PikMike's blog

Educational Codeforces Round 49 Editorial

By **PikMike**, history, 23 hours ago,

1027A - Palindromic Twist

Tutorial

1027A - Palindromic Twist

If some string can't be transformed to palindrom then it has some pair of positions $(i, n - i + 1)$ with different letters on them (as no such pair affects any other pair). Thus you need to check each pair for i from 1 to $\frac{n}{2}$ and verify that the distance between the corresponding letters is either 0 or 2.

Overall complexity: $O(T \cdot n)$.

Solution (PikMike)

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

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

using namespace std;


int main() {
    int T;
    cin >> T;

    int n;
    string s;
    forn(_, T){
        cin >> n >> s;
        bool ok = true;
        forn(i, n){
            int k = abs(s[i] - s[n - i - 1]);
            if (k > 2 || k % 2 == 1){
                ok = false;
                break;
            }
        }
        cout << (ok ? "YES" : "NO") << endl;
    }
    return 0;
}
```

1027B - Numbers on the Chessboard

Tutorial

1027B - Numbers on the Chessboard

Let's see the following fact: if we will decrease $\lceil \frac{n^2}{2} \rceil$ from all numbers written in cells with an odd sum of coordinates and write out the numbers obtained on the board from left to right from top to bottom, the sequence will looks like $1, 1, 2, 2, \ldots, \lceil \frac{n^2}{2} \rceil, \lceil \frac{n^2}{2} \rceil$ for even n (for odd n there is only one number $\lceil \frac{n^2}{2} \rceil$ at the end of the sequence, but, in general, it does not matter).

Let's try to find out the answer for some query (x, y). Let $cnt = (x - 1) \cdot n + y$ (1-indexed). There cnt is the position of our cell in order of the written sequence. The first approximation of the answer is $\lceil \frac{cnt}{2} \rceil$. But now we are remember that we decreased $\lceil \frac{n^2}{2} \rceil$ from all numbers written in cells with an odd sum of coordinates. So if $x + y$ is even then the answer is $\lceil \frac{cnt}{2} \rceil$, otherwise the answer is $\lceil \frac{cnt}{2} \rceil + \lceil \frac{n^2}{2} \rceil$. Note that you should be careful with integer overflow in C++, Java or similar languages. 64-bit datatype is quite enough.

Time complexity: $O(q)$.

Solution (Vovuh)

```
import sys

lst = sys.stdin.readlines()
n, q = map(int, lst[0].split())

for i in range(q):
    x, y = map(int, lst[i + 1].split())
    cnt = (x - 1) * n + y
    ans = (cnt + 1) // 2
    if ((x + y) % 2 == 1): ans += (n * n + 1) // 2
    sys.stdout.write(str(ans) + '\n')
```

1027C - Minimum Value Rectangle

Tutorial

1027C - Minimum Value Rectangle

Let's work with the formula a bit:

$$\frac{P^2}{s} = \frac{4(x+y)^2}{xy} = 4(2 + \frac{x}{y} + \frac{y}{x}) = 8 + 4(\frac{x}{y} + \frac{y}{x})$$

Let $a = \frac{x}{y}$, then the formula becomes $8 + 4(a + \frac{1}{a})$. Considering $x \geq y$, $a = \frac{x}{y} \geq 1$, thus $(a + \frac{1}{a})$ is strictly increasing and has its minimum at $a = 1$.

So the solution will be to sort the list, extract the pairs of sticks of equal length and check only neighbouring pairs in sorted order for the answer.

Overall complexity: $O(n \log n)$.

Solution (PikMike)

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

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

typedef long long li;

using namespace std;

const int N = 1000 * 1000 + 13;

int n, m;
int a[N];
int b[N];
```

→ Recent actions

Radewoosh → Blogewoosh #3

GreenGrape → Codeforces Round #505 Editorial

gojira → Codeforces Round #196 — Problems Analysis

fjzzq2002 → Linear Recurrence and Berlekamp-Massey Algorithm

GreenGrape → Codeforces Round #505

abhayps → Maximum Independent Set

fatemehkarimi → how to solve this dynamic programming problem??

