chemthan's blog

Codeforces Round #604 Editorial

By chemthan, history, 22 hours ago,

Sorry for delay, we are working on remaining problem. Below are draft editorials.

1265A - Beautiful String

If string s initially contains 2 equal consecutive letters ("aa", "bb" or "cc") then the answer is obviously -1.

Otherwise, it is always possible to replacing all characters '?' to make s beautiful. We will replacing one '?' at a time and in any order (from left to right for example). For each '?', since it is adjacent to at most 2 other characters and we have 3 options ('a', 'b' and 'c') for this '?', there always exists at least one option which differ from 2 characters that are adjacent with this '?'. Simply find one and replace '?' by it.

Time comlexity: $\mathcal{O}(n)$ where n is length of s.

Python Solution

```
T = int(input())
for tc in range(T):
    s = [c for c in input()]
    n = len(s)
    i = 0
    while (i < n):
        if (s[i] == '?'):
            prv = 'd' if i == 0 else s[i - 1]
            nxt = 'e' if i + 1 >= n else s[i + 1]
            for x in ['a', 'b', 'c']:
                if (x != prv) and (x != nxt):
                    s[i] = x
                    break
        else:
            i += 1

    ok = True
    for i in range(n - 1):
        if (s[i] == s[i + 1]):
            print("-1")
            ok = False
            break
    if (ok == True):
        print("".join(s))
```

Author: isaf27, prepare laoriu, I_love_Hoang_Yen

1265B - Beautiful Numbers

A number m is beautiful if and only if all numbers in range $[1, m]$ occupies m consecutive positions in the given sequence p. This is equivalent to $pos_{max} - pos_{min} + 1 = m$ where pos_{max}, pos_{min} are the largest, smallest position of $1, 2, \ldots, m$ in sequence p respectively.

We will consider m in increasing order, that its $m = 1, 2, \ldots, n$. For each m we will find a way to update pos_{max}, pos_{min} so we can tell either m is a beautiful number or not in constant time. Denote pos_i is the position of number i in the sequence p. When $m = 1$, we have $pos_{max} = pos_{min} = pos_1$. When $m > 1$, the value of pos_{max}, pos_{min} can be updated by the following formula:

- $new\_pos_{max} = max(old\_pos_{max}, pos_m)$
- $new\_pos_{min} = max(old\_pos_{min}, pos_m)$

So we can update them in constant time. The correctness of above formula is based on the definition of pos_{max}, pos_{min}.
Total time comlexity: $\mathcal{O}(n)$

C++ solution

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

using namespace std;

const int M = 2e5 + 239;

int n, p[M], x;

void solve()
{
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cin >> x;
        p[x - 1] = i;
    }
    int l = n;
    int r = 0;
    string ans = "";
    for (int i = 0; i < n; i++)
    {
        l = min(l, p[i]);
        r = max(r, p[i]);
        if (r - l == i)
            ans += '1';
        else
            ans += '0';
    }
    cout << ans << "\n";
}

int main()
{
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int t;
    cin >> t;
    while (t--)
        solve();
    return 0;
}
```

Author: **isaf27**, prepare **isaf27**

1265C - Beautiful Regional Contest

To solve this problem, we have to make the following observations:

- All participants who solved the same number of problems must be either not awarded at all or all are awarded a same type of medal.
- All $g + s + b$ awarded participants are the first $g + s + b$ participants.

They can be proved based on the rules that the number of problems solved by a gold medalist > ones solved by a silver medalist > ones solved by a bronze medalist > ones solved by un-awarded participants.
Suppose we have an optimal solution with g gold medals, s silver medals and b bronze medals where $g + s + b$ is maximized. We can make the followings changes that resulted in another valid (and still optimal) solution:

- Only keep gold medalist who solved most problem. For others, we change their medal types from gold to silver. After this change, $g + s + b$ is unchanged and other rules are still satisfied.

→ Recent actions
ExposingACPCthieves → ACPC committee (The Greatest thieves to ever exist in the competitive programming world)
MikeMirzayanov → Rule about third-party code is changing
snapdragon3101 → CP Editor: An Editor for Competitive Programming
tom → How to run successful algorithm class in high school?
chemthan → Codeforces Round #604 Editorial
Rose_max → Something about third-party code in Codeforces Round#604 Div1
--Someone-- → How hard is it to become red in CF
chemthan → Codeforces Round #604 (div1 & div2)
jonathanirvings → Live online mirror of The 2019 ICPC Asia Jakarta Regional Contest
trouvaille → Use of "~" operator
ojuz → A blog post for people who want to connect their account with oj.uz
Gaunt → Some Heap Problems
I_love_Saundarya → How to minimize the sum of all nodes in a tree?
ATSTNG → [Tutorial] Matroid intersection in simple words
ijxjdjd → Incorrect Recursive Behavior in Java
-B1nary- → Tips on reading input from UVA problems.
jtnydv25 → Invitation to ICPC Asia-Kharagpur Onsite Mirror Contest 2019
PikMike → Educational Codeforces Round 75 Editorial
revivedDevil → Teams going to Kharagpur Regionals
snapdragon3101 → CP Editor 2.0.3 : Everything you wanted is here
dreamr → USACO
Spsk1313 → New feature demand
jiry_2 → A simple introduction to "Segment tree beats"
LightRay → Let's share problems with most unexpected solutions
Enigma27 → Manthan, Codefest'19 Editorial
Detailed →

- Similarly, only keep a minimized number of silver medalist who solved most problems among all silver medalist and the number of kept silver must strictly larger than the number gold medals. For others, we change their medal types from silver to bronze. After this changed, $g + s + b$ is unchanged and other rules are still satisfied.

Therefore, there exists an optimal solution where: g is minimized then s is minimized next. We can determine them greedily. When g and s are fixed, b should be maximized such that all rules are still satisfied.
Time complexity: $\mathcal{O}(n)$

C++ solution

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

using namespace std;

#define forn(i, n) for (int i = 0; i < int(n); i++)

int main() {
    int t;
    cin >> t;
    forn(tt, t) {
        int n;
        cin >> n;
        map<int,int> c;
        forn(i, n) {
            int pi;
            cin >> pi;
            c[-pi]++;
        }
        vector<int> pp;
        for (auto p: c)
            pp.push_back(p.second);
        bool ok = false;
        int g = pp[0];
        int i = 1;
        int s = 0;
        while (s <= g && i < pp.size())
            s += pp[i++];
        if (g < s) {
           int b = 0;
           while (b <= g && i < pp.size())
               b += pp[i++];
           while (i < pp.size() && g + s + b + pp[i] <= n / 2)
               b += pp[i++];
           if (g < b && g + s + b <= n / 2) {
               ok = true;
               cout << g << " " << s << " " << b << endl;
           }
        }
        if (!ok)
            cout << 0 << " " << 0 << " " << 0 << endl;
    }
}
```

