Содержание

1 Общее

1.1 Пробный тур

1. Первым делом почекать, считаются ли WA и CE за попытки. Можно ли засылать в уже сданную задачу?
2. Пописать код и посдавать каждому члену команды.
3. Сравнить скорости компа и тестирующей системы на Флойде (n $\approx$ 1050).
4. Запустить все IDE, проверить, что все настройки работают, проверить, что работает c++11.
5. Проверить, есть ли int128.
6. Проверить работу прагм в тестирующей системе.
7. Почекать максимальный размер отправляемого кода.
8. Проверить рабочесть стресса.

1.2 Чеклист при отправке

1. Протестить на всех тестах из примера и других рандомных тестах.
2. Протестить все крайние случаи.
3. Убрать дебаг вывод.
4. Точно убедиться в правильности ответа и формата вывода.
5. Перечитать формат ввода/вывода.
6. Проверить, все ли хорошо с мультитестом

1.3 Даты

1 января 2000 года – суббота, 1 января 1900 года – понедельник.

1.4 Формулы

- Расстояние между точками по сфере: $L = R \cdot \arccos(\cos\theta_1 \cdot \cos\theta_2 + \sin\theta_1 \cdot \sin\theta_2 \cdot \cos(\varphi_1 - \varphi_2))$, где θ - широты, φ - долготы (от $-\pi$ до π)
- Объем шарового сегмента: $V = \pi h^2(R - \frac{h}{3})$, где h – высота от вершины сектора до секущей плоскости
- Площадь поверхности шарового сегмента: $S = 2\pi Rh$, где h – высота
- $2^{23} \cdot 7 \cdot 17 + 1 = 998244353$ - простое, первообразный корень – 3
- Код Грея: $g_n = n \oplus \frac{n}{2}$
- Числа Стирлинга: s(n, k) — количество перестановок на n элементах, в которых ровно k циклов. S(n, k) – это количество способов разбить n-элементное множество на k непустых подмножеств.
- $s(n, k) = (n - 1) \cdot s(n - 1, k) + s(n - 1, k - 1)$.
- $S(n, k) = k \cdot S(n - 1, k) + S(n - 1, k - 1)$.
- Gambler's ruin. У первого игрока есть n_1 монет, у второго n_2. На каждом шаге с вероятностью p второй отдает одну монету первому, а с вероятностью $q = 1 - p$ первый отдает одну монету второму. Игра заканчивается, когда у кого-нибудь не остается монет. Тогда первый выигрывает с вероятностью $\frac{1-\left(\frac{p}{q}\right)^{n_1}}{1-\left(\frac{p}{q}\right)^{n_1+n_2}}$

1.5 Простые числа

N	Первое простое после N
10^5	$10^5 + 3$
10^6	$10^6 + 3$
10^9	$10^9 + 7$
$10^9 + 7$	$10^9 + 9$
10^{12}	$10^{12} + 39$
$2^{55} - 56$	$2^{55} - 55$

1.6 Максимальное количество делителей

$\leq N$	n	Факторизация	$d(n)$
20	12	$2^2 \cdot 3^1$	6
50	48	$2^4 \cdot 3^1$	10
100	60	$2^2 \cdot 3^1 \cdot 5^1$	12
10^3	840	$2^3 \cdot 3^1 \cdot 5^1 \cdot 7^1$	32
10^4	9240	$2^3 \cdot 3^1 \cdot 5^1 \cdot 7^1 \cdot 11^1$	64
10^5	83 160	$2^3 \cdot 3^3 \cdot 5^1 \cdot 7^1 \cdot 11^1$	128
10^6	720 720	$2^4 \cdot 3^2 \cdot 5^1 \cdot 7^1 \cdot 11^1 \cdot 13^1$	240
10^7	8 648 640	$2^6 \cdot 3^3 \cdot 5^1 \cdot 7^1 \cdot 11^1 \cdot 13^1$	448
10^8	91 891 800	$2^3 \cdot 3^3 \cdot 5^2 \cdot 7^1 \cdot 11^1 \cdot 13^1 \cdot 17^1$	768
10^9	931 170 240	$2^6 \cdot 3^2 \cdot 5^2 \cdot 7^1 \cdot 11^1 \cdot 13^1 \cdot 17^1 \cdot 19^1$	1344
10^{11}	97 772 875 200	$2^6 \cdot 3^3 \cdot 5^2 \cdot 7^2 \cdot 11^1 \cdot 13^1 \cdot 17^1 \cdot 19^1$	4032
10^{12}	963 761 198 400	$2^6 \cdot 3^4 \cdot 5^2 \cdot 7^1 \cdot 11^1 \cdot 13^1 \cdot 17^1 \cdot 19^1 \cdot 23^1$	6720
10^{15}	866 421 317 361 600	$2^6 \cdot 3^4 \cdot 5^2 \cdot 7^1 \cdot 11^1 \cdot 13^1 \cdot 17^1 \cdot 19^1 \cdot 23^1 \cdot 29^1 \cdot 31^1$	26880
10^{18}	89 612 484 786 617 600	$2^8 \cdot 3^4 \cdot 5^2 \cdot 7^2 \cdot 11^1 \cdot 13^1 \cdot 17^1 \cdot 19^1 \cdot 23^1 \cdot 29^1 \cdot 31^1 \cdot 37^1$	103680

1.7 Чеклист при WA

1. Распечатать.
2. Отдать комп следующему.
3. Проверить код по строчке, даже условия в форах и т.д.
4. Проверить, не забыл ли случаи.
5. Забрать комп, потестить, предварительно составив много тестов и ответы к ним.
6. Написать стресс.
7. Возможно, решение вообще неправильное.
8. Возможно, WA где-то не там, где ты думаешь.

1.8 Правила

1. Перед написанием кода проверить решение на сэмплах.
2. Если задача неочевидная, рассказать решение другому перед написанием, подумать над тестами перед написанием.
3. В первый час 5 минут на исправление бага. Во второй час 10. В третий час 15.
4. После 3 неправильных посылок необходимо обсудить с кем-то решение и провести полное тестирование перед посылкой.
5. В первой половине контеста скипать задачу, если ничего не получается 15 минут.
6. Все задачи должны быть кем-то прочитаны в первые полтора часа.
7. Если кто-то уже прочитал задачу с длинной легендой и не занят, то узнай условие у него вместо того, чтобы читать самому, если он уверен, что правильно понял задачу.
8. В табличке отмечать краткое описание задачи в пару слов.
9. Через 3,5 часа все должны знать все несданные задачи.
10. Перед написанием адовой жести надо пообмениваться идеями по остальным задачам с остальными.
11. Пытаться применять стандартные трюки, решать более простые версии задачи.
12. 30 секунд ни на что не влияют, можно остановиться и подумать над происходящим.
13. Если придумал решение, но не можешь сейчас написать, продумай детали, проверь, все ли правильно.
14. Если не решается, посмотри на тесты, попридумывать тесты, возможно, ты не видишь какую-то очевидную вещь.
15. Если ты думал, что уже точно доказал свое решение, а после этого вылез еще какой-то случай/баг, то надо рассказать решение другому / не подходить к компу еще 15 минут.
16. Когда в конце контеста кто-то пишет код, другой пока делает тесты, чекает, нет ли других случаев.
17. После посылки минуту не чекать, зашло ли.

