



vovuh's blog

Codeforces Round #634 (Div. 3) Editorial

 By [vovuh](#), [history](#), 11 hours ago,  

1335A - Candies and Two Sisters

Idea: [vovuh](#)[Tutorial](#)

1335A - Candies and Two Sisters

 The answer is $\lfloor \frac{n-1}{2} \rfloor$, where $\lfloor x \rfloor$ is x rounded down.
[Solution](#)

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

using namespace std;

int main() {
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
    //    freopen("output.txt", "w", stdout);
#endif

    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        cout << (n - 1) / 2 << endl;
    }

    return 0;
}
```

1335B - Construct the String

Idea: [MikeMirzayanov](#)[Tutorial](#)

1335B - Construct the String

If we represent letters with digits, then the answer can be represented as $1, 2, \dots, b, 1, 2, \dots, b, \dots$. There is no substring containing more than b distinct characters and each substring of length a contains exactly b distinct characters because of the condition $b \leq a$.

Time complexity: $O(n)$.[Solution](#)

→ Pay attention

 Before contest
[Codeforces Round #635 \(Div. 1\)](#)
 25:19:22

 Before contest
[Codeforces Round #635 \(Div. 2\)](#)
 25:19:22
[Register now »](#)
 *has extra registration

 Like 105 people like this. [Sign Up](#) to see what your friends like.

→ hawkvine

 Rating: **1413**
 Contribution: 0


hawkvine

- [Settings](#)
- [Blog](#)
- [Teams](#)
- [Submissions](#)
- [Talks](#)
- [Contests](#)

→ Top rated

#	User	Rating
1	tourist	3733
2	MiFaFaOvO	3681
3	Um_nik	3493
4	apiadu	3397
5	300iq	3317
6	ecnerwala	3273
7	maroonrk	3243
8	LHiC	3229
9	TLE	3223
10	Benq	3213

[Countries](#) | [Cities](#) | [Organizations](#)
[View all →](#)

→ Top contributors

#	User	Contrib.
1	antontrygubO_o	192
2	Errichto	186
3	vovuh	172
3	pikmike	172
5	tourist	167
6	ko_osaga	164
6	McDic	164
8	Um_nik	163
9	Radewoosh	162
10	300iq	155

[View all →](#)

→ Find user

Handle:

Find

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

using namespace std;

int main() {
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
    //    freopen("output.txt", "w", stdout);
#endif

    int t;
    cin >> t;
    while (t--) {
        int n, a, b;
        cin >> n >> a >> b;
        for (int i = 0; i < n; ++i) {
            cout << char('a' + i % b);
        }
        cout << endl;
    }

    return 0;
}
```

1335C - Two Teams Composing

Idea: MikeMirzayanov

Tutorial

1335C - Two Teams Composing

Let the number of students with the skill i is cnt_i and the number of different skills is $diff$. Then the size of the first team can not exceed $diff$ and the size of the second team can not exceed $maxcnt = \max(cnt_1, cnt_2, \dots, cnt_n)$. So the answer is not greater than the minimum of these two values. Moreover, let's take a look at the skill with a maximum value of cnt . Then there are two cases: all students with this skill go to the second team then the sizes of teams are at most $diff - 1$ and $maxcnt$ correspondingly. Otherwise, at least one student with this skill goes to the first team and the sizes are at most $diff$ and $maxcnt - 1$ correspondingly. So the answer is $\max(\min(diff - 1, maxcnt), \min(diff, maxcnt - 1))$.

