max(f_{a_i}, dp_i)$. Ta có thể thực hiện điều này bằng cách đưa mảng f lên segment tree để cập nhật 1 vị trí và truy vấn số lớn nhất trong 1 đoạn nhanh hơn.
Tuy nhiên là $1 \le a_i \le 10^9$ nên ta không đủ bộ nhớ để lưu trữ được. Để giải quyết điều này ta có thể tạo mảng b_i = vị trí tương đối của a_i trong mảng a. Vậy bây giờ công thức dp của ta sẽ là $dp_i = max(f_{b_i}) + 1$ rồi cập nhật $f_{b_i} = max(f_{b_i}, dp_i)$ và đưa mảng f lên segment tree để có độ phức tạp là O($nlogn$).

4.2 Code mẫu

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

using namespace std;

const int MAX_N = 1e5 + 5;

int n;
pair<int, int> a[MAX_N];
int b[MAX_N];
int dp[MAX_N];
int ST[4 * MAX_N];

void update(int id, int l, int r, int i, int val) {
        if (i > r || i < l) {
                return;
        }

        if (l == r) {
                ST[id] = max(ST[id], val);
                return;
        }

        int mid = (l + r) / 2;

        update(id * 2, l, mid, i, val);
        update(id * 2 + 1, mid + 1, r, i, val);

        ST[id] = max(ST[id * 2], ST[id * 2 + 1]);
```

```
}

int get(int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return 0;
        }

        if (u <= l && r <= v) {
                return ST[id];
        }

        int mid = (l + r) / 2;
        return max(get(id * 2, l, mid, u, v), get(id * 2 + 1, mid + 1, r, u, v));
}


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

        cin >> n;
        for (int i = 1; i <= n; i++) {
                cin >> a[i].first;
                a[i].second = i;
        }

        // se uu tien sort theo first truoc tuc la uu tien theo gia tri truoc
        sort(a + 1, a + n + 1);

        int id = 0;
        for (int i = 1; i <= n; i++) {
                if (a[i].first != a[i - 1].first) {
                        id++;
                }
                b[a[i].second] = id;
        }

        int ans = 0;
        for (int i = 1; i <= n; i++) {
                dp[i] = get(1, 1, n, 1, b[i]) + 1;
                ans = max(ans, dp[i]);
                update(1, 1, n, b[i], dp[i]);
        }

        cout << ans;

        return 0;
}
```

Đọc code đầy đủ hơn với chú thích ở *đây*

5 Bài E - NKINV

5.1 Hướng dẫn

Bài này ta có thể làm thuật toán O(n^2) là với mỗi vị trí i, ta chỉ cần for j ngược từ i về 1 và đếm xem có bao nhiêu vị trí j sao cho $a_j > a_i$.
Ta nhận thấy rằng cái quan trọng cần lấy chỉ là số lượng số ở trước i có giá trị lớn hơn a_i thôi. Nên ta có thể duy trì mảng cnt_k có ý nghĩa là số lượng số k ở trước vị trí i, vậy số lượng số có thể tạo ra nghịch thế với i sẽ là tổng của các cnt_k với $a_i < k \leq 10^5$ rồi tăng cnt_{a_i} lên 1 cho những lần tính toán sau.
Và từ đây ta có thể cải tiến thuật toán lên thành O($nlogn$) bằng cách đưa mảng cnt lên segment tree quản lý tổng tức 1 node sẽ lưu lại tổng của các phần tử trong đoạn mà nó quản lý.

5.2 Code mẫu

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

using namespace std;

const int MAX_N = 1e5 + 5;

int n;
int a[MAX_N];
long long ST[4 * MAX_N];

void update(int id, int l, int r, int i, int val) {
        if (i > r || i < l) {
                return;
        }

        if (l == r) {
                ST[id] += val;
                return;
        }

        int mid = (l + r) / 2;

        update(id * 2, l, mid, i, val);
        update(id * 2 + 1, mid + 1, r, i, val);

        ST[id] = ST[id * 2] + ST[id * 2 + 1];
}

long long get(int id, int l, int r, int u, int v) {
```

```
        if (u > r || v < l) {
                return 0;
        }

        if (u <= l && r <= v) {
                return ST[id];
        }

        int mid = (l + r) / 2;
        return get(id * 2, l, mid, u, v) + get(id * 2 + 1, mid + 1, r, u, v);
}


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

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

        long long ans = 0;
        for (int i = 1; i <= n; i++) {
                ans += get(1, 1, 1e5, a[i] + 1, 1e5);
                update(1, 1, 1e5, a[i], 1);
        }

        cout << ans;

        return 0;
}
```

Đọc code đầy đủ hơn với chú thích ở *đây*

6 Bài F - Kinv

6.1 Hướng dẫn

Bài này ta có thể gọi $dp_{i,k}$ = số lượng dãy nghịch thế có độ dài k và kết thúc tại phần tử a_i. Công thức là $dp_{i,k}$ = tổng của các $dp_{j,k-1}$ với mọi $1 \le j < i$ thoả $a_j > a_i$ với độ phức tạp là O(n^2k).
Trước khi cải tiến thuật toán, ta đi qua 1 bước đệm cho dễ hiểu hơn. Ta gọi mảng $cnt_{z,k}$ bằng tổng của các $dp_{j,k}$ sao cho a_j = z và $j < i$. Công thức dp của ta có thể viết thành $dp_{i,k}$ = tổng của các $cnt_{z,k-1}$ với mọi $z > a_i$. Sau khi tính dp ở vị trí i xong ta lại tăng $cnt_{a_i,k}$ thêm $dp_{i,k}$.
Ta có thể cải thiện độ phức tạp bằng duy trì cây segment tree có chức năng tăng giá trị của 1 phần tử và truy vấn ra tổng của 1 đoạn. Ta sẽ đưa mảng cnt lên segment tree thôi =)), độ phức tạp sẽ là O(nlognk).

6.2 Code mẫu

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

using namespace std;

const int MAX_N = 1e5 + 5;
const int mod = 1e9;

int n, k;
int a[MAX_N];
int dp[MAX_N][11];
int ST[11][4 * MAX_N];

void update(int j, int id, int l, int r, int i, int val) {
        if (i > r || i < l) {
                return;
        }

        if (l == r) {
                ST[j][id] = (ST[j][id] + val) % mod;
                return;
        }

        int mid = (l + r) / 2;
        update(j, id * 2, l, mid, i, val);
        update(j, id * 2 + 1, mid + 1, r, i, val);

        ST[j][id] = (ST[j][id * 2] + ST[j][id * 2 + 1]) % mod;
}
```

```
int get(int j, int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return 0;
        }

        if (u <= l && r <= v) {
                return ST[j][id];
        }

        int mid = (l + r) / 2;
        return (get(j, id * 2, l, mid, u, v) + get(j, id * 2 + 1, mid + 1, r, u, v)) % mod
}

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

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

        for (int i = 1; i <= n; i++) {
                dp[i][1] = 1;
                for (int j = 2; j <= k; j++) {
                        dp[i][j] = get(j - 1, 1, 1, n, a[i] + 1, n);
                }
                for (int j = 1; j < k; j++) {
                        update(j, 1, 1, n, a[i], dp[i][j]);
                }
        }

        int ans = 0;
        for (int i = 1; i <= n; i++) {
                ans = (ans + dp[i][k]) % mod;
        }

        cout << ans;

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

7 Bài G - GCD

7.1 Hướng dẫn

Bài này ta có thể dễ dàng làm thuật O(nk), để cải tiến ta có thể tìm cách để tính gcd của 1 đoạn liên tiếp nhanh hơn.
Nên ta sẽ sử dụng segment tree, mỗi node sẽ lưu lại gcd của các số trong đoạn mà nó quản lý.
Lưu ý là khi truy vấn ra gcd của một đoạn, lúc mà ta đi ra ngoài đoạn cần đến, giá trị mà hàm get trả lại phải là 1 số gì đó để khi lấy gcd nó sẽ không ảnh hưởng đến kết quả của mình, ta có thể xử lý bằng cách trả về a_u khi truy vấn đoạn [u; v] (xem code để hiểu hơn), chỗ này có nhiều cách để xử lý, đây chỉ là 1 trong số các cách.

7.2 Code mẫu

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

using namespace std;

const int MAX_N = 5e5 + 5;

int n, k;
int a[MAX_N];
int ST[4 * MAX_N];

void build(int id, int l, int r) {
        if (l == r) {
                ST[id] = a[l];
                return;
        }

        int mid = (l + r) / 2;
        build(id * 2, l, mid);
        build(id * 2 + 1, mid + 1, r);

        ST[id] = __gcd(ST[id * 2], ST[id * 2 + 1]);
}

int get(int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return a[u];
        }

        if (u <= l && r <= v) {
                return ST[id];
        }
```

```
        int mid = (l + r) / 2;
        return __gcd(get(id * 2, l, mid, u, v), get(id * 2 + 1, mid + 1, r, u, v));
}

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

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

        build(1, 1, n);

        int ans = 0;
        for (int i = k; i <= n; i++) {
                ans = max(ans, get(1, 1, n, i - k + 1, i));
        }

        cout << ans;

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

8 Bài H - SPSEQ

8.1 Hướng dẫn

Bài này ý tưởng là ta sẽ tính 2 mảng, $pref_i$ = độ dài của dãy tăng nghiêm ngặt dài nhất kết thúc tại i gồm 1 số phần tử tăng dần từ 1 đến i, $suff_i$ = độ dài của dãy tăng nghiêm ngặt dài nhất kết thúc tại i gồm 1 số phần tử tăng dần từ n về i.
Đáp án của bài toán, giả sử nó nằm ở vị trí i, sẽ có N phần tử tạo thành dãy tăng nghiêm ngặt ở bên trái i, và N phần tử tạo thành dãy tăng nghiêm ngặt ở bên phải i, N đây nó là min($pref_i$,$suff_i$) - 1.
Cách tính mảng $pref_i$ cũng giống như cách tính mảng dp_i trong bài D ở trên khác ở chỗ các phần tử trong dãy trong bài D có thể bằng nhau ở bài này các phần tử phải khác nhau hay **tăng nghiêm ngặt**. Cách tính mảng $suff_i$ cũng tương tự như cách tính mảng $pref_i$ chỉ khác ở chỗ là ta đi ngược từ n về 1.