2 Коды

2.1 Basic setup

1. Запустить все IDE.
2. Сразу сделать файлы для всех задач.
3. Для каждой задачи свой input file.
4. Сделать template file, из него скопировать во все, не забыв поменять название input file.
5. Сделать проект для тестинга времени работы.

```
// #pragma GCC optimize("Ofast")
#include <bits/stdc++.h>

using namespace std;

#define forn(i, s, f) for (int i = (int)s; i < (int)f; i++)
#define ll long long
#define ull unsigned long long
#define ld long double
#define pii pair <int, int>
#define fs first
#define sc second
#define pf push_front
#define pb push_back
#define pop_f pop_front
#define pop_b pop_back
#define eb emplace_back
#define all(x) (x).begin(), (x).end()
#define rall(x) (x).rbegin(), (x).rend()
#define sz(x) (int)(x).size()

#ifdef DEBUG
#else
#define cerr if (false) cerr
#endif

template <typename T> istream& operator>>(istream& in, vector <T>& a) {
    for (auto& i : a) in >> i; return in;
}
template <typename T> ostream& operator<<(ostream& out, vector <T>& a) {
    for (auto& i : a) out << i << " "; return out;
}
template <typename T, typename U> void chkmin(T& a, U b) {if (a > b) a = b;}
template <typename T, typename U> void chkmax(T& a, U b) {if (a < b) a = b;}

#define int long long

void solve() {
}

signed main() {
    #ifdef DEBUG
    freopen("input.txt", "r", stdin);
    #endif
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    cout << fixed << setprecision(20);

    int t = 1;
    cin >> t;
    while (t --> 0) {
        solve();
    }
```

```
    #ifdef DEBUG
    cerr << "Runtime is: " << clock() * 1.0 / CLOCKS_PER_SEC << endl;
    #endif
    return 0;
}
```

2.2 Полезное

```
Прагмы:
#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,avx,avx2")
Встроенный декартач:
#include <ext/pb_ds/assoc_container.hpp> // Общий файл.
#include <ext/pb_ds/tree_policy.hpp> // Содержит класс tree_order_statistics_node_update
//#include <ext/pb_ds/detail/standard_policies.hpp>
using namespace __gnu_pbds;
typedef
tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>
}
ordered_set;
ordered set q;
q.find_by_order(1);
q.order_of_key(2);
Atomic hashset:
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
gp_hash_table<int, int> table;
-----------------------------------------------
const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash {
int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<key, int, chash> table;
Atomic hashmap:
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef cc_hash_table<int, int, hash<int>> ht;
Фишки битсета:
bitset<N> a;
for (int i = a._Find_first(); i != a.size(); i = a._Find_next(i)) { cout << i };
Перебор всех подмасок:
for (int s=m; ; s=(s-1)&m) {
    ... можно использовать s ...
    if (s==0) break;
}
```

2.3 Пример Python

```
def main():
    n, k = list(map(int, input().split()))
    for i in range(1, n):
        if i in lst:
```

```
            print(i ** 2)
```

2.4 Стресс-тест

Нужно создать файлы gen.cpp (для генерации тестов), solve.cpp (для корректного перебора) и test.cpp (для тестирования решения)

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    srand(time(0));
    cout << "Enter compiler flags (Generator/Test solution/Correct solution): ";
    string raw;
    cin >> raw;
    string compil;
    for (char i : raw)
    if (i == '1' || i == '0')
    compil += i;
    if (compil[0] == '1') {
        cout << "Compiling generator..." << endl;
        system("g++ -std=c++14 -o gen gen.cpp");
        cout << "Compilation is finished." << endl;
    }
    if (compil[1] == '1') {
        cout << "Compiling test solution..." << endl;
        system("g++ -std=c++14 -o test test.cpp");
        cout << "Compilation is finished." << endl;
    }
    if (compil[2] == '1') {
        cout << "Compiling correct solution..." << endl;
        system("g++ -std=c++14 -o solve solve.cpp");
        cout << "Compilation is finished." << endl;
    }
    cout << "Amount of tests: ";
    int n;
    cin >> n;
    for (int test = 1; test <= n; test++) {
        system("gen > input.txt");
        system("test < input.txt > output.txt");
        system("solve < input.txt > answer.txt");
        ifstream fout("output.txt");
        vector<string> output, answer;
        string cur;
        while (fout >> cur)
        output.push_back(cur);
        fout.close();
        ifstream fans("answer.txt");
        while (fans >> cur)
        answer.push_back(cur);
        fans.close();
        if (output == answer) {
            cout << "Test #" << test << ": OK" << endl;
        } else {
            cout << "Test #" << test << ": WA" << endl;
            cout << "Input:" << endl;
            ifstream fin("input.txt");
            while (getline(fin, cur))
            cout << cur << endl;
            fin.close();
            cout << "Got:" << endl;
```

```
            ifstream fout("output.txt");
            while (getline(fout, cur))
            cout << cur << endl;
            fout.close();
            cout << "Expected:" << endl;
            ifstream fans("answer.txt");
            while (getline(fans, cur))
            cout << cur << endl;
            fans.close();
            cout << "Enter anything to exit." << endl;
            cin.get();
            return 0;
        }
    }
    cout << "All tests passed." << endl;
    cout << "Enter anything to exit." << endl;
    cin >> raw;
    return 0;
}
```

2.5 Дробь с минимальным знаменателем между двумя другими

```
rat find_best(rat l, rat r) {
    if (l.x >= l.y) {
        ll d = l.x / l.y;
        rat res = find_best(rat(l.x - d * l.y, l.y), rat(r.x - d * r.y, r.y));
        res.x += res.y * d;
        return res;
    }
    if (r.x > r.y)
    return rat(1, 1);
    rat res = find_best(rat(r.y, r.x), rat(l.y, l.x));
    return rat(res.y, res.x);
}
```

2.6 Битсет

```
const int SZ = 6;
const int BASE = (1 << SZ);
const int MOD = BASE - 1;
struct Bitset {
    typedef unsigned long long T;
    vector<T> data;
    int n;
    void resize(int nn) {
        n = nn;
        data.resize((n + BASE - 1) / BASE);
    }
    void set(int pos, int val) {
        int id = pos >> SZ;
        int rem = pos & MOD;
        data[id] ^= data[id] & (1LL << rem);
        data[id] |= val * (1LL << rem);
    }
    int get(int pos) {
        return (data[pos >> SZ] >> (pos & MOD)) & 1;
    }
    // k > 0 -> (*this) << k
    // k < 0 -> (*this) >> (-k)
    Bitset shift (int k) {
        Bitset res;
```

```
        res.resize(n);
        int s = k / BASE;
        int rem = k % BASE;
        if (rem < 0) {
            rem += BASE;
            s--;
        }
        int p1 = BASE - rem;
        T mask = (p1 == 64)? -1: (1LL << p1) - 1;
        for (int i = max(0, -s); i < sz(data) - max(s, 0); i++) {
            res.data[i + s] |= (data[i] & mask) << rem;
        }
        if (rem != 0) {
            for (int i = max(0, -s - 1); i < sz(data) - max(s + 1, 0); i++) {
                res.data[i + s + 1] |= (data[i] >> p1) & ((1LL << rem) - 1);
            }
        }
        int cc = data.size() * BASE - n;
        res.data.back() <<= cc;
        res.data.back() >>= cc;
        return res;
    }
};
```

2.7 FFT