farmersrice → Disappearance of comments

himadree → Insertion Sort problem 362C: please explain the given sample outputs easily

300iq → Codeforces Round #504

tehqin → Algorithms Live - Episode 32

MikeMirzayanov → About 505

MikeMirzayanov → Rule about third-party code is changing

PikMike → Educational Codeforces Round 49 [Rated for Div. 2]

hadyko.hk → denial of judgement!

alex_ita → Voting problems

allllekssssa → Invitation for August Cook Off, 2018

lostbattle → Codechef cookoff division 1 que : "GukiZ and Candies"

scribblingcode → 1B — SPREADSHEETS

PikMike → Educational Codeforces Round 49 Editorial

LoneFox → Facebook Hacker Cup 2018: Round 2

hrOarr → Need help on Rotating Calipers technique

joisino → JOI Open Contest 2018

komendart → Codeforces Round #340 (Div. 2) Editorial

Meenachinmay → Can we do DFS in backward manner or in reverse direction?

Detailed →

```
int main() {
    int T;
    scanf("%d", &T);
    forn(_, T){
        scanf("%d", &n);
        forn(i, n)
            scanf("%d", &a[i]);
        sort(a, a + n);

        m = 0;
        forn(i, n - 1){
            if (a[i] == a[i + 1]){
                b[m++] = a[i];
                ++i;
            }
        }

        int A = b[0], B = b[1];
        li P2 = (A + B) * li(A + B);
        li S = A * li(B);

        forn(i, m - 1){
            li p2 = (b[i] + b[i + 1]) * li(b[i] + b[i + 1]);
            li s = b[i] * li(b[i + 1]);
            if (s * P2 > S * p2){
                A = b[i];
                B = b[i + 1];
                S = s;
                P2 = p2;
            }
        }

        printf("%d %d %d %d\n", A, A, B, B);
    }
}
```

1027D - Mouse Hunt

Tutorial

1027D - Mouse Hunt

Mouse jumps on a cycle at some point, no matter the starting vertex, thus it's always the most profitable to set traps on cycles. The structure of the graph implies that there are no intersecting cycles. Moreover, mouse will visit each vertex of the cycle, so it's enough to set exactly one trap on each cycle. The only thing left is to find the cheapest vertex of each cycle. This can be done by a simple dfs.

Overall complexity: $O(n)$.

Solution (adedalic)

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

using namespace std;

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

#define mp make_pair
#define pb push_back

#define sz(a) int((a).size())
#define all(a) (a).begin(), (a).end()
#define sqr(a) ((a) * (a))
```

```
#define x first
#define y second

typedef long long li;
typedef long double ld;
typedef pair<li, li> pt;

template<class A, class B> ostream& operator <<(ostream& out, const pair<A,
B> &p) {
    return out << "(" << p.x << ", " << p.y << ")";
}
template<class A> ostream& operator <<(ostream& out, const vector<A> &v) {
    out << "[";
    forn(i, sz(v)) {
        if(i) out << ", ";
        out << v[i];
    }
    return out << "]";
}

const int INF = int(1e9);
const li INF64 = li(1e18);
const ld EPS = 1e-9;

int n;
vector<int> a, c;

inline bool read() {
    if(!(cin >> n))
        return false;
    c.assign(n, 0);
    a.assign(n, 0);

    forn(i, n)
        assert(scanf("%d", &c[i]) == 1);
    forn(i, n) {
        assert(scanf("%d", &a[i]) == 1);
        a[i]--;
    }
    return true;
}

vector< vector<int> > cycles;

vector<char> used;
vector<int> path;

void dfs(int v) {
    path.push_back(v);
    used[v] = 1;

    int to = a[v];
    if(used[to] != 2) {
        if(used[to] == 1) {
            cycles.emplace_back();

            int id = sz(path) - 1;
            while(path[id] != to)
                cycles.back().push_back(path[id--]);
            cycles.back().push_back(to);
        } else
            dfs(to);
    }
```