8.2 Code mẫu

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

using namespace std;

const int MAX_N = 1e5 + 5;

int n;
pair<int, int> a[MAX_N];
int b[MAX_N];
int ST[4 * MAX_N];
int pref[MAX_N], suff[MAX_N];

void update(int id, int l, int r, int i, int val) {
        if (i > r || i < l) {
                return;
        }

        if (l == r) {
                ST[id] = max(ST[id], val);
                return;
        }

        int mid = (l + r) / 2;
        update(id * 2, l, mid, i, val);
        update(id * 2 + 1, mid + 1, r, i, val);

        ST[id] = max(ST[id * 2], ST[id * 2 + 1]);
}
```

```
int get(int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return 0;
        }

        if (u <= l && r <= v) {
                return ST[id];
        }

        int mid = (l + r) / 2;
        return max(get(id * 2, l, mid, u, v), get(id * 2 + 1, mid + 1, r, u, v));
}

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

        cin >> n;
        for (int i = 1; i <= n; i++) {
                cin >> a[i].first;
                a[i].second = i;
        }

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

        int id = 0;
        for (int i = 1; i <= n; i++) {
                if (a[i].first != a[i - 1].first) {
                        id++;
                }
                b[a[i].second] = id;
        }

        for (int i = 1; i <= n; i++) {
                pref[i] = get(1, 1, n, 1, b[i] - 1) + 1;
                update(1, 1, n, b[i], pref[i]);
        }

        memset(ST, 0, sizeof ST);

        for (int i = n; i; i--) {
                suff[i] = get(1, 1, n, 1, b[i] - 1) + 1;
                update(1, 1, n, b[i], suff[i]);
        }

        int ans = 0;
        for (int i = 1; i <= n; i++) {
```

```
                ans = max(ans, min(pref[i], suff[i]) * 2 - 1);
        }

        cout << ans;

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

9 Bài I - Đi bộ trên cây

9.1 Hướng dẫn

Bài này ta sẽ sử dụng kỹ thuật *Walk on segment tree*.
Ý tưởng sẽ là, ở truy vấn tìm vị trí trái nhất vẫn nằm trong đoạn [u; v] và có giá trị ≤ k, khi đi trên segment tree, mỗi node của segment tree sẽ lưu giá trị nhỏ nhất trong đoạn mà nó quản lý.
khi ta đi đến 1 đoạn [l; r] nào đấy, nếu đoạn [l; r] đó hoàn toàn nằm ở ngoài đoạn [u; v] ta cần xét trả về -1 luôn, còn không nếu giá trị nhỏ nhất của mọi số trong đoạn [l; r] > k ta cũng trả về -1 tại điều này có nghĩa là ta không thể tìm được vị trí nào thoả mãn điều kiện mà ta mong muốn trong đoạn đó nữa tại số nào trong đoạn [l; r] cũng > k. Nếu không có gì xảy ra, ta sẽ gọi đệ quy, ưu tiên tìm kiếm ở bên trái trước, bên kết quả của bên trái trả về -1 có nghĩa là bên trái không tồn tại vị trí ta mong muốn ta tìm qua bên phải, còn không ta lấy luôn kết quả của bên trái.
Còn ở truy vấn tìm vị trí bên phải nhất mà vẫn ở trong đoạn [u; v] và có giá trị ≤ k ta cũng làm tương tự mà thay vì đó ta ưu tiên đi bên phải thôi.
Đọc code để dễ hình dung hơn ha.

9.2 Code mẫu

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

using namespace std;

const int MAX_N = 3e5 + 5;

int n, m;
int ST[4 * MAX_N];

void update(int id, int l, int r, int i, int val) {
        if (i > r || i < l) {
                return;
        }

        if (l == r) {
                ST[id] = val;
                return;
        }

        int mid = (l + r) / 2;
        update(id * 2, l, mid, i, val);
        update(id * 2 + 1, mid + 1, r, i, val);

        ST[id] = min(ST[id * 2], ST[id * 2 + 1]);
```

```
}

int walkLeft(int id, int l, int r, int u, int v, int k) {
        if (u > r || v < l || ST[id] > k) {
                return -1;
        }

        if (l == r) return l;

        int mid = (l + r) / 2;
        int left = walkLeft(id * 2, l, mid, u, v, k);
        if (left != -1) {
                return left;
        }
        return walkLeft(id * 2 + 1, mid + 1, r, u, v, k);
}

int walkRight(int id, int l, int r, int u, int v, int k) {
        if (u > r || v < l || ST[id] > k) {
                return -1;
        }

        if (l == r) return l;

        int mid = (l + r) / 2;
        int right = walkRight(id * 2 + 1, mid + 1, r, u, v, k);
        if (right != -1) {
                return right;
        }
        return walkRight(id * 2, l, mid, u, v, k);
}

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

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

        while (m--) {
                int type;
                cin >> type;

                if (type == 1) {
```

```
                int pos, k;
                cin >> pos >> k;
                update(1, 1, n, pos, k);
            }
            else {
                int l, r, k;
                cin >> l >> r >> k;
                cout << walkLeft(1, 1, n, l, r, k) << " " << walkRight(1, 1, n, l,
            }
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

10 Bài J - Đi bộ lười trên cây

10.1 Hướng dẫn

Giống như bài I nhưng khác ở chỗ phần cập nhật đoạn ta sử dụng segmentree lazy thôi.

10.2 Code mẫu

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

using namespace std;

const int MAX_N = 3e5 + 5;

int n, m;
long long ST[4 * MAX_N];
long long lazy[4 * MAX_N];

void down(int id) {
        long long k = lazy[id];
        if (k) {
                lazy[id] = 0;
                lazy[id * 2] += k;
                ST[id * 2] += k;
                lazy[id * 2 + 1] += k;
                ST[id * 2 + 1] += k;
        }
}

void update(int id, int l, int r, int u, int v, int val) {
        if (u > r || v < l) {
                return;
        }

        if (u <= l && r <= v) {
                ST[id] += val;
                lazy[id] += val;
                return;
        }

        down(id);

        int mid = (l + r) / 2;
        update(id * 2, l, mid, u, v, val);
        update(id * 2 + 1, mid + 1, r, u, v, val);

        ST[id] = min(ST[id * 2], ST[id * 2 + 1]);
}
```

```
int walkLeft(int id, int l, int r, int u, int v, long long k) {
        if (u > r || v < l || ST[id] > k) {
                return -1;
        }

        if (l == r) return l;

        down(id);
        int mid = (l + r) / 2;
        int left = walkLeft(id * 2, l, mid, u, v, k);
        if (left != -1) {
                return left;
        }
        return walkLeft(id * 2 + 1, mid + 1, r, u, v, k);
}

int walkRight(int id, int l, int r, int u, int v, long long k) {
        if (u > r || v < l || ST[id] > k) {
                return -1;
        }

        if (l == r) return l;

        down(id);
        int mid = (l + r) / 2;
        int right = walkRight(id * 2 + 1, mid + 1, r, u, v, k);
        if (right != -1) {
                return right;
        }
        return walkRight(id * 2, l, mid, u, v, k);
}

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

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

        while (m--) {
                int type;
                cin >> type;
```

```
                if (type == 1) {
                        int l, r, add;
                        cin >> l >> r >> add;
                        update(1, 1, n, l, r, add);
                }
                else {
                        int l, r, k;
                        cin >> l >> r >> k;
                        cout << walkLeft(1, 1, n, l, r, k) << " " << walkRight(1, 1, n, l,
                }
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

11 Bài K - petrol

11.1 Hướng dẫn

Bài này tất nhiên đầu tiên ta sẽ sắp xếp các trạm xăng lại theo vị trí của nó.
Ta sẽ lần lượt kiểm tra ở phía bên trái trước rồi đến bên phải coi có trạm xăng nào trong khoảng cách D và có độ cao ≥ 2 lần độ cao mình đang xét.
Ở đây sẽ hướng dẫn kiểm tra ở phía bên trái, ở phía bên phải làm tương tự.
khi xét đến trạm xăng thứ i, ta sẽ quan tâm đến trạm xăng j xa nhất ở phía bên trái i, sao cho vẫn năm trong khoảng cách D, việc này có thể thực hiện nhanh bằng kỹ thuật 2 con trỏ.
Sau khi xác định được j, ta chỉ cần quan tâm đến trạm xăng lớn nhất trong các trạm xăng từ j đến i - 1, tại càng cao càng có cơ hội có độ cao ≥ 2 lần độ cao của trạm xăng i.
Để lấy ra độ cao lớn nhất trong 1 đoạn liên tiếp, ta có thể thực hiện nhanh bằng segment tree.

11.2 Code mẫu

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

using namespace std;

const int MAX_N = 1e5 + 5;

int n, D;
pair<int, int> a[MAX_N];
bool have[MAX_N];
int ST[4 * MAX_N];

void build(int id, int l, int r) {
        if (l == r) {
                ST[id] = a[l].second;
                return;
        }

        int mid = (l + r) / 2;
        build(id * 2, l, mid);
        build(id * 2 + 1, mid + 1, r);

        ST[id] = max(ST[id * 2], ST[id * 2 + 1]);
}

int get(int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return 0;
        }
```

```
        if (u <= l && r <= v) {
                return ST[id];
        }

        int mid = (l + r) / 2;
        return max(get(id * 2, l, mid, u, v), get(id * 2 + 1, mid + 1, r, u, v));
}

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

        cin >> n >> D;
        for (int i = 1; i <= n; i++) {
                cin >> a[i].first >> a[i].second;
        }
        sort(a + 1, a + n + 1);

        build(1, 1, n);

        int j = 1;
        for (int i = 1; i <= n; i++) {
                while (a[j].first < a[i].first - D) {
                        j++;
                }
                if (get(1, 1, n, j, i - 1) >= 2 * a[i].second) {
                        have[i] = true;
                }
        }

        int ans = 0;
        j = n;
        for (int i = n; i; i--) {
                while (a[j].first > a[i].first + D) {
                        j--;
                }
                if (have[i] && get(1, 1, n, i + 1, j) >= 2 * a[i].second) {
                        ans++;
                }
        }

        cout << ans;

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

12 Bài L - Good Subset

12.1 Hướng dẫn

+) Tại vì đáp án sẽ được tính bằng chênh lệch giữa giá trị của đoạn thẳng lớn và nhỏ nhất nên là ta sẽ sắp xếp các đoạn thẳng theo thứ tự tăng dần giá trị.
Giả sử cố định đoạn thẳng thứ i là đoạn thẳng có giá trị lớn nhất mà ta sẽ chọn, và tìm đoạn thẳng j là đoạn thẳng có giá trị nhỏ nhất mà ta sẽ chọn, j phải thoả:

- j <= i
- sử dụng các đoạn thẳng từ j đến i sẽ tạo ra được đường đi từ 1 đến m.

và chi phí của chúng ta sẽ là chênh lệch giữa giá trị của đoạn i và j. Bởi vì vậy, và do i cố định, nên ta sẽ cố gắng tìm vị trí j lớn nhất để giảm thiểu chi phí.
Với nhận xét là nếu sử dụng các đoạn thẳng từ j đến i có thể tạo ra được đường đi từ 1 đến m, sử dụng các đoạn thẳng từ j - 1 đến i cũng vậy. Cho nên ta sẽ sử dụng kỹ thuật 2 con trỏ để giảm thiểu độ phức tạp.

+) Để kiểm tra coi sử dụng các đoạn thẳng từ j nào đó đến i nào đó có thoả mãn hay không. Ta sẽ tạo mảng cnt_k có kích thước là m, ý nghĩa là số lượng đoạn thẳng đã bao lên vị trí k, đơn giản khi thêm 1 đoạn thẳng [l; r] nào đó ta sẽ tăng $cnt_{l->r}$ lên 1, hoặc khi bỏ đoạn thẳng ra -1.
Để kiểm tra liệu có tạo ra 1 đường đi hợp lệ từ 1 đến m hay không nếu tồn tại 1 điểm chưa được bao, gọi là z, khi này cnt_z = 0 có nghĩa là ta chưa tạo ra được.
Làm thế này cũng khá hợp lý nhưng mà sẽ sai. Ví dụ ta có 2 đoạn thẳng là [1; 3] và [4; 6] $cnt_{1->6}$ đều là 1 có nghĩa là từ 1 đến 6 có thể tạo ra được đường đi từ 1 đến 6, nhưng trên thực tế lại không.
Cho nên ta sẽ thay đổi đôi chút là khi thêm hoặc bớt đoạn thẳng, ta chỉ cập nhật trong đoạn từ **[l; r - 1]** và khi kiểm tra chỉ kiểm tra từ **1 đến m - 1** thôi.
Ta cần làm như thế này bởi như trường hợp trên sẽ có $cnt_3 = 0$ giúp ta biết được rằng không thể tạo ra được đường đi từ 1 đến 6.

Và để thực hiện việc cập nhật mảng cnt và hỏi xem coi có vị trí nào có giá trị bằng 0 không ta sẽ sử dụng segment tree lazy để thực viện cập nhật đoạn, mà mỗi node sẽ lưu giá trị nhỏ nhất của các vị trí mà nó quản lý. Tại ta chỉ cần biết là liệu có vị trí nào giá trị bằng 0 hay không, có nghĩa là min của $cnt_{1->m-1} = 0$.