```
const long double PI = acosl(-1.0);
template<typename T = long double>
class Complex {
    private:
    T real_, imag_;
    public:
    Complex() = default;
    Complex(T real, T imag = 0) : real_(real), imag_(imag) {}
    Complex operator+(const Complex& other) const {
        return { real() + other.real(), imag() + other.imag() };
    }
    Complex operator*(const Complex& other) const {
        return { real() * other.real() - imag() * other.imag(),
            real() * other.imag() + imag() * other.real() };
    }
    Complex& operator*=(const Complex& other) {
        *this = (*this) * other;
        return *this;
    }
    T real() const { return real_; }
    T imag() const { return imag_; }
};
template<typename T = long double>
class FourierTransform {
    private:
    const T PI = acosl(-1.0l);
    vector<Complex<T>> roots_;
    public:
    FourierTransform(size_t n) {
        assert((n & (n - 1)) == 0);
        roots_.reserve(n);
        for (size_t i = 0; i < n; ++i) {
            roots_.emplace_back(cosl(2.0l * PI * i / n), sinl(2.0l * PI * i / n));
        }
    }
```

```
    template<typename U = int>
    vector<Complex<T>> FFT(const vector<U>& poly, size_t root_index = 1) {
        if (poly.size() == 1) {
            return { static_cast<Complex<T>>(poly[0]) };
        }
        vector<U> vecs[2];  // even, odd
        for (size_t i = 0; i < poly.size(); ++i) {
            vecs[i & 1].push_back(poly[i]);
        }
        vector<Complex<T>> result_even = FFT(vecs[0], root_index << 1);
        vector<Complex<T>> result_odd = FFT(vecs[1], root_index << 1);
        vector<Complex<T>> result(poly.size());
        size_t current_index = 0;
        size_t m = result.size() / 2;
        for (size_t i = 0; i < m; ++i) {
            result[i] = result_even[i] + result_odd[i] * roots_[cur_index];
            cur_index += root_index;
        }
        for (size_t i = m; i < result.size(); ++i) {
            size_t d = i - m;
            result[i] = result_even[d] + result_odd[d] * roots_[current_index];
            current_index += root_index;
        }
        return result;
    }
    template<typename U = int>
    vector<U> InverseFFT(const vector<Complex<T>>& transformed) {
        assert(transformed.size() == roots_.size());
        vector<Complex<T>> fft = FFT(transformed);
        reverse(fft.begin() + 1, fft.end());
        vector<U> result(fft.size());
        for (size_t i = 0; i < fft.size(); ++i) {
            //assert(fft[i].imag() < 1e-5);
            result[i] = roundl(fft[i].real() / fft.size());
        }
        return result;
    }
};
template<typename T>
void FitForMultiply(vector<T>& vector) {
    while (vector.size() & (vector.size() - 1)) {
        vector.push_back(0);
    }
    vector.resize(vector.size() * 2, 0);
}
template<typename T = int>
vector<T> Multiply(vector<T> A, vector<T> B) {
    FitForMultiply(A);
    FitForMultiply(B);
    FourierTransform<> F(A.size());
    vector<Complex<>> Af = F.FFT(A), Bf = F.FFT(B);
    for (size_t i = 0; i < Bf.size(); ++i) {
        Af[i] *= Bf[i];
    }
    return F.InverseFFT<T>(Af);
}
```

2.8 Сумма линейных функций по модулю

```
// sum(i=0..n-1) (a + b * i) div m
ll solve(ll n, ll a, ll b, ll m) {
```

```
    if (b == 0) return n * (a / m);
    if (a >= m) return n * (a / m) + solve(n, a % m, b, m);
    if (b >= m) return n * (n - 1) / 2 * (b / m) + solve(n, a, b % m, m);
    return solve((a + b * n) / m, (a + b * n) % m, m, b);
}
```

2.9 Штор-Вагнер

```
const int MAXN = 500;
int n, g[MAXN][MAXN];
int best_cost = 1e9;
vector<int> best_cut, v[MAXN];
int w[MAXN];
bool exist[MAXN], in_a[MAXN];
void mincut() {
    for (int i = 0; i < n; i++)
    v[i].assign(1, i);
    memset(exist, true, sizeof exist);
    for (int ph = 0; ph < n - 1; ph++) {
        memset(in_a, false, sizeof in_a);
        memset(w, 0, sizeof w);
        for (int it = 0, prev = 0; it < n - ph; it++) {
            int sel = -1;
            for (int i = 0; i < n; i++)
            if (exist[i] && !in_a[i] && (sel == -1 || w[i] > w[sel]))
            sel = i;
            if (it == n - ph - 1) {
                if (w[sel] < best_cost)
                best_cost = w[sel], best_cut = v[sel];
                v[prev].insert(v[prev].end(), v[sel].begin(), v[sel].end());
                for (int i = 0; i < n; i++)
                g[prev][i] = g[i][prev] += g[sel][i];
                exist[sel] = false;
            }
            else {
                in_a[sel] = true;
                for (int i = 0; i < n; i++)
                w[i] += g[sel][i];
                prev = sel;
            }
        }
    }
    // ans: best_cost, best_cut
}
```

2.10 Диниц

```
template <class T = int> struct Dinic {
    int lim = 1;
    const T INF = numeric_limits<T>::max();
    struct edge {
        int to, rev;
        T cap, flow;
    };

    int s, t;
    vector<int> level;
    vector<uint32_t> ptr;
    vector<vector<edge>> adj;

    Dinic(int n, int s, int t): s(s), t(t), level(vector<int>(n))
```

```
    , ptr(vector<uint32_t>(n)), adj(vector<vector<edge>>(n)) {}
    void addEdge(int a, int b, T cap, bool isDirected = true) {
        adj[a].push_back({b, int(adj[b].size()), cap, 0});
        adj[b].push_back({a, int(adj[a].size()) - 1, isDirected ? 0 : cap, 0});
    }
    bool bfs() {
        queue<int> q({s});
        level.assign(level.size(), -1);
        level[s] = 0;
        while (!q.empty() && level[t] == -1) {
            int v = q.front();
            q.pop();
            for (auto e : adj[v]) {
                if (level[e.to] == -1 && e.flow < e.cap && e.cap - e.flow >= lim) {
                    q.push(e.to);
                    level[e.to] = level[v] + 1;
                }
            }
        }
        return level[t] != -1;
    }
    T dfs(int v, T flow) {
        if (v == t || !flow)
        return flow;
        for (; ptr[v] < adj[v].size(); ptr[v]++) {
            edge &e = adj[v][ptr[v]];
            if (level[e.to] != level[v] + 1)
            continue;
            if (T pushed = dfs(e.to, min(flow, e.cap - e.flow))) {
                e.flow += pushed;
                adj[e.to][e.rev].flow -= pushed;
                return pushed;
            }
        }
        return 0;
    }
    long long calc() {
        long long flow = 0;
        for (lim = (1 << 30); lim > 0; lim >>= 1) {
            while (bfs()) {
                ptr.assign(ptr.size(), 0);
                while (T pushed = dfs(s, INF))
                flow += pushed;
            }
        }
        return flow;
    }
};
```

2.11 Dynamic CHT

```
const ll is_query = -(1LL << 62);
struct Line {
    ll m, b;
    mutable function<const Line *()> succ;
    bool operator<(const Line &rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line *s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
```

```
    }
};
struct HullDynamic : public multiset<Line> {
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b) * (z->m - y->m) >= (y->b - z->b) * (y->m - x->m);
    }
    void insert_line(ll m, ll b) {
        auto y = insert({m, b});
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) {
            erase(y);
            return;
        }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) {x, is_query});
        return l.m * x + l.b;
    }
};
```

2.12 Пересечение матроидов