Author: **MikeMirzayanov**, prepare **MikeMirzayanov**

1265D - Beautiful Sequence

Firstly, let's arrange even numbers. It is optimal to arrange those numbers as $0, 0, 0, \dots, 0, 2, 2, \dots 2$. Because we can place number 1 anywhere while number 3 only between two numbers 2 or at the end beside one number 2. So we need to maximize the number of positions where we can place number 3. The above gives us an optimal way. The next step is to place remaining numbers $1, 3$. Insert them inside positions first then ends later.

Base on above argument, we can do as following way that eliminates corner case issues:

Starting from the smallest or second smallest number. At any moment, if x is the last element and there is number $x - 1$ remains, we add $x - 1$ otherwise we add $x + 1$ or stop if there is no

$x+1$ left. If we manage to use all numbers then we have a beautiful sequence and answer is 'YES'.

C++ Solution

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

int main() {
    map<int, int> hs;
    cin >> hs[0] >> hs[1] >> hs[2] >> hs[3];
    int total = hs[0] + hs[1] + hs[2] + hs[3];
    for (int st = 0; st < 4; st++) if (hs[st]) {
        vector<int> res;
        auto ths = hs;
        ths[st]--;
        res.push_back(st);
        int last = st;
        for (int i = 0; i < total - 1; i++) {
            if (ths[last - 1]) {
                ths[last - 1]--;
                res.push_back(last - 1);
                last--;
            }
            else if (ths[last + 1]) {
                ths[last + 1]--;
                res.push_back(last + 1);
                last++;
            }
            else {
                break;
            }
        }
        if ((int) res.size() == total) {
            cout << "YES\n";
            for (int i = 0; i < (int) res.size(); i++) {
                cout << res[i] << " \n"[i == (int) res.size() - 1];
            }
            return 0;
        }
    }
    cout << "NO\n";
    return 0;
}
```

Author: **laoriu**, prepare **laoriu**, **I_love_Hoang_Yen**

1265E - Beautiful Mirrors

Let p_i be the probability that the i-th mirror will answer YES when it is asked. That is, this p_i equals to p_i in the problem statement divide by 100.

Let e_i be the expected number of days until Creatnx becomes happy when initially he is at the i-th mirror. For convenient, let $e_{n+1}=0$ (because when Creatnx is at $(n+1)$-th mirror he is happy already). The answer of the problem will be e_1.

By the definition of expectation value and its basic properties, the following must holds for all $1 \le i \le n$:

$$e_i = 1 + p_i \times e_{i+1} + (1-p_i) \times e_1$$

Let explain this equation for those who are not familiar with probability. Expectation value is just average of all possible outcomes. The first number 1 in the right hand side means Creatnx spends 1 day to ask the i-th mirror. With probability of p_i the i-th mirror will answer YES and Creatnx will move to the $(i+1)$-th mirror in the next day. At the $(i+1)$-th mirror, Creatnx on average needs to spend e_{i+1} days more to become happy. The second term $p_i \times e_{i+1}$ explains this case. Similarly, the third term $(1-p_i) \times e_1$ represents the case where the i-th mirror answers NO.

To find e_1 we need to solve n equations:

(1) $e_1 = 1 + p_1 \times e_2 + (1-p_1) \times e_1$

(2) $e_2 = 1 + p_2 \times e_3 + (1 - p_2) \times e_1$

$\ldots$

(n) $e_n = 1 + p_n \times e_{n+1} + (1 - p_n) \times e_1$

We can solve this system of equations by using substitution - a common technique. From equation (1) we have $e_2 = e_1 - \frac{1}{p_1}$. Substituting this in (2) we obtained $e_3 = e_1 - \frac{1}{p_1 \times p_2} - \frac{1}{p_2}$. See the pattern now? Similarly by substituting to that last equation we have:

$$0 = e_{n+1} = e_1 - \frac{1}{p_1 \times p_2 \times \ldots \times p_n} - \frac{1}{p_2 \times p_3 \times \ldots \times p_n} - \ldots - \frac{1}{p_n}$$

$$e_1 = \frac{1}{p_1 \times p_2 \times \ldots \times p_n} + \frac{1}{p_2 \times p_3 \times \ldots \times p_n} + \ldots + \frac{1}{p_n}$$

$$e_1 = \frac{1 + p_1 + p_1 \times p_2 + \ldots + p_1 \times p_2 \times \ldots \times p_{n-1}}{p_1 \times p_2 \times \ldots \times p_n}$$

We can compute e_1 according to the above formula in $\mathcal{O}(n)$.

C++ Solution

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

const int MOD = 119 << 23 | 1;

int inv(int a) {
    int r = 1, t = a, k = MOD - 2;
    while (k) {
        if (k & 1) r = (long long) r * t % MOD;
        t = (long long) t * t % MOD;
        k >>= 1;
    }
    return r;
}

int main() {
    int n; cin >> n;
    vector<int> p(n);
    for (int i = 0; i < n; i++) cin >> p[i], p[i] = (long long) p[i] *
inv(100) % MOD;
    int a = 1, b = 0;
    for (int i = 0; i < n; i++) {
        a = (long long) a * inv(p[i]) % MOD;
        b = (long long) b * inv(p[i]) % MOD;
        a = (a + (long long) (p[i] - 1) * inv(p[i])) % MOD;
        b = (b - inv(p[i]) + MOD) % MOD;
    }
    cout <<  (long long) (MOD - b) * inv(a) % MOD << "\n";
    return 0;
}
```

