Algorithm Library

st1vdy

2021 年 10 月 27 日

目录

1 多项式

1.1 FFT - tourist

```cpp
/* copy from tourist */
namespace FFT {
    typedef double dbl;

    struct num {
        dbl x, y;
        num() { x = y = 0; }
        num(dbl x, dbl y) : x(x), y(y) { }
    };

    inline num operator+(num a, num b) { return num(a.x + b.x, a.y + b.y); }
    inline num operator-(num a, num b) { return num(a.x - b.x, a.y - b.y); }
    inline num operator*(num a, num b) { return num(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
    inline num conj(num a) { return num(a.x, -a.y); }

    int base = 1;
    vector<num> roots = { {0, 0}, {1, 0} };
    vector<int> rev = { 0, 1 };

    const dbl PI = acosl(-1.0);

    void ensure_base(int nbase) {
        if (nbase <= base) {
            return;
        }
        rev.resize(1 << nbase);
        for (int i = 0; i < (1 << nbase); i++) {
            rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
        }
        roots.resize(1 << nbase);
        while (base < nbase) {
            dbl angle = 2 * PI / (1 << (base + 1));
            for (int i = 1 << (base - 1); i < (1 << base); i++) {
                roots[i << 1] = roots[i];
                dbl angle_i = angle * (2 * i + 1 - (1 << base));
                roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
            }
            base++;
        }
```

```
    }

    void fft(vector<num>& a, int n = -1) {
        if (n == -1) {
            n = a.size();
        }
        assert((n & (n - 1)) == 0);
        int zeros = __builtin_ctz(n);
        ensure_base(zeros);
        int shift = base - zeros;
        for (int i = 0; i < n; i++) {
            if (i < (rev[i] >> shift)) {
                swap(a[i], a[rev[i] >> shift]);
            }
        }
        for (int k = 1; k < n; k <<= 1) {
            for (int i = 0; i < n; i += 2 * k) {
                for (int j = 0; j < k; j++) {
                    num z = a[i + j + k] * roots[j + k];
                    a[i + j + k] = a[i + j] - z;
                    a[i + j] = a[i + j] + z;
                }
            }
        }
    }

    vector<num> fa, fb;

    vector<long long> multiply(vector<int>& a, vector<int>& b) {
        int need = a.size() + b.size() - 1;
        int nbase = 1;
        while ((1 << nbase) < need) nbase++;
        ensure_base(nbase);
        int sz = 1 << nbase;
        if (sz > (int)fa.size()) {
            fa.resize(sz);
        }
        for (int i = 0; i < sz; i++) {
            int x = (i < (int)a.size() ? a[i] : 0);
            int y = (i < (int)b.size() ? b[i] : 0);
            fa[i] = num(x, y);
        }
        fft(fa, sz);
```

```
        num r(0, -0.25 / (sz >> 1));
        for (int i = 0; i <= (sz >> 1); i++) {
            int j = (sz - i) & (sz - 1);
            num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
            if (i != j) {
                fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
            }
            fa[i] = z;
        }
        for (int i = 0; i < (sz >> 1); i++) {
            num A0 = (fa[i] + fa[i + (sz >> 1)]) * num(0.5, 0);
            num A1 = (fa[i] - fa[i + (sz >> 1)]) * num(0.5, 0) * roots[(sz
                >> 1) + i];
            fa[i] = A0 + A1 * num(0, 1);
        }
        fft(fa, sz >> 1);
        vector<long long> res(need);
        for (int i = 0; i < need; i++) {
            if (i % 2 == 0) {
                res[i] = fa[i >> 1].x + 0.5;
            } else {
                res[i] = fa[i >> 1].y + 0.5;
            }
        }
        return res;
    }

    vector<long long> square(const vector<int>& a) {
        int need = a.size() + a.size() - 1;
        int nbase = 1;
        while ((1 << nbase) < need) nbase++;
        ensure_base(nbase);
        int sz = 1 << nbase;
        if ((sz >> 1) > (int)fa.size()) {
            fa.resize(sz >> 1);
        }
        for (int i = 0; i < (sz >> 1); i++) {
            int x = (2 * i < (int)a.size() ? a[2 * i] : 0);
            int y = (2 * i + 1 < (int)a.size() ? a[2 * i + 1] : 0);
            fa[i] = num(x, y);
        }
        fft(fa, sz >> 1);
        num r(1.0 / (sz >> 1), 0.0);
```

```
        for (int i = 0; i <= (sz >> 2); i++) {
            int j = ((sz >> 1) - i) & ((sz >> 1) - 1);
            num fe = (fa[i] + conj(fa[j])) * num(0.5, 0);
            num fo = (fa[i] - conj(fa[j])) * num(0, -0.5);
            num aux = fe * fe + fo * fo * roots[(sz >> 1) + i] * roots[(sz
                >> 1) + i];
            num tmp = fe * fo;
            fa[i] = r * (conj(aux) + num(0, 2) * conj(tmp));
            fa[j] = r * (aux + num(0, 2) * tmp);
        }
        fft(fa, sz >> 1);
        vector<long long> res(need);
        for (int i = 0; i < need; i++) {
            if (i % 2 == 0) {
                res[i] = fa[i >> 1].x + 0.5;
            } else {
                res[i] = fa[i >> 1].y + 0.5;
            }
        }
        return res;
    }

    vector<int> multiply_mod(vector<int>& a, vector<int>& b, int m, int eq =
        0) {
        int need = a.size() + b.size() - 1;
        int nbase = 0;
        while ((1 << nbase) < need) nbase++;
        ensure_base(nbase);
        int sz = 1 << nbase;
        if (sz > (int)fa.size()) {
            fa.resize(sz);
        }
        for (int i = 0; i < (int)a.size(); i++) {
            int x = (a[i] % m + m) % m;
            fa[i] = num(x & ((1 << 15) - 1), x >> 15);
        }
        fill(fa.begin() + a.size(), fa.begin() + sz, num{ 0, 0 });
        fft(fa, sz);
        if (sz > (int) fb.size()) {
            fb.resize(sz);
        }
        if (eq) {
            copy(fa.begin(), fa.begin() + sz, fb.begin());
```

```
        } else {
            for (int i = 0; i < (int)b.size(); i++) {
                int x = (b[i] % m + m) % m;
                fb[i] = num(x & ((1 << 15) - 1), x >> 15);
            }
            fill(fb.begin() + b.size(), fb.begin() + sz, num{ 0, 0 });
            fft(fb, sz);
        }
        dbl ratio = 0.25 / sz;
        num r2(0, -1);
        num r3(ratio, 0);
        num r4(0, -ratio);
        num r5(0, 1);
        for (int i = 0; i <= (sz >> 1); i++) {
            int j = (sz - i) & (sz - 1);
            num a1 = (fa[i] + conj(fa[j]));
            num a2 = (fa[i] - conj(fa[j])) * r2;
            num b1 = (fb[i] + conj(fb[j])) * r3;
            num b2 = (fb[i] - conj(fb[j])) * r4;
            if (i != j) {
                num c1 = (fa[j] + conj(fa[i]));
                num c2 = (fa[j] - conj(fa[i])) * r2;
                num d1 = (fb[j] + conj(fb[i])) * r3;
                num d2 = (fb[j] - conj(fb[i])) * r4;
                fa[i] = c1 * d1 + c2 * d2 * r5;
                fb[i] = c1 * d2 + c2 * d1;
            }
            fa[j] = a1 * b1 + a2 * b2 * r5;
            fb[j] = a1 * b2 + a2 * b1;
        }
        fft(fa, sz);
        fft(fb, sz);
        vector<int> res(need);
        for (int i = 0; i < need; i++) {
            long long aa = fa[i].x + 0.5;
            long long bb = fb[i].x + 0.5;
            long long cc = fa[i].y + 0.5;
            res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
        }
        return res;
    }

    vector<int> square_mod(vector<int>& a, int m) {
```

```
        return multiply_mod(a, a, m, 1);
    }
};
```