12.2 Code mẫu

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

using namespace std;

const int MAX_N = 3e5 + 5;
```

```
const int MAX_M = 1e6 + 6;

int n, m;
pair<int, pair<int, int>> a[MAX_N];
int ST[4 * MAX_M];
int lazy[4 * MAX_M];

void down(int id) {
        long long k = lazy[id];
        if (k) {
                lazy[id] = 0;
                lazy[id * 2] += k;
                ST[id * 2] += k;
                lazy[id * 2 + 1] += k;
                ST[id * 2 + 1] += k;
        }
}

void update(int id, int l, int r, int u, int v, int val) {
        if (u > r || v < l) {
                return;
        }

        if (u <= l && r <= v) {
                ST[id] += val;
                lazy[id] += val;
                return;
        }

        down(id);

        int mid = (l + r) / 2;
        update(id * 2, l, mid, u, v, val);
        update(id * 2 + 1, mid + 1, r, u, v, val);

        ST[id] = min(ST[id * 2], ST[id * 2 + 1]);
}

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

        cin >> n >> m;
        for (int i = 1; i <= n; i++) {
                cin >> a[i].second.first >> a[i].second.second >> a[i].first;
        }
        sort(a + 1, a + n + 1);
```

```
        int j = 1;
        int ans = 1e9;
        for (int i = 1; i <= n; i++) {
                update(1, 1, m - 1, a[i].second.first, a[i].second.second - 1, 1);
                while (ST[1]) {
                        ans = min(ans, a[i].first - a[j].first);
                        update(1, 1, m - 1, a[j].second.first, a[j].second.second - 1, -1)
                        j++;
                }
        }

        cout << ans;

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

13 Bài M - C1

13.1 Hướng dẫn

Trong số những đường đi ngắn nhất từ (1, 1) đến (n, n) và có đi qua ô (i, j), đường đi nào có số lượng máy loại $a_{i,j}$ nhiều nhất, số lượng máy loại $a_{i,j}$ trong đường đi đó là độ lặp của ô (i, j). Ta có thể chia đường đi đó thành 2 phần là:

- Đường đi từ (1, 1) đến (i, j).
- Đường đi từ (i, j) đến (n, m).

tổng số lượng máy loại $a_{i,j}$ ở 2 phần đó là độ lập của ô (i, j).

Việc tính độ lập ở ô (i, j) ta sẽ chia ra làm 2 phần như trên để xử lý. Ở đây sẽ hướng dẫn ở phần đầu tiên là đường đi từ (1, 1) đến (i, j) còn phần thứ 2 cũng làm tương tự, thay đổi tý xíu thôi.
ta cần quan tâm đến đường đi ngắn nhất từ (1, 1) đến ô (i, j) mà chứa nhiều máy loại $a_{i,j}$ nhất. Ta có thể gọi $dp_{x,y}$ = số lượng máy loại $a_{i,j}$ nhiều nhất khi ta đi trên đường đi ngắn nhất nào đó từ (1, 1) đến (x, y). Công thức sẽ là $dp_{x,y} = max(dp_{x,y-1}, dp_{x-1,y})$ và + 1 nếu $a_{x,y} = a_{i,j}$. bằng cách này ta đã có thể tính được cái ta muốn đó là $dp_{i,j}$ nhưng độ phức tạp vẫn còn khá lớn. Bởi với mỗi ô ta tính dp lại nên độ phức tạp sẽ là O(n^4).

Ta có thể cải tiến bằng cách, có nhận xét rằng với mỗi loại máy bán hàng, ta chỉ cần quan tâm đến những ô cùng loại với nó. Và xét 1 loại máy bán hàng, ta dp 1 lần và sử dụng được luôn đáp án cho những ô có cùng loại. Có nghĩa là thay vì với mỗi ô ta lại dp 1 lần để biết kết quả cho ô (i, j) ta có thể xét mỗi loại và dp sau đó sử dụng đáp án dp cho những ô có cùng loại đó luôn, sẽ tiết kiệm hơn thay vì mỗi khi đi đến những ô cùng loại đó ta lại tính dp lại từ đầu như cách cũ nó sẽ lâu hơn. Độ phức tạp của ta lúc này sẽ là O(số lượng loại khác nhau * n^2).

Để ý là khi dp, ta sẽ duyệt từng hàng, rồi mới duyệt cột với để tính $dp_{x,y}$. Nếu mỗi lần tính xong, ta lại cập nhật max dp cho mỗi cột, hay ta có mảng mx_k lưu lại max dp của những hàng ở cột k mà ta đã đi qua. công thức dp bây giờ sẽ là $dp_{x,y}$ = max(mx_y, mx_{y-1}) và + 1 nếu $a_{x,y}$ là loại ta đang xét rồi cập nhật mx_y = max(mx_y, $dp_{x,y}$) công thức này vẫn đúng tại càng đi đáp án ta càng tăng chứ không có chuyện giảm.

Từ đây ta có ý tưởng để cải tiến lên **thuật full** của bài này.
Vẫn như ý tưởng cũ, ta xét riêng từng loại một. Cũng như cũ, ta cũng chỉ quan tâm đến vị trí của những ô có cùng loại đang xét và ta cũng tính dp giống trên nhưng thay vì duyệt n^2 cả mảng để dp ta chỉ duyệt qua những ô có cùng loại thôi (ta có thể chuẩn bị trước 1 cái danh sách để thực hiện việc này), nhưng khi này, công thức dp dựa trên mảng mx trên sẽ không còn đúng nữa, tại vì vị trí của các ô lúc này không phải lúc nào cũng cạnh nhau, nên là thay vì lấy mx ở cột y và y - 1 ta sẽ lấy từ cột 1 -> cột y luôn, như thế sẽ không bị bỏ xót

và vẫn đúng. để thực hiện việc này, ta có thể đưa mảng mx lên segment tree thôi. Độ phức tạp của ta sẽ là O(n^2logn). Bởi vì với mỗi loại, ta chỉ duyệt qua những ô có loại đó, nên là tổng số ô ta duyệt là n^2.
Đọc code để hiểu thêm ha.

13.2 Code mẫu

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

using namespace std;

const int MAX_N = 700 + 5;

int n;
int a[MAX_N][MAX_N];
vector<pair<int, int>> b[MAX_N * MAX_N];
int ST[4 * MAX_N];
int f[MAX_N * MAX_N], g[MAX_N * MAX_N];
int ans[2 * MAX_N];

void update(int id, int l, int r, int i, int val) {
        if (i > r || i < l) {
                return;
        }

        if (l == r) {
                ST[id] = max(ST[id], val);
                return;
        }

        int mid = (l + r) / 2;
        update(id * 2, l, mid, i, val);
        update(id * 2 + 1, mid + 1, r, i, val);

        ST[id] = max(ST[id * 2], ST[id * 2 + 1]);
}

void update_mark(int id, int l, int r, int i, int val) {
        if (i > r || i < l) {
                return;
        }

        if (l == r) {
                ST[id] = val;
                return;
        }
```

```
        int mid = (l + r) / 2;
        update_mark(id * 2, l, mid, i, val);
        update_mark(id * 2 + 1, mid + 1, r, i, val);

        ST[id] = max(ST[id * 2], ST[id * 2 + 1]);
}

int get(int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return 0;
        }

        if (u <= l && r <= v) {
                return ST[id];
        }

        int mid = (l + r) / 2;
        return max(get(id * 2, l, mid, u, v), get(id * 2 + 1, mid + 1, r, u, v));
}

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

        cin >> n;
        for (int i = 1; i <= n; i++) {
                for (int j = 1; j <= n; j++) {
                        cin >> a[i][j];
                        b[a[i][j]].push_back(pair<int, int>(i, j));
                }
        }

        for (int x = 1; x <= n * n; x++) {
                sort(b[x].begin(), b[x].end());

                for (int i = 0; i < b[x].size(); i++) {
                        f[i] = g[i] = 0;
                }

                // tinh phan truoc
                for (int i = 0; i < b[x].size(); i++) {
                        f[i] = 1 + get(1, 1, n, 1, b[x][i].second);
                        update(1, 1, n, b[x][i].second, f[i]);
                }

                for (int i = 0; i < b[x].size(); i++) {
                        update_mark(1, 1, n, b[x][i].second, 0);
                }
```

```
                // tinh phan sau
                for (int i = b[x].size() - 1; i >= 0; i--) {
                        g[i] = 1 + get(1, 1, n, b[x][i].second, n);
                        update(1, 1, n, b[x][i].second, g[i]);
                }

                for (int i = 0; i < b[x].size(); i++) {
                        update_mark(1, 1, n, b[x][i].second, 0);
                }

                // cap nhat dap an
                for (int i = 0; i < b[x].size(); i++) {
                        ans[f[i] + g[i] - 1]++;
                }
        }

        for (int i = 1; i <= 2 * n - 1; i++) {
                cout << ans[i] << '\n';
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

14 Bài N - Kquery

14.1 Hướng dẫn

Ta có nhận xét rằng, ở mỗi truy vấn ta chỉ quan tâm đến những phần tử lớn hơn k, cụ thể hơn là trong khoảng [i; j].

Bài này ta có thể xử lý offline là sắp xếp mảng a theo thứ tự tăng dần của giá trị và các truy vấn theo thứ tự tăng dần của k. Và ta cũng phải lưu lại vị trí gốc của các giá trị trong mảng a sau khi sắp xếp.

Vậy nếu ta sắp xếp lại như vậy, sử dụng kỹ thuật 2 con trỏ, khi duyệt các truy vấn theo thứ tự giảm dần k, ta di chuyển con trỏ trên mảng a theo thứ tự giảm dần giá trị miễn là giá trị nó vẫn lớn hơn k của truy vấn hiện tại mình đang duyệt đến.
Bằng cách duyệt như này, ta đã giải quyết được điều kiện là các phần tử phải có giá trị phải lớn hơn k, cái ta cần quan tâm bây giờ chỉ là trong số những phần tử đó, có bao nhiêu phần tử nằm trong khoảng [i; j] của truy vấn ta đang xét, đó chính là đáp án của truy vấn ấy.

Nếu ta có mảng cnt với ý nghĩa là $cnt_x = 1$ tương đương với $a_x >$ k của truy vấn hiện tại đang xét. Khi ta duyệt tới một truy vấn nào đó, ta di chuyển con trỏ z trên mảng a trong khi giá trị của phần tử ta duyệt đến vẫn còn lớn hơn k của truy vấn ta đang xét, ta cập nhật $cnt_{pos_{a_z}} = 1$, pos_{a_z} là vị trí gốc của phần tử a_z ở mảng a ban đầu (trước khi sắp xếp), tại đấy mới là vị trí mà ta quan tâm đến. Và đáp án của truy vấn sẽ là tổng của $cnt_{i->j}$.

Ta có thể đưa mảng cnt lên segment tree để cái tiến độ phức tạp thuật toán để full bài này.
Đọc code để dễ hình dung hơn ha.