Author: **chemthan**, prepare **laoriu**, **I_love_Hoang_Yen**

1265F - Beautiful Bracket Sequence (easy version)

We calculate the depth of a sequence as follow:

- Let two pointers at each end of the sequence.
- If character at the left pointer is ')', we move the left pointer one position to the right.
- If character at the right pointer is '(', we move the right pointer one position to the left.
- If the character at the left pointer is '(' and the right pointer is ')' we increase our result and move the left pointer to the right and the right one to the left each with one position.
- We repeat while the left one is at the left of the right one.

After that, we can use dynamic programming to solve the problem. Let $dp[l][r]$ be total number of depth of all sequences induced by subsequence $a_l a_{l+1} \ldots a_r$. We have following transitives:

- $dp[l][r] += dp[l+1][r]$ if $a_l \neq$ '('.
- $dp[l][r] += dp[l][r-1]$ if $a_r \neq$ ')'.

- $dp[l][r] -= dp[l+1][r-1]$ if $a_l \neq$ '(' and $a_r \neq$ ')'.
- $dp[l][r] += dp[l+1][r-1] + 2^k$ if $a_l \neq$ ')' and $a_r \neq$ '(', where k is the number of '?' character in $s_{l+1} s_{l+2} \ldots s_{r-1}$.

C++ Solution

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

const int MOD = 119 << 23 | 1;

int fpow(int a, int k) {
    int r = 1, t = a;
    while (k) {
        if (k & 1) r = (long long) r * t % MOD;
        t = (long long) t * t % MOD;
        k >>= 1;
    }
    return r;
}