```
    path.pop_back();
    used[v] = 2;
}

inline void solve() {
    used.assign(n, 0);

    forn(i, n) {
        if (!used[i])
            dfs(i);
    }

    li ans = 0;
    for(auto &cur : cycles) {
        int mn = c[cur[0]];
        for(int v : cur)
            mn = min(mn, c[v]);
        ans += mn;
    }
    cout << ans << endl;
}

int main() {
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
    int tt = clock();
#endif
    cout << fixed << setprecision(15);

    if(read()) {
        solve();

#ifdef _DEBUG
        cerr << "TIME = " << clock() - tt << endl;
        tt = clock();
#endif
    }
    return 0;
}
```

1027E - Inverse Coloring

Tutorial

1027E - Inverse Coloring

You can notice that every coloring can be encoded by the two binary strings of length n. You firstly generate one string to put as a first row and then use the second string to mark if you put the first string as it is or inverting each color.

That way, you can also guess that the area of maximum rectangle of a single color you will get in you coloring is the product of maximum lengths of segments of a single color in both of the strings.

Let's consider the following dynamic programming solution:

$dp[i][j][k]$ is the number of binary strings of length i, such the current last segment of a single color has length j and the maximum segment of a single color has length k.

The transitions are:

- $dp[i+1][j+1][max(k, j+1)]$ += $dp[i][j][k]$ — color the new tile the same as the previous one;
- $dp[i+1][1][max(k, 1)]$ += $dp[i][j][k]$ — color the new tile the opposite from the previous one.

The starting state is $dp[0][0][0] = 1$.

Let's sum the values of this dp to an array $cnt[i]$ — the number of binary strings of length n such that the maximum segment of a single color in them has length i. You can also do another dp to calculate this not in $O(n^3)$ but in $O(n^2)$ using some partial sums in it.

Finally, you iterate over the first side of the resulting rectangle (the maximum length of segment of a single color in a first binary string) and multiply the number of ways to get it by the total number of ways to get the second side of the resulting rectangle, so that the area doesn't $(k - 1)$.

Overall complexity: $O(n^3)$ or $O(n^2)$.

Solution (PikMike) O(n^3)

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

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

using namespace std;

const int N = 500 + 7;
const int MOD = 998244353;

int n, k;
int dp[2][N][N];
int cnt[N];
int pr[N];

void add(int &a, int b){
    a += b;
    if (a >= MOD)
        a -= MOD;
}

int main() {
    scanf("%d%d", &n, &k);

    dp[0][0][0] = 1;
    forn(ii, n){
        int i = ii & 1;
        int ni = i ^ 1;
        memset(dp[ni], 0, sizeof(dp[ni]));
        forn(j, n + 1){
            forn(k, n + 1){
                add(dp[ni][j + 1][max(j + 1, k)], dp[i][j][k]);
                add(dp[ni][1][max(1, k)], dp[i][j][k]);
            }
        }
    }

    forn(i, n + 1)
        forn(j, n + 1)
            add(cnt[i], dp[n & 1][j][i]);

    forn(i, n + 1){
        add(pr[i + 1], pr[i]);
        add(pr[i + 1], cnt[i]);
    }

    int ans = 0;
    for (int i = 1; i <= n; ++i)
        add(ans, cnt[i] * (long long)(pr[min(n + 1, (k - 1) / i + 1)]) %
MOD);

    ans = (ans * (long long)((MOD + 1) / 2)) % MOD;
    printf("%d\n", ans);
```

```
    return 0;
}
```

Solution (BledDest) O(n^2)

```
def norm(x):
    return (x % 998244353 + 998244353) % 998244353

n, k = map(int, input().split())

dp1 = [0]
dp2 = [0]

for i in range(n):
    l = [1]
    cur = 0
    for j in range(n + 1):
        cur += l[j]
        if(j > i):
            cur -= l[j - i - 1]
        cur = norm(cur)
        l.append(cur)
    dp1.append(l[n])
    dp2.append(norm(dp1[i + 1] - dp1[i]))

ans = 0
for i in range(n + 1):
    for j in range(n + 1):
        if(i * j < k):
            ans = norm(ans + dp2[i] * dp2[j])

ans = norm(ans * 2)

print(ans)
```

1027F - Session in BSU

Tutorial

1027F - Session in BSU

This problem has many approaches (as Hall's theorem, Kuhn algorithm (???) and so on), I will explain one (or two) of them.

