F. Partition Score

Intuition

আমাদের একটি array দেওয়া আছে, যেটিকে দুটি disjoint subset S1 এবং S2-তে ভাগ করতে হবে, যেন প্রত্যেকটার সাইজ কমপক্ষে K হয়।
Score নির্ধারণ করা হয়েছে এভাবে:
score = min(S1) + max(S1) + min(S2) + max(S2)
আমাদের কাজ হলো এই score টা যত বড় সম্ভব করা।

Observation

- Score বড় করতে হলে subset গুলোর **max** মানগুলো যত বড় হবে, তত ভালো।
- আর **min** মানগুলো যত ছোট, তত ভালো না (কারণ min ছোট মান যোগ হয়), কিন্তু যেহেতু দুটো subset লাগবে, তাই একটা subset এর min ছোট রাখলেও অন্যটার max বড় রাখতে পারি।

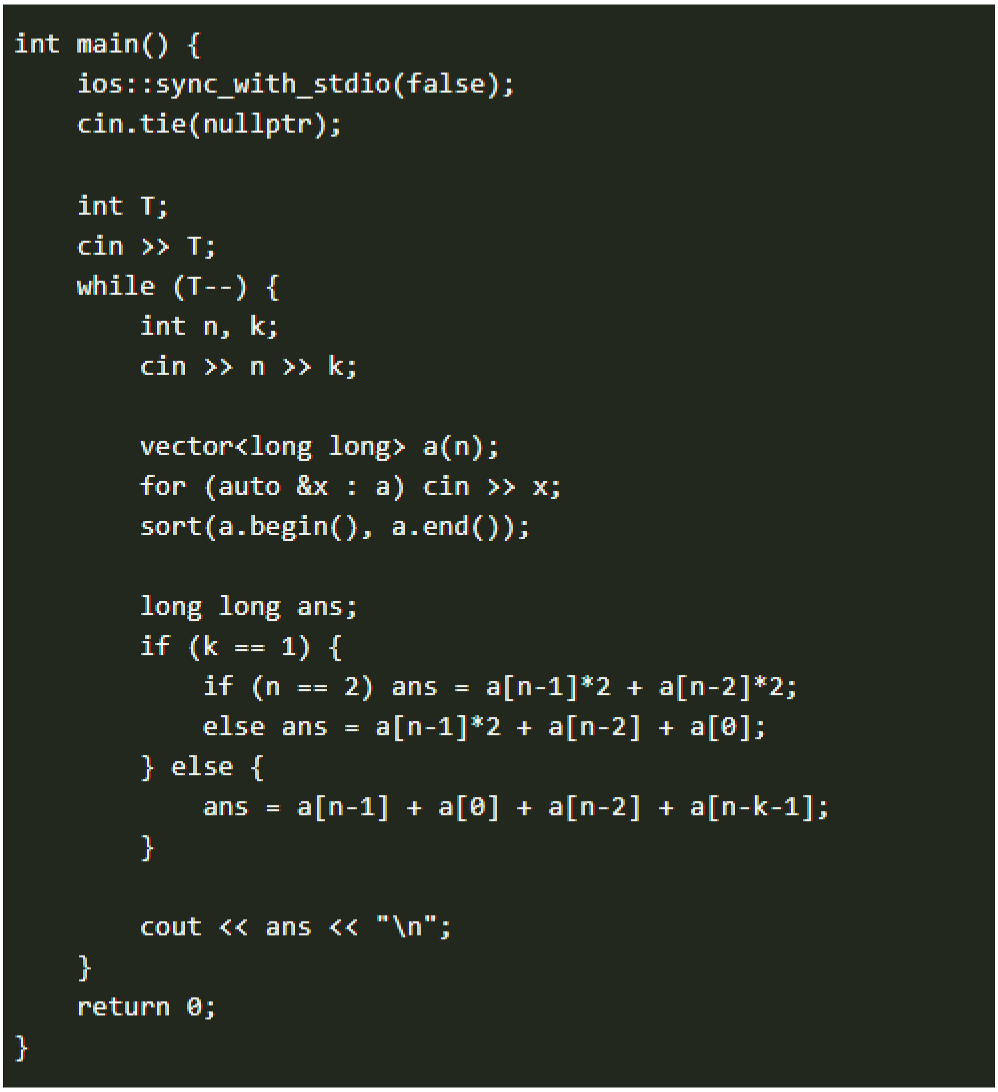
Array টা sort করলে দেখা যায়:

- সবচেয়ে বড় মানগুলো শেষে থাকে,
- সবচেয়ে ছোট মানগুলো শুরুতে।

তাহলে, sorting এর পর সহজে আমরা এই চারটি মান নিতে পারি:
max(S1) = A[n-1]
min(S1) = A[0]
max(S2) = A[n-2]
min(S2) = A[n-k-1]

Solution Idea:

1. Array sort করো।
2. যদি K = 1 হয়:
    - তখন একটিমাত্র এলিমেন্টও subset হতে পারে।
    - এই ক্ষেত্রে আমরা max(S1) = v[n-1], max(S2) = v[n-2], min(S1) = v[0] নিয়ে সরাসরি হিসাব করব।
3. অন্য ক্ষেত্রে:
   score = v[n-1] + v[0] + v[n-2] + v[n-k-1]



```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int T;
    cin >> T;
    while (T--) {
        int n, k;
        cin >> n >> k;

        vector<long long> a(n);
        for (auto &x : a) cin >> x;
        sort(a.begin(), a.end());

        long long ans;
        if (k == 1) {
            if (n == 2) ans = a[n-1]*2 + a[n-2]*2;
            else ans = a[n-1]*2 + a[n-2] + a[0];
        } else {
            ans = a[n-1] + a[0] + a[n-2] + a[n-k-1];
        }

        cout << ans << "\n";
    }
    return 0;
}
```

G - Maximize Hamming Distance

Intuition

আমাদের M সংখ্যক স্ট্রিং দেওয়া আছে, প্রতিটি স্ট্রিংয়ের দৈর্ঘ্য N। প্রতিটি ক্যারেক্টার হয় ‘0’, ‘1’, অথবা ‘?’ হতে পারে।
আমাদের কাজ হলো সবগুলো ‘?’ এমনভাবে 0 বা 1 দিয়ে replace করা, যেন সকল স্ট্রিংয়ের **pair-**এর মধ্যে **Hamming distance-**এর যোগফল সর্বোচ্চ হয়।

Hamming distance মানে হলো — দুটি সমান দৈর্ঘ্যের স্ট্রিংয়ে যত পজিশনে তাদের ক্যারেক্টার ভিন্ন, সেই সংখ্যা।

Observation

প্রতিটি কলাম (column) আলাদা করে ভাবলে দেখা যায়,

- একটি কলামে কিছু ‘0’, কিছু ‘1’, আর কিছু ‘?’ আছে।

- এই কলামেই নির্ধারণ হবে কতগুলো pair ভিন্ন হবে।

ধরো কোনো কলামে,

- count0 = কতগুলো 0 আছে
- count1 = কতগুলো 1 আছে
- other = কতগুলো '?' আছে

যদি আমরা ‘?’ গুলো এমনভাবে বসাই যাতে ‘0’ আর ‘1’-এর সংখ্যা কাছাকাছি হয়, তাহলে pair গুলোর মধ্যে পার্থক্য (difference) সবচেয়ে বেশি হবে।

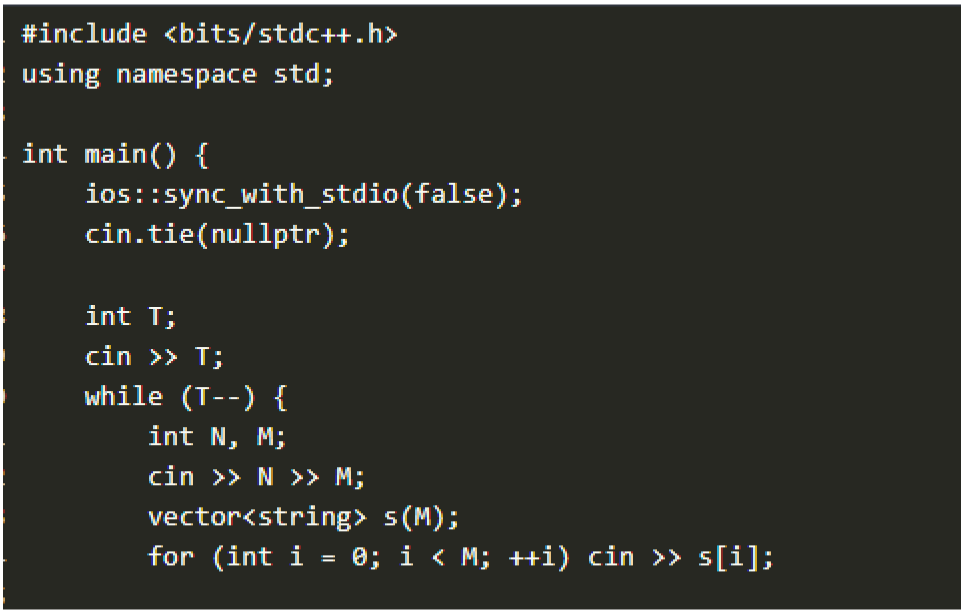
প্রতিটি কলামে Hamming contribution হবে:
count0 × count1