14.2 Code mẫu

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

using namespace std;

const int MAX_N = 3e4 + 5;
const int MAX_Q = 2e5 + 5;

int n, numQ;
pair<int, int> a[MAX_N];
pair<pair<int, int>, pair<int, int>> q[MAX_Q];
int ans[MAX_Q];
int ST[4 * MAX_N];
```

```
void update(int id, int l, int r, int i, int val) {
        if (i > r || i < l) {
                return;
        }

        if (l == r) {
                ST[id] += val;
                return;
        }

        int mid = (l + r) / 2;
        update(id * 2, l, mid, i, val);
        update(id * 2 + 1, mid + 1, r, i, val);

        ST[id] = ST[id * 2] + ST[id * 2 + 1];
}

int get(int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return 0;
        }

        if (u <= l && r <= v) {
                return ST[id];
        }

        int mid = (l + r) / 2;
        return get(id * 2, l, mid, u, v) + get(id * 2 + 1, mid + 1, r, u, v);
}

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

        cin >> n;
        for (int i = 1; i <= n; i++) {
                cin >> a[i].first;
                a[i].second = i;
        }
        cin >> numQ;
        for (int i = 1; i <= numQ; i++) {
                cin >> q[i].second.first >> q[i].second.second >> q[i].first.first;
                q[i].first.second = i;
        }

        sort(a + 1, a + n + 1);
        sort(q + 1, q + numQ + 1);
```

```
        int j = n;
        for (int i = numQ; i; i--) {
                while (j && a[j].first > q[i].first.first) {
                        update(1, 1, n, a[j].second, 1);
                        j--;
                }
                ans[q[i].first.second] = get(1, 1, n, q[i].second.first, q[i].second.secon
        }

        for (int i = 1; i <= numQ; i++) {
                cout << ans[i] << "\n";
        }

        return 0;
}
```

Đọc code đầy đủ hơn *ở đây*.

15 Bài O - Tân và kiến

15.1 Hướng dẫn

Bài này ta sẽ đi đếm số lượng kiến chúa trong đoạn mà ta truy vấn, rồi đáp án của truy vấn sẽ là độ dài của đoạn - số lượng kiến chúa.

Ta có nhận xét là 1 con kiến là kiến chúa khi nó là ước của mọi số trong đoạn, hay nó = **gcd** của cả đoạn và nó phải là số nhỏ nhất của đoạn.

Vậy để giải quyết bài này ta chỉ cần cài 2 cây segment tree, 1 cây để lấy ra gcd của đoạn truy vấn, và 1 cây để lấy ra số nhỏ nhất cùng với số lượng của số đó trong đoạn.
Nếu số nhỏ nhất trong đoạn bằng với gcd của cả đoạn, số đó chính là giá trị của con kiến chúa, và số lượng số nhỏ nhất trong đoạn đó chính là số lượng kiến chúa mà ta cần tìm.

15.2 Code mẫu

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

using namespace std;

const int MAX_N = 3e5 + 5;

int n;
int a[MAX_N];
int gcdST[MAX_N * 4];
pair<int, int> minST[MAX_N * 4];

void build(int id, int l, int r) {
        if (l == r) {
                gcdST[id] = a[l];
                minST[id] = {a[l], 1};
                return;
        }

        int mid = (l + r) / 2;
        build(id * 2, l, mid);
        build(id * 2 + 1, mid + 1, r);

        gcdST[id] = __gcd(gcdST[id * 2], gcdST[id * 2 + 1]);
        if (minST[id * 2].first == minST[id * 2 + 1].first) {
                minST[id] = {minST[id * 2].first, minST[id * 2].second + minST[id * 2 + 1]
        }
        else {
                minST[id] = min(minST[id * 2], minST[id * 2 + 1]);
        }
```

```
}

int get_gcd(int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return a[u];
        }

        if (u <= l && r <= v) {
                return gcdST[id];
        }

        int mid = (l + r) / 2;
        return __gcd(get_gcd(id * 2, l, mid, u, v), get_gcd(id * 2 + 1, mid + 1, r, u, v))
}

pair<int, int> get_min(int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return {0, 0};
        }

        if (u <= l && r <= v) {
                return minST[id];
        }

        int mid = (l + r) / 2;
        pair<int, int> left = get_min(id * 2, l, mid, u, v);
        pair<int, int> right = get_min(id * 2 + 1, mid + 1, r, u, v);
        if (!left.second) {
                return right;
        }
        if (!right.second) {
                return left;
        }
        if (left.first == right.first) {
                return {left.first, left.second + right.second};
        }
        else {
                return min(left, right);
        }
}

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

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

```
        }
        build(1, 1, n);

        int t;
        cin >> t;
        while (t--) {
                int l, r;
                cin >> l >> r;
                pair<int, int> mn = get_min(1, 1, n, l, r);
                if (mn.first == get_gcd(1, 1, n, l, r)) {
                        cout << r - l + 1 - mn.second;
                }
                else {
                        cout << r - l + 1;
                }
                cout << '\n';
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

16 Bài P - SUM and REPLACE

16.1 Hướng dẫn

Bài này ta nhận thấy rằng, 1 số sẽ thực hiện được việc thay thế nó bằng số lượng ước của nó với nó số lần rất nhỏ, sau đó số đó sẽ trở thành 1 hoặc 2, khi đó số lượng ước của nó bằng chính nó luôn, nên khi này ta không cần quan tâm đến số này nữa bởi vì có thay đổi cũng không ảnh hưởng gì đến việc lấy tổng.
Vậy nên ở truy vấn Replace, ta sẽ lưu lại 1 cái set để quản lý những vị trí mà tại đó ta còn có thể thay thế, những vị trí đã bị bỏ ra là những vị trí tại đó số lượng ước của số đó bằng chính nó rồi, nên ta không cần quan tâm đến nó nữa. ở truy vấn Replace l r, ta sẽ thực hiện việc thay thế số ở trên những vị trí còn lại trong set nằm trong l r. Và ta sẽ duy trì 1 cây segment tree để truy vấn tổng đoạn. Ban đầu ở mọi vị trí i trên segment tree có giá trị là a[i], mỗi lần thay đổi số a[i] nào đó thành D(a[i]) ta sẽ thêm vào ở vị trí i, -a[i] + D(a[i]) tức là bỏ đi số a[i] và thay thế nó thành D(a[i]).

16.2 Code mẫu

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

using namespace std;

const int MAX_N = 3e5 + 5;
const int MAX_VAL = 1e6 + 1;

int lp[MAX_VAL];

int n, m;
int a[MAX_N];
long long ST[4 * MAX_N];
set<int> s;
vector<int> eraseList;

int D(int x) {
        int ans = 1;
        while (x > 1) {
                int k = lp[x];
                int cnt = 0;
                while (x % k == 0) {
                        x /= k;
                        cnt++;
                }
                ans *= cnt + 1;
        }
        return ans;
}
```

```
void update(int id, int l, int r, int i, int val) {
        if (i > r || i < l) {
                return;
        }

        if (l == r) {
                ST[id] += val;
                return;
        }

        int mid = (l + r) / 2;
        update(id * 2, l, mid, i, val);
        update(id * 2 + 1, mid + 1, r, i, val);

        ST[id] = ST[id * 2] + ST[id * 2 + 1];
}

long long get(int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return 0;
        }

        if (u <= l && r <= v) {
                return ST[id];
        }

        int mid = (l + r) / 2;
        return get(id * 2, l, mid, u, v) + get(id * 2 + 1, mid + 1, r, u, v);
}

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

        for (int i = 2; i * i < MAX_VAL; i++) {
                if (!lp[i]) {
                        for (int j = i * i; j < MAX_VAL; j += i) {
                                lp[j] = i;
                        }
                }
        }
        for (int i = 2; i < MAX_VAL; i++) {
                if (!lp[i]) {
                        lp[i] = i;
                }
        }
```

```
        cin >> n >> m;
        for (int i = 1; i <= n; i++) {
                cin >> a[i];
                s.insert(i);
                update(1, 1, n, i, a[i]);
        }

        while (m--) {
                int t, l, r;
                cin >> t >> l >> r;

                if (t == 1) {
                        auto it = s.lower_bound(l);
                        eraseList.clear();
                        while (it != s.end() && *it <= r) {
                                int tmp = D(a[*it]);
                                if (tmp == a[*it]) {
                                        eraseList.push_back(*it);
                                }
                                else {
                                        update(1, 1, n, *it, -a[*it] + tmp);
                                        a[*it] = tmp;
                                }
                                it++;
                        }
                        for (auto i : eraseList) {
                                s.erase(i);
                        }
                }
                else {
                        cout << get(1, 1, n, l, r) << '\n';
                }
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

17 Bài Q - COLORS

17.1 Hướng dẫn

Nếu ta có mảng $last_k$ = vị trí của cùng xuất hiện của màu k. Giả sử xét 1 truy vấn nào đó, nếu ta đi lần lượt từ 1 -> R, vừa đi vừa cập nhật mảng last, $last_{a_i} = i$ xét các màu j lần lượt từ c đến d, nếu $last_j$ < L tức là trong đoạn L R mà truy vấn ta quan tâm, sẽ không có màu j.
Từ đây ta nhận thấy rằng, ta có thể sắp xếp các truy vấn theo thứ tự tăng dần của R, bởi ta có thể sử dụng kỹ thuật 2 con trỏ để cập nhật mảng last hiệu quả.
Từ đây ta đã có thuật O(n^2), để cải tiến, ta có thể đưa mảng last lên segment tree và mỗi node sẽ lưu min trong đoạn mà nó quản lý và ta có thể sử dụng walk on segment tree để tìm màu $c \le j \le$ d nhỏ nhất mà $last_j$ < L.

17.2 Code mẫu

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

using namespace std;

const int MAX_N = 4e5 + 5;

int n, m;
int a[MAX_N];
struct Query {
        int c, d, L, R, id;
};
Query q[MAX_N];
int ans[MAX_N];
int ST[4 * MAX_N];

void update(int id, int l, int r, int i, int val) {
        if (i > r || i < l) {
                return;
        }

        if (l == r) {
                ST[id] = val;
                return;
        }

        int mid = (l + r) / 2;
        update(id * 2, l, mid, i, val);
        update(id * 2 + 1, mid + 1, r, i, val);

        ST[id] = min(ST[id * 2], ST[id * 2 + 1]);
```

```
}

int walk(int id, int l, int r, int u, int v, int L) {
        if (u > r || v < l || ST[id] >= L) {
                return -1;
        }

        if (l == r) {
                return l;
        }

        int mid = (l + r) / 2;
        int left = walk(id * 2, l, mid, u, v, L);
        if (left == -1) {
                return walk(id * 2 + 1, mid + 1, r, u, v, L);
        }
        return left;
}

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

        cin >> n >> m;
        for (int i = 1; i <= n; i++) {
                cin >> a[i];
        }
        for (int i = 1; i <= m; i++) {
                cin >> q[i].c >> q[i].d >> q[i].L >> q[i].R;
                q[i].id = i;
        }

        sort(q + 1, q + m + 1, [](Query x, Query y){
                return x.R < y.R;
        });

        int j = 1;
        for (int i = 1; i <= m; i++) {
                while (j <= q[i].R) {
                        update(1, 1, n, a[j], j);
                        j++;
                }
                ans[q[i].id] = walk(1, 1, n, q[i].c, q[i].d, q[i].L);
        }

        for (int i = 1; i <= m; i++) {
                if (ans[i] == -1) {
                        cout << "OK";
```

```
                }
                else {
                        cout << ans[i];
                }
                cout << '\n';
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

18 Bài R - Sum

18.1 Hướng dẫn

Bài này ta có thể thử sắp xếp các truy vấn theo thứ tự u tăng dần, nếu thế ta cần nghĩ ra cách nào đó để sử dụng lại những thông tin ở trước cho những truy vấn sau liên quan đến việc u các truy vấn tăng dần.
Ta sẽ sử dụng ý tưởng khá tương tự bài O, nếu ta xét 1 truy vấn nào đó, và xét toàn bộ tổng của các đoạn con [l; r], sao cho $l \le r \le v$, và ta có mảng $last_s$ sẽ lưu l lớn nhất nếu có đoạn [l; r] nào đó trong những đoạn trên tạo ra tổng s. ta có tính chất là nếu tổng s nào có $last_s \ge u$, có nghĩa là tổng s có thể được tạo thành bằng 1 đoạn con nào đó nằm trong [u; v], bởi tổng s đó sẽ của 1 đoạn con [$last_s$; r] mà $u \le last_s \le r \le v$ đáp án cho truy vấn nó là tổng s lớn nhất ≤ k có $last_s \ge u$.
Việc bây giờ là nếu mỗi truy vấn nào cũng làm như thế độ phức tạp thuật toán của ta là O(qn^2). Để cải tiến thuật toán, ta để ý rằng, các truy vấn đã được sắp xếp theo thứ tự tăng dần của u, cho nên là mỗi lần ta duyệt đến các truy vấn mới, tức là giá trị u tăng hay là giá trị r của các đoạn con mà ta xét cũng tăng dần. Điều đó có nghĩa là, ta có thể cập nhật thêm các đoạn con mới vào mảng last bằng kỹ thuật 2 con trỏ, để tổng độ phức tạp của việc thêm các đoạn con bây giờ chỉ còn là O(n^2), cụ thể hơn là ta có con trỏ r, mỗi lần ta xét truy vấn mới, ta sẽ lần lượt cập nhật các đoạn con [l; r], $1 \le l \le r$ với r cố định và tăng sau đó r lên, sau đó cập nhật tiếp miễn là r vẫn ≤ u.