Let's find the answer using binary search. It is obvious that if we can pass all the exams in k days we can also pass them in $k + 1$ days.

For the fixed last day k let's do the following thing: firstly, if there exists some exam with the day of the first opportunity to pass it greater than k, then the answer for the day k is false.

Next, while there exist exams having only one possibility to pass them (because of the upper bound of the maximum possible day or constraints imposed by the other exams), we choose this day for this exam and continue (after choosing such day there can appear some new exams with the same property). Now there are no exams having only one day to pass them.

Let's take a look on the graph where vertices represent days and edges represent exams (the edge between some vertices u and v ($u < v$) exists iff there is an exam with the first day to pass it equal to u and the second day to pass it equal to v). Let's remove all the exams for which we identified the answer. Now let's take a look on the connected components of this graph and analyze the problem which we have now. Our problem is to choose exactly one vertex incident to each edge of the connected component such that no vertex is chosen twice (and we have to do this for all the connected components we have).

Let $cntv$ be the number of vertices in the current connected component and $cnte$ be the number of edges in the current connected component. The answer for the connected component is true iff $cnte \le cntv$, for obvious reasons. There is very easy constructive method to see how we can do this. If $cnte < cntv$ then the current connected component is a tree. Let's remove some leaf of this tree and set it as the chosen vertex for the edge incident to this leaf (and remove this edge too). If $cnte = cntv$ then let's remove all leaves as in the algorithm for the tree. For the remaining cycle let's choose any edge and any vertex incident to it, set this vertex as the chosen to this edge and remove them. Now we have a chain. Chain is a tree, so let's apply the algorithm for the tree to this chain.

So, if for some connected component c holds $cnte_c > cntv_c$ then the answer for the day k is false. Otherwise the answer is true.

Overall complexity $O(n \log n)$ because of numbers compressing or using logarithmic data structures to maintain the graph.

Also there is another solution (which can be too slow, I don't know why it works). It is well-known fact that if we will apply Kuhn algorithm to the some bipartite graph in order of increasing indices of vertices of the left part, then the last vertex in the left part of this graph which is in the matching will be minimum possible. Oh, that's what we need! Let the left part of this graph consist of days and the right part consist of exams. The edge between some vertices u from the left part and v from the right part exists iff u is one of two days to pass the exam v. Let's apply Kuhn algorithm to this graph, considering days in increasing order. The first day when matching becomes n (all exams are in the matching) will be the answer. I don't know its complexity, really. Maybe it works too fast because of the special properties of the graph... If someone can explain in which time it works I will very happy!

Solution (Vovuh)

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

using namespace std;

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

const int N = 2 * 1000 * 1000 + 11;
const int INF = 1e9 + 1;

int n;
int a[N][2];
vector<int> s;
bool used[N];
bool exam[N];
vector<pair<int, int>> g[N];

bool deny(int v) {
    if (used[v]) return false;
    used[v] = true;
    for (auto it : g[v]) {
        int to = it.first;
        int ex = it.second;
        if (exam[ex]) continue;
        if (used[to])
            return false;
        exam[ex] = true;
        if (!deny(to))
            return false;
    }
    return true;
}

int cntv, cnte;

void dfs(int v) {
    ++cntv;
```

```
    used[v] = true;
    for (auto it : g[v]) {
        int to = it.first;
        int ex = it.second;
        if (exam[ex]) continue;
        exam[ex] = true;
        ++cnte;
        if (!used[to])
            dfs(to);
    }
}

bool check(int day) {
    memset(used, false, sizeof(used));
    memset(exam, false, sizeof(exam));
    forn(i, s.size()) if (i > day) {
        if (!deny(i)) {
            return false;
        }
    }

    forn(i, s.size()) {
        if (!used[i]) {
            cntv = 0;
            cnte = 0;
            dfs(i);
            if (cntv < cnte)
                return false;
        }
    }

    return true;
}

int main() {
    scanf("%d", &n);
    forn(i, n) forn(j, 2) {
        scanf("%d", &a[i][j]);
        s.push_back(a[i][j]);
    }