```
// check(ctaken, 1) -- first matroid
// check(ctaken, 2) -- second matroid
vector<char> taken(m);
while (1) {
    vector<vector<int>> e(m);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            if (taken[i] && !taken[j]) {
                auto ctaken = taken;
                ctaken[i] = 0;
                ctaken[j] = 1;
                if (check(ctaken, 2)) {
                    e[i].push_back(j);
                }
            }
            if (!taken[i] && taken[j]) {
                auto ctaken = taken;
                ctaken[i] = 1;
                ctaken[j] = 0;
                if (check(ctaken, 1)) {
                    e[i].push_back(j);
                }
            }
        }
    }
    vector<int> type(m);
    // 0 -- cant, 1 -- can in \2, 2 -- can in \1
    for (int i = 0; i < m; i++) {
        if (!taken[i]) {
            auto ctaken = taken;
```

```
            ctaken[i] = 1;
            if (check(ctaken, 2)) type[i] |= 1;
        }
        if (!taken[i]) {
            auto ctaken = taken;
            ctaken[i] = 1;
            if (check(ctaken, 1)) type[i] |= 2;
        }
    }
    vector<int> w(m);
    for (int i = 0; i < m; i++) {
        w[i] = taken[i] ? ed[i].c : -ed[i].c;
    }
    vector<pair<int, int>> d(m, {INF, 0});
    for (int i = 0; i < m; i++) {
        if (type[i] & 1) d[i] = {w[i], 0};
    }
    vector<int> pr(m, -1);
    while (1) {
        vector<pair<int, int>> nd = d;
        for (int i = 0; i < m; i++) {
            if (d[i].first == INF) continue;
            for (int to : e[i]) {
                if (nd[to] > make_pair(d[i].first + w[to], d[i].second + 1)) {
                    nd[to] = make_pair(d[i].first + w[to], d[i].second + 1);
                    pr[to] = i;
                }
            }
        }
        if (d == nd) break;
        d = nd;
    }
    int v = -1;
    for (int i = 0; i < m; i++) {
        if ((d[i].first < INF && (type[i] & 2)) && (v == -1 || d[i] < d[v])) v = i;
    }
    if (v == -1) break;
    while (v != -1) {
        sum += w[v];
        taken[v] ^= 1;
        v = pr[v];
    }
    ans[--cnt] = sum;
}
```

2.13 Пожилой суффиксный автомат

```
struct state {
    int len, link;
    map<char,int> next;
};
const int MAXLEN = 112345;
state st[MAXLEN*2];
int sz, last;
// чтобы получить терминальные состояния,
// надо пройтись по суффиксным ссылкам от last до 0.
// вершин <= 2n-1, переходов <= 3n-4
// считаем размеры endpos(v) для количества вхождений строки в нашу.
// Для этого поступим следующим образом. Для каждого состояния, если оно не было получено путём клониро
// Затем будем идти по всем состояниям в порядке убывания их длины len и
// пробрасывать текущее значение cnt[v] по суффиксной ссылке:
```

```
// cnt[link(v)] += cnt[v].
void sa_init() {
    sz = last = 0;
    st[0].len = 0;
    st[0].link = -1;
    ++sz;
    /*
    // этот код нужен, только если автомат
    // строится много раз для разных строк:
    for (int i=0; i<MAXLEN*2; ++i)
    st[i].next.clear();
    */
}
void sa_extend (char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p;
    for (p=last; p!=-1 && !st[p].next.count(c); p=st[p].link)
    st[p].next[c] = cur;
    if (p == -1)
    st[cur].link = 0;
    else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len)
        st[cur].link = q;
        else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            for (; p!=-1 && st[p].next[c]==q; p=st[p].link)
            st[p].next[c] = clone;
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
```

2.14 Быстрый ввод-вывод

```
#include <cstdio>
/** Interface */
inline int readChar();
template <class T = int> inline T readInt();
template <class T> inline void writeInt( T x, char end = 0 );
inline void writeChar( int x );
inline void writeWord( const char *s );
/** Read */
static const int buf_size = 4096;
inline int getChar() {
    static char buf[buf_size];
    static int len = 0, pos = 0;
    if (pos == len)
    pos = 0, len = fread(buf, 1, buf_size, stdin);
if (pos == len)
return -1;
return buf[pos++];
}
inline int readChar() {
int c = getChar();
while (c <= 32)
```

```
c = getChar();
return c;
}
template <class T>
inline T readInt() {
int s = 1, c = readChar();
T x = 0;
if (c == '-')
s = -1, c = getChar();
while ('0' <= c && c <= '9')
x = x * 10 + c - '0', c = getChar();
return s == 1 ? x : -x;
}
/** Write */
static int write_pos = 0;
static char write_buf[buf_size];
inline void writeChar( int x ) {
if (write_pos == buf_size)
fwrite(write_buf, 1, buf_size, stdout), write_pos = 0;
write_buf[write_pos++] = x;
}
template <class T>
inline void writeInt( T x, char end ) {
if (x < 0)
writeChar('-'), x = -x;
char s[24];
int n = 0;
while (x || !n)
s[n++] = '0' + x % 10, x /= 10;
while (n--)
writeChar(s[n]);
if (end)
writeChar(end);
}
inline void writeWord( const char *s ) {
while (*s)
writeChar(*s++);
}
struct Flusher {
~Flusher() {
    if (write_pos)
    fwrite(write_buf, 1, write_pos, stdout), write_pos = 0;
}
} flusher;
// cin - 3.02, scanf - 1.2, cin+sync - 0.71
// fastRead getchar - 0.53, fastRead fread - 0.15
```

2.15 Пересечение полуплоскостей

```
namespace hpi{
const double eps = 1e-8;
typedef pair<long double, long double> pi;
bool z(long double x){ return fabs(x) < eps; }
long double ccw(pi a, pi b, pi c){
    long double dx1 = b.first - a.first;
    long double dy1 = b.second - a.second;
    long double dx2 = c.first - a.first;
    long double dy2 = c.second - a.second;
    return dx1 * dy2 - dy1 * dx2;
}
struct line{
```

```
    long double a, b, c;
    bool operator<(const line &l)const{
        bool flag1 = pi(a, b) > pi(0, 0);
        bool flag2 = pi(l.a, l.b) > pi(0, 0);
        if(flag1 != flag2) return flag1 > flag2;
        long double t = ccw(pi(0, 0), pi(a, b), pi(l.a, l.b));
        return z(t) ? c * hypot(l.a, l.b) < l.c * hypot(a, b) : t > 0;
    }
    pi slope(){ return pi(a, b); }
};
pi cross(line a, line b){
    long double det = a.a * b.b - b.a * a.b;
    return pi((a.c * b.b - a.b * b.c) / det, (a.a * b.c - a.c * b.a) / det);
}
bool bad(line a, line b, line c){
    if(ccw(pi(0, 0), a.slope(), b.slope()) <= 0) return false;
    pi crs = cross(a, b);
    return crs.first * c.a + crs.second * c.b >= c.c;
}
bool solve(vector<line> v, vector<pi> &solution){ // ax + by <= c;
    sort(v.begin(), v.end());
    deque<line> dq;
    for(auto &i : v){
        if(!dq.empty() && z(ccw(pi(0, 0), dq.back().slope(), i.slope()))) continue;
        while(dq.size() >= 2 && bad(dq[dq.size()-2], dq.back(), i)) dq.pop_back();
        while(dq.size() >= 2 && bad(i, dq[0], dq[1])) dq.pop_front();
        dq.push_back(i);
    }
    while(dq.size() > 2 && bad(dq[dq.size()-2], dq.back(), dq[0])) dq.pop_back();
    while(dq.size() > 2 && bad(dq.back(), dq[0], dq[1])) dq.pop_front();
    vector<pi> tmp;
    for(int i=0; i<dq.size(); i++){
        line cur = dq[i], nxt = dq[(i+1)%dq.size()];
        if(ccw(pi(0, 0), cur.slope(), nxt.slope()) <= eps) return false;
        tmp.push_back(cross(cur, nxt));
    }
    solution = tmp;
    return true;
}
}
```

2.16 Поллард