অর্থাৎ, যতগুলো ‘0’ আছে × যতগুলো ‘1’ আছে — ততগুলো pair এই কলামে ভিন্ন হবে।
তাই আমাদের লক্ষ্য — প্রতিটি কলামে 0 এবং 1-এর সংখ্যা যতটা সম্ভব balance করা।

Solution Idea:

1. প্রতিটি test case-এর জন্য N এবং M ইনপুট নাও।
2. সব স্ট্রিং একটি vector<string> এ রাখো।
3. প্রতিটি কলামের জন্য:
    - count0, count1, এবং other গণনা করো।
    - ‘?’ গুলো এমনভাবে বসাও যাতে 0 এবং 1 এর সংখ্যা যতটা সম্ভব কাছাকাছি হয়।
    - অর্থাৎ যদি ‘?’ গুলো যোগ করে balance করা যায়, তাহলে অর্ধেক 0 তে, অর্ধেক 1 তে দাও।

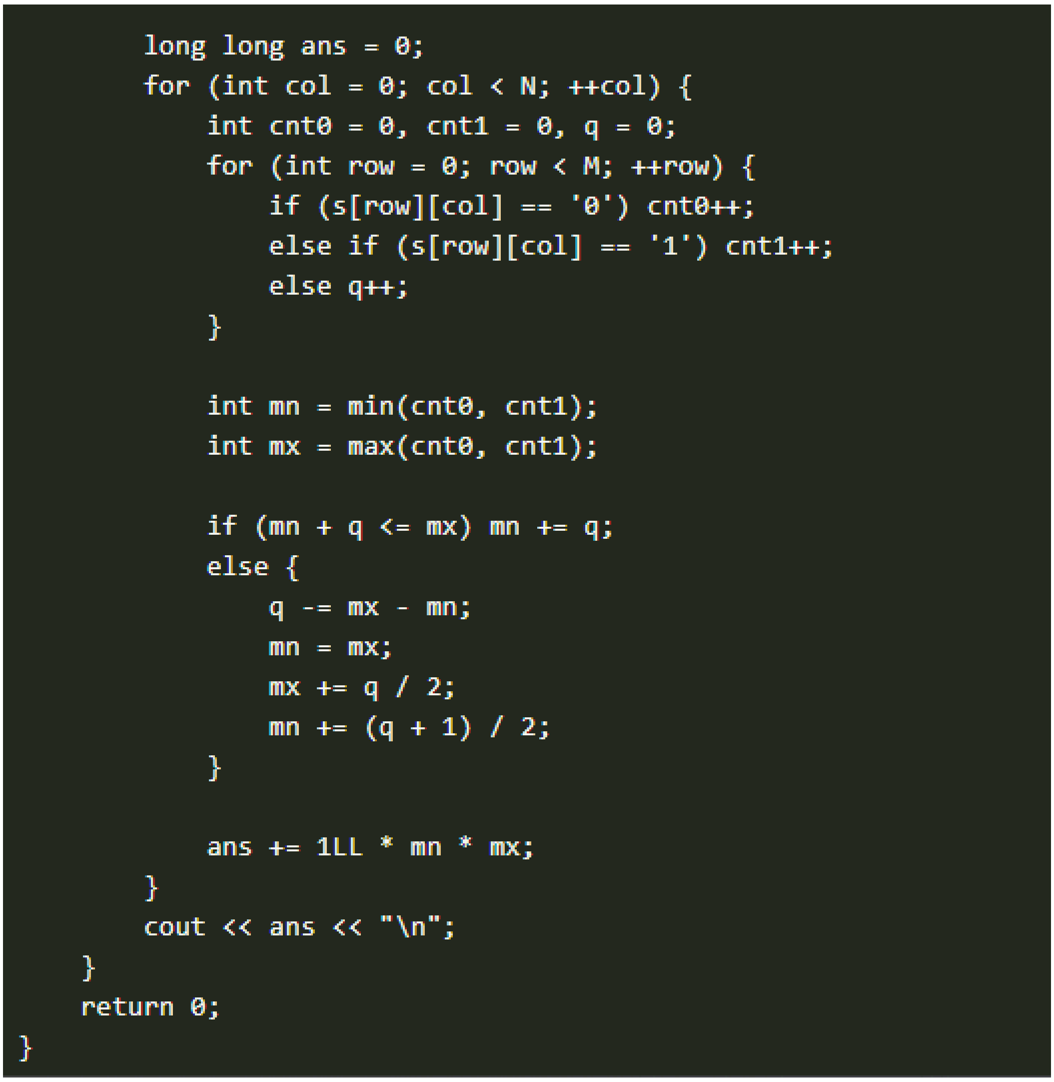
4. প্রতিটি কলামের জন্য min_count × max_count যোগ করো।
5. সব কলামের যোগফলই হবে চাওয়া maximum Hamming distance।