1.2 形式幂级数

```
#define db double
#ifndef ONLINE_JUDGE // 这三个函数是给MSVC用的，G++不需要
inline int __builtin_clz(int v) { // 返回前导0的个数
    return __lzcnt(v);
}
inline int __builtin_ctz(int v) { // 返回末尾0的个数
    if (v == 0) {
        return 0;
    }
    __asm {
        bsf eax, dword ptr[v];
    }
}
inline int __builtin_popcount(int v) { // 返回二进制中1的个数
    return __popcnt(v);
}
#endif
struct Complex {
    db real, imag;
    Complex(db x = 0, db y = 0) :real(x), imag(y) {}
    Complex& operator+=(const Complex& rhs) {
        real += rhs.real; imag += rhs.imag;
        return *this;
    }
    Complex& operator-=(const Complex& rhs) {
        real -= rhs.real; imag -= rhs.imag;
        return *this;
    }
    Complex& operator*=(const Complex& rhs) {
        db t_real = real * rhs.real - imag * rhs.imag;
        imag = real * rhs.imag + imag * rhs.real;
        real = t_real;
        return *this;
    }
    Complex& operator/=(double x) {
        real /= x, imag /= x;
        return *this;
```

```cpp
    }
    friend Complex operator + (const Complex& a, const Complex& b) { return Complex(a) += b; }
    friend Complex operator - (const Complex& a, const Complex& b) { return Complex(a) -= b; }
    friend Complex operator * (const Complex& a, const Complex& b) { return Complex(a) *= b; }
    friend Complex operator / (const Complex& a, const db& b) { return Complex(a) /= b; }
    Complex power(long long p) const {
        assert(p >= 0);
        Complex a = *this, res = { 1,0 };
        while (p > 0) {
            if (p & 1) res = res * a;
            a = a * a;
            p >>= 1;
        }
        return res;
    }
    static long long val(double x) { return x < 0 ? x - 0.5 : x + 0.5; }
    inline long long Real() const { return val(real); }
    inline long long Imag() const { return val(imag); }
    Complex conj()const { return Complex(real, -imag); }
    explicit operator int()const { return Real(); }
    friend ostream& operator<<(ostream& stream, const Complex& m) {
        return stream << complex<db>(m.real, m.imag);
    }
};
constexpr int MOD = 998244353;
constexpr int Phi_MOD = 998244352;
inline int exgcd(int a, int md = MOD) {
    a %= md;
    if (a < 0) a += md;
    int b = md, u = 0, v = 1;
    while (a) {
        int t = b / a;
        b -= t * a; swap(a, b);
        u -= t * v; swap(u, v);
    }
    assert(b == 1);
    if (u < 0) u += md;
    return u;
}
```

```
inline int add(int a, int b) { return a + b >= MOD ? a + b - MOD : a + b; }
inline int sub(int a, int b) { return a - b < 0 ? a - b + MOD : a - b; }
inline int mul(int a, int b) { return 1LL * a * b % MOD; }
inline int powmod(int a, long long b) {
    int res = 1;
    while (b > 0) {
        if (b & 1) res = mul(res, a);
        a = mul(a, a);
        b >>= 1;
    }
    return res;
}

vector<int> inv, fac, ifac;
void prepare_factorials(int maximum) {
    inv.assign(maximum + 1, 1);
    // Make sure MOD is prime, which is necessary for the inverse algorithm
       below.
    for (int p = 2; p * p <= MOD; p++)
        assert(MOD % p != 0);
    for (int i = 2; i <= maximum; i++)
        inv[i] = mul(inv[MOD % i], (MOD - MOD / i));

    fac.resize(maximum + 1);
    ifac.resize(maximum + 1);
    fac[0] = ifac[0] = 1;

    for (int i = 1; i <= maximum; i++) {
        fac[i] = mul(i, fac[i - 1]);
        ifac[i] = mul(inv[i], ifac[i - 1]);
    }
}
namespace FFT {
    vector<Complex> roots = { Complex(0, 0), Complex(1, 0) };
    vector<int> bit_reverse;
    int max_size = 1 << 20;
    const long double pi = acosl(-1.0l);
    constexpr int FFT_CUTOFF = 150;
    inline bool is_power_of_two(int n) { return (n & (n - 1)) == 0; }
    inline int round_up_power_two(int n) {
        assert(n > 0);
        while (n & (n - 1)) {
            n = (n | (n - 1)) + 1;
```

```
        }
        return n;
    }
    // Given n (a power of two), finds k such that n == 1 << k.
    inline int get_length(int n) {
        assert(is_power_of_two(n));
        return __builtin_ctz(n);
    }
    // Rearranges the indices to be sorted by lowest bit first, then second
       lowest, etc., rather than highest bit first.
    // This makes even-odd div-conquer much easier.
    void bit_reorder(int n, vector<Complex>& values) {
        if ((int)bit_reverse.size() != n) {
            bit_reverse.assign(n, 0);
            int length = get_length(n);
            for (int i = 0; i < n; i++) {
                bit_reverse[i] = (bit_reverse[i >> 1] >> 1) + ((i & 1) << (
                    length - 1));
            }
        }
        for (int i = 0; i < n; i++) {
            if (i < bit_reverse[i]) {
                swap(values[i], values[bit_reverse[i]]);
            }
        }
    }
    void prepare_roots(int n) {
        assert(n <= max_size);
        if ((int)roots.size() >= n)
            return;
        int length = get_length(roots.size());