```
const int maxc = 500010;
ll n, x[maxc];
ll mul(ll a, ll b, ll m) { // m <= 8e18
    ll k = ((ld)a * b) / m;
    ll r = a * b - k * m;
    while (r < 0) r += m;
    while (r >= m) r -= m;
    return r;
}
void slow(int n) {
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0) {
            cout << i << ' ' << n / i << endl;
            exit(0);
        }
    cout << "IMPOSSIBLE" << endl;
    exit(0);
}
int main() {
    cin >> n;
    if (n <= (int)1e6)
        Slow(n);
    ll C = 2 * pow(n, 1.0 / 4);
    for (int cnt = 0; cnt < 4; cnt++) {
        x[0] = abs((int)rnd()) % (n - 1) + 1;
        for (i = 0; i < C; i++)
            x[i + 1] = (mul(x[i], x[i], n) + 3) % n;
        for (int i = 0; i < C; i++) {
            ll g = gcd(abs(x[i] - x[C]), n);
            if (g != 1 && g != n) {
                cout << g << ' ' << n / g << endl;
                return 0;
            }
        }
    }
    cout << "IMPOSSIBLE" << endl;
    return 0;
}
```

2.17 mincostmaxflow

```
#define _MAX_COST 1000111
#define _MAX_FLOW 1000111
template<class Flow = int, class Cost = int>
struct MinCostFlow {
    struct Edge {
        int t;
        Flow f;
        Cost c;
        Edge*next, *rev;
        Edge(int _t, Flow _f, Cost _c, Edge*_next) :
        t(_t), f(_f), c(_c), next(_next) {
        }
    };
    vector<Edge*> E;
    int addV() {
        E.push_back((Edge*) 0);
        return E.size() - 1;
    }
    Edge* makeEdge(int s, int t, Flow f, Cost c) {
        return E[s] = new Edge(t, f, c, E[s]);
    }
    Edge* addEdge(int s, int t, Flow f, Cost c) {
        Edge*e1 = makeEdge(s, t, f, c);
        Edge*e2 = makeEdge(t, s, 0, -c);
        e1->rev = e2, e2->rev = e1;
        return e1;
    }
    pair<Flow, Cost> minCostFlow(int vs, int vt) {
        //flow,cost
        int n = E.size();
        Flow flow = 0;
        Cost cost = 0;
        const Cost MAX_COST = _MAX_COST;
        const Flow MAX_FLOW = _MAX_FLOW;
        for (;;) {
            vector<Cost> dist(n, MAX_COST);
            vector<Flow> am(n, 0);
            vector<Edge*> prev(n);
            vector<bool> inQ(n, false);
            queue<int> que;
            dist[vs] = 0;
            am[vs] = MAX_FLOW;
            que.push(vs);
            inQ[vs] = true;
            while (!que.empty()) {
                int u = que.front();
                Cost c = dist[u];
                que.pop();
                inQ[u] = false;
                for (Edge*e = E[u]; e; e = e->next)
                if (e->f > 0) {
                    Cost nc = c + e->c;
                    if (nc < dist[e->t]) {
                        dist[e->t] = nc;
                        prev[e->t] = e;
                        am[e->t] = min(am[u], e->f);
                        if (!inQ[e->t]) {
                            que.push(e->t);
                            inQ[e->t] = true;
                        }
                    }
                }
            }
            if (dist[vt] == MAX_COST)
            break;
            Flow by = am[vt];
            int u = vt;
            flow += by;
            cost += by * dist[vt];
            while (u != vs) {
                Edge*e = prev[u];
                e->f -= by;
                e->rev->f += by;
                u = e->rev->t;
            }
        }
        return make_pair(flow, cost);
    }
```

```
};
```

2.18 Гаусс

```
int gauss (vector<vector<double>> a,
vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;
    vector<int> where (m, -1);
    for (int col=0,row=0;col<m && row<n;++col){
        int sel = row;
        for (int i=row; i<n; ++i)
        if (abs (a[i][col]) > abs (a[sel][col]))
        sel = i;
        if (abs (a[sel][col]) < EPS)
        continue;
        for (int i=col; i<=m; ++i)
        swap (a[sel][i], a[row][i]);
        where[col] = row;
        for (int i=0; i<n; ++i)
        if (i != row) {
            double c =
            a[i][col] / a[row][col];
            for (int j=col; j<=m; ++j)
            a[i][j] -= a[row][j] * c;
        }
        ++row;
    }
    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
    if (where[i] != -1)
    ans[i] =
    a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
        sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
        return 0;
    }
    for (int i=0; i<m; ++i)
    if (where[i] == -1)
    return INF;
    return 1;
}
```

Бинарный

```
int gauss (vector < bitset<N> > a, int n, int m,
bitset<N> & ans) {
    vector<int> where (m, -1);
    for (int col=0,row=0;col<m&&row<n;++col){
        for (int i=row; i<n; ++i)
        if (a[i][col]) {
            swap (a[i], a[row]);
            break;
        }
        if (! a[row][col])
        continue;
        where[col] = row;
        for (int i=0; i<n; ++i)
        if (i != row && a[i][col])
        a[i] ^= a[row];
        ++row;
    }
}
```

2.19 Atomic Kuhn

```
const int N = 300005;
int curcol, used[N], covered[N], match[N];
vector<int> g[N];
bool dfs(int v) {
    used[v] = curcol;
    for (int u : g[v])
    if (match[u] == -1) {
        covered[v] = 1;
        match[u] = v;
        return 1;
    }
    for (int u : g[v])
    if (used[match[u]] != curcol
    && dfs(match[u])) {
        covered[v] = 1;
        match[u] = v;
        return 1;
    }
    return 0;
}
void kuhn(int n)
{
    curcol = 0;
    for (int i = 0; i < n; i++)
    {
        used[i] = 0, match[i] = -1;
        covered[i] = 0;
    }
    while (true) {
        curcol++;
        bool ch = 0;
        for (int i = 0; i < n; i++)
        if (!covered[i] &&
        used[i] != curcol)
        ch |= dfs(i);
        if (!ch)
        break;
    }
}
```

2.20 Обратный по любому модулю

```
ll rev(ll a, ll m) // a, m > 0 {
    if (a == 1)
    return 1;
    return (1LL - rev(m % a, a) * m) / a + m;
}
int gcd(int a, int b, int &x, int &y) {
    if (a == 0) {
        x = 0; y = 1;
        return b;
    }
    int newx = 0, newy = 0;
    int d = gcd(b % a, a, newx, newy);
```

```
    x = newy - (b / a) * newx;
    y = newx;
    return d;
}
```

2.21 Произведение по модулю

```
ll mul(ll a, ll b, ll m) { // m <= 8e18
    ll k = ((ld)a * b) / m;
    ll r = a * b - k * m;
    while (r < 0) r += m;
    while (r >= m) r -= m;
    return r;
}
```

2.22 Миллер-Рабин

```
bool check_prime(ll n) {
    ll m = n - 1, k = 0;
    while (m % 2 == 0) {
    m /= 2;
    k++;
}
ll S = 50; // error_prob <= 1/4^S
for (int i = 0; i < S; i++) {
    ll a = random(1, n - 1);
    ll x = binpow(a, m, n);
    for (int j = 0; j < k; j++) {
        ll y = x * x % n;
        if (y == 1 && x != 1 && x != n - 1)
        return false;
        x = y;
    }
    if (x != 1)
    return false;
}
return true;
}
```

2.23 Венгерка