int main() {
    string s; cin >> s;
    int n = s.size();
    vector<vector<int>> dp(n, vector<int>(n));
    vector<int> f(n + 1);
    for (int i = 0; i < n; i++) {
        f[i + 1] = f[i];
        f[i + 1] += s[i] == '?';
    }
    for (int len = 2; len <= n; len++) {
        for (int i = 0; i < n - len + 1; i++) {
            int j = i + len - 1;
            if (s[i] != '(') {
                dp[i][j] += dp[i + 1][j];
                dp[i][j] %= MOD;
            }
            if (s[j] != ')') {
                dp[i][j] += dp[i][j - 1];
                dp[i][j] %= MOD;
            }
            if (s[i] != '(' && s[j] != ')') {
                dp[i][j] -= dp[i + 1][j - 1];
                dp[i][j] += MOD;
                dp[i][j] %= MOD;
            }
            if (s[i] != ')' && s[j] != '(') {
                dp[i][j] += dp[i + 1][j - 1];
                dp[i][j] %= MOD;
                dp[i][j] += fpow(2, f[j] - f[i + 1]);
                dp[i][j] %= MOD;
            }
        }
    }
    cout << dp[0][n - 1] << "\n";
    return 0;
}
```

Author: **chemthan**, prepare **laoriu**, **I_love_Hoang_Yen**

1264C - Beautiful Mirrors with queries

Assuming that currently there are k checkpoints $1 = x_1 < x_2 < \ldots < x_k \leq n$, the journey becoming happy of Creatnx can be divided to k stages where in i-th stage Creatnx "moving" from mirror x_i to mirror at position $x_{i+1} - 1$. Denote the i-th stage as $(x_i, x_{i+1} - 1)$. These k stages are independent so the sum of expected number of days Creatnx spending in each stage will be the answer to this problem.

When a new checkpoint b appear between 2 old checkpoints a and c, stage $(a, c-1)$ will be removed from the set of stages and 2 new stages will be added, they are $(a, b-1)$ and $(b, c-1)$. Similarly, when a checkpoint b between 2 checkpoints a and c is no longer a checkpoint, 2 stages $(a, b-1)$ and $(b, c-1)$ will be removed from the set of stages and new stage $(a, c-1)$ will be added. These removed/added stages can be fastly retrieved by storing all checkpoints in an ordered data structure such as *set* in C++. For removed/added stages, we subtract/add its expected number of days from/to the current answer. We see that when a query occurs, the number of stages removed/added is small (just 3 in total). Therefore, if we can calculate the expected number of days for an arbitrary stage fast enough, we can answer any query in a reasonable time.

From the solution of problem Beautiful Mirror, we know that the expected number of days Creatnx spending in stage (u, v) is:

$$\frac{1+p_u+p_u \times p_{u+1}+\ldots+p_u \times p_{u+1} \times \ldots \times p_{v-1}}{p_u \times p_{u+1} \times \ldots \times p_v} = \frac{A}{B}$$

The denominator B can be computed by using a prefix product array — a common useful trick. We prepare an array s where $s_i = p_1 \times p_2 \times \ldots \times p_i$. After that, B can be obtained by using 1 division: $B = \frac{s_v}{s_{u-1}}$.

For numerator A, we also use the same trick. An array t will be prepare where $t_i = p_1 + p_1 \times p_2 + \ldots + p_1 \times p_2 \ldots \times p_i$. We have $t_{v-1} = t_{u-2} + p_1 \times p_2 \times \ldots \times p_{u-1} \times A$ so $A = \frac{t_{v-1}-t_{u-2}}{p_1 \times p_2 \times \ldots \times p_{u-1}} = \frac{t_{v-1}-t_{u-2}}{s_{u-1}}$.

C++ Solution

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

const int MOD = 119 << 23 | 1;

struct node_t {
    int prd;
    int val;
    node_t() {
        prd = val = 0;
    }
};

node_t operator + (node_t a, node_t b) {
    if (!a.prd) return b;
    if (!b.prd) return a;
    node_t c;
    c.prd = (long long) a.prd * b.prd % MOD;
    c.val = (long long) a.val * b.prd % MOD;
    c.val = (c.val + b.val) % MOD;
    return c;
}

int inv(int a) {
    int r = 1, t = a, k = MOD - 2;
    while (k) {
        if (k & 1) r = (long long) r * t % MOD;
        t = (long long) t * t % MOD;
        k >>= 1;
    }
    return r;
}

int main() {
    int n, q; cin >> n >> q;
    vector<int> p(n);
    for (int i = 0; i < n; i++) cin >> p[i], p[i] = (long long) p[i] *
inv(100) % MOD;
    vector<node_t> st(n << 2);
    auto upd = [&] (int p, node_t val) {
        for (st[p += n] = val; 1 < p; ) p >>= 1, st[p] = st[p << 1] + st[p <<
1 | 1];
    };
    auto query = [&] (int l, int r) {
        node_t lres, rres;
```

```
        for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
            if (l & 1) {
                lres = lres + st[l++];
            }
            if (r & 1) {
                rres = st[--r] + rres;
            }
        }
        return (lres + rres).val;
    };
    for (int i = 0; i < n; i++) {
        node_t c;
        c.val = c.prd = inv(p[i]);
        upd(i, c);
    }
    set<int> ss;
    ss.insert(0);
    int res = query(0, n - 1);
    for (int i = 0; i < q; i++) {
        string op; int u; cin >> u; u--;
        auto it = ss.lower_bound(u);
        if (it == ss.end() || *it != u) {
            it = ss.insert(u).first;
            int lo = *(--it);
            it++;
            int hi = n;
            if (++it != ss.end()) {
                hi = *it;
            }
            res -= query(lo, hi - 1);
            res += MOD;
            res %= MOD;
            res += query(lo, u - 1);
            res %= MOD;
            res += query(u, hi - 1);
            res %= MOD;
        }
        else {
            int lo = *(--it);
            it++;
            int hi = n;
            if (++it != ss.end()) {
                hi = *it;
            }
            res += query(lo, hi - 1);
            res %= MOD;
            res -= query(lo, u - 1);
            res += MOD;
            res %= MOD;
            res -= query(u, hi - 1);
            res += MOD;
            res %= MOD;
            ss.erase(u);
        }
        cout << res << "\n";
    }
    return 0;
}
```

Author: **chemthan**, prepare **laoriu**, **I_love_Hoang_Yen**

1264D2 - Beautiful Bracket Sequence (hard version)

By the way calculate the depth in easy version we can construct a maximal depth correct bracket S. At i-th position containing '(' or '?' we will count how many times it appears in S.

Let x be the number of '(' before the i-th position i, y be the number of ')' after the i-th position, c be the number of '?' before the i-th position and d be the number of '?' after the i-th position. The i-th position appears in S iff the number of '(' before the i-th position must less than the number of

') ' after the i-th position.

So we can derive a mathematics formula:

$$\sum_{a+i<b+j} C(c,i) \cdot C(d,j) = \sum_{a+i<b+j} C(c,i) \cdot C(d,d-j) = \sum_{i+j<b+d-c} C(c,i) \cdot C(d,j)$$

.