        roots.resize(n);
        // The roots array is set up such that for a given power of two n >=
            2, roots[n / 2] through roots[n - 1] are
        // the first half of the n-th primitive roots of MOD.
        while (1 << length < n) {
            for (int i = 1 << (length - 1); i < 1 << length; i++) {
                roots[2 * i] = roots[i];
                long double angle = pi * (2 * i + 1) / (1 << length);
                roots[2 * i + 1] = Complex(-cos(angle), -sin(angle));
            }
            length++;
        }
```

```
    }
    void fft_iterative(int N, vector<Complex>& values) {
        assert(is_power_of_two(N));
        prepare_roots(N);
        bit_reorder(N, values);
        for (int n = 1; n < N; n *= 2) {
            for (int start = 0; start < N; start += 2 * n) {
                for (int i = 0; i < n; i++) {
                    Complex& even = values[start + i];
                    Complex odd = values[start + n + i] * roots[n + i];
                    values[start + n + i] = even - odd;
                    values[start + i] = even + odd;
                }
            }
        }
    }
    vector<long long> multiply(vector<int> a, vector<int> b) { // 普通FFT
        int n = a.size();
        int m = b.size();
        if (min(n, m) < FFT_CUTOFF) {
            vector<long long> res(n + m - 1);
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < m; j++) {
                    res[i + j] += 1LL * a[i] * b[j];
                }
            }
            return res;
        }
        int N = round_up_power_two(n + m - 1);
        vector<Complex> tmp(N);
        for (int i = 0; i < a.size(); i++) tmp[i].real = a[i];
        for (int i = 0; i < b.size(); i++) tmp[i].imag = b[i];
        fft_iterative(N, tmp);
        for (int i = 0; i < N; i++) tmp[i] = tmp[i] * tmp[i];
        reverse(tmp.begin() + 1, tmp.end());
        fft_iterative(N, tmp);
        vector<long long> res(n + m - 1);
        for (int i = 0; i < res.size(); i++) {
            res[i] = tmp[i].imag / 2 / N + 0.5;
        }
        return res;
    }
    vector<int> mod_multiply(vector<int> a, vector<int> b, int lim =
```

```
    max_size) { // 任意模数FFT
    int n = a.size();
    int m = b.size();
    if (min(n, m) < FFT_CUTOFF) {
        vector<int> res(n + m - 1);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                res[i + j] += 1LL * a[i] * b[j] % MOD;
                res[i + j] %= MOD;
            }
        }
        return res;
    }
    int N = round_up_power_two(n + m - 1);
    N = min(N, lim);
    vector<Complex> P(N);
    vector<Complex> Q(N);
    for (int i = 0; i < n; i++) {
        P[i] = Complex(a[i] >> 15, a[i] & 0x7fff);
    }
    for (int i = 0; i < m; i++) {
        Q[i] = Complex(b[i] >> 15, b[i] & 0x7fff);
    }
    fft_iterative(N, P);
    fft_iterative(N, Q);
    vector<Complex>A(N), B(N), C(N), D(N);
    for (int i = 0; i < N; i++) {
        Complex P2 = P[(N - i) & (N - 1)].conj();
        A[i] = (P2 + P[i]) * Complex(0.5, 0),
            B[i] = (P2 - P[i]) * Complex(0, 0.5);
        Complex Q2 = Q[(N - i) & (N - 1)].conj();
        C[i] = (Q2 + Q[i]) * Complex(0.5, 0),
            D[i] = (Q2 - Q[i]) * Complex(0, 0.5);
    }
    for (int i = 0; i < N; i++) {
        P[i] = (A[i] * C[i]) + (B[i] * D[i]) * Complex(0, 1),
            Q[i] = (A[i] * D[i]) + (B[i] * C[i]) * Complex(0, 1);
    }
    reverse(P.begin() + 1, P.end());
    reverse(Q.begin() + 1, Q.end());
    fft_iterative(N, P);
    fft_iterative(N, Q);
    for (int i = 0; i < N; i++) {
```

```cpp
                P[i] /= N, Q[i] /= N;
            }
            int size = min(n + m - 1, lim);
            vector<int> res(size);
            for (int i = 0; i < size; i++) {
                long long ac = P[i].Real() % MOD, bd = P[i].Imag() % MOD,
                    ad = Q[i].Real() % MOD, bc = Q[i].Imag() % MOD;
                res[i] = ((ac << 30) + bd + ((ad + bc) << 15)) % MOD;
            }
            return res.resize(n + m - 1), res;
        }
        vector<int> mod_inv(vector<int> a) { // 多项式逆
            int n = a.size();
            int N = round_up_power_two(a.size());
            a.resize(N * 2);
            vector<int> res(1);
            res[0] = exgcd(a[0]);
            for (int i = 2; i <= N; i <<= 1) {
                vector<int> tmp(a.begin(), a.begin() + i);
                int n = (i << 1);
                tmp = mod_multiply(tmp, mod_multiply(res, res, n), n);
                res.resize(i);
                for (int j = 0; j < i; j++) {
                    res[j] = add(res[j], sub(res[j], tmp[j]));
                }
            }
            res.resize(n);
            return res;
        }
        vector<int> integral(vector<int> a) { // 多项式积分
            assert(a.size() <= inv.size());
            a.push_back(0);