Time complexity: $O(n)$.Solution

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

using namespace std;

int main() {
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
    //    freopen("output.txt", "w", stdout);
#endif

    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> cnt(n + 1);
        for (int i = 0; i < n; ++i) {
            int x;
            cin >> x;
```

→ Recent actions

- vovuh → [Codeforces Round #634 \(Div. 3\)](#) 🔒
- Sooke → [Codeforces Round #635](#) 🔒
- Errichto → [Coding Until I hit 100k Subscribers](#) 🔒
- vovuh → [Codeforces Round #634 \(Div. 3\) Editorial](#) 🔒
- Yandex → [Happy Birthday, Codeforces!](#) 🔒
- gourav_raj → [1<x vs pow\(2,x\)](#) 🌱
- kuroniorz → [Cheating in Codejam Round 1A](#) 🔒
- TheDark_Knight → [Stuck with TLE for a problem](#) 🔒
- thedeater → [Div-3 F Problem Not clear](#) 🔒
- MikeMirzayanov → [Rule about third-party code is changing](#) 🔒
- vovuh → [Codeforces Round #627 \(Div. 3\) Editorial](#) 🌱
- overwrite → [Combinatorics resources](#) 🔒
- kazama460 → [Graph Theory Course For Beginners : CodeNCode \(YouTube\)](#) 🔒
- sava-cska → [TCO20 Algorithm — Registration on Round 1A](#) 🔒
- w33z8kqrqk8zzx33 → [\[Bug\] An extremely DP problem](#) 🔒
- Surya1231 → [All Div 3 , Div 2 , Educational Round links](#) 🌱
- yash2040 → [Adding default values in a Map in C++ STL](#) 🔒
- pikmike → [Educational Codeforces Round 83 Editorial](#) 🔒
- McDic → [Codeforces Round #633 Editorial](#) 🔒
- 126378 → [Is here any plagiarism checker in Codeforces?](#) 🌱
- scanhex → [Codeforces Round #534 — Editorial](#) 🔒
- Arpa → [Call for problem setting on HackerEarth](#) 🔒
- retr0coderxyz → [CSES Movie Festival II](#) 🔒
- pikmike → [Educational Codeforces Round 85 \[Rated for Div. 2\]](#) 🔒
- Mr_No_One → [Recieved Mail from codeforces false plagiarism](#) 🔒

[Detailed →](#)

```

        ++cnt[x];
    }
    int mx = *max_element(cnt.begin(), cnt.end());
    int diff = n + 1 - count(cnt.begin(), cnt.end(), 0);
    cout << max(min(mx - 1, diff), min(mx, diff - 1)) <<
endl;
}

return 0;
}

```

1335D - Anti-Sudoku

Idea: **vovuh**Tutorial

1335D - Anti-Sudoku

Well, if we replace all occurrences of the number 2 with the number 1, then the initial solution will be anti-sudoku. It is easy to see that this replacement will make exactly two copies of 1 in every row, column, and block. There are also other correct approaches but I found this one the most pretty.

Solution

```

#include <bits/stdc++.h>

using namespace std;

int main() {
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
    //    freopen("output.txt", "w", stdout);
#endif

    int t;
    cin >> t;
    while (t--) {
        for (int i = 0; i < 9; ++i) {
            string s;
            cin >> s;
            for (auto &c : s) if (c == '2') c = '1';
            cout << s << endl;
        }
    }