Expanding both hand sides of indentity $(x+1)^{c+d} = (x+1)^c \cdot (x+1)^d$, the above sum is simplified as: $\sum_{0 \le i < b+d-c} C(c+d,i)$. Notice that $c+d$ doesn't change much, it only is one of two possible values. So we can prepare and obtain $O(n)$ complexity.

C++ Solution

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

const int MOD = 119 << 23 | 1;

int inv(int a) {
    int r = 1, t = a, k = MOD - 2;
    while (k) {
        if (k & 1) r = (long long) r * t % MOD;
        t = (long long) t * t % MOD;
        k >>= 1;
    }
    return r;
}

int main() {
    string s; cin >> s;
    int k = s.size() + 1;
    int a = 0, b = 0, c = 0, d = 0;
    for (char e : s) {
        if (e == ')') {
            b++;
        }
        if (e == '?') {
            d++;
        }
    }
    map<int, vector<int>> dps;
    auto calc = [&] (int a, int b, int c, int d) {
        int x = b + d - a;
        if (x < 0) return 0;
        int k = c + d;
        if (k < x) x = k;
        if (dps.count(k)) return dps[k][x];
        auto& dp = dps[k];
        dp.resize(k + 1);
        int t = 0, w = 1;
        for (int i = 0; i <= k; i++) {
            t += w;
            t %= MOD;
            dp[i] = t;
            w = (long long) w * (c + d - i) % MOD;
            w = (long long) w * inv(i + 1) % MOD;
        }
        return dp[x];
    };
    int res = 0;
    for (int i = 0; i < (int) s.size(); i++) {
        char e = s[i];
        if (e == '(') {
            a++;
        }
        if (e == ')') {
            b--;
        }
        if (e == '?') {
```

```
            c++;
            d--;
        }
        if (e == '(') {
            res += calc(a, b, c, d);
            res %= MOD;
        }
        else if (e == '?') {
            a++, c--;
            res += calc(a, b, c, d);
            res %= MOD;
            a--, c++;
        }
    }
    cout << res << "\n";
    return 0;
}
```

Author: **chemthan**, prepare **laoriu**, **I_love_Hoang_Yen**

1264E - Красивая лига

Firstly, Let's calculate the number of non-beautiful triples given all result of matches. It is obivous that for each non-beautiful triple (A, B, C) there exactly is one team that wins over the others. So if a team A wins k other teams $B_1, B_2, \ldots, B_k$ then team A corresponds to $\frac{k \cdot (k-1)}{2}$ non-beautiful triple (A, B_i, B_j). If we define $(p_1, p_2, \ldots, p_n)$ as the number of wins of n teams. Then the number of non-beautiful triples will be $\frac{p_1 \cdot (p_1-1) + p_2 \cdot (p_2-1) + \ldots + p_n \cdot (p_n-1)}{2}$. Notice that $p_1 + p_2 + \ldots + p_n = \frac{n \cdot (n-1)}{2}$. So we only need to minimize the square sum: $p_1^2 + p_2^2 + \ldots + p_n^2$.

The remain can be solved easily by Mincost-Maxflow:

- Creating source S, sink T, each '`match node`' for each match haven't played yet, each '`team node`' for each team.
- Add an edge between source S and each '`match node`' with capacity 1 cost 0.
- Add an edge between each '`match node`' and each of two '`team nodes`' with capacity 1 cost 0.
- Assuming, after m matches played, i-th team wins q_i matches. We add $n - q_i - 1$ edges between the i-th'`team node`' and sink T with capacity 1 and costs $2 \cdot q_i + 1, 2 \cdot q_i + 3, \ldots, 2 \cdot n - 1$ The min cost after run Mincost-Maxflow plus $q_1^2 + q_2^2 + \ldots + q_n^2$ will give us the minimal square sum.

Base on Mincost-Maxflow idea, we can solve in more sophisticated way. At any moment, we will try to pick a team with minimal number of wins. Then we try to give it one more win by setting result of a match (that haven't used so far) and/or changing results of a path of matches to keep the number of win of others. If not pick next one and so on.

Complexity: $O(n^4)$.

C++ Solution

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

int main() {
    int n, m; cin >> n >> m;
    vector<int> d(n);
    vector<vector<int>> g(n, vector<int>(n));
    vector<vector<int>> f(n, vector<int>(n, 1));
    for (int i = 0; i < m; i++) {
        int u, v; cin >> u >> v; u--, v--;
        d[u]++;
        g[u][v] = 1, g[v][u] = 2;
        f[u][v] = f[v][u] = 0;
    }
    priority_queue<pair<int, int>> heap;
    for (int i = 0; i < n; i++) {
        if (d[i] < n - 1) {
```

```
            heap.push({-d[i], i});
        }
    }
    while (!heap.empty()) {
        int u = heap.top().second;
        heap.pop();
        queue<int> que;
        vector<int> vis(n), from(n);
        que.push(u), vis[u] = 1;
        int id = -1;
        while (!que.empty()) {
            int v = que.front(); que.pop();
            int found = 0;
            for (int w = 0; w < n; w++) if (w != v && !g[v][w]) {
                found = 1;
                break;
            }
            if (found) {
                id = v;
                break;
            }
            for (int w = 0; w < n; w++) if (!vis[w] && g[v][w] == 2 &&
f[v][w] == 1) {
                que.push(w), vis[w] = 1;
                from[w] = v;
            }
        }
        if (id == -1) {
            continue;
        }
        for (int v = 0; v < n; v++) if (v != id && !g[id][v]) {
            g[id][v] = 1, g[v][id] = 2;
            break;
        }
        while (id != u) {
            int nid = from[id];
            swap(g[id][nid], g[nid][id]);
            id = nid;
        }
        d[u]++;
        if (d[u] < n - 1) {
            heap.push({-d[u], u});
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                cout << 0;
            }
            else {
                cout << 2 - g[i][j];
            }
        }
        cout << "\n";
    }
    return 0;
}
```

Author: **chemthan**, prepare **laoriu**, **I_love_Hoang_Yen**

Another cool problem: WEICOM.