Còn về đáp án của truy vấn, thay vì ta phải duyệt qua các tổng để xét, ta có thể đưa mảng last lên segment tree với vị trí của 1 tổng s nào đó trên segment tree sẽ là vị trí tương đối của nó so với toàn bộ tổng của các đoạn con của mảng a. để tìm tổng s lớn nhất mà $s \le k$ và $last_s \ge u$ nhanh ta có thể sử dụng walk on segment tree và mỗi node của segment tree sẽ lưu max các số trong đoạn mà nó quản lý.

18.2 Code mẫu

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

using namespace std;

const int MAX_N = 1e3 + 5;
const int MAX_M = 1e6 + 5;

int n, m, numQ;
int a[MAX_N];
vector<long long> ns;
struct Query {
        int l, k, id;
};
```

```
vector<Query> q[MAX_N];
int ans[MAX_M];
int ST[4 * MAX_M];
unordered_map<long long, int> pos;

void update(int id, int l, int r, int i, int val) {
        if (l == r) {
                ST[id] = max(ST[id], val);
                return;
        }

        int mid = (l + r) / 2;
        if (i <= mid) {
                update(id * 2, l, mid, i, val);
        }
        else {
                update(id * 2 + 1, mid + 1, r, i, val);
        }
        ST[id] = max(ST[id * 2], ST[id * 2 + 1]);
}

int walk(int id, int l, int r, int k, int L) {
        if (l > k || ST[id] < L) {
                return -1;
        }

        if (l == r) {
                return l;
        }

        int mid = (l + r) / 2;
        int right = walk(id * 2 + 1, mid + 1, r, k, L);
        if (right == -1) {
                return walk(id * 2, l, mid, k, L);
        }
        return right;
}

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

        cin >> n >> numQ;
        for (int i = 1; i <= n; i++) {
                cin >> a[i];
                long long sum = 0;
                for (int j = i; j; j--) {
                        sum += a[j];
```

```
                ns.push_back(sum);
            }
        }
        for (int i = 1; i <= numQ; i++) {
            int l, r, k;
            cin >> l >> r >> k;
            q[r].push_back({l, k, i});
        }

        sort(ns.begin(), ns.end());
        ns.erase(unique(ns.begin(), ns.end()), ns.end());
        m = ns.size();
        for (int i = 0; i < (int) ns.size(); i++) {
            pos[ns[i]] = i + 1;
        }

        for (int i = 1; i <= n; i++) {
            long long sum = 0;
            for (int j = i; j; j--) {
                sum += a[j];
                update(1, 1, m, pos[sum], j);
            }
            for (auto it : q[i]) {
                int x = lower_bound(ns.begin(), ns.end(), it.k) - ns.begin() + 1;
                if (ns[x - 1] > it.k) x--;
                ans[it.id] = walk(1, 1, m, x, it.l);
            }
        }

        for (int i = 1; i <= numQ; i++) {
            if (ans[i] == -1) {
                cout << "NONE";
            }
            else {
                cout << ns[ans[i] - 1];
            }
            cout << '\n';
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

19 Bài S - Tấn và các chiếc hộp 1

19.1 Hướng dẫn

Bài này ta có thuật trâu là lưu lại n cái multiset tương ứng với n vị trí của mảng, việc ta thực hiện thao tác loại 1 và 2 rất dễ dàng, còn thao tác loại 3 ta lần lượt duyệt các multiset từ l sang r rồi tìm số nhỏ nhất > val trong đó, có thể tìm bằng cách sử dụng hàm upperbound.
Để cải tiến thuật toán, ta có thể đưa mảng ấy lên thành 1 cây segment tree mà mỗi node là 1 multiset chứa những số trong đoạn mà nó quản lý. Ta vẫn có thể lưu đủ 1 cây segment tree multiset bởi vì tại mỗi tầng của segment tree có tổng cộng nhiều nhất là q phần tử thôi, tổng cộng số lượng node tối đa ta lưu chỉ có qlogn phần tử.

19.2 Code mẫu

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

using namespace std;

const int MAX_N = 3e5 + 5;
const int inf = 1e9 + 10;

int n, q;
multiset<int> ST[4 * MAX_N];
multiset<int> a[MAX_N];

void add(int id, int l, int r, int i, int val) {
        ST[id].insert(val);

        if (l == r) {
                return;
        }

        int mid = (l + r) / 2;
        if (i <= mid) {
                add(id * 2, l, mid, i, val);
        }
        else {
                add(id * 2 + 1, mid + 1, r, i, val);
        }
}

void remove(int id, int l, int r, int i, int val) {
        ST[id].erase(ST[id].find(val));

        if (l == r) {
```

```
                return;
        }

        int mid = (l + r) / 2;
        if (i <= mid) {
                remove(id * 2, l, mid, i, val);
        }
        else {
                remove(id * 2 + 1, mid + 1, r, i, val);
        }
}

int query(int id, int l, int r, int u, int v, int val) {
        if (u > r || v < l) {
                return inf;
        }

        if (u <= l && r <= v) {
                auto it = ST[id].upper_bound(val);
                return (it == ST[id].end() ? inf : *it);
        }

        int mid = (l + r) / 2;
        return min(query(id * 2, l, mid, u, v, val), query(id * 2 + 1, mid + 1, r, u, v, v
}

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

        cin >> n >> q;
        while (q--) {
                int type;
                cin >> type;

                if (type == 1) {
                        int pos, val;
                        cin >> pos >> val;
                        a[pos].insert(val);
                        add(1, 1, n, pos, val);
                        continue;
                }

                if (type == 2) {
                        int pos, val;
                        cin >> pos >> val;
                        if (a[pos].find(val) != a[pos].end()) {
                                a[pos].erase(a[pos].find(val));
```

```
                        remove(1, 1, n, pos, val);
                }
                continue;
        }

        if (type == 3) {
                int l, r, val;
                cin >> l >> r >> val;
                int ans = query(1, 1, n, l, r, val);
                if (ans == inf) {
                        cout << "NONE";
                }
                else {
                        cout << ans;
                }
                cout << '\n';
                continue;
        }
    }

    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

20 Bài T - Tấn và các chiếc hộp 2

20.1 Hướng dẫn

Bài này ta có thuật trâu là lưu lại n cái multiset tương ứng với n vị trí của mảng, việc ta thực hiện thao tác loại 1 và 2 rất dễ dàng, còn thao tác loại 3 ta lần lượt duyệt các multiset từ l sang r rồi tìm vị trí lớn nhất mà ở đó multiset có chứa ít nhất 1 số lớn hơn val hay số lớn nhất trong multiset đó lớn hơn val.
Để cải tiến thuật toán, ta có thể đưa mảng ấy lên thành 1 cây segment tree mà mỗi node là 1 multiset chứa những số trong đoạn mà nó quản lý và ở thao tác thứ 3 ta sẽ sử dụng walk on segment tree.

20.2 Code mẫu

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

using namespace std;

const int MAX_N = 3e5 + 5;

int n, q;
multiset<int> ST[4 * MAX_N];
multiset<int> a[MAX_N];

void add(int id, int l, int r, int i, int val) {
        ST[id].insert(val);

        if (l == r) {
                return;
        }

        int mid = (l + r) / 2;
        if (i <= mid) {
                add(id * 2, l, mid, i, val);
        }
        else {
                add(id * 2 + 1, mid + 1, r, i, val);
        }
}

void remove(int id, int l, int r, int i, int val) {
        ST[id].erase(ST[id].find(val));

        if (l == r) {
                return;
        }
```

```
        int mid = (l + r) / 2;
        if (i <= mid) {
                remove(id * 2, l, mid, i, val);
        }
        else {
                remove(id * 2 + 1, mid + 1, r, i, val);
        }
}

int walk(int id, int l, int r, int u, int v, int val) {
        if (u > r || v < l || ST[id].empty() || *(ST[id].rbegin()) <= val) {
                return -1;
        }

        if (l == r) {
                return l;
        }

        int mid = (l + r) / 2;
        int right = walk(id * 2 + 1, mid + 1, r, u, v, val);
        if (right == -1) {
                return walk(id * 2, l, mid, u, v, val);
        }
        return right;
}

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

        cin >> n >> q;
        while (q--) {
                int type;
                cin >> type;

                if (type == 1) {
                        int pos, val;
                        cin >> pos >> val;
                        a[pos].insert(val);
                        add(1, 1, n, pos, val);
                        continue;
                }

                if (type == 2) {
                        int pos, val;
                        cin >> pos >> val;
                        if (a[pos].find(val) != a[pos].end()) {
```

```
                        a[pos].erase(a[pos].find(val));
                        remove(1, 1, n, pos, val);
                }
                continue;
        }

        if (type == 3) {
                int l, r, val;
                cin >> l >> r >> val;
                int ans = walk(1, 1, n, l, r, val);
                if (ans == -1) {
                        cout << "NONE";
                }
                else {
                        cout << ans;
                }
                cout << '\n';
                continue;
        }
    }

    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

21 Bài U - Impossible mission???

21.1 Hướng dẫn

Ta có nhận xét rằng ở truy vấn truy xuất ra b_x, giá trị của b_x hiện tại sẽ do truy vấn cuối cùng mà có gán lên vị trí x thực hiện, nên ta chỉ cần quan tâm đến truy vấn đó.
Vậy nên ta có ý tưởng là, ở truy vấn gán, ta không đi gán vào mảng b mà ta tạo mảng $last_i$ tức là id của truy vấn cuối cùng có gán lên vị trí i ở truy vấn gán, ta chỉ cập nhập $last_{y->y+k-1}$ = id của truy vấn hiện tại độ phức tạp thuật toán của ta sẽ là O(mn).
Ta có thể bằng cách đưa mảng last lên segment tree và ở thao tác cập nhật ta sẽ dùng lazy để gán nhanh vị trí từ y đến y + k - 1.