```
// graph matrix - a[][]
// 1-indexation
// p - matching
vector<int> u (n+1), v (m+1);
vector<int> p (m+1), way (m+1);
for (int i=1; i<=n; ++i) {
    p[0] = i;
    int j0 = 0;
    vector<int> minv (m+1, INF);
    vector<char> used (m+1, false);
    do {
        used[j0] = true;
        int i0 = p[j0], delta = INF, j1;
        for (int j=1; j<=m; ++j)
        if (!used[j]) {
            int cur = a[i0][j]-u[i0]-v[j];
            if (cur < minv[j]) {
                minv[j] = cur;
                way[j] = j0;
            }
            if (minv[j] < delta) {
                delta = minv[j];
                j1 = j;
            }
        }
    for (int j=0; j<=m; ++j)
        if (used[j]) {
            u[p[j]] += delta;
            v[j] -= delta;
        } else
            minv[j] -= delta;
        j0 = j1;
    } while (p[j0] != 0);
    do {
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
    } while (j0);
}
```

2.24 Мосты

```
void dfs(int v, int p = -1) {
    used[v] = true;
    tin[v] = fup[v] = timer++;
    for (int i = 0; i < sz(g[v]); i++) {
        int to = g[v][i];
        if (to == p) continue;
        if (used[to])
        fup[v] = min(fup[v], tin[to]);
        else {
            dfs(to, v);
            fup[v] = min(fup[v], fup[to]);
            if (fup[to] > tin[v])
            IS_BRIDGE(v, to);
        }
    }
}
```

2.25 Точки сочленения

```
void dfs(int v, int p = -1) {
    used[v] = true;
    tin[v] = fup[v] = timer++;
    int children = 0;
    for (int i = 0; i < sz(g[v]); i++) {
        int to = g[v][i];
        if (to == p) continue;
        if (used[to])
        fup[v] = min(fup[v], tin[to]);
        else {
            dfs(to, v);
            fup[v] = min(fup[v], fup[to]);
            if (fup[to] >= tin[v] && p != -1)
            IS_CUTPOINT(v);
            ++children;
        }
    }
    if (p == -1 && children > 1)
    IS_CUTPOINT(v);
}
```

2.26 Z-функция

```
vector<int> z_function(string s) {
    int n = sz(s);
    vector<int> z(n, 0);
    for (int i = 1, l = 0, r = 0; i < n; i++) {
        if (i <= r)
        z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < n &&
        s[z[i]] == s[i + z[i]])
        z[i]++;
        if (i + z[i] - 1 > r)
        l = i, r = i + z[i] - 1;
    }
    return z;
}
```

2.27 Префикс-функция

```
vector<int> prefix_function(string s) {
    int n = sz(s);
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j])
        j = pi[j - 1];
        if (s[i] == s[j])
        j++;
        pi[i] = j;
    }
    return pi;
}
```

2.28 Алгоритм Манакера

```
vector<int> man_nech(string s) {
    vector<int> p(sz(s), 0);
    int l = -1, r = -1;
    for (int i = 0; i < sz(s); i++) {
        if (i < r)
        p[i] = min(r - i, p[l + r - i]);
        while (i > p[i] && i + p[i] + 1 < sz(s)
        && s[i - p[i] - 1] == s[i + p[i] + 1])
        p[i]++;
        if (i + p[i] > r)
        l = i - p[i], r = i + p[i];
    }
    return p;
}
vector<int> man_ch(string s) {
    vector<int> p(sz(s), 0);
    int l = -1, r = -1;
    for (int i = 0; i < sz(s); i++) {
        if (i < r)
        p[i] = min(r - i, p[l + r - i - 1]);
        while (i >= p[i] && i + p[i] + 1 < sz(s)
        && s[i - p[i]] == s[i + p[i] + 1])
        p[i]++;
        if (i + p[i] > r)
        l = i - p[i] + 1, r = i + p[i];
    }
    return p;
}
```

2.29 Укконен

```
const int N=1123456,INF=1e9 + 7;
string a;
int t[N][26], // массив переходов (состояние, буква)
l[N], r[N],// лев и прав границ подстр ребра в предка
p[N], // предок вершины
s[N], // суффиксная ссылка
tv, // вершина текущего суффикса
tp, // положение в строке соотв месту на ребре
ts, // количество вершин
la; // текущий символ строки
void ukkadd (int c) {
    suff:;
    if (r[tv]<tp) {
        if (t[tv][c]==-1) {
            t[tv][c]=ts; l[ts]=la;
            p[ts++]=tv; tv=s[tv];
            tp=r[tv]+1; goto suff;
        }
        tv=t[tv][c]; tp=l[tv];
    }
    if (tp==-1 || c==a[tp]-'a') tp++; else {
        l[ts+1]=la; p[ts+1]=ts;
        l[ts]=l[tv]; r[ts]=tp-1; p[ts]=p[tv];
        t[ts][c]=ts+1; t[ts][a[tp]-'a']=tv;
        l[tv]=tp; p[tv]=ts;
        t[p[ts]][a[l[ts]]-'a']=ts; ts+=2;
        tv=s[p[ts-2]]; tp=l[ts-2];
        while (tp<=r[ts-2]) {
            tv=t[tv][a[tp]-'a']; tp+=r[tv]-l[tv]+1;
        }
        if (tp==r[ts-2]+1)
        s[ts-2]=tv;
        else
        s[ts-2]=ts;
        tp=r[tv]-(tp-r[ts-2])+2; goto suff;
    }
}
void build() {
    ts=2; tv=0; tp=0; fill(r,r+N,sz(a)-1);
    s[0]=1; l[0]=-1; r[0]=-1; l[1]=-1; r[1]=-1;
    memset (t, -1, sizeof t); fill(t[1],t[1]+26,0);
    for (la=0; la<sz(a); ++la)
    ukkadd (a[la]-'a');
}
```

2.30 Фенвик

```
int sum (int r)
{
    int result = 0;
    for (; r >= 0; r = (r & (r+1)) - 1)
    result += t[r];
    return result;
}
void inc (int i, int delta) {
```

```
    for (; i < n; i = (i | (i+1)))
        t[i] += delta;
}
```

2.31 OR-XOR-AND-свертки