1264F - Beautiful Fibonacci Problem

Intuitively, we want to a formula for F_{m+n}. This is one we need:
$F_{m+n} = F_m \cdot F_{n+1} + F_{m-1} \cdot F_n$.

Denote $n = 12 \cdot 10^k$, one can verify that $F_{i \cdot n} \equiv 0$ modulo 10^k (directly calculate or use '`Pisano Period`').

So we have following properties:

- $F_{2\cdot n+1} = F_{n+1}^2 + F_n^2 \Rightarrow F_{2\cdot n+1} = F_{n+1}^2$ modulo $10^{2\cdot k}$.
- $F_{3\cdot n+1} = F_{2\cdot n+1} \cdot F_{n+1} + F_{2\cdot n} \cdot F_n \Rightarrow F_{3\cdot n+1} = F_{n+1}^3$ modulo $10^{2\cdot k}$.
- So on, $F_{u\cdot n+1} = F_{n+1}^u$ modulo $10^{2\cdot k}$.

Notice that $F_{n+1} = 8 \cdot 10^k \cdot t + 1$, where $gcd(t, 10) = 1$.

So $F_{u\cdot n+1} = (8 \cdot 10^k \cdot t + 1)^u = 8 \cdot u \cdot t \cdot 10^k + 1$ modulo $10^{2\cdot k}$.

Let $u = 125.a.t^{-1}$ modulo 10^k, $v = 125.d.t^{-1}$ modulo 10^k. Then we choose $b = u \cdot n + 1, e = v \cdot n$.

Python Solution

```
n,a,d=map(int,input().split())
print(368131125*a%10**9*12*10**9+1,368131125*d%10**9*12*10**9)
```

Author: **chemthan**, prepare **laoriu**, **I_love_Hoang_Yen**

Tutorial of Codeforces Round #604 (Div. 1)
Tutorial of Codeforces Round #604 (Div. 2)

+46 | chemthan | 22 hours ago | 20

Comments (20)

Write comment?

Nieb_Hasan_077

21 hour(s) ago, # | ← Rev. 2 **0**

Was eagerly waiting for the editorial. Thanks.

→ Reply

yuguotianqing

20 hours ago, # | **+46**

Problem 1264E has a similar which can be found at https://www.luogu.com.cn/problem/P4249, I have used the code in https://www.luogu.com.cn/problemnew/solution/P4249 to solve 1264E during the contest, which was published in 2018.

Thus some people's (such as **jiangly**, **Rose_max**, **KotonohaKatsura**, **disposrestfully**, **SpiderDance** and me) submissions was skipped by mistake, they are unrated. But it's not fair.

Can you help us to solve this problem? Greatly thanks.

The skipped submissions are jiangly/66346874, Rose_max/66347301, KotonohaKatsura/66350584, disposrestfully/66342618, yuguotianqing/66351221, SpiderDance/66352851.

→ Reply

chemthan

20 hours ago, # ^ | **0**

Let ask **MikeMirzayanov**.

→ Reply

yuguotianqing

20 hours ago, # ^ | **0**

Thanks.

→ Reply

Rose_max

18 hours ago, # ^ | **0**

please help us.. i am really anxious now

→ Reply

Mucosolvan

20 hours ago, # | ☆ 0

https://codeforces.com/contest/1264/submission/66453921 — Div 1C WA4

Looking at the comments my idea is similar, we first calculate answer with no checkpoints (which works, since I pass Div2E), and afterwards if we add a checkpoint j in interval [i, k], we have to decrease the answer by prod(i, j — 1) * prod(j, k) and add val[j — 1], when we delete j we need to merge intervals [i, j — 1] and [j, k], we do the opposite (add products, subtract val). This fails on first checkpoint addition on big test, so I cannot research that manually. Any ideas on what I did wrong?

→ Reply

paulica

18 hours ago, # ^ | ☆ +8

Your formula for updates is wrong.

Try this testcase:

```
3 1
100 100 50
3
```

The answer is 4, but your output is 5.

→ Reply

nikhilsarda

19 hours ago, # | ☆ 0

Can someone please explain,how to solve Beautiful sequence(1265D) as there was a lot of confusions while trying to get the proper solution.Also explain how did you get to the solution please. Thanks in advance.

→ Reply

saquib2508

4 hours ago, # ^ | ☆ +1

What I tried to do is this. I tried to make a pattern of $01010101|21212121|23232323$ form. The 01 sequence needs to terminate with a 1, the 23 sequence needs to start with 2. Now, if there are *too many* 0's than 1's, the sequence cannot be formed as you need 1's beside the 0's. So, if $a \geq b+2$ or $d \geq c+2$, answer is NO. The other cases are also similar. The code is pretty much self-explanatory, so I'd suggest you have a look at it. My submission: 66486337

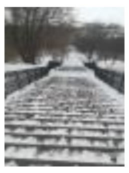
→ Reply

laoriu

18 hours ago, # | ☆ -8

Please be aware that the tutorial is not completed yet. There may be some typo or ambiguous sentences. We are fixing them gradually.