    sort(s.begin(), s.end());
    s.resize(unique(s.begin(), s.end()) - s.begin());

    forn(i, n) {
        forn(j, 2) {
            a[i][j] = lower_bound(s.begin(), s.end(), a[i][j]) - s.begin();
        }
        g[a[i][0]].push_back(make_pair(a[i][1], i));
        g[a[i][1]].push_back(make_pair(a[i][0], i));
    }

    int l = 0, r = int(s.size()) - 1;
    while (r - l > 1) {
        int mid = (l + r) >> 1;
        if (check(mid))
            r = mid;
        else
            l = mid;
    }
    for (int i = l; i <= r; ++i) {
        if (check(i)) {
            printf("%d\n", s[i]);
            return 0;
```

```
        }
    }
    puts("-1");
}
```

Solution (Vovuh) Kuhn's Algorithm

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

using namespace std;

mt19937 rnd(time(NULL));

const int N = 2 * 1000 * 1000 + 11;
const int INF = 1e9;

int n;
int a[N][2];

vector<int> nums;
int m;

vector<int> g[N];
int mt[N];
int used[N];
int T = 1;

bool try_kuhn(int v) {
    if (used[v] == T)
        return false;
    used[v] = T;

    for (auto to : g[v]) {
        if (mt[to] == -1) {
            mt[to] = v;
            return true;
        }
    }

    for (auto to : g[v]) {
        if (try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }

    return false;
}

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

    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < 2; ++j) {
            scanf("%d", &a[i][j]);
            nums.push_back(a[i][j]);
        }
    }

    sort(nums.begin(), nums.end());
```

```
    nums.resize(unique(nums.begin(), nums.end()) - nums.begin());
    m = nums.size();

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < 2; ++j) {
            a[i][j] = lower_bound(nums.begin(), nums.end(), a[i][j]) -
nums.begin();
        }
    }

    for (int i = 0; i < n; ++i) {
        g[a[i][0]].push_back(i);
        g[a[i][1]].push_back(i);
    }

    memset(mt, -1, sizeof(mt));
    int match = 0;
    for (int i = 0; i < m; ++i) {
        if (try_kuhn(i)) {
            ++T;
            ++match;
            if (match == n) {
                printf("%d\n", nums[i]);
                return 0;
            }
        }
    }

    puts("-1");

    return 0;
}
```

1027G - X-mouse in the Campus

Tutorial

1027G - X-mouse in the Campus

Some notes:

At first, there is $x^{-1} \mod m$ since $(x, m) = 1$ (lets define $\mathrm{GCD}(a, b)$ as (a, b)). That means that for each $v = 0..m-1$ there is exactly one u that $u \cdot x = v$. So if we look at this problem as the graph then it consists of cycles (consider loops as cycles of length one). So we need to know number of cycles in this graph.

At second, $(v, m) = (v \cdot x \mod m, m)$ since $(x, m) = 1$, $v \cdot x \mod m = v \cdot x - m \cdot k$ and $(v, m) \mid v$ and $(v, m) \mid m$. So all v can be divided in groups by its (v, m). And we can calculate number of cylces in each group independently.

Let fix some GCD equal to g. All numbers v such that $(v, m) = g$ can be represented as $v = g \cdot v'$ and $(v', m) = 1$. Number of such v equals to $\varphi(\frac{m}{g})$. Moreover $gv' \cdot x \equiv gv' \cdot (x \mod \frac{m}{g}) \mod m$. Here we can shift from v, x and m to v', $(x \mod \frac{m}{g})$ and $\frac{m}{g}$.

In result we need for each $d \mid m$ calculate number of cycles created by $x \mod d$ from numbers v, that $1 \le v < d$ and $(v, d) = 1$. Lets set $x = x \mod d$.

Next step is to find minimal $k > 0$ such that $x^k \equiv 1 \mod d$, let's name it order of x or $\mathrm{ord}(x)$. Then for each $0 \le i, j < k$ if $i \neq j$ then $x^i \not\equiv x^j$ and $v \cdot x^i \not\equiv v \cdot x^j$, so each cycle will have length equal to $\mathrm{ord}(x)$ and number of cycles will be equal to $\frac{\varphi(d)}{\mathrm{ord}(x \mod d)}$.