            for (int i = (int)a.size() - 1; i >= 1; i--) {
                a[i] = mul(a[i - 1], inv[i]);
            }
            return a;
        }
        vector<int> differential(vector<int> a) { // 多项式求导
            for (int i = 0; i < (int)a.size() - 1; i++) {
                a[i] = mul(i + 1, a[i + 1]);
            }
            a.pop_back();
            return a;
```

```
    }
    vector<int> ln(vector<int> a) { // 多项式对数函数
        assert((int)a[0] == 1);
        auto b = mod_multiply(differential(a), mod_inv(a));
        b = integral(b);
        b[0] = 0;
        return b;
    }
    vector<int> exp(vector<int> a) { // 多项式指数函数
        int N = round_up_power_two(a.size());
        int n = a.size();
        a.resize(N * 2);
        vector<int> res{ 1 };
        for (int i = 2; i <= N; i <<= 1) {
            auto tmp = res;
            tmp.resize(i);
            tmp = ln(tmp);
            for (int j = 0; j < i; j++) {
                tmp[j] = sub(a[j], tmp[j]);
            }
            tmp[0] = add(tmp[0], 1);
            res.resize(i);
            res = mod_multiply(res, tmp, i << 1);
            fill(res.begin() + i, res.end(), 0);
        }
        res.resize(n);
        return res;
    }
    // Multiplies many polynomials whose total degree is n in O(n log^2 n).
    vector<int> mod_multiply_all(const vector<vector<int>>& polynomials) {
        if (polynomials.empty())
            return { 1 };
        struct compare_size {
            bool operator()(const vector<int>& x, const vector<int>& y) {
                return x.size() > y.size();
            }
        };
        priority_queue<vector<int>, vector<vector<int>>, compare_size> pq(
            polynomials.begin(), polynomials.end());
        while (pq.size() > 1) {
            vector<int> a = pq.top(); pq.pop();
            vector<int> b = pq.top(); pq.pop();
            pq.push(mod_multiply(a, b));
```

```
        }
        return pq.top();
    }
    tuple<int, int, bool> power_reduction(string s, int n) { // 多项式快速幂预处理
        int p = 0, q = 0; bool zero = false;
        for (int i = 0; i < s.length(); i++) {
            p = mul(p, 10);
            p = add(p, s[i] - '0');
            q = 1LL * q * 10 % Phi_MOD; // Phi_MOD 是MOD的欧拉函数值
            q = (q + s[i] - '0');
            if (q >= Phi_MOD) q -= Phi_MOD;
            if (q >= (int)n) zero = true;
        }
        return { p,q,zero };
    }
    vector<int> power(vector<int> a, string s) { // 多项式快速幂 a^s O(nlogn)
        int n = a.size();
        auto [p, q, zero] = power_reduction(s, (int)a.size()); // 不需要降幂的话可以省去这部分
        if (a[0] == 1) {
            auto res = ln(a);
            while ((int)res.size() > n) res.pop_back();
            for (auto& i : res) {
                i = mul(p, i);
            }
            res = exp(res);
            return res;
        } else {
            int mn = -1;
            vector<int> copy_a;
            for (int i = 0; i < (int)a.size(); i++) {
                if (a[i]) {
                    mn = i;
                    break;
                }
            }
            if ((mn == -1) || (mn && (zero || (1LL * mn * p > n)))) { // a中所有元素都是0 或 偏移过大
                return vector<int>(n, 0);
            }
            int inverse_amin = exgcd(a[mn]);
```

```
            for (int i = mn; i < n; i++) {
                copy_a.emplace_back(mul(a[i], inverse_amin));
            }
            copy_a = ln(copy_a);
            while ((int)copy_a.size() > n) copy_a.pop_back();
            for (auto& i : copy_a) {
                i = mul(i, p);
            }
            copy_a = exp(copy_a);
            vector<int> res(n, 0);
            // shift是偏移量 power_k 是a_min^q(q是扩展欧拉定理降出来的幂次)
            int shift = mn * p, power_k = powmod(a[mn], q);
            for (int i = 0; i + shift < n; i++) {
                res[i + shift] = mul(copy_a[i], power_k);
            }
            return res;
        }
    }
    vector<long long> sub_convolution(vector<int> a, vector<int> b) { // 减法卷积 只保留非负次项
        int n = b.size();
        reverse(b.begin(), b.end());
        auto res = multiply(a, b);
        return vector<long long>(res.begin() + n - 1, res.end());
    }
    int bostan_mori(vector<int> p, vector<int> q, long long n) { // [x^n]p(x)/q(x) O(2/3dlog(d)log(n+1)) d是多项式度数
        int i;
        for (; n; n >>= 1) {
            auto r = q;
            for (i = 1; i < r.size(); i += 2) {
                r[i] = MOD - r[i];
            }
            p = mod_multiply(p, r);
            q = mod_multiply(q, r);
            for (i = (n & 1); i < p.size(); i += 2) {
                p[i / 2] = p[i];
            }
            p.resize(i / 2);
            for (i = 0; i < q.size(); i += 2) {
                q[i / 2] = q[i];
            }
            q.resize(i / 2);
```

```
        }
        return p[0];
    }
};
```