    return 0;
}

```

1335E1 - Three Blocks Palindrome (easy version)

Idea: **MikeMirzayanov**Tutorial

1335E1 - Three Blocks Palindrome (easy version)

Let's precalculate for each number c (0-indexed) the array $pref_c$ of length $n + 1$, where $pref_c[i]$ is the number of occurrences of the number c on the prefix of length i . This can be done with easy dynamic programming (just compute prefix sums). Also let $sum(c, l, r)$ be $pref_c[r + 1] - pref_c[l]$ and it's meaning is the number of occurrences of the number c on the segment $[l; r]$ (0-indexed).

Firstly, let's update the answer with $\max_{c=0}^{25} pref_c[n]$ (we can always take all occurrences of the same element as the answer). Then let's iterate over all possible segments of the

array. Let the current segment be $[l; r]$. Consider that all occurrences of the element in the middle block belong to a_l, a_{l+1}, \dots, a_r . Then we can just take the most frequent number on this segment ($cntin = \max_{c=0}^{25} sum(c, l, r)$). We also have to choose the number for the first and the last blocks. It is obvious that for the number num the maximum amount of such numbers we can take is $\min(sum(num, 0, l-1), sum(num, r+1, n-1))$. So $cntout = \max_{c=0}^{25} \min(sum(c, 0, l-1), sum(c, r+1, n-1))$. And we can update the answer with $2cntout + cntin$.

Time complexity: $O(n^2 \cdot AL)$, where AL is the size of alphabet (the maximum value of a_i).

Solution

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

using namespace std;

#define sz(a) int((a).size())
#define forn(i, n) for (int i = 0; i < int(n); ++i)
#define fore(i, l, r) for (int i = int(l); i < int(r); ++i)

int main() {
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
    //    freopen("output.txt", "w", stdout);
#endif

    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> a(n);
        for (auto &it : a) cin >> it;
        vector<vector<int>> cnt(26, vector<int>(n + 1));
        forn(i, n) {
            forn(j, 26) cnt[j][i + 1] = cnt[j][i];
            ++cnt[a[i] - 1][i + 1];
        }
        int ans = 0;
        forn(i, 26) ans = max(ans, cnt[i][n - 1]);
        forn(l, n) fore(r, l, n) {
            int cntin = 0, cntout = 0;
            forn(el, 26) {
                cntin = max(cntin, cnt[el][r + 1] - cnt[el][l]);
                cntout = max(cntout, min(cnt[el][l], cnt[el][n] - cnt[el][r + 1]) * 2);
            }
            ans = max(ans, cntin + cntout);
        }
        cout << ans << endl;
    }

    return 0;
}
```

1335E2 - Three Blocks Palindrome (hard version)

Idea: MikeMirzayanov

Tutorial

1335E2 - Three Blocks Palindrome (hard version)

I'll take some definitions from the solution of the easy version, so you can read it first if you don't understand something. Let $sum(c, l, r)$ be the number of occurrences of c on the segment $[l; r]$.

We will try to do almost the same solution as in the easy version. The only difference is how do we iterate over all segments corresponding to the second block of the required palindrome. Consider some number which we want to use as the number for the first and third blocks. If we take k occurrences in the first block then we also take k occurrences in the third block. Let's take these occurrences greedily! If we take k elements in the first block (and also in the third block) then it is obviously better to take k leftmost and k rightmost elements correspondingly.

Define $pos_c[i]$ be the position of the i -th occurrence of the number c (0-indexed). So, if pos_c is the array of length $sum(c, 0, n - 1)$ and contains all occurrences of c in order from left to right then let's iterate over its left half and fix the amount of numbers c we will take in the first block (and also in the third block). Let it be k . Then the left border of the segment for the second block is $l = pos_c[k - 1] + 1$ and the right border is $r = pos_c[sum(c, 0, n - 1) - k] - 1$. So let $cntin = \max_{c=0}^{199} sum(c, l, r)$ and $cntout = k$ and we can update the answer with $2cntout + cntin$.

It is easy to see that the total number of segments we consider is $O(n)$ so the total time complexity is $O(n \cdot AL)$ (there are also solutions not depending on the size of the alphabet, at least Mo's algorithm in $O(n\sqrt{n})$, but all of them are pretty hard to implement so I won't describe them here).

And I'm also interested in $O(n \log n)$ solution, so if you know it, share it with us!