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

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

    int T;
    cin >> T;
    while (T--) {
        int N, M;
        cin >> N >> M;
        vector<string> s(M);
        for (int i = 0; i < M; ++i) cin >> s[i];
```



```
        long long ans = 0;
        for (int col = 0; col < N; ++col) {
            int cnt0 = 0, cnt1 = 0, q = 0;
            for (int row = 0; row < M; ++row) {
                if (s[row][col] == '0') cnt0++;
                else if (s[row][col] == '1') cnt1++;
                else q++;
            }

            int mn = min(cnt0, cnt1);
            int mx = max(cnt0, cnt1);

            if (mn + q <= mx) mn += q;
            else {
                q -= mx - mn;
                mn = mx;
                mx += q / 2;
                mn += (q + 1) / 2;
            }

            ans += 1LL * mn * mx;
        }
        cout << ans << "\n";
    }
    return 0;
}
```

H - Make K Most Frequent

Intuition

আমাদের একটি array AAA দেওয়া আছে, যেখানে কিছু সংখ্যা একাধিকবার আছে।
আমরা চাই একটি নির্দিষ্ট সংখ্যা KKK কে সবচেয়ে বেশি বার থাকা সংখ্যা **(most frequent)** করতে।

তবে আমরা কেবল দুটি ধরণের অপারেশন করতে পারি:

- Array-এর **prefix (শুরুর দিক)** মুছে ফেলা।
- অথবা **suffix (শেষের দিক)** মুছে ফেলা।
  (পুরো array একসাথে মুছে ফেলা যাবে না)

আমাদের লক্ষ্য, যত কম সংখ্যক অপারেশনে KKK-কে সবচেয়ে frequent বানানো।

Observation

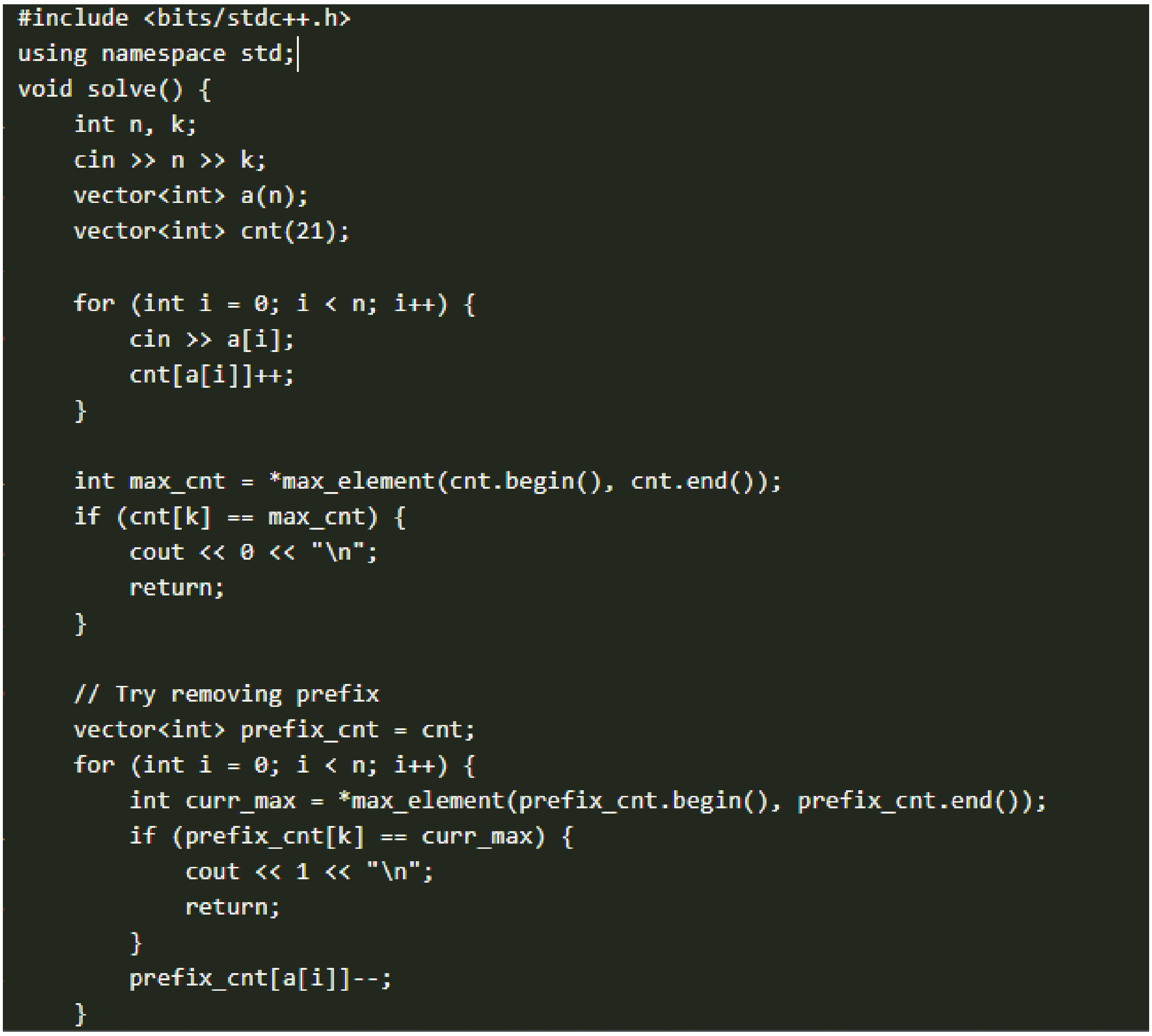
প্রথমে দেখা যাক, KKK-এর frequency কত, এবং অন্য সংখ্যাগুলোর মধ্যে সবচেয়ে বেশি frequency কার।
যদি KKK-এর frequency আগেই সর্বাধিক হয়, তাহলে **0 operation** লাগবে।

অন্যথায়:

- Prefix বা Suffix মুছে এমন অংশ রাখতে হবে, যেখানে KKK-এর frequency অন্যদের চেয়ে বেশি হয়ে যায়।
- যদি কোনো এক পাশে থেকে কিছু মুছে KKK most frequent হয়, তাহলে **1 operation** যথেষ্ট।
- যদি না হয়, তাহলে **2 operation** (একবার prefix, একবার suffix মুছে) করলেই হবে, কারণ KKK আছে , গ্যারান্টি করা আছে।

Solution Idea:

1. প্রতিটি টেস্ট কেসে N,KN, KN,K ইনপুট নাও।
2. array AAA ইনপুট নিয়ে প্রতিটি সংখ্যার frequency গণনা করো।
3. যদি KKK-এর frequency সর্বাধিক হয় → উত্তর 0।
4. Prefix থেকে একে একে বাদ দাও এবং প্রত্যেক ধাপে frequency চেক করো,
   যদি কোনো সময় KKK-এর frequency সর্বাধিক হয়, উত্তর 1।
5. একইভাবে Suffix দিক থেকেও চেক করো।
6. যদি prefix বা suffix কোনোটিতেই সম্ভব না হয়, উত্তর 2।