2 数论

2.1 简单的防爆模板

2.1.1 类型 1

```
namespace SimpleMod {
    constexpr int MOD = (int)1e9 + 7;
    inline int norm(long long a) { return (a % MOD + MOD) % MOD; }
    inline int add(int a, int b) { return a + b >= MOD ? a + b - MOD : a + b
        ; }
    inline int sub(int a, int b) { return a - b < 0 ? a - b + MOD : a - b; }
    inline int mul(int a, int b) { return (int)((long long)a * b % MOD); }
    inline int powmod(int a, long long b) {
        int res = 1;
        while (b > 0) {
            if (b & 1) res = mul(res, a);
            a = mul(a, a);
            b >>= 1;
        }
        return res;
    }
    inline int inv(int a) {
        a %= MOD;
        if (a < 0) a += MOD;
        int b = MOD, u = 0, v = 1;
        while (a) {
            int t = b / a;
            b -= t * a; swap(a, b);
            u -= t * v; swap(u, v);
        }
        assert(b == 1);
        if (u < 0) u += MOD;
        return u;
    }
}
```

2.1.2 类型 2

```cpp
// copy from jiangly
constexpr int P = 1e9 + 7;
// assume -P <= x < 2P
int norm(int x) {
    if (x < 0) {
        x += P;
    }
    if (x >= P) {
        x -= P;
    }
    return x;
}
template<class T>
T power(T a, int b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    int val() const {
        return x;
    }
    Z operator-() const {
        return Z(norm(P - x));
    }
    Z inv() const {
        assert(x != 0);
        return power(*this, P - 2);
    }
    Z& operator*=(const Z& rhs) {
        x = 1LL * x * rhs.x % P;
        return *this;
    }
    Z& operator+=(const Z& rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
```

```cpp
    Z& operator-=(const Z& rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    Z& operator/=(const Z& rhs) {
        return *this *= rhs.inv();
    }
    friend Z operator*(const Z& lhs, const Z& rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend Z operator+(const Z& lhs, const Z& rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend Z operator-(const Z& lhs, const Z& rhs) {
        Z res = lhs;
        res -= rhs;
        return res;
    }
    friend Z operator/(const Z& lhs, const Z& rhs) {
        Z res = lhs;
        res /= rhs;
        return res;
    }
};
```

2.2 筛法

2.2.1 线性素数筛

```cpp
vector<bool> isPrime; // true 表示非素数  false 表示是素数
vector<int> prime; // 保存素数
int sieve(int n) {
    isPrime.resize(n + 1, false);
    isPrime[0] = isPrime[1] = true;
    for (int i = 2; i <= n; i++) {
        if (!isPrime[i]) prime.emplace_back(i);
        for (int j = 0; j < (int)prime.size() && prime[j] * i <= n; j++) {
            isPrime[prime[j] * i] = true;
            if (!(i % prime[j])) break;
        }
```

```
    }
    return (int)prime.size();
}
```

2.2.2 线性欧拉函数筛

```
bool is_prime[SIZE];
int prime[SIZE], phi[SIZE]; // phi[i] 表示 i 的欧拉函数值
int Phi(int n) { // 线性筛素数的同时线性求欧拉函数
    phi[1] = 1; is_prime[1] = true;
    int p = 0;
    for (int i = 2; i <= n; i++) {
        if (!is_prime[i]) prime[p++] = i, phi[i] = i - 1;
        for (int j = 0; j < p && prime[j] * i <= n; j++) {
            is_prime[prime[j] * i] = true;
            if (!(i % prime[j])) {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
    return p;
}
```

2.2.3 线性约数个数函数筛

```
bool is_prime[SIZE];
int prime[SIZE], d[SIZE], num[SIZE]; // d[i] 表示 i 的因子数  num[i] 表示 i
    的最小质因子出现次数
int getFactors(int n) { // 线性筛因子数
    d[1] = 1; is_prime[1] = true;
    int p = 0;
    for (int i = 2; i <= n; i++) {
        if (!is_prime[i]) prime[p++] = i, d[i] = 2, num[i] = 1;
        for (int j = 0; j < p && prime[j] * i <= n; j++) {
            is_prime[prime[j] * i] = true;
            if (!(i % prime[j])) {
                num[i * prime[j]] = num[i] + 1;
                d[i * prime[j]] = d[i] / num[i * prime[j]] * (num[i * prime[
                    j]] + 1);
                break;
            }
```

```
            num[i * prime[j]] = 1;
            d[i * prime[j]] = d[i] + d[i];
        }
    }
    return p;
}
```

2.2.4 线性素因子个数函数筛

```
bool is_prime[SIZE];
int prime[SIZE], num[SIZE]; // num[i] 表示 i 的质因子数
int getPrimeFactors(int n) { // 线性筛质因子数
    is_prime[1] = true;
    int p = 0;
    for (int i = 2; i <= n; i++) {
        if (!is_prime[i]) prime[p++] = i, num[i] = 1;
        for (int j = 0; j < p && prime[j] * i <= n; j++) {
            is_prime[prime[j] * i] = true;
            if (!(i % prime[j])) {
                num[i * prime[j]] = num[i];
                break;
            }
            num[i * prime[j]] = num[i] + 1;
        }
    }
    return p;
}
```

2.2.5 线性约数和函数筛

```
bool is_prime[SIZE];
int prime[SIZE], f[SIZE], g[SIZE]; // f[i] 表示 i 的约数和
int getSigma(int n) {
    g[1] = f[1] = 1; is_prime[1] = true;
    int p = 0;
    for (int i = 2; i <= n; i++) {
        if (!is_prime[i]) prime[p++] = i, f[i] = g[i] = i + 1;
        for (int j = 0; j < p && prime[j] * i <= n; j++) {
            is_prime[prime[j] * i] = true;
            if (!(i % prime[j])) {
                g[i * prime[j]] = g[i] * prime[j] + 1;
                f[i * prime[j]] = f[i] / g[i] * g[i * prime[j]];
                break;
```

```
            }
            f[i * prime[j]] = f[i] * f[prime[j]];
            g[i * prime[j]] = 1 + prime[j];
        }
    }
    return p;
}
```

2.2.6 线性莫比乌斯函数筛

```
bool is_prime[SIZE];
int prime[SIZE], mu[SIZE]; // mu[i] 表示 i 的莫比乌斯函数值
int getMu(int n) { // 线性筛莫比乌斯函数
    mu[1] = 1; is_prime[1] = true;
    int p = 0;
    for (int i = 2; i <= n; i++) {
        if (!is_prime[i]) prime[p++] = i, mu[i] = -1;
        for (int j = 0; j < p && prime[j] * i <= n; j++) {
            is_prime[prime[j] * i] = true;
            if (!(i % prime[j])) {
                mu[i * prime[j]] = 0;
                break;
            }
            mu[i * prime[j]] = -mu[i];
        }
    }
    return p;
}
```