Solution

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

using namespace std;

#define sz(a) int((a).size())
#define forn(i, n) for (int i = 0; i < int(n); ++i)

int main() {
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
    // freopen("output.txt", "w", stdout);
#endif

    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> a(n);
        for (auto &it : a) cin >> it;
        vector<vector<int>> cnt(200, vector<int>(n + 1));
        vector<vector<int>> pos(200);
        forn(i, n) {
            forn(j, 200) cnt[j][i + 1] = cnt[j][i];
            ++cnt[a[i] - 1][i + 1];
            pos[a[i] - 1].push_back(i);
        }
        int ans = 0;
        forn(i, 200) {
            ans = max(ans, sz(pos[i]));
            forn(p, sz(pos[i]) / 2) {
                int l = pos[i][p] + 1, r = pos[i]
[sz(pos[i]) - p - 1] - 1;

                forn(e1, 200) {
```

```

        cnt[e1][1];

        sum);

        }

    }

    cout << ans << endl;

}

return 0;

}

```

1335F - Robots on a Grid

Idea: **MikeMirzayanov**Tutorial

1335F - Robots on a Grid

First of all, I want to say about $O(nm)$ solution. You can extract cycles in the graph, do some dynamic programming on trees, use some hard formulas and so on, but it is a way harder to implement than the other solution that only has an additional log, so I'll describe the one which is easier to understand and much easier to implement.

Firstly, consider the problem from the other side. What is this grid? It is a functional graph (such a directed graph that each its vertex has exactly one outgoing edge). This graph seems like a set of cycles and ordered trees going into these cycles. How can it help us? Let's notice that if two robots meet after some move, then they'll go together infinitely from this moment. It means that if we try to make at least nm moves from each possible cell, we will obtain some "equivalence classes" (it means that if endpoints of two cells coincide after nm moves then you cannot place robots in both cells at once).

So, if we could calculate such endpoints, then we can take for each possible endpoint the robot starting from the black cell (if such exists) and otherwise the robot starting from the white cell (if such exists).

How can we calculate such endpoints? Let's number all cells from 0 to $nm - 1$, where the number of the cell (i, j) is $i \cdot m + j$. Let the next cell after (i, j) is (ni, nj) (i.e. if you make a move from (i, j) , you go to (ni, nj)). Also, let's create the two-dimensional array nxt , where $nxt[v][deg]$ means the number of the cell in which you will be if you start in the cell number v and make 2^{deg} moves. What is the upper bound of deg ? It is exactly $\log nm = \lceil \log_2(nm + 1) \rceil$.

Well, we need to calculate this matrix somehow. It is obvious that if the number of the cell (i, j) is id and the number of the next cell (ni, nj) is nid then $nxt[id][0] = nid$. Then let's iterate over all degrees deg from 1 to $\log nm - 1$ and for each vertex v set $nxt[v][deg] = nxt[nxt[v][deg - 1]][deg - 1]$. The logic behind this expression is very simple: if we want to jump 2^{deg} times from v and we have all values for 2^{deg-1} calculated then let's just jump 2^{deg-1} times from v and 2^{deg-1} times from the obtained vertex. This technique is called binary lifting.

Now we can jump from every cell nm times in $O(\log nm)$ time: just iterate over all degrees deg from 0 to $\log nm - 1$ and if nm has the deg -th bit on, just jump from the current vertex 2^{deg} times (set $v := nxt[v][deg]$). The rest of solution is described above.

Time complexity: $O(nm \log_2 nm)$.

Solution

```

#include <bits/stdc++.h>

using namespace std;

int n, m, lognm;
vector<string> c, s;
vector<vector<int>> used, nxt;

```

```

void getnext(int x, int y, int &nx, int &ny) {
    if (s[x][y] == 'U') nx = x - 1, ny = y;
    if (s[x][y] == 'R') nx = x, ny = y + 1;
    if (s[x][y] == 'D') nx = x + 1, ny = y;
    if (s[x][y] == 'L') nx = x, ny = y - 1;
}

void dfs(int x, int y) {
    used[x][y] = 1;
    int nx = -1, ny = -1;
    getnext(x, y, nx, ny);
    assert(0 <= nx && nx < n && 0 <= ny && ny < m);
    int v = x * m + y, to = nx * m + ny;
    if (!used[nx][ny]) dfs(nx, ny);
    nxt[v][0] = to;
}

int main() {
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
    //    freopen("output.txt", "w", stdout);
#endif