21.2 Code mẫu

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

using namespace std;

const int MAX_N = 1e5 + 5;

int n, m;
vector<pair<int, int>> q;
int a[MAX_N];
int b[MAX_N];
int ST[4 * MAX_N];
int lazy[4 * MAX_N];

void down(int id) {
        int k = lazy[id];
        if (k) {
                ST[id * 2] = k;
                lazy[id * 2] = k;
                ST[id * 2 + 1] = k;
                lazy[id * 2 + 1] = k;
                lazy[id] = 0;
        }
}

void update(int id, int l, int r, int u, int v, int val) {
        if (u > r || v < l) {
                return;
        }

        if (u <= l && r <= v) {
                ST[id] = val;
                lazy[id] = val;
```

```
                return;
        }

        down(id);

        int mid = (l + r) / 2;
        update(id * 2, l, mid, u, v, val);
        update(id * 2 + 1, mid + 1, r, u, v, val);
}

int get(int id, int l, int r, int i) {
        if (l == r) {
                return ST[id];
        }

        down(id);

        int mid = (l + r) / 2;
        if (i <= mid) {
                return get(id * 2, l, mid, i);
        }
        else {
                return get(id * 2 + 1, mid + 1, r, i);
        }
}

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

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

        while (m--) {
                int type;
                cin >> type;

                if (type == 2) {
                        int x;
                        cin >> x;
                        int queryPos = get(1, 1, n, x);
                        if (!queryPos) {
                                cout << b[x];
```

```
                        }
                        else {
                                cout << a[q[queryPos - 1].first + x - q[queryPos - 1].secon
                        }
                        cout << '\n';
                }
                else {
                        int x, y, k;
                        cin >> x >> y >> k;
                        q.push_back({x, y});
                        update(1, 1, n, y, y + k - 1, q.size());
                }
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

22 Bài V - S1mple

22.1 Hướng dẫn

Ta sẽ nói về thao tác sắp xếp tăng dần còn thao tác sắp xếp giảm dần làm tương tự. Ta để ý là sâu s chỉ bao gồm các chữ cái tiếng anh viết thường, nên khi ta sắp xếp giảm dần 1 đoạn nào đó, đoạn đó sẽ có dạng như là:

aaaaaaabbbbbbbccccddddeeee....xxxxxyyyzzzz

tất nhiên có thể 1 vài loại ký tự sẽ không có, nhưng nhìn chung là như vậy.

Vậy nên thay vì sắp xếp lại đoạn đấy theo các thông thường, ta có cách khác để sắp xếp đó nếu ta biết được cnt_h là số lượng ký tự h trong đoạn trong đoạn, đoạn đó sau khi được sắp xếp sẽ có dạng là cnt_a ký tự a ở đầu, sau đó là cnt_b ký tự b ở sau cnt_a ký tự a, tiếp đến là cnt_c ký tự c ở sau cnt_b ký tự b, tiếp tục như vậy đến cnt_z ký tự z ở cuối.

Để cải tiến độ phức tạp, ta sẽ thay đổi mảng cnt 1 chút là $cnt_{h,i}$ = 1 nếu tại vị trí i, có ký tự h. Vậy khi sắp xếp tăng dần 1 đoạn [l; r] nào đấy, ta sẽ có biến h xét lần lượt các ký tự từ a đến z, khi xét đến ký tự h nào đó, ta gọi i là vị trí sau vị trí cuối cùng khi ta đã đặt theo đúng thứ tự các ký tự trước ký tự h, gọi x = tổng của $cnt_{h,l->r}$ từ vị trí i đến i + x - 1 ta sẽ đặt ký tự h hay thực chất ta sẽ làm là ở các vị trí $cnt_{h,j}$, $l \le j \le r$ nào bằng 1 ta sẽ gán bằng 0 (xoá đi các ký tự cũ) rồi sau đó cập nhật $cnt_{h,j}$, $i \le j \le i+x-1$ = 1 (cập nhật các ký tự mới) sau đó i sẽ tăng lên x, hay i = i + x.
Và cuối cùng là ta đưa mảng cnt lên segment tree, còn thao tác xoá đi các ký tự, thay vì tìm vị trí nào bằng 1 mới xoá, ta có thể gán luôn cả $cnt_{h,l->r}$ = 0 hết tại nó không ảnh hưởng gì đến sự đúng đắn, để thực hiện việc gán 1 đoạn liên tục 1 số nào đấy, ta có thể sử dụng lazy. Đọc code để dễ hình dung hơn ha.

22.2 Code mẫu

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

using namespace std;

const int MAX_N = 1e5 + 5;

int n, q;
int ST[26][4 * MAX_N];
int lazy[26][4 * MAX_N];

void down(int t, int id, int l, int r) {
        if (!lazy[t][id]) return;
        int mid = (l + r) / 2;
        lazy[t][id * 2] = lazy[t][id * 2 + 1] = lazy[t][id];
```

```
        ST[t][id * 2] = (mid - l + 1) * (lazy[t][id] - 1);
        ST[t][id * 2 + 1] = (r - mid) * (lazy[t][id] - 1);
        lazy[t][id] = 0;
}

void update(int t, int id, int l, int r, int u, int v, int val) {
    if (u > r || v < l) {
        return;
    }

    if (u <= l && r <= v) {
        ST[t][id] = (r - l + 1) * val;
        lazy[t][id] = val + 1;
        return;
    }

    down(t, id, l, r);

    int mid = (l + r) / 2;
    update(t, id * 2, l, mid, u, v, val);
    update(t, id * 2 + 1, mid + 1, r, u, v, val);

    ST[t][id] = ST[t][id * 2] + ST[t][id * 2 + 1];
}

int get(int t, int id, int l, int r, int u, int v) {
    if (u > r || v < l) {
        return 0;
    }

    if (u <= l && r <= v) {
        return ST[t][id];
    }

    down(t, id, l, r);

    int mid = (l + r) / 2;
    return get(t, id * 2, l, mid, u, v) + get(t, id * 2 + 1, mid + 1, r, u, v);
}

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

        cin >> n >> q;
        for (int i = 1; i <= n; i++) {
                char h;
                cin >> h;
```

```
                update(h - 'a', 1, 1, n, i, i, 1);
        }

        while (q--) {
                int i, j, k;
                cin >> i >> j >> k;

                if (k) {
                        int id = i;
                        for (int h = 0; h < 26; h++) {
                                int cnt = get(h, 1, 1, n, i, j);
                                update(h, 1, 1, n, i, j, 0);
                                update(h, 1, 1, n, id, id + cnt - 1, 1);
                                id += cnt;
                        }
                }
                else {
                        int id = j;
                        for (int h = 0; h < 26; h++) {
                                int cnt = get(h, 1, 1, n, i, j);
                                update(h, 1, 1, n, i, j, 0);
                                update(h, 1, 1, n, id - cnt + 1, id, 1);
                                id -= cnt;
                        }
                }
        }

        for (int i = 1; i <= n; i++) {
                for (int h = 0; h < 26; h++) {
                        if (get(h, 1, 1, n, i, i)) {
                                cout << char(h + 'a');
                                break;
                        }
                }
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

23 Bài W - Tấn và phép Xor

23.1 Hướng dẫn

Ta xét bài toán đơn giản hơn, nếu có 1 tập n số nguyên, và ta có các truy vấn cần tìm giá trị lớn nhất x xor 1 số nào đó trong tập. ta có thể giải được bằng cách đưa tập n số kia lên 1 cây trie, rồi khi truy vấn ta chỉ đi trên cây trie, đi sao cho ưu tiên tạo ra bit 1 lớn nhất nếu có thể thôi.
Trở về bài toán của ta ta cần tìm trên 1 tập số là 1 đoạn [l; r] nào đó của mảng a, ta có thể sử dụng segment tree để quản lý việc truy vấn đoạn, tư tưởng ta cũng giống như mấy bài truy vấn trên đoạn bình thường, mỗi node trên segment tree ta sẽ lưu lại 1 cây trie để trả ra kết quả là đáp án lớn nhất x xor 1 số nào đó trong đoạn mà node đó nó quản lý.

23.2 Code mẫu

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

using namespace std;

const int MAX_N = 3e5 + 5;
const int TOP_BIT = 20;

struct TrieNode {
        int child[2];
};

struct Trie {
        vector<TrieNode> data;

        Trie() {
                data.push_back(TrieNode());
        }

        void add(int x) {
                int p = 0;
                for (int i = TOP_BIT; i >= 0; i--) {
                        bool v = x >> i & 1;
                        if (!data[p].child[v]) {
                                data[p].child[v] = data.size();
                                data.push_back(TrieNode());
                        }
                        p = data[p].child[v];
                }
        }

        int get(int x) {
```

```
                int p = 0;
                int ans = 0;
                for (int i = TOP_BIT; i >= 0; i--) {
                        bool v = x >> i & 1;
                        if (data[p].child[v ^ 1]) {
                                ans |= (1 << i);
                                p = data[p].child[v ^ 1];
                        }
                        else {
                                p = data[p].child[v ^ 0];
                        }
                }
                return ans;
        }
};

Trie ST[4 * MAX_N];
int n;
int a[MAX_N];

void update(int id, int l, int r, int i, int val) {
        ST[id].add(val);

        if (l == r) {
                return;
        }

        int mid = (l + r) / 2;

        if (i <= mid) {
                update(id * 2, l, mid, i, val);
        }
        else {
                update(id * 2 + 1, mid + 1, r, i, val);
        }
}

int get(int id, int l, int r, int u, int v, int val) {
        if (u > r || v < l) {
                return 0;
        }

        if (u <= l && r <= v) {
                return ST[id].get(val);
        }

        int mid = (l + r) / 2;
        return max(get(id * 2, l, mid, u, v, val), get(id * 2 + 1, mid + 1, r, u, v, val))
```

```
}

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

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

        int t;
        cin >> t;
        while (t--) {
                int l, r, x;
                cin >> l >> r >> x;
                cout << get(1, 1, n, l, r, x) << '\n';
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

24 Bài X - Tấn và bữa ăn vui vẻ

24.1 Hướng dẫn

Ta để ý là, các học sinh sẽ mua theo thứ tự là, bắt đầu từ học sinh nhiều tiền nhất, sẽ mua món đắt nhất có thể, rồi tiếp đến từ học sinh có nhiều tiền thứ nhì, sẽ mua món đắt nhất có thể trong số những món còn lại, và cứ tiếp như vậy.

Ta có thể trả lời kết quả cho bài toán dễ nhưng với bài có dạng truy vấn như này, ta cần phải làm gì để trả lời truy vấn nhanh:

- Ta có mảng f_i = x, nếu x > 0 sẽ có ý nghĩa là xét từ giá trị i trở lên trên (những món hàng hay người có giá trị < i không tính), sẽ còn x món hàng chưa được mua có giá trị ≥ i sau khi các học sinh hoàn tất việc mua hàng, x < 0 có nghĩa là xét từ giá trị i trở lên trên, sẽ còn -x người chưa được mua có giá trị ≥ i sau khi các học sinh đã hoàn tất việc mua hàng.

- Đáp án sẽ là giá trị i lớn nhất sao cho f_i > 0, còn nếu không có giá trị i nào có f_i > 0 thì là -1.

- Để tạo ra mảng f, với mỗi món hàng có giá trị x, ta cập nhật $f_{1->x}$ lên 1, với mỗi học sinh có x VNĐ, ta cập nhật $f_{1->x}$ xuống -1.

Và để thực hiện việc ấy nhanh ta có thể đưa mảng f lên segment tree, việc cập nhật ta có thể sử dụng lazy để add thêm 1 đoạn 1 giá trị nào đó. Để lấy ra kết quả, mỗi node của segment tree ta sẽ lưu max trong đoạn mà nó quản lý và sử dụng walk, ưu tiên đi về bên phải hơn nếu có tồn tại 1 vị trí nào đó mà đoạn bên phải quản lý có giá trị > 0, hay giá trị node bên phải lớn hơn 0.