2.3 扩展欧几里得

2.3.1 线性同余方程最小非负整数解

exgcd 求 $ax+by=c$ 的最小非负整数解详解：

1. 求出 a,b 的最大公约数 $g=\gcd(a,b)$ ，根据裴蜀定理检查是否满足 $c\%g=0$ ，不满足则无解；
2. 调整系数 a,b,c 为 $a'=\frac{a}{g},b'=\frac{b}{g},c'=\frac{c}{g}$ ，这是因为 $ax+by=c$ 和 $a'x+b'y=c'$ 是完全等价的；
3. 实际上 exgcd 求解的方程是 $a'x+b'y=1$ ，求解前需要注意让系数 $a',b'\geq 0$ （举个例子，如果系数 b' 原本 <0 ，我们可以翻转 b' 的符号然后令解 (x,y) 为 $(x,-y)$ ，但是求解的时候要把 y 翻回来）；
4. 我们通过 exgcd 求出一组解 (x_0,y_0) ，这组解满足 $a'x_0+b'y_0=1$ ，为了使解合法我们需要令 $x_0=c'x_0,y_0=c'y_0$ ，于是有 $a'(c'x_0)+b'(c'y_0)=c''$ ；
5. 考虑到 $a'x_0+b'y_0=1$ 等价于同余方程 $a'x_0\equiv 1 \pmod{b'}$ ，因此为了求出最小非负整数解，我们最后还需要对 b' 取模；

6. 最后注意特判 $c' = 0$ 的情况，如果要求解 y 且系数 b 发生了翻转，将其翻转回来。

```
long long exgcd(long long a, long long b, long long& x, long long& y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    long long g = exgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return g;
}

ll x, y; // 最小非负整数解
bool solve(ll a, ll b, ll c) { // ax+by=c
    ll g = gcd(a, b);
    if (c % g) return false;
    a /= g, b /= g, c /= g;
    bool flag = false;
    if (b < 0) b = -b, flag = true;
    exgcd(a, b, x, y);
    x = (x * c % b + b) % b;
    if (flag) b = -b;
    y = (c - a * x) / b;
    if (!c) x = y = 0; // ax+by=0
    return true;
}
```

2.4 欧拉定理

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(p)}, & \gcd(a,\, p) = 1 \\ a^b, & \gcd(a,\, p) \neq 1,\, b < \varphi(p) \\ a^{b \bmod \varphi(p) + \varphi(p)}, & \gcd(a,\, p) \neq 1,\, b \geq \varphi(p) \end{cases} \pmod p$$

2.5 欧拉函数

2.5.1 暴力单点查询

```
long long phi(long long n) { // O(sqrt(N))
    int m = int(sqrt(n + 0.5));
    long long ans = n;
    for (int i = 2; i <= m; i++) {
        if (n % i == 0) {
            ans = ans / i * (i - 1);
            while (n % i == 0) n /= i;
```

```
        }
    }
    if (n > 1) ans = ans / n * (n - 1);
    return ans;
}
```

2.5.2 预处理单点查询

```
vector<int> prime;  // 求 n 的欧拉函数需要先把 <= ceil(sqrt(n)) 的素数筛出
long long phi(long long n) { // O(sqrt(N)/log(N))
    long long res = n;
    for (int i = 0; i < (int)prime.size(); i++) {
        long long p = prime[i];
        if (p * p > n) break;
        if (n % p == 0) {
            res = res / p * (p - 1);
            while (n % p == 0) n /= p;
        }
    }
    if (n > 1) res = res / n * (n - 1);
    return res;
}
```

2.6 中国剩余定理

2.6.1 CRT

```
// 求解形如 x = ci (mod mi) 的线性方程组 (mi, mj)必须两两互质
long long CRT(vector<long long>& c, vector<long long>& m) {
    long long M = m[0], ans = 0;
    for (int i = 1; i < (int)m.size(); ++i) M *= m[i];
    for (int i = 0; i < (int)m.size(); ++i) { // Mi * ti * ci
        long long mi = M / m[i];
        long long ti = inv(mi, m[i]); // mi 模 m[i] 的逆元
        ans = (ans + mi * ti % M * c[i] % M) % M;
    }
    ans = (ans + M) % M; // 返回模 M 意义下的唯一解
    return ans;
}
```

2.6.2 EXCRT

```
long long exgcd(long long a, long long b, long long& x, long long& y) {
```

```cpp
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    long long g = exgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return g;
}

long long mulmod(long long x, long long y, const long long z) { // x * y % z 防爆
    return (x * y - (long long)(((long double)x * y + 0.5) / (long double)z) * z + z) % z;
}

// 求解形如 x = ci (mod mi) 的线性方程组
long long EXCRT(vector<long long>& c, vector<long long>& m) {
    long long M = m[0], ans = c[0];
    for (int i = 1; i < (int)m.size(); ++i) { // M * x - mi * y = ci - C
        long long x, y, C = ((c[i] - ans) % m[i] + m[i]) % m[i]; // ci - C
        long long G = exgcd(M, m[i], x, y);
        if (C % G) return -1; // 无解
        long long P = m[i] / G;
        x = mulmod(C / G, x, P); // 防爆求最小正整数解 x
        ans += x * M;
        M *= P; // LCM(M, mi)
        ans = (ans % M + M) % M;
    }
    return ans;
}
```

2.7 BSGS

2.8 迪利克雷卷积

$$g(1)S(n) = \sum_{i=1}^{n}(f*g)(i) - \sum_{i=2}^{n} g(i)S(\lfloor\frac{n}{i}\rfloor)$$