→ Reply



Teamforces

17 hours ago, # | ☆ 0

Better late than never

→ Reply

sorry_marymarine

17 hours ago, # | ☆ -15

1265E is a super standard bayan problem

→ Reply

DeD_TihoN

16 hours ago, # | ☆ ← Rev. 3 +4

Problem E div 2

By the definition of expectation value and its basic properties, the following must holds for all 1≤i≤n:

ei=1+pi×ei+1+(1−pi)×e1

I know the definition of expectation value, but this equality is not obvious for me.

Can somebody explain this formula or give some information about expectation value to learn?

→ Reply

12 hours ago, # ^ | +22

birka0

It is the linearity of expected value, i.e. if X and Y are two random variables, then $E[\alpha X + \beta Y] = \alpha E[X] + \beta E[Y]$ for any α, β constants. The variable X_i is the number of days to get happy if she's at the i-th mirror. For this random variable, we must have the relation $X_i = p_i \cdot (1 + X_{i+1}) + (1 - p_i) \cdot (1 + X_1) = 1 + p_i X_{i+1} + (1 - p_i) X_1$, since with probability p_i she can move on to the next mirror with a day gone, and with $1 - p_i$ she goes back to the first. Now, $E[X_i] = E[1 + p_i X_{i+1} + (1 - p_i) X_1] = 1 + p_i E[X_{i+1}] + (1 - p_i) E[X_1]$.

Or, without intuitive useage of random variables, you could say that the probability of X_i taking the value d (i.e. exactly d days to get happy), we must have $P(X_i = d) = p_i \cdot P(X_{i+1} = d - 1) + (1 - p_i) \cdot P(X_1 = d - 1)$, since after passing i-th with p_i probability she has to get happy in exactly $d - 1$ days from $i + 1$-st, or failing at the i-th gives her $d - 1$ days to pass from the first one. So we need only the definition of expected value to get $E[X_i] = \sum_{d=1}^{\infty} d \cdot P(X_i = d) = p_i \sum_{d=1}^{\infty} d \cdot P(X_{i+1} = d - 1) + (1 - p_i) \sum_{d=1}^{\infty} d \cdot P(X_1 = d - 1)$,

where

$$d \cdot P(X_{i+1} = d - 1) = (d - 1) \cdot P(X_{i+1} = d - 1) + P(X_{i+1} = d - 1)$$

means

$$E[X_i] = p_i \sum_{d=0}^{\infty} d \cdot P(X_{i+1} = d) + p_i \sum_{d=0}^{\infty} P(X_{i+1} = d) +$$
$$+ (1 - p_i) \sum_{d=0}^{\infty} d \cdot P(X_1 = d) + (1 - p_i) \sum_{d=0}^{\infty} P(X_1 = d)$$

Finally this leads to

$$E[X_i] = p_i E[X_{i+1}] + p_i \cdot 1 + (1 - p_i) E[X_1] + (1 - p_i) \cdot 1.$$

→ Reply

13 hours ago, # | +64

birka0

Here's my approach to 1264F - Beautiful Fibonacci Problem, hoping it provides some additional insight:

I was aware that the Fibonacci sequence is periodic for any modulus, and the period is called Pisano period. I checked that for $x \geq 3$, the Pisano period $\mod 10^x$ is $15 \cdot 10^{x-1}$. At first, I was considering an easier task, with limits at 10^3 instead of 10^6, i.e. I was looking for arithmetic sequences of indices such that the corresponding Fibonacci numbers contain all the 3-length digit substrings, hoping to discover some useful relation. I found out that over a period, the last three digits won't produce all substrings, so i was looking at the next block of 3 digits, for which i realized that if I jump the lengths of Pisano period $p = 1500$ in the index, then the numbers $F_{1+k \cdot p} / 1000 \mod 1000$ (the second 3-digit blocks from the end of the number) are in arithmetic progression $\mod 1000$, and since in this case $F_{1+1 \cdot p} / 1000 \mod 1000 = 949$ is relative prime to 1000, so it generates all the digit strings. I will provide a proof in the end of this post.

Now, the value of index k in order to have the string 001 in $F_{1+k \cdot p}$ is easy to get by Euler-theorem, it is $k = 949^{\varphi(1000)-1} \mod 1000 = 949^{399} \mod 1000 = 549$. So, if we need a digit string a to be found, we could just take $F_{1+a \cdot k \cdot p}$, since the strings are generated in arithmetic progression, as I mentioned before. Moreover for $a + l \cdot d$ we could choose $F_{1+(a+l \cdot d) \cdot k \cdot p} = F_{1+a \cdot k \cdot p + d \cdot l \cdot k \cdot p}$, i.e. obviously the indices here would be in arithmetic progression too, so we can simply output our answer $b = 1 + a \cdot k \cdot p$ and $e = d \cdot k \cdot p$.

