Codefoces Round 941, Div.2

<https://codeforces.com/contest/1966?locale=en>

***Problem 1 (1966A) Card Exchange.***

https://codeforces.com/contest/1966/problem/A?locale=en

***Code:***

#include <iostream>

#include <unordered\_map>

#include <vector>

using namespace std;

int main() {

int t;

cin >> t;

for (int tc = 0; tc < t; tc++) {

int n, k;

cin >> n >> k;

vector<int> cards(n);

unordered\_map<int, int> ct;

int ans = n;

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

cin >> cards[i];

if (ct.find(cards[i]) != ct.end()) {

ct[cards[i]]++;

} else {

ct[cards[i]] = 1;

}

if (ct[cards[i]] >= k) {

ans = k - 1;

}

}

cout << ans << endl;

}

return 0;

}

***Idea:***

If you initially do not have at least k copies of any number, you cannot perform any operations, and the answer will be n.

Otherwise, you can always reduce the number of cards to k−1 using the following algorithm:

1. Select any card that you have k copies of and delete those k copies.

2. If there are no more cards, just take any k−1 cards and finish the process.

3. Otherwise, let x be the number on any card you have. Take k−1 copies of x. Now you have at least k copies of x, so go back to step 1.

Since the total number of cards decreases at each step, this process is always completed and you always get k−1 cards.

Also, since the total number of cards decreases by one at each step, and you can't perform any operations if you have fewer than k cards, it's impossible to do better than k−1, so our k−1 solution is optimal.

***Algorithm complexity:***

O(n)

***Problem 2. (1966B) Rectangle Filling.*** <https://codeforces.com/contest/1966/problem/B?locale=en>

***Code:***

#include <iostream>

#include <vector>

#include <string>

using namespace std;

int main() {

int t;

cin >> t;

for (int tc = 0; tc < t; tc++) {

int n, m;

cin >> n >> m;

vector<string> gr(n);

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

cin >> gr[i];

}

string ans = "YES";

if (gr[0][0] != gr[n - 1][m - 1]) {

bool impossible = true;

for (int j = 0; j < m - 1; j++) {

if (gr[0][j] != gr[0][j + 1] || gr[n - 1][j] != gr[n - 1][j + 1]) {

impossible = false;

break;

}

}

if (impossible) {

ans = "NO";

}

impossible = true;

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

if (gr[i][0] != gr[i + 1][0] || gr[i][m - 1] != gr[i + 1][m - 1]) {

impossible = false;

break;

}

}

if (impossible) {

ans = "NO";

}

}

cout << ans << endl;

}

return 0;

}

***Idea:***

NO, if all the squares in the top row are the same color, all the squares in the bottom row are the same color and these two colors are different

NO, if all the squares in the leftmost column are the same color, all the squares in the rightmost column are the same color and the two colors are different

YES in all other cases

***Algorithm complexity:***

O(n\*m)

***Problem 3.(1966C) Everything Nim***

<https://codeforces.com/contest/1966/problem/C?locale=en>

***Code:***

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

int t;

cin >> t;

for (int tc = 0; tc < t; tc++) {

int n;

cin >> n;

vector<int> a(n);

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

cin >> a[i];

}

int maxsize = \*max\_element(a.begin(), a.end());

sort(a.begin(), a.end());

int mexsize = 1;

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

if (a[i] == mexsize) {

mexsize = mexsize + 1;

}

}

if (mexsize > maxsize) {

cout << (maxsize % 2 == 1 ? "Alice" : "Bob") << endl;

} else {

cout << (mexsize % 2 == 1 ? "Alice" : "Bob") << endl;

}

}

return 0;

}

***Idea:***

If the smallest pile has size 1, that Alice should choose k=1 on her first move. Therefore, we can imagine that 1 is subtracted from all the piles and determine who wins, provided that Bob goes first. We can repeat this process by swapping the first player back and forth until there is a pile of size 1. At this moment, we are in one of two states:

1. If there are no more piles left, the first player loses because he cannot make any moves.
2. Otherwise, the smallest heap has size x≥2. We can show that the first player will always win. To do this, consider what happens if the first player chooses k=x:

- If this leads to a losing state for the next player, then the first player can choose k=x and win.

- Otherwise, the state reached by choosing k=x is a winning one for the next player. Therefore, the first player can choose k=x−1, forcing the second player to choose k = 1. Now the first player will be in a winning state and will be able to win the game.

To implement this solution, we need to keep track of the largest heap size a and the smallest positive integer b, which is not the size of the heap

If b>a, then Alice and Bob will be forced to choose k=1 until the end of the game, so the parity of a determines the winner.

Otherwise, they will eventually reach a state where the minimum heap size is at least 2, so the parity of b determines the winner.

***Algorithm complexity:***

O(n)