24.2 Code mẫu

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

using namespace std;

const int MAX_N = 3e5 + 5;
const int MAX_VAL = 1e6;

int n, m, q;
int a[MAX_N];
int b[MAX_N];
int ST[4 * MAX_VAL + 5];
int lazy[4 * MAX_VAL + 5];

void down(int id) {
        if (!lazy[id]) return;
        lazy[id * 2] += lazy[id];
        ST[id * 2] += lazy[id];
```

```
        lazy[id * 2 + 1] += lazy[id];
        ST[id * 2 + 1] += lazy[id];
        lazy[id] = 0;
}

void update(int id, int l, int r, int u, int v, int val) {
        if (u > r || v < l) {
                return;
        }

        if (u <= l && r <= v) {
                ST[id] += val;
                lazy[id] += val;
                return;
        }

        down(id);

        int mid = (l + r) / 2;
        update(id * 2, l, mid, u, v, val);
        update(id * 2 + 1, mid + 1, r, u, v, val);

        ST[id] = max(ST[id * 2], ST[id * 2 + 1]);
}

int walk(int id, int l, int r) {
        if (l == r) {
                return (ST[id] > 0 ? l : -1);
        }

        down(id);

        int mid = (l + r) / 2;

        if (ST[id * 2 + 1] > 0) {
                return walk(id * 2 + 1, mid + 1, r);
        }
        return walk(id * 2, l, mid);
}

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

        cin >> n >> m;
        for (int i = 1; i <= n; i++) {
                cin >> a[i];
                update(1, 1, MAX_VAL, 1, a[i], 1);
```

```
        }
        for (int i = 1; i <= m; i++) {
                cin >> b[i];
                update(1, 1, MAX_VAL, 1, b[i], -1);
        }

        cin >> q;
        while (q--) {
                int type, i, x;
                cin >> type >> i >> x;

                if (type == 1) {
                        update(1, 1, MAX_VAL, 1, a[i], -1);
                        a[i] = x;
                        update(1, 1, MAX_VAL, 1, a[i], 1);
                }
                else {
                        update(1, 1, MAX_VAL, 1, b[i], 1);
                        b[i] = x;
                        update(1, 1, MAX_VAL, 1, b[i], -1);
                }

                cout << walk(1, 1, MAX_VAL) << '\n';
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

25 Bài Y - Tấn và cú nhảy thần sầu

25.1 Hướng dẫn

Bài này ta có thể gọi dp_i = số bước nhảy nhiều nhất kết thúc tại vị trí i, dp_i = max(1, max(dp_j) + 1) với $1 \le j < i$ và $|h_i - h_j| \ge d$ đáp án bài toán là lấy max toàn bộ dp_i. Ta có thể dễ dàng thực hiện trong độ phức tạp là O(n^2).

Ta cải tiến thuật toán bằng cách thay đổi tý cách gọi dp, dp_{h_i} = số bước nhảy nhiều nhất kết thúc tại toà nhà cao h_i. Khi đến vị trí i, ta tính $dp_{h_i} = max(dp_{h_i}, dp_{h_j} + 1)$ với $|h_i - h_j| \ge d$, thì ta nhận thấy h_j sẽ ở 2 phía là:

- $h_j > h_i$, thì h_j sẽ từ h_k nhỏ nhất sao cho $h_k - h_i \ge d$ đến max độ cao.
- $h_j < h_i$, thì h_j sẽ từ min độ cao đến h_k lớn nhất sao cho $h_i - h_k \ge d$.

Thì để lấy max h_j nhanh, ta nhận xét ở 2 phần thì ta sẽ lấy max 1 đoạn độ cao liên tiếp, nên ta sẽ đưa mảng dp lên segment tree.

Và nhớ $1 \le h_i \le 10^{15}$ nên ta phải nén số độ cao các toà nhà lại rồi mới cho vào mảng dp, tức thay vì dp_{h_i} thì nó sẽ là $dp_{b_{h_i}}$ với b_{h_i} là vị trí tương đối của h_i trong toàn bộ tất cả toà nhà.

25.2 Code mẫu

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

using namespace std;

const int MAX_N = 1e5 + 5;

int n, d, m;
long long a[MAX_N];
int ST[4 * MAX_N];
vector<long long> ns;

void update(int id, int l, int r, int i, int val) {
        if (l == r) {
                ST[id] = max(ST[id], val);
                return;
        }

        int mid = (l + r) / 2;
        if (i <= mid) {
                update(id * 2, l, mid, i, val);
        }
        else {
                update(id * 2 + 1, mid + 1, r, i, val);
```

```
        }

        ST[id] = max(ST[id * 2], ST[id * 2 + 1]);
}

int get(int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return 0;
        }

        if (u <= l && r <= v) {
                return ST[id];
        }

        int mid = (l + r) / 2;
        return max(get(id * 2, l, mid, u, v), get(id * 2 + 1, mid + 1, r, u, v));
}

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

        cin >> n >> d;
        for (int i = 1; i <= n; i++) {
                cin >> a[i];
                ns.push_back(a[i]);
        }

        sort(ns.begin(), ns.end());
        ns.erase(unique(ns.begin(), ns.end()), ns.end());
        m = ns.size();

        int ans = 1;
        for (int i = 1; i <= n; i++) {
                int curr = lower_bound(ns.begin(), ns.end(), a[i]) - ns.begin() + 1;
                int dp = 1;
                int pos = 0;
                int low = curr, high = m;
                while (low <= high) {
                        int mid = (low + high) / 2;
                        if (ns[mid - 1] - ns[curr - 1] >= d) {
                                pos = mid;
                                high = mid - 1;
                        }
                        else {
                                low = mid + 1;
                        }
                }
```

```
                if (pos) {
                        dp = get(1, 1, m, pos, m) + 1;
                }

                pos = 0;
                low = 1, high = curr;
                while (low <= high) {
                        int mid = (low + high) / 2;
                        if (ns[curr - 1] - ns[mid - 1] >= d) {
                                pos = mid;
                                low = mid + 1;
                        }
                        else {
                                high = mid - 1;
                        }
                }

                if (pos) {
                        dp = max(dp, get(1, 1, m, 1, pos) + 1);
                }

                update(1, 1, m, curr, dp);

                ans = max(ans, dp);
        }

        cout << ans;

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

26 Bài Z - Tấn và mùa hè đáng nhớ

26.1 Hướng dẫn

Xét truy vấn a > b, thì ta sẽ quan tâm đến các cây trong đoạn [b + 1; a - 1]. Để trả lời nhanh truy vấn, ta có thể đưa các cây lên segment tree, mỗi node quản lý đoạn [l; r] sẽ lưu:

- val: kết quả tối ưu của việc đi giữa 2 cây nào đó trong đoạn mà node quản lý.

- pref: 1 đoạn đường đi từ l đến 1 cây i ($l \le i \le r$) nào đó (bao gồm cả việc leo cây i) lớn nhất, tức $sum_{d_{l->i-1}} + 2 * h_i$ lớn nhất.

- suff: 1 đoạn đường đi từ 1 cây i nào đó ($l \le i \le r$) (bao gồm cả việc leo cây i) đến r lớn nhất, tức $sum_{d_{i->r}} + 2 * h_i$ lớn nhất (ta tính cả d_r để kết nối với đoạn khác).

Xét 1 đoạn [l; r], gọi mid = (l + r) / 2, curr là đoạn hiện tại, left là đoạn bên trái [l; mid], right là đoạn bên phải [mid + 1; r]:

- $curr_{val} = max(left_{val}, right_{val}, left_{suff} + right_{pref})$, tức giá trị của đoạn [l; r] có thể từ việc đi giữa 2 cây trong đoạn [l; mid] ($left_{val}$), hoặc trong [mid + 1; r] ($right_{val}$), hoặc từ 1 cây nào đó ở đoạn left đến 1 cây nào đến ở đoạn right ($left_{suff} + right_{pref}$).

- $curr_{pref} = max(left_{pref}, sum_{d_{l->mid}} + right_{pref})$, tức giá trị pref của đoạn [l; r] có thể là việc đi từ l đến 1 cây i nào đấy ≤ mid ($left_{pref}$) hoặc đến 1 cây i nào đấy > mid ($sum_{d_{l->mid}} + right_{pref}$).

- $curr_{suff} = max(right_{suff}, left_{suff} + sum_{d_{mid+1->r}})$, tức giá trị suff của đoạn [l; r] có thể là việc đi từ 1 cây i nào đấy > mid đến r ($right_{suff}$) hoặc từ 1 cây $l \le i \le mid$ đến r ($left_{suff} + sum_{d_{mid+1->r}}$).

Nhưng cách này chỉ làm được trên 1 đoạn liên tiếp thôi, lỡ ở truy vấn a < b, thì ta chỉ có thể đi từ b + 1 -> n vòng lại 1 -> a - 1, như này thì nó không còn liên tiếp để ta tạo trên segment tree nữa.
Vậy nên ta có cách là tạo mảng $a_{n+1->n+n} = a_{1->n}$ bằng cách gán $a_{i+n} = a_i, 1 \le i \le n$.
Và segment tree ta cũng tạo trên mảng 2 * n trên, ở truy vấn a > b thì ta vẫn get từ đoạn [b + 1; a - 1] như thường, còn truy vấn a < b thì ta sẽ get từ đoạn [b + 1, a + n - 1].

26.2 Code mẫu

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

using namespace std;

const int MAX_N = 2e5 + 5;

int n, m;
int d[MAX_N], h[MAX_N];
struct Node {
```

```
        long long val, pref, suff, sum;
};
Node ST[4 * MAX_N];

Node merge(const Node& Left, const Node& Right) {
        Node res;

        res.sum = Left.sum + Right.sum;

        res.val = max({Left.val, Right.val, Left.suff + Right.pref});

        res.pref = max({Left.pref, Left.sum + Right.pref});

        res.suff = max({Right.suff, Left.suff + Right.sum});

        return res;
}

void build(int id, int l, int r) {
        if (l == r) {
                ST[id] = {0, 2l * h[l], 2ll * h[l] + d[l], d[l]};
                return;
        }

        int mid = (l + r) / 2;
        build(id * 2, l, mid);
        build(id * 2 + 1, mid + 1, r);

        ST[id] = merge(ST[id * 2], ST[id * 2 + 1]);
}

Node get(int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return {-1, -1, -1, -1};
        }

        if (u <= l && r <= v) {
                return ST[id];
        }

        int mid = (l + r) / 2;
        Node Left = get(id * 2, l, mid, u, v);
        Node Right = get(id * 2 + 1, mid + 1, r, u, v);

        if (Left.val == -1) {
                return Right;
        }
```

```
        if (Right.val == -1) {
                return Left;
        }

        return merge(Left, Right);
}

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

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

        build(1, 1, 2 * n);

        while (m--) {
                int a, b;
                cin >> a >> b;

                if (a <= b) {
                        cout << get(1, 1, 2 * n, b + 1, a + n - 1).val;
                }
                else {
                        cout << get(1, 1, 2 * n, b + 1, a - 1).val;
                }
                cout << '\n';
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

27 Bài ZA - Tấn và dãy số fibonacci

27.1 Hướng dẫn

27.2 Code mẫu

Đọc code đầy đủ hơn ở *đây*.

28 Bài ZB - Tấn và bài toán hóc búa

28.1 Hướng dẫn

Ta nhận thấy rằng, số lượng nghịch thế chỉ tạo ra giữa các số bị thay đổi và những số ở giữa chúng (những số không bị thay đổi nhưng nằm giữa số lớn nhất và số nhỏ nhất bị thay đổi). Số lượng nghịch thế có thể được chia ra làm 2 phần:

- Giữa những số bị thay đổi với nhau, để đếm ta làm bằng cách duyệt qua từng số bị thay đổi theo thứ tự sau khi thay đổi (sau thay đổi, số nào đứng trước thì duyệt trước). Khi duyệt đến số i nào đó, ta muốn biết có bao số đứng trước và có giá trị lớn hơn, để thực hiện điều này, ta có thể nén các số bị thay đổi lại lại (rời rạc hoá) sau đó tạo 1 mảng cnt_j là số lượng số j, thì thứ ta quan tâm sẽ là tổng $cnt_{i+1->m}$ (với m là số lượng số sau khi nén số lại) sau đó cập nhật $cnt_i = 1$. Để thực hiện điều này nhanh, ta có thể đưa mảng cnt lên segment tree.