2.9 杜教筛

$$(f*g)(n) = \sum_{d|n} f(d)g(\frac{n}{d}) = \sum_{xy=n} f(x)g(y)$$

2.10 Berlekamp Massey

```cpp
namespace Berlekamp_Massey {
    typedef long long ll;
    constexpr ll MOD = 1e9 + 7;
    constexpr int N = 10010;
    ll res[N], base[N], _c[N], _md[N];
    vector<int> Md;
    ll powmod(ll a, ll b) {
        ll res = 1;
        while (b > 0) {
            if (b & 1) res = res * a % MOD;
            a = a * a % MOD;
            b >>= 1;
        }
        return res;
    }
    void mul(ll* a, ll* b, int k) {
        for (int i = 0; i < k + k; i++)
            _c[i] = 0;
        for (int i = 0; i < k; i++) {
            if (!a[i]) continue;
            for (int j = 0; j < k; j++) {
                _c[i + j] = (_c[i + j] + a[i] * b[j]) % MOD;
            }
        }
        for (int i = k + k - 1; i >= k; i--) {
            if (!_c[i]) continue;
            for (int j = 0; j < Md.size(); j++) {
                _c[i - k + Md[j]] = (_c[i - k + Md[j]] - _c[i] * _md[Md[j]])
                    % MOD;
            }
        }
        for (int i = 0; i < k; i++)
            a[i] = _c[i];
    }
    int solve(ll n, vector<int> a, vector<int> b) { //a系数 b初值 b[n+1]=a
        [0]*b[n]+...
        //printf("%d\n", (int)b.size());
        //for (int i = 0; i < b.size(); i++)
        //    printf("b[%d] = %d\n", i, b[i]);
        //printf("%d\n", (int)a.size());
        //printf("b[n]");
        //for (int i = 0; i < a.size(); i++) {
```

```
        //     if (!i)putchar('='); else putchar('+');
        //     printf("%d*b[n-%d]", a[i], i + 1);
        //}
        //puts("");
        ll ans = 0, pnt = 0;
        int k = a.size();
        for (int i = 0; i < k; i++) {
            _md[k - 1 - i] = -a[i];
        }
        _md[k] = 1;
        Md.clear();
        for (int i = 0; i < k; i++) {
            if (_md[i]) {
                Md.push_back(i);
            }
            res[i] = base[i] = 0;
        }
        res[0] = 1;
        while ((1LL << pnt) <= n) pnt++;
        for (int p = pnt; p >= 0; p--) {
            mul(res, res, k);
            if ((n >> p) & 1) {
                for (int i = k - 1; i >= 0; i--)
                    res[i + 1] = res[i];
                res[0] = 0;
                for (int j = 0; j < Md.size(); j++) {
                    res[Md[j]] = (res[Md[j]] - res[k] * _md[Md[j]]) % MOD;
                }
            }
        }
        for (int i = 0; i < k; i++)
            ans = (ans + res[i] * b[i]) % MOD;
        return (ans < 0 ? ans + MOD : ans);
    }
    vector<int> BM(vector<int> s) { // O(n^2)
        vector<int> C(1, 1), B(1, 1);
        int L = 0, m = 1, b = 1;
        for (int n = 0; n < (int)s.size(); n++) {
            ll d = 0;
            for (int i = 0; i <= L; i++)
                d = (d + (ll)C[i] * s[n - i]) % MOD;
            if (!d) {
                ++m;
```

```
            } else if (2 * L <= n) {
                auto T = C;
                ll c = MOD - d * powmod(b, MOD - 2) % MOD;
                while (C.size() < B.size() + m)
                    C.push_back(0);
                for (int i = 0; i < B.size(); i++)
                    C[i + m] = (C[i + m] + c * B[i]) % MOD;
                L = n + 1 - L; B = T; b = d; m = 1;
            } else {
                ll c = MOD - d * powmod(b, MOD - 2) % MOD;
                while (C.size() < B.size() + m) C.push_back(0);
                for (int i = 0; i < B.size(); i++) {
                    C[i + m] = (C[i + m] + c * B[i]) % MOD;
                }
                ++m;
            }
        }
        return C;
    }
    int work(vector<int> a, ll n) { // 这里的n不是数组大小 是求数列第n项的值
        vector<int> c = BM(a);        // 求第n项的复杂度为 O(k^2logn) k是递推数列大小
        c.erase(c.begin());
        for (int i = 0; i < c.size(); i++)
            c[i] = (MOD - c[i]) % MOD;
        return solve(n, c, vector<int>(a.begin(), a.begin() + (int)c.size()));
    }
}
```

3 线性代数

3.1 矩阵

3.2 高斯-约旦消元法

```
/*
 * 高斯-约旦消元法
 * 可以修改为解异或方程组 修改策略为
 * a+b -> a^b
 * a-b -> a^b
 * a*b -> a&b
 * a/b -> a*(b==1)
 * */
```

```cpp
constexpr double eps = 1e-7;
double a[SIZE][SIZE], ans[SIZE];
void gauss(int n) {
    vector<bool> vis(n, false);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (vis[j]) continue;
            if (fabs(a[j][i]) > eps) {
                vis[i] = true;
                for (int k = 0; k <= n; k++) swap(a[i][k], a[j][k]);
                break;
            }
        }
        if (fabs(a[i][i]) < eps) continue;
        for (int j = 0; j <= n; j++) {
            if (i != j && fabs(a[j][i]) > eps) {
                double res = a[j][i] / a[i][i];
                for (int k = 0; k <= n; k++) a[j][k] -= a[i][k] * res;
            }
        }
    }
}

int check(int n) { // 解系检测
    int status = 1;
    for (int i = n - 1; i >= 0; i--) {
        if (fabs(a[i][i]) < eps && fabs(a[i][n]) > eps) return -1; // 无解
        if (fabs(a[i][i]) < eps && fabs(a[i][n]) < eps) status = 0; // 无穷解
        ans[i] = a[i][n] / a[i][i];
        if (fabs(ans[i]) < eps) ans[i] = 0;
    }
    return status; // 唯一解或无穷解
}
```

3.3 高斯消元法-bitset

```cpp
constexpr int SIZE = 1001;
bitset<SIZE> a[SIZE];
int ans[SIZE];
void gauss(int n) { // bitset版高斯消元 用于求解异或线性方程组
    bitset<SIZE> vis;
    for (int i = 0; i < n; i++) {
```

```
        for (int j = 0; j < n; j++) {
            if (vis[j]) continue;
            if (a[j][i]) {
                vis.set(i);
                swap(a[i], a[j]);
                break;
            }
        }
        if (!a[i][i]) continue;
        for (int j = 0; j <= n; j++) {
            if (i != j && (a[j][i] & a[i][i])) {
                a[j] ^= a[i];
            }
        }
    }
}
```