Last step is calculate $\mathrm{ord}(x)$ for each $d \mid m$. There is a fact that $\mathrm{ord}(x) \mid \varphi(d)$ so can try to iterate over all divisors df of $\varphi(d)$ and check $x^{df} \equiv 1$ by binary exponentiation (It

seems as $O(divs(m)^2 \log m)$ but it's faster and author's version work around 2 seconds. It doesn't pass but somebody can write better). But we'll speed it up. Let $\varphi(d) = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$. So we can independently for each p_i find its minimal power $b_i \le a_i$ such that $x^{\varphi(d) \cdot p_i^{b_i - a_i}} \equiv 1$. We can just iterate over all p_i and a_i since $\sum a_i = O(\log m)$.

Some words about finding $\varphi(d)$ — its factorization differs from factorization of d just by lowering degrees of primes and adding factorizations of some $(p_i - 1)$. But we can manually find factorization of $(p_i - 1)$ with memorization (or even without it) since $\sum \sqrt{p_i} \le \prod \sqrt{p_i} = O(\sqrt{m})$.

So our steps are next: factorize m, recursively iterate over all divisors of m, find $\varphi(d)$ and $\mathrm{ord}(x \mod d)$, and add to the answer $\frac{\varphi(d)}{\mathrm{ord}(x \mod d)}$.

Result complexity is $O(\sqrt{m} + \mathrm{divs}(m) \cdot \log^2 m)$.

And the last note is how to multiply $a, b < 10^{14}$ modulo $mod \le 10^{14}$. You can use binary multiplification which will give you extra $\log m$ what is not critically in this task (in C++, of course). Or you can use multiplification from hashes, which will work with 64 bit double, since it's only 10^{14}.

Solution (adedalic)

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

using namespace std;

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

#define x first
#define y second

typedef long long li;
typedef long double ld;
typedef pair<li, li> pt;

const int INF = int(1e9);
const li INF64 = li(1e18);
const ld EPS = 1e-9;

li m, x;

inline bool read() {
    if(!(cin >> m >> x))
        return false;
    return true;
}

map< li, vector<pt> > dFact;
vector<pt> fact(li v) {
    if(dFact.count(v))
        return dFact[v];

    li oldv = v;
    vector<pt> fs;
    for(li d = 2; d * d <= v; d++) {
        int cnt = 0;
        while(v % d == 0)
            v /= d, cnt++;

        if(cnt > 0)
            fs.emplace_back(d, cnt);
    }
```

```
    if(v > 1)
        fs.emplace_back(v, 1), v = 1;
    return dFact[oldv] = fs;
}

vector<pt> merge(const vector<pt> &a, const vector<pt> &b) {
    vector<pt> ans;
    int u = 0, v = 0;
    while(u < sz(a) && v < sz(b)) {
        if(a[u].x < b[v].x)
            ans.push_back(a[u++]);
        else if(a[u].x > b[v].x)
            ans.push_back(b[v++]);
        else {
            ans.emplace_back(a[u].x, a[u].y + b[v].y);
            u++, v++;
        }
    }
    while(u < sz(a))
        ans.push_back(a[u++]);
    while(v < sz(b))
        ans.push_back(b[v++]);
    return ans;
}

li getPw(li a, li b) {
    li ans = 1;
    fore(i, 0, b)
        ans *= a;
    return ans;
}

li mul(li a, li b, li mod) {
    li m = li(ld(a) * b / mod);
    li rem = a * b - m * mod;
    while(rem < 0)
        rem += mod;
    while(rem >= mod)
        rem -= mod;
    return rem;
}

li binPow(li a, li k, li mod) {
    li ans = 1 % mod;
    while(k > 0) {
        if(k & 1)
            ans = mul(ans, a, mod);
        a = mul(a, a, mod);
        k >>= 1;
    }
    return ans;
}

li findOrder(li x, li mod, const vector<pt> &f) {
    li phi = 1;
    fore(i, 0, sz(f)) fore(k, 0, f[i].y)
        phi *= f[i].x;