- Giữa những số bị thay đổi với những số không bị thay đổi. Gọi pos_i là vị trí của số i sau n thay đổi, ta có nhận xét rằng số lượng những số không bị thay đổi tạo ra nghịch thế với số i sẽ là số lượng số không bị thay đổi trong đoạn từ i đến pos_i ($pos_i > i$) hoặc từ pos_i đến i ($i > pos_i$) = $|pos_i - i| + 1-$ (số lượng số bị thay đổi = $|i - j|$ + 1 với j là vị trí của số pos_i) để tìm j ta có thể sử dụng chặt nhị phân.

28.2 Code mẫu

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

using namespace std;

const int MAX_N = 2e5 + 5;

int n, m;
vector<int> ns;
pair<int, int> a[MAX_N];
int pos[MAX_N];
int val[MAX_N];
int ST[4 * MAX_N];

void update(int id, int l, int r, int i, int val) {
        if (l == r) {
                ST[id] += val;
                return;
        }

        int mid = (l + r) / 2;
        if (i <= mid) {
                update(id * 2, l, mid, i, val);
        }
```

```
        else {
                update(id * 2 + 1, mid + 1, r, i, val);
        }

        ST[id] = ST[id * 2] + ST[id * 2 + 1];
}

int get(int id, int l, int r, int u, int v) {
        if (u > r || v < l) {
                return 0;
        }

        if (u <= l && r <= v) {
                return ST[id];
        }

        int mid = (l + r) / 2;
        return get(id * 2, l, mid, u, v) + get(id * 2 + 1, mid + 1, r, u, v);
}

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

        cin >> n;
        for (int i = 1; i <= n; i++) {
                cin >> a[i].first >> a[i].second;
                ns.push_back(a[i].first);
                ns.push_back(a[i].second);
        }

        sort(ns.begin(), ns.end());
        ns.erase(unique(ns.begin(), ns.end()), ns.end());
        m = ns.size();

        for (int i = 1; i <= m; i++) {
                pos[i] = ns[i - 1];
                val[i] = i;
        }

        for (int i = 1; i <= n; i++) {
                int u = lower_bound(ns.begin(), ns.end(), a[i].first) - ns.begin() + 1;
                int v = lower_bound(ns.begin(), ns.end(), a[i].second) - ns.begin() + 1;
                swap(pos[val[u]], pos[val[v]]);
                swap(val[u], val[v]);
        }

        long long ans = 0;
```

```
        // for theo pos
        for (int i = 1; i <= m; i++) {
                ans += abs(ns[i - 1] - pos[i]) - abs(lower_bound(ns.begin(), ns.end(), pos
                ans += get(1, 1, m, val[i], m);
                update(1, 1, m, val[i], 1);
        }

        cout << ans;

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.

29 Bài ZC - MARK

29.1 Hướng dẫn

Chưa xét đến các truy vấn thay đổi, để tìm đáp án với mảng a, ta sẽ lần lượt thực hiện thao tác chọn 2 vị trí i j sao cho $a_i = a_j$ rồi thay thế nó thành số $a_i + 1$, ta sẽ làm việc đó từ các số có giá trị nhỏ nhất lên dần cho tới khi không thể thực hiện được nữa thì đáp án của ta là số lớn nhất trong số những số còn lại.

Ta có thể nhìn theo 1 gốc độ khác, gọi mảng cnt_i là số lượng số i trong mảng a khi các số đẩy lên đến số i.
Cách trên nó cũng tương đương với việc đi giá trị i từ nhỏ đến lớn, ta đẩy lên i + 1, mỗi 2 số i thành 1 số i + 1 nên số lượng mà số i đẩy lên i + 1 là $\lfloor \frac{cnt_i}{2} \rfloor$ thì ta thực hiện thế cho đến khi không làm được nữa. Đáp án của ta là số i lớn nhất sao cho $cnt_i > 0$.

Giờ xét đến những truy vấn thay đổi $a_k = l$.
Khi thay đổi sẽ gồm 2 lần là, bỏ đi giá trị a_k cũ và thêm vào giá trị $a_k = l$ mới:

- Ở việc bỏ đi 1 giá trị x, nhận thấy rằng:

+ Nếu cnt_x **lẻ**, số lượng mà nó đẩy lên x + 1 là không đổi *(ví dụ cnt_x = 5, số lượng nó đẩy lên x + 1 là $\lfloor \frac{5}{2} \rfloor$ = 2, số lượng x giảm đi 1 là còn 4, số lượng nó đẩy lên x + 1 là $\lfloor \frac{4}{2} \rfloor$ = 2 không thay đổi so với trước)*, thì ngoài cnt_x giảm đi 1 ra thì còn lại giữ nguyên.

+ Nếu cnt_x **chẵn**, thì số lượng mà nó đẩy lên x + 1 sẽ giảm đi 1 *(ví dụ cnt_x = 4, số lượng nó đẩy lên x + 1 là $\lfloor \frac{4}{2} \rfloor$ = 2, x giảm đi 1 số còn 3, giờ số lượng nó đẩy lên x + 1 là $\lfloor \frac{3}{2} \rfloor$ = 1)*. Tương tự, nếu cnt_{x+1} trước đó chẵn, bị giảm đi 1 từ x, thì số lượng nó đẩy lên x + 2 giảm đi 1,... đến giá trị y ($y > x$) đầu tiên mà cnt_y **lẻ**, lúc này tại số lượng y - 1 đẩy lên y giảm đi 1, làm cho số lượng y đẩy lên y + 1 không thay đổi *(lý do đã giải thích ở trên, trong trường hợp cnt_x lẻ bị giảm đi 1)*, dừng lại tại đây bởi không còn số nào bị thay đổi nữa.
→ Tổng kết lại là ta sẽ tìm số y ($y \geq x$) đầu tiên mà cnt_y **lẻ**, giảm $cnt_{x->y}$ xuống 1.

- Ở việc thêm vào 1 giá trị x, nhận thấy rằng:

+ Nếu cnt_x **chẵn**, tăng lên 1, số lượng nó đẩy lên x + 1 không thay đổi *(ví dụ cnt_x = 4, số lượng nó đẩy lên x + 1 là $\lfloor \frac{4}{2} \rfloor$ = 2, khi tăng thêm 1 thành 5, số lượng x đẩy lên x + 1 là $\lfloor \frac{5}{2} \rfloor$ = 2)*, vậy nên ngoài việc cnt_x tăng lên 1 thì không còn gì thay đổi nữa.

+ Nếu cnt_x **lẻ**, tăng lên 1 thì số lượng đẩy sang x + 1 sẽ tăng lên 1 *ví dụ cnt_x = 3, số lượng nó đẩy lên x + 1 là $\lfloor \frac{3}{2} \rfloor$ = 1, được tăng lên 1 thành 4, số lượng nó sẽ đẩy sang x + 1 bây giờ là $\lfloor \frac{4}{2} \rfloor$ = 2*. Tương tự vậy nếu cnt_{x+1} trước đó là lẻ, được tăng lên 1 từ x, số lượng đẩy sang x + 2 sẽ tăng lên 1,... đến giá trị y **chẵn** đầu tiên ($y > x$), nhận được thêm 1 từ y - 1 nhưng số lượng y đẩy sang y + 1 sẽ không đổi *(lý do đã được giải thích ở trên, trong trường hợp cnt_x chẵn được tăng lên 1)*, nên dừng lại tại đây.
→ Tổng kết lại là ta sẽ tìm số y ($y \geq x$) đầu tiên mà cnt_y **chẵn**, tăng $cnt_{x->y}$ lên 1.

Đến đây thì cũng dễ cài đặt và cũng có nhiều cách cài đặt, ví dụ như sử dụng segment tree. Ở thao tác cập nhật thêm xoá thì ta có thể sử dụng lazy, thao tác tìm vị trí y, hoặc tìm đáp án thì ta có thể sử dụng kết hợp segment tree và chặt nhị phân hoặc sử dụng walk để giảm độ phức tạp.
Ở code mẫu cài đặt theo cách, tại vì ta có nhận xét, mọi thứ ta làm, chỉ liên quan đến tính chẵn lẻ của giá trị cnt, còn giả sử đáp án sau lần thay đổi là x, thì chắc chắn $cnt_x = 1$ hay cnt_x lẻ, nên ta có thể cài cnt ở dưới dạng chẵn lẻ là được rồi, rồi lên đưa lên mảng segment tree. Mỗi node của segment tree sẽ lưu số lượng số có giá trị cnt chẵn, giá trị cnt lẻ trong đoạn mà nó quản lý, từ đây việc tìm giá trị y đầu tiên ≥ x có cnt chẵn hay lẻ bằng walk cũng rất dễ, lấy ra đáp án cũng thuận tiện bằng walk.

29.2 Code mẫu

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

using namespace std;

const int MAX_N = 2e5 + 100;

int n, q;
int ST[4 * MAX_N][2];
int lazy[4 * MAX_N];
int a[MAX_N];

void init(int id, int l, int r) {
        ST[id][0] = r - l + 1;
        lazy[id] = -1;

        if (l == r) {
                return;
        }

        int mid = (l + r) / 2;
        init(id * 2, l, mid);
        init(id * 2 + 1, mid + 1, r);
}

void down(int id, int l, int r) {
        if (lazy[id] == -1) {
                return;
        }

        bool k = lazy[id];
        lazy[id] = -1;
```

```
        int mid = (l + r) / 2;

        lazy[id * 2] = lazy[id * 2 + 1] = k;
        ST[id * 2][!k] = ST[id * 2 + 1][!k] = 0;
        ST[id * 2][k] = mid - l + 1;
        ST[id * 2 + 1][k] = r - mid;
}

void update(int id, int l, int r, int u, int v, bool k) {
        if (u > r || v < l) {
                return;
        }

        if (u <= l && r <= v) {
                ST[id][!k] = 0;
                ST[id][k] = r - l + 1;
                lazy[id] = k;
                return;
        }

        down(id, l, r);

        int mid = (l + r) / 2;
        update(id * 2, l, mid, u, v, k);
        update(id * 2 + 1, mid + 1, r, u, v, k);

        ST[id][0] = ST[id * 2][0] + ST[id * 2 + 1][0];
        ST[id][1] = ST[id * 2][1] + ST[id * 2 + 1][1];
}

int find(int id, int l, int r, int x, int k) {
        if (r < x || !ST[id][k]) {
                return -1;
        }

        if (l == r) {
                return l;
        }

        down(id, l, r);

        int mid = (l + r) / 2;
        int left = find(id * 2, l, mid, x, k);
        if (left == -1) {
                return find(id * 2 + 1, mid + 1, r, x, k);
        }
        return left;
}
```

```
int get(int id, int l, int r) {
        if (l == r) {
                return l;
        }

        down(id, l, r);

        int mid = (l + r) / 2;
        if (ST[id * 2 + 1][1]) {
                return get(id * 2 + 1, mid + 1, r);
        }
        return get(id * 2, l, mid);
}

void add(int x) {
        int y = find(1, 1, MAX_N - 1, x, 0);
        update(1, 1, MAX_N - 1, x, y - 1, 0);
        update(1, 1, MAX_N - 1, y, y, 1);
}

void erase(int x) {
        int y = find(1, 1, MAX_N - 1, x, 1);
        update(1, 1, MAX_N - 1, x, y - 1, 1);
        update(1, 1, MAX_N - 1, y, y, 0);
}

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

        init(1, 1, MAX_N - 1);

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

        while (q--) {
                int k, l;
                cin >> k >> l;

                erase(a[k]);
                a[k] = l;
                add(a[k]);

                cout << get(1, 1, MAX_N - 1) << '\n';
```

```
        }

        return 0;
}
```

Đọc code đầy đủ hơn ở *đây*.