```
vector<ll> fib, cnt, cntab, cntc, cntde;
ll revers(ll x) {
    return binpow(x % MOD, MOD - 2);
}
void adamar(vector<ll> &a, int l, int r) {
    if (l + 1 >= r) return;
    int mid = (r + l) / 2;
    adamar(a, l, mid);
    adamar(a, mid, r);
    for (int i = l; i < mid; i++) {
        ll u = a[i], v = a[i + mid - l];
        a[i] = (u + v) % MOD;
        a[i + mid - l] = (u - v) % MOD;
    }
}
void andamar(vector<ll> &a, int l, int r) {
    if (l + 1 >= r) return;
    int mid = (r + l) / 2;
    andamar(a, l, mid);
    andamar(a, mid, r);
    for (int i = l; i < mid; i++)
    a[i] = (a[i] + a[i + mid - l]) % MOD;
}
void revandamar(vector<ll> &a, int l, int r) {
    if (l + 1 >= r) return;
    int mid = (r + l) / 2;
    revandamar(a, l, mid);
    revandamar(a, mid, r);
    for (int i = l; i < mid; i++)
    a[i] = (a[i] - a[i + mid - l]) % MOD;
}
void ordamar(vector<ll> &a, int l, int r) {
    if (l + 1 >= r) return;
    int mid = (r + l) / 2;
    ordamar(a, l, mid);
    ordamar(a, mid, r);
    for (int i = l; i < mid; i++)
    a[i + mid - l] = (a[i] + a[i + mid - l]) % MOD;
}
void revordamar(vector<ll> &a, int l, int r) {
    if (l + 1 >= r) return;
    int mid = (r + l) / 2;
    revordamar(a, l, mid);
    revordamar(a, mid, r);
    for (int i = l; i < mid; i++)
    a[i + mid - l] = (-a[i] + a[i + mid - l]) % MOD;
}
vector<ll> xormult(vector<ll> a, vector<ll> b) {
    adamar(a, 0, a.size());
    adamar(b, 0, b.size());
    for (int i = 0; i < a.size(); i++)
    a[i] = (a[i] * b[i]) % MOD;
    adamar(a, 0, a.size());
    ll revn = revers(a.size());
    for (int i = 0; i < a.size(); i++)
    a[i] = (a[i] * revn) % MOD;
    return a;
}
vector<ll> ormult(vector<ll> a, vector<ll> b) {
    ordamar(a, 0, a.size());
    ordamar(b, 0, b.size());
    for (int i = 0; i < a.size(); i++)
    a[i] = (a[i] * b[i]) % MOD;
    revordamar(a, 0, a.size());
    return a;
}
vector<ll> andmult(vector<ll> a, vector<ll> b) {
    andamar(a, 0, a.size());
    andamar(b, 0, b.size());
    for (int i = 0; i < a.size(); i++)
    a[i] = (a[i] * b[i]) % MOD;
    revandamar(a, 0, a.size());
    return a;
}
int main() {
    int n;
    cin >> n;
    // assign(N, 0) for
    // fib, cnt, cntab
    // cntc, cnde
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        cnt[x]++;
    }
    fib[0] = 0, fib[1] = 1;
    for (int i = 2; i < N; ++i)
    fib[i] = (fib[i - 1] + fib[i - 2]) % MOD;
    for (int i = 0; i < N; i++) {
        for (int j = i; j > 0; j = ((j - 1) & i))
        cntab[i] = (cntab[i] + cnt[j] * cnt[i ^ j]) %
        cntab[i] = (cntab[i] + cnt[0] * cnt[i]) % MOD;
        cntab[i] %= MOD;
    }
    cntc = cnt;
    /*ordamar(cntab, 0, N);
    for (int i = 0; i < N; i++)
    cntab[i] = (cntab[i] * cntab[i]) % MOD;
    revordamar(cntab, 0, N);
    adamar(cnt, 0, N);
    for (int i = 0; i < N; ++i)
    cntde[i] = (cnt[i] * cnt[i]) % MOD;
    adamar(cntde, 0, N);
    for (int i = 0; i < N; ++i)
    cntde[i] = (cntde[i] * revers(N)) % MOD;*/
    cntde = xormult(cnt, cnt);
    for (int i = 0; i < N; i++) {
        cntab[i] = (cntab[i] * fib[i]) % MOD;
        cntc[i] = (cntc[i] * fib[i]) % MOD;
        cntde[i] = (cntde[i] * fib[i]) % MOD;
    }
    /* andamar(cntab, 0, N);
    andamar(cntc, 0, N);
    andamar(cntde, 0, N);
    for (int i = 0; i < N; ++i)
    cntc[i] = cntc[i] * cntab[i] % MOD
```

```
    * cntde[i] % MOD;
    revandamar(cntc, 0, N);*/
    cntc = andmult(andmult(cntab, cntc), cntde);
    ll ans = 0;
    for (int i = 0; i < L; ++i)
    ans = (ans + cntc[(1 << i)]) % MOD;
    cout << (ans % MOD + MOD) % MOD;
    return 0;
}
```

2.32 Суфмас

```
const char C = 'a' - 1;
// before first letter // change
const char maxchar = 'z'; // change
vector<int> suffarray(string s)
// without $ at the end
{
    vector<int> p, c, pn, cn, cnt;
    int n = (int)s.size();
    c.assign(n, 0);
    for (int i = 0; i < n; i++)
    c[i] = s[i] - C;
    for (int j = 0; j <= (maxchar - C); j++)
    for (int i = 0; i < n; i++)
    if (c[i] == j)
    p.push_back(i);
    int maxc = c[p.back()];
    pn.resize(n);
    for (int k = 0; (1 << k) <= 2 * n; k++) {
        for (int i = 0; i < n; i++)
        pn[i] =
        ((p[i] - (1 << k)) % n + n) % n;
        cnt.assign(maxc + 3, 0);
        for (int i = 0; i < n; i++)
        cnt[c[i] + 1]++;
        for (int i = 1; i <= maxc + 2; i++)
        cnt[i] += cnt[i - 1];
        for (int i = 0; i < n; i++)
        p[cnt[c[pn[i]]]++] = pn[i];
        cn.assign(n, 0);
        cn[p[0]] = 1;
        for (int i = 1; i < n; i++)
        if (c[p[i]] == c[p[i - 1]] &&
        c[(p[i] + (1 << k)) % n] ==
        c[(p[i - 1] + (1 << k)) % n])
        cn[p[i]] = cn[p[i - 1]];
        else
        cn[p[i]] = cn[p[i - 1]] + 1;
        maxc = cn[p.back()];
        c = cn;
    }
    return p;
}

vector<int> findlcp(string s, vector<int> p) {
    vector<int> lcp, mem;
    int n = (int)s.size();
    mem.resize(n);
    for (int i = 0; i < n; i++)
    mem[p[i]] = i;
    lcp.assign(n, 0);for (int i = 0; i < n; i++) {
        if (i)
        lcp[mem[i]] = max(lcp[mem[i - 1]] - 1, 0);
        if (mem[i] == n - 1)
        continue;
        while (max(i, p[mem[i] + 1]) +
        lcp[mem[i]] < n
        && s[i + lcp[mem[i]]] ==
        s[p[mem[i] + 1] + lcp[mem[i]]])
        lcp[mem[i]]++;
    }
    return lcp;
}
```

2.33 ConvexHullTrick

```
typedef long long integer;
typedef long double ld;
struct Line {
    integer k, b;
    Line():
    k(0), b(0) {}
    Line(integer k, integer b):
    k(k), b(b) {}
    ld operator()(ld x) {
        return x * (ld)k + (ld)b;
    }
};
const integer INF = 2e18; // change
struct CHT {
    vector<Line> lines;
    bool mini; // cht on minimum
    ld f(Line l1, Line l2) {
        return (ld)(l1.b - l2.b) / (ld)(l2.k - l1.k);
    }
    void addLine(integer k, integer b) {
        if (!mini) {
            k = -k;
            b = -b;
        }
        Line l(k, b);
        while (lines.size() > 1) {
            if (lines.back().k == k) {
                if (lines.back().b > b)
                lines.pop_back();
                else
                break;
                continue;
            }
            ld x1 = f(lines.back(), l);
            ld x2 = f(lines.back(),
            lines[lines.size() - 2]);
            if (x1 > x2)
            break;
            lines.pop_back();
        }
        if (!lines.size() || lines.back().k != k)
        lines.push_back(l);
    }
    CHT(vector<pair<integer, integer> > v,
```

```
    bool ok = 1) // change{
        mini = ok;
        lines.clear();
        for (int i = 0; i < v.size(); i++)
        addLine(v[i].first, v[i].second);
    }
    integer getmin(integer x)
    //find of integer!
    {
        if (!lines.size())
        return (mini ? INF : -INF);
        int l = 0, r = lines.size();
        while (r - l > 1) {
            int mid = (r + l) / 2;
            if (f(lines[mid], lines[mid - 1])
            <= (ld)x)
            l = mid;
            else
            r = mid;
        }
        integer ans = lines[l].k * x +
        lines[l].b;
        return (mini ? ans : -ans);
    }
};
```

2.34 Конденсация орграфа