    if(phi == 1 || x == 0)
        return 1;

    li ord = 1;
    fore(i, 0, sz(f)) {
        li basePw = phi;
```

```
        fore(k, 0, f[i].y)
            basePw /= f[i].x;

        li curV = binPow(x, basePw, mod);

        int curPw = -1;
        fore(k, 0, f[i].y + 1) {
            if(curV != 1)
                curPw = k;
            curV = binPow(curV, f[i].x, mod);
        }

        ord *= getPw(f[i].x, curPw + 1);
    }
    return ord;
}

vector<pt> fm;
vector<int> pw;

li calc(int pos, li dv) {
    if(pos >= sz(fm)) {
        vector<pt> cf = fm;
        fore(i, 0, sz(fm))
            cf[i].y = max(pw[i] - 1, 0);

        li phi = dv;
        fore(i, 0, sz(fm)) {
            if(pw[i] > 0) {
                cf = merge(cf, fact(fm[i].x - 1));
                phi -= phi / fm[i].x;
            }
        }
        li k = findOrder(x % dv, dv, cf);
        return phi / k;
    }

    li ans = 0;
    fore(i, 0, fm[pos].y + 1) {
        pw[pos] = i;
        ans += calc(pos + 1, dv);
        dv *= fm[pos].x;
    }
    return ans;
}

inline void solve() {
    fm = fact(m);

    pw.assign(sz(fm), 0);
    li ans = calc(0, 1);

    cout << ans << endl;
}

int main() {
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
    int tt = clock();
#endif
    cout << fixed << setprecision(15);

    if(read()) {
        solve();
```

```
#ifdef _DEBUG
        cerr << "TIME = " << clock() - tt << endl;
        tt = clock();
#endif
    }
    return 0;
}
```

+39 PikMike 23 hours ago 24

Comments (24)

Write comment?

Roundgod

22 hours ago, # | +46

You don't need a binary search in problem F. The complexity is O(nlogn) for coordinate compression. Build the graph, then apply DSU or SCC(Tarjan or Korasaju) with do the job. If there exists a connected component with number of edges greater than the number of vertices, then the answer is -1. If the number of edges is equal to the number of vertices, record the largest value, If the number of edges is equal to the number of vertices minus one, record the second largest value, the answer is the maximum over all connected components.

→ Reply

zeref_dragoneel

22 hours ago, # | 0

Can you please elaborate on the part of partial sums in problem E to solve the problem in N^2 time. I am unable to get it.

→ Reply

gohan123

19 hours ago, # ^ | ← Rev. 2 +26

let lte[i] be the n.of binary strings of length n such that maximum segment of a single color has length <= i. (lte[i]-lte[i-1]) will be the n.of binary strings of length n such that maximum segment of a single color has length = i. Hence if we have lte array we can solve the problem. We can calculate lte[i] in O(n) as follows. let dp[j] be the n.of binary strings of length j such that maximum segment of a single color has length <= i.

If j>i dp[j]=dp[j-1]+dp[j-2]....dp[j-i]

else dp[j]=dp[j-1]+dp[j-2]..+dp[1]+1.

This is because in a string of length j at the beginning we can have at most i bits with the same color and after that, it's just dp[remaining length]. Now, lte[i]=dp[n]. Clearly, we can calculate the dp array by using partial sums in O(n). Since each element of the lte array takes O(n) time to calculate overall complexity is O(n^2). P.S: Don't forget to multiply by 2 at the end since all this is for a fixed color of the top left tile and it has 2 options (white or black).

AC: O(n^2)

→ Reply

zeref_dragoneel

18 hours ago, # ^ | 0

Thanks a lot.

Can you just tell me why are we adding 1 in case of j <= i.

→ Reply

18 hours ago, # ^ | +4

I think I got it. You are assuming that 1st element is fixed and 1 is for assuming all the j elements are same. Thanks

zeref_dragoneel
same. Thanks.
→ Reply


gohan123

17 hours ago, # ^ | ☆ +18
yes, glad that I was of some help :D
→ Reply


napatnicky

21 hour(s) ago, # | ☆ 0
Can someone find out what is time complexity on my code? (problem C)

Here is my code 41811657 I think it should be O(T*n*log(n)) . But why it got TLE in case 6. Thank you!!
→ Reply


adedalic

21 hour(s) ago, # ^ | ☆ 0
Maybe because $T \cdot n \cdot \log n > 10^{11}$?

In fact, for each test you create arrays `a` and `cnt` istead of making it global and clearing after each test.
→ Reply


g_abilash

20 hours ago, # ^ | ☆ 0
Why is this getting tle sir??please help me 41788605
→ Reply


Kuroni

20 hours ago, # ^ | ☆ ← Rev. 4 +1
Try using `ios_base::sync_with_stdio(false)` . I optimized my solution from TLE to AC.

TLE on test 6: 41767762

AC: 41767854
→ Reply


g_abilash

20 hours ago, # ^ | ☆ 0
Thanks _Kuroni_
→ Reply


saluev

12 hours ago, # ^ | ☆ 0
Wow, that helped me too. Thank you!
→ Reply


SJ151

20 hours ago, # ^ | ☆ +3
Try scanf and printf rather than cin and cout it worked for me
→ Reply


g_abilash

19 hours ago, # ^ | ☆ 0
but ,will i be able to use sort() function , if i am taking input through scanf() ,because i have coded in c++
→ Reply

19 hours ago, # ^ | ☆ 0
Ya Of course
→ Reply

SJ151

19 hours ago, # ^ |0

This code is not working , dude can you figure out the error