    int t;
    cin >> t;
    while (t--) {
        cin >> n >> m;
        lognm = 0;
        int nm = n * m;
        while ((1 << lognm) <= nm) ++lognm;
        c = s = vector<string>(n);
        for (auto &it : c) cin >> it;
        for (auto &it : s) cin >> it;

        used = vector<vector<int>>(n, vector<int>(m));
        nxt = vector<vector<int>>(n * m, vector<int>(lognm));
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                if (!used[i][j]) dfs(i, j);
            }
        }
        for (int deg = 1; deg < lognm; ++deg) {
            for (int i = 0; i < n; ++i) {
                for (int j = 0; j < m; ++j) {
                    int id = i * m + j;
                    nxt[id][deg] = nxt[nxt[id][deg - 1]][deg - 1];
                }
            }
        }

        vector<vector<int>> black, white;
        black = white = vector<vector<int>>(n, vector<int>(m));
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                int v = i * m + j, to = v;
                for (int deg = lognm - 1; deg >= 0; --deg) {
                    if ((nm >> deg) & 1) to =
nxt[to][deg];
                }
                if (c[i][j] == '0') ++black[to / m][to % m];
                else ++white[to / m][to % m];
            }
        }
    }
}

```

```

    }

    int all = 0, good = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (black[i][j]) {
                ++all;
                ++good;
            } else if (white[i][j]) {
                ++all;
            }
        }
    }
    cout << all << " " << good << endl;
}

return 0;
}

```

Solution (actually overrated, fastest $O(nm \log nm)$)

```

#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <queue>
#include <algorithm>
#include <string>
#include <cmath>
#include <cstdio>
#include <iomanip>
#include <fstream>
#include <cassert>
#include <cstring>
#include <unordered_set>
#include <unordered_map>
#include <numeric>
#include <ctime>
#include <bitset>
#include <complex>
#include <chrono>
#include <random>
#include <functional>

using namespace std;

const int N = 1e6 + 7;
const int K = 24;

int f[N];
int dp[N];
int ndp[N];
int sel[N];

int fans[N];
int sans[N];

void solve() {
    int n, m;
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        string s;
        cin >> s;
        for (int j = 0; j < m; j++) {
            sel[i * m + j] = (s[j] == '0');
        }
    }
}

```



```


    }
}
for (int i = 0; i < n; i++) {
    string s;
    cin >> s;
    for (int j = 0; j < m; j++) {
        int to = -1;
        if (s[j] == 'U') to = (i - 1) * m + j;
        if (s[j] == 'D') to = (i + 1) * m + j;
        if (s[j] == 'L') to = i * m + (j - 1);
        if (s[j] == 'R') to = i * m + (j + 1);
        f[i * m + j] = to;
    }
}
n *= m;
for (int i = 0; i < n; i++) {
    dp[i] = f[i];
}
for (int j = 0; j < K; j++) {
    for (int i = 0; i < n; i++) {
        ndp[i] = dp[dp[i]];
    }
    for (int i = 0; i < n; i++) {
        dp[i] = ndp[i];
    }
}
fill(fans, fans + n, 0);
fill(sans, sans + n, 0);
for (int i = 0; i < n; i++) {
    fans[dp[i]]++;
    sans[dp[i]] += sel[i];
}
int fs = 0, ss = 0;
for (int i = 0; i < n; i++) {
    fs += (fans[i] > 0);
    ss += (sans[i] > 0);
}
cout << fs << ' ' << ss << '\n';
}

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

    int q;
    cin >> q;
    while (q--) {
        solve();
    }
}


```

 Tutorial of Codeforces Round #634 (Div. 3)

 codeforces, 634, third division, editorial

 +48  

 [vovuh](#)

 11 hours ago

 [101](#)



Comments (101)

[Write comment?](#)



Sadat999

11 hours ago, # | ☆

That anti-sudoku approach is the coolest approach ever possible!!

→ [Reply](#)

 +19 