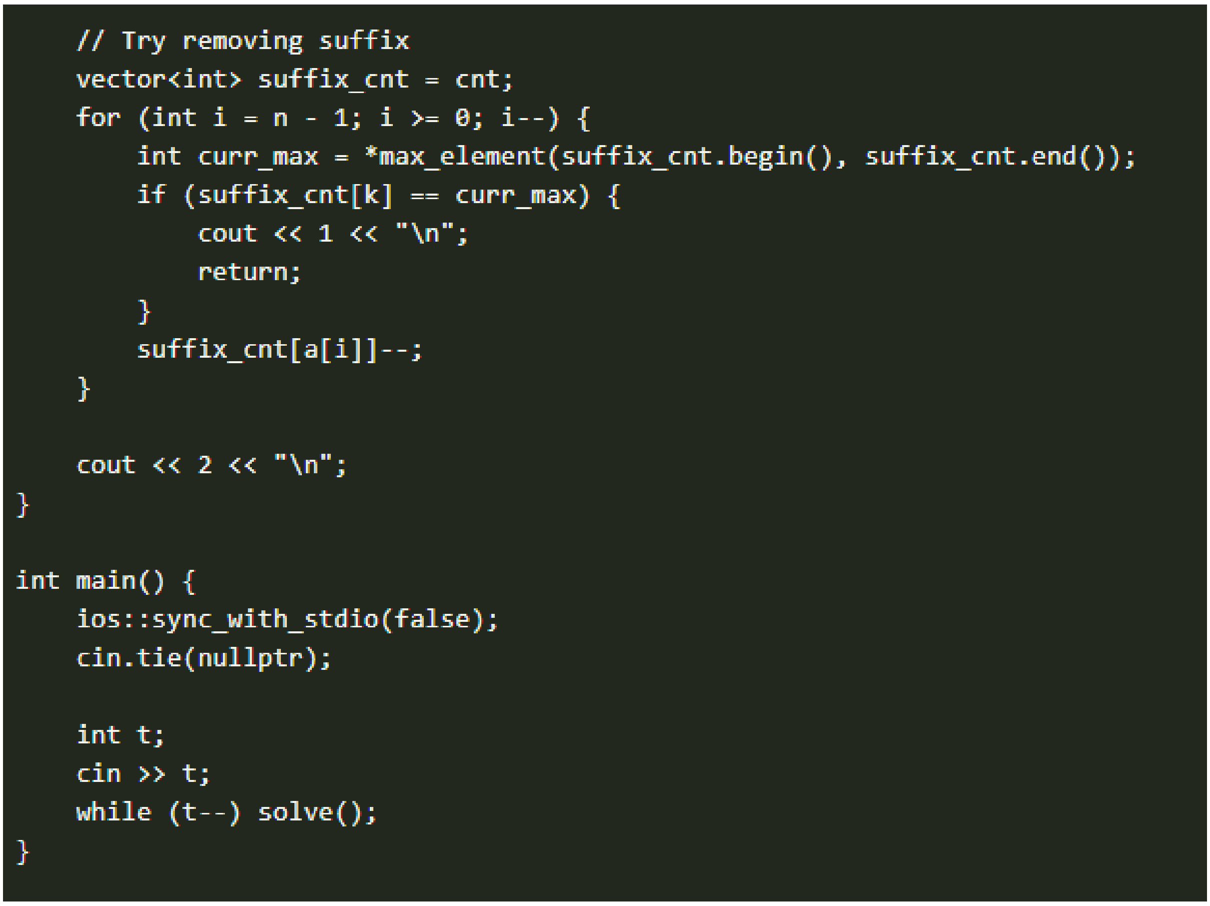


```
#include <bits/stdc++.h>
using namespace std;
void solve() {
    int n, k;
    cin >> n >> k;
    vector<int> a(n);
    vector<int> cnt(21);

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

    int max_cnt = *max_element(cnt.begin(), cnt.end());
    if (cnt[k] == max_cnt) {
        cout << 0 << "\n";
        return;
    }

    // Try removing prefix
    vector<int> prefix_cnt = cnt;
    for (int i = 0; i < n; i++) {
        int curr_max = *max_element(prefix_cnt.begin(), prefix_cnt.end());
        if (prefix_cnt[k] == curr_max) {
            cout << 1 << "\n";
            return;
        }
        prefix_cnt[a[i]]--;
    }
```



```
    // Try removing suffix
    vector<int> suffix_cnt = cnt;
    for (int i = n - 1; i >= 0; i--) {
        int curr_max = *max_element(suffix_cnt.begin(), suffix_cnt.end());
        if (suffix_cnt[k] == curr_max) {
            cout << 1 << "\n";
            return;
        }
        suffix_cnt[a[i]]--;
    }

    cout << 2 << "\n";
}

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

    int t;
    cin >> t;
    while (t--) solve();
}
```