```
#include<bits/stdc++.h>
main()
{
   int i,arr[5];
   for( i=0;i<5;i++)
   {
      arr[i]=5-i;
   }
    sort(arr,arr+5);
   for(i=0;i<5;i++)
   {
      printf("%d ",arr[i]);
   }
}
```

g_abilash

→ Reply

SJ151

~~19 hours~~ Rev. 2 ago, 0 # ^ |

Add using namespace std;

→ Reply

g_abilash

17 hours ago, 0 # ^ |

Yeah , Got it !!! Thanks.

→ Reply

VinBat

19 hours ago, # | ← Rev. 2 0

What does this line do in the code for Problem E? `ans = (ans * (long long)((MOD + 1) / 2)) % MOD;` Edit : Understood. For anyone having trouble it is modular inverse

→ Reply

bazsi700

19 hours ago, # ^ | +10

It divides ans by 2 modulo MOD, as $\frac{MOD+1}{2}$ is the modular inverse of 2, because $\frac{MOD+1}{2} * 2 \equiv MOD + 1 \equiv 1 (\mod MOD)$

→ Reply

elizarov

17 hours ago, # | ← Rev. 3 +13

Kuhn's algo for F works only with this particular modification found in your solution -- you first check all the edges and if you find a suitable one, then you use it. Only if there are no suitable edges found, then you do a second loop and recurse into them (DFS). This versions passes tests. I don't know why.

However, a "vanilla Kuhn" that just uses a regular DFS (tries every edge to a non-visited vertex recursively) receives TLE, which is kind of expected, since it has complexity O(N*M) -- too big for this problem.

→ Reply

12 hours ago, # | 0

I've sent a solution for C (after the end of the contest) which sounds precisely like

saluev

I've sent a solution for C (after the end of the contest) which sounds precisely like the given here (41817149), but it still didn't fit into given time :(What did I do wrong?

→ Reply

shazly555

12 hours ago, # | ☆ 0

In problem D, Why it is better to put trap on cycle, although there may be cheaper way to put trap on each path leads to this cycle like that.
https://drive.google.com/open?id=19ff-GgeGhbhGXh5MykElhYXB3EErpq_9

→ Reply

lukijzaw97

10 hours ago, # ^ | ☆ ← Rev. 2 0

Because we don't know where the mouse starts (it may start in any vertex). Mouse can be on a cycle at the beginning and if the mouse is on a cycle at the beginning, it will be on this cycle forever. Thus it is necessary to put a trap on cycle, otherwise mouse beginning on cycle will never be caught. And according to your example, if we put a trap in vertex with cost 100, mouse will go through this vertex regardless of its beginning vertex, so it is enough to put there a trap.

→ Reply