The same holds if we have the problem's original limits and we need the 6-length substrings: consider the above with $\mod 10^6$, when the Pisano period is $p = 15 \cdot 10^5$, get the generator of the 6-length digit strings $F_{1+1 \cdot p} / 10^6 \mod 10^6 = 677499$, compute the index of 000001 which is $k = 677499^{399999} \mod 10^6 = 445049$, and so the answer is $b = 1 + a \cdot 445049 \cdot 15 \cdot 10^5$ and $e = d \cdot 445049 \cdot 15 \cdot 10^5$.

Link to submission: 66473963

The fact that the numbers $F_{1+k \cdot 15 \cdot 10^{x-1}} / 10^x \mod 10^x$ are in arithmetic progression $\mod 10^x$ can be verified for the given limits by a simple program, but we can prove it formally too: for this, we need the formula $F_{1+(k+1) \cdot p} = F_{1+kp} \cdot F_{1+p} + F_{kp} \cdot F_p$, which is a special case of the formula mentioned by the editorial, but nonetheless its easy to verify either using the generator function or the recurrence matrix of the sequence. Since p is the Pisano period, we know that both terms of the second product are congruent to $0 \mod 10^x$, i.e. they end with x zeroes, so the product ends with at least $2x$ zeroes, so it won't affect the second x-length digit block from the end, consequently $F_{1+(k+1) \cdot p} / 10^x \mod 10^x = (F_{1+kp} \cdot F_{1+p}) / 10^x \mod 10^x$. The last x digit block in both of these right hand side terms is $00 \ldots 01$ (by Pisano period), so the second x digit block from the end (notating by B_l the second x digit block of F_l for any l index), is obtained by multiplying blockwise $(A_{1+kp} \cdot 10^{2x} + B_{1+kp} \cdot 10^x + 00 \ldots 01) \cdot (A_{1+p} \cdot 10^{2x} + B_{1+p} \cdot 10^x + 00 \ldots 01)$

to get $X \cdot 10^{2x} + (B_{1+kp} + B_{1+p}) \cdot 10^x + 00 \ldots 01$, i.e.

$B_{1+(k+1)p} = B_{1+kp} + B_{1+p} \mod 10^x$, the second x-digit block is the sum of the second x-digit blocks of the $1 + kp$ and $1 + p$ Fibonacci numbers, thats what we wanted to show.

→ Reply

quickers

9 hours ago, # | 0

How to derive the mathematics formula for D2?

→ Reply

pritishn

7 hours ago, # | 0

Would this approach work for 1265E Beautiful Mirrors?

Since we need to calculate expected number of days..

So let Pi=Probablity of **becoming happy** at ith day

So that implies that P1,P2,....P(n-1) = 0 as he cant be happy on days he's not asking the last mirror,

So , he can be happy on Pn,P2n,..... days.

Now probability of having success on nth day is = p1*p2*p3*....*pn

And on (2n)th day is =(Pn)*(Pn)

And on (3n)th day is =(Pn)*(Pn)*(Pn)

And so on...

So now according to formula

E= n*(Pn) + (2n)*(P2n) + (3n)*(P3n) + ...

=> E = n*(Pn) + (2n)*(Pn)^2 + (3n)*(P3n)^3 + ...

So now the answer turns out to be an infinite Arithmetic-Geometric Progression series

and we should now be able to use the summation formula for it,

Would this approach work??

→ Reply

saquib2508

5 hours ago, # | 0

We can solve the Div2E problem WITHOUT having to find the modular inverse of all the P_i existing. Let, E_k is the expected number of days *after* having asked k mirrors i.e. about to ask the $k+1^{th}$ mirror. So, clearly from this definition, $E_n = 0$, and E_0 is our answer to this problem. Now, we can see that

$$E_k = 1 + \frac{P_{k+1}}{100} \cdot E_{k+1} + (1 - \frac{P_{k+1}}{100}) \cdot E_0$$

Let this be *eqn1*, which holds for all $k(0 \leq k \leq n-1)$. Let's express every $E_k(0 \leq k \leq n)$ in $constant + efficient \cdot E_0$ form. So, $E_n = 0 + 0 \cdot E_0$. Using that, we can find $E_{n-1}, E_{n-2} \ldots E_0$ by plugging in the values of $constant$ and $efficient$ in *eqn1* and updating them. So, at the end of the iteration, we will have $E_0 = constant + efficient \cdot E_0$. Thus, $E_0 = \frac{const}{1-eff} \% MOD$. Do your modular operations properly, be careful of overflow, and it will lead to the answer. All we need to do is to evaluate modular inverse of 100 at the beginning, and that of $(1 - eff)$ towards the end. My submission: 66363163

→ Reply

Spheniscine

9 minutes ago, # ^ | 0

This was my approach too. Unfortunately, I think it's harder to get the insight needed to solve the Div1 version from this approach.

→ Reply

Falcon__

3 hours ago, # | +4

Let us solve D by dp!!! you may say are you crazy?Be patient,my friend we can naively denote the status dp[i][e][a][b][c][d] is when we are at i position and the end is "e"(0,1,2,3) element we has a zeros ,b ones ,c twos ,d threes ,so the transform is obvious so, we have many status....TLE? NO,many status can just be neglected let us denote x,y is the element we want to put ,and denote nd:=min(cnt[x],cnt[y]) nd+1,nd is the number we can put ,other condition in fact will finally transfer into the status my code: Your text to link here...66490372

→ Reply