```
using Graph = vector<vector<int>>;
class Condensator {
    private:
    Graph graph_, reversed_;
    vector<char> visited_;
    vector<int> color_;
    int current_color_ = 0;
    stack<uint32_t> vertex_order_;
    // Kosaraju Algorithm
    void FillVertexOrder(uint32_t cur) {
        visited_[cur] = 1;
        for (uint32_t next : graph_[cur]) {
            if (!visited_[next]) {
                FillVertexOrder(next);
            }
        }
        vertex_order_.push(cur);
    }
    void PaintVertexes(uint32_t cur) {
        color_[cur] = current_color_;
        for (uint32_t next : reversed_[cur]) {
            if (color_[next] == -1) {
                PaintVertexes(next);
            }
        }
    }
    public:
    Condensator(const Graph& g) : graph_(g)
    , reversed_(Graph(g.size()))
    , visited_(vector<char>(g.size(), 0))
    , color_(vector<int>(g.size(), -1))
    {
        for (size_t i = 0; i < g.size(); ++i){
            for (uint32_t j : g[i]) {
                reversed_[j].push_back(i);
            }
        }
    }
    pair<Graph, vector<vector<int>>> Condensate() {
        for (size_t v = 0; v < graph_.size(); ++v) {
            if (!visited_[v]) {
                FillVertexOrder(v);
            }
        }
        while (!vertex_order_.empty()) {
            if (color_[vertex_order_.top()] == -1) {
                PaintVertexes(vertex_order_.top());
                ++current_color_;
                vertex_order_.pop();
            }
        }
        Graph condensation(size_t(cur_color_));
        vector<vector<int>> to_colors(cur_color_);
        for (size_t v = 0; v < graph_.size(); ++v) {
            size_t a = size_t(color_[v]);
            to_colors[a].push_back(v);
            for (uint32_t u : graph_[v]) {
                size_t b = size_t(color_[u]);
                if (a != b) {
                    condensation[a].push_back(b);
                }
            }
        }
        return { condensation, to_colors };
    }
};
```

2.35 Поиск эйлерова цикла

```
vector<int> GetEulerCycle(SetGraph g) {
    for (size_t i = 0; i < g.size(); ++i) {
            if (g[i].size() % 2) {
            return {};
        }
    }

    vector<int> ans;
    stack<int> dfs;
    dfs.push(0);
    while (!dfs.empty()) {
        int cur = dfs.top();
        if (g[cur].empty()) {
            dfs.pop();
            ans.push_back(cur);
        } else {
            int nxt = *g[cur].begin();
            g[cur].erase(g[cur].find(nxt));
            g[nxt].erase(g[nxt].find(cur));
            dfs.push(nxt);
        }
    }
    return ans;
}
```

```
vector<int> GetEulerCycleOriented(SetGraph g) {
    vector<size_t> incoming(g.size(), 0);
    for (size_t i = 0; i < g.size(); ++i) {
        for (int nxt : g[i]) {
            ++incoming[nxt];
        }
    }
    for (size_t i = 0; i < g.size(); ++i) {
        if (incoming[i] != g[i].size()) {
            return {};
        }
    }

    vector<int> ans;
    stack<int> dfs;
    dfs.push(0);
    while (!dfs.empty()) {
        int cur = dfs.top();
        if (g[cur].empty()) {
            dfs.pop();
            ans.push_back(cur);
        } else {
            int nxt = *g[cur].begin();
            g[cur].erase(g[cur].find(nxt));
            dfs.push(nxt);
        }
    }
    std::reverse(ans.begin(), ans.end());
    return ans;
}
```

2.36 Выпуклая оболочка

```
// Returns convex hull in clockwise order
template<typename T>
Polygon<T> ConvexHull(vector<Point<T>> p) {
    sort(p.begin(), p.end(),
    [](Point<T> a, Point<T> b) {
        return (a.x == b.x)
        ? a.y < b.y
        : a.x < b.x;
    });
    int n = p.size();
    if (n < 3)
    return p;
    vector<Point<T>> hull;
    for (int i = 0; i < n; ++i) {
        while (hull.size() >= 2) {
            Point<T> a = hull[hull.size() - 2];
            Point<T> b = hull.back();
            Point<T> c = p[i];
            Vector<T> ab = { a, b }, bc = { b, c };
            if (sgn(ab % bc) >= 0)
            hull.pop_back();
            else
            break;
        }
        hull.pb(p[i]);
    }
    hull.pb(p[n - 2]);
    int lim = hull.size();
    for (int i = n - 3; i >= 0; i--) {
        while (hull.size() >= lim) {
            Point<T> a = hull[hull.size() - 2];
            Point<T> b = hull.back();
            Point<T> c = p[i];
            Vector<T> ab = { a, b }, bc = { b, c };
            if (sgn(ab % bc) >= 0)
            hull.pop_back();
            else
            break;
        }
        hull.pb(p[i]);
    }
    hull.pop_back();
    return hull;
}
```

2.37 Пересечение двух прямых

```
template<typename T>
constexpr inline bool Intersect(
    const Line<T>& line1,
    const Line<T>& line2,
    Point<long double> * const result = nullptr
) {
    long double vector_product = line1.a * line2.b -
    line1.b * line2.a;
    if (abs(vector_product) < 1e-9) {
        if (abs(line1.c - line2.c) < 1e-9) {
            if (result != nullptr) {
                // Any point is good
                *result = line1.GetPointByOffset(0);
            }
            return true;
        } else {
            return false;
        }
    }
    if (result != nullptr) {
        result->x = (line2.c * line1.b - line2.b *
        line1.c) / vector_product;
        result->y = (line2.a * line1.c - line2.c *
        line1.a) / vector_product;
    }
    return true;
}
```

2.38 Менеджер отрезков

```
class SegmentManager {
    private:
    int united_length_ = 0;
    set<pair<int, int>> segments_rl_;

    public:
    int AddSegment(int l, int r) {
        int border_intersection = 0;
        for (auto iter = segments_rl_.lower_bound({ l, -1 }); iter != segments_rl_.end();
        iter = segments_rl_.lower_bound({ l, -1 })) {
            int other_l = (*iter).second, other_r = (*iter).first;
            if (other_l < l) {  // other_l...l
                if (r < other_r) {  // other_l...l...r...other_r
                    return border_intersection;
                } else {  // other_l...l...other_r...r
                    ++border_intersection;
                    united_length_ -= other_r - other_l;
                    l = other_l;
                }
            } else {  // l...other_l
                if (other_l <= r) {  // l...other_l...r
                    ++border_intersection;
                    if (other_r <= r) {  // l...other_l...other_r...r
                        ++border_intersection;
                        united_length_ -= other_r - other_l;
                    } else { // l...other_l...r...other_r
                        united_length_ -= other_r - other_l;
                        r = other_r;
                    }
                } else {  // l...r...other_l...other_r
                    break;
                }
            }
            segments_rl_.erase(iter);
        }
        segments_rl_.insert({ r, l });
        united_length_ += r - l;
        return border_intersection;
    }

    constexpr int GetUnitedLength() const { return united_length_; }

    inline bool HasPoint(int point) const {
        auto iter = segments_rl_.lower_bound({ point, -1 });
        return (iter != segments_rl_.end() && (*iter).second <= point);
    }

    set<pair<int, int>>& GetSegmentsRL() const { return segments_rl_; }
};
```