3.4 线性基

```
struct linearBasis {
    /* 线性基性质:
     * 1.若a[i]!=0 (即主元i存在)
     *   则线性基中只有a[i]的第i位是1 (只存在一个主元)
     *   且此时a[i]的最高位就是第i位
     * 2.将数组a插入线性基 假设有|B|个元素成功插入
     *   则数组a中每个不同的子集异或和都出现 2^(n-|B|) 次
     * */
    static const int MAXL = 60;
    long long a[MAXL + 1];
    int id[MAXL + 1];
    int zero;
    /* 0的标志位 =1则表示0可以被线性基表示出来
     * 求第k大元素时 需要注意题意中线性基为空时是否可以表示0
     * 默认不可以表示 有必要时进行修改即可
     * */
    linearBasis() {
        zero = 0;
        fill(a, a + MAXL + 1, 0);
    }
    long long& operator[](int k) { return a[k]; }
    bool insert(long long x) {
        for (int j = MAXL; ~j; j--) {
            if (!(x & (1LL << j))) { // 如果 x 的第 j 位为 0, 则跳过
```

```
                continue;
            }
            if (a[j]) { // 如果 a[j] != 0，则用 a[j] 消去 x 的第 j 位上的 1
                x ^= a[j];
            } else { // 找到插入位置
                for (int k = 0; k < j; k++) {
                    if (x & (1LL << k)) { // 如果x存在某个低位线性基的主元k则消去
                        x ^= a[k];
                    }
                }
                for (int k = j + 1; k <= MAXL; k++) {
                    if (a[k] & (1LL << j)) { // 如果某个高位线性基存在主元j则消去
                        a[k] ^= x;
                    }
                }
                a[j] = x;
                return true;
            }
        }
        zero = 1;
        return false;
    }
    long long query_max() { // 最大值
        long long res = 0;
        for (int i = MAXL; ~i; i--) {
            res ^= a[i];
        }
        return res;
    }
    long long query_max(long long x) { // 线性基异或x的最大值
        for (int i = MAXL; ~i; i--) {
            if ((x ^ a[i]) > x) {
                x ^= a[i];
            }
        }
        return x;
    }
    long long query_min() { // 最小值
        for (int i = 0; i < MAXL; i++) {
            if (a[i]) {
                return a[i];
```

```
            }
        }
        return -1; // 线性基为空
    }
    long long query_min(long long x) { // 线性基异或x的最小值
        for (int i = MAXL; ~i; i--) {
            if ((x ^ a[i]) < x) {
                x ^= a[i];
            }
        }
        return x;
    }
    int count(long long x) { // 元素 x 能否被线性基内元素表示
        int res = 0;
        vector<long long> b(MAXL + 1);
        for (int i = 0; i <= MAXL; i++) {
            b[i] = a[i];
        }
        res = this->insert(x);
        for (int i = 0; i <= MAXL; i++) {
            a[i] = b[i];
        }
        return !res; // 成功插入则无法表示 失败则可以表示
    }
    int size() { // 线性基有效元素数量
        int res = 0;
        for (int i = 0; i <= MAXL; i++) {
            if (a[i]) {
                res++;
            }
        }
        return res;
    }
    long long kth_element(long long k) { // 第k大元素
        vector<long long> b;
        for (int i = 0; i <= MAXL; i++) {
            if (a[i]) {
                b.push_back(a[i]);
            }
        }
        if (zero) {
            if (--k == 0) {
                return 0;
```

```cpp
            }
        }
        if (k >= (1LL << this->size())) { // k超过了线性基可以表示的最大数量
            return -1;
        }
        long long res = 0;
        for (int i = 0; i <= MAXL; i++) {
            if (k & (1LL << i)) {
                res ^= b[i];
            }
        }
        return res;
    }
    long long rank(long long x) { // 元素x在线性基内的排名（默认不考虑0）
        vector<long long> b;
        for (int i = 0; i <= MAXL; i++) {
            if (a[i]) {
                b.push_back(1LL << i);
            }
        }
        long long res = 0;
        for (int i = 0; i < (int)b.size(); i++) {
            if (x & b[i]) {
                res |= (1LL << i);
            }
        }
        return res;
    }
    void clear() {
        zero = 0;
        fill(a, a + MAXL + 1, 0);
    }
};
```

3.5 矩阵树定理

```cpp
/*
 * 矩阵树定理
 * 有向图: 若 u->v 有一条权值为 w 的边 基尔霍夫矩阵 a[v][v] += w, a[v][u] -= w
 * 生成树数量为除去 根所在行和列 后的n-1阶行列式的值
 * 无向图: 若 u->v 有一条权值为 w 的边 基尔霍夫矩阵 a[v][v] += w, a[v][u] -= w, a[u][u] += w, a[u][v] -= w
```

```
 * 生成树数量为除去 任意一行和列 后的n-1阶行列式的值
 * 无权图则边权默认为1
 * */
typedef long long ll;
typedef unsigned long long u64;
int a[SIZE][SIZE];
int gauss(int a[][SIZE], int n) { // 任意模数求行列式 O(n^2(n + log(mod)))
    int ans = 1;
    for (int i = 1; i <= n; i++) {
        int* x = 0, * y = 0;
        for (int j = i; j <= n; j++) {
            if (a[j][i] && (x == NULL || a[j][i] < x[i])) {
                x = a[j];
            }
        }
        if (x == 0) {
            return 0;
        }
        for (int j = i; j <= n; j++) {
            if (a[j] != x && a[j][i]) {
                y = a[j];
                for (;;) {
                    int v = md - y[i] / x[i], k = i;
                    for (; k + 3 <= n; k += 4) {
                        y[k + 0] = (y[k + 0] + u64(x[k + 0]) * v) % md;
                        y[k + 1] = (y[k + 1] + u64(x[k + 1]) * v) % md;
                        y[k + 2] = (y[k + 2] + u64(x[k + 2]) * v) % md;
                        y[k + 3] = (y[k + 3] + u64(x[k + 3]) * v) % md;
                    }
                    for (; k <= n; ++k) {
                        y[k] = (y[k] + u64(x[k]) * v) % md;
                    }
                    if (!y[i]) break;
                    swap(x, y);
                }
            }
        }
        if (x != a[i]) {
            for (int k = i; k <= n; k++) {
                swap(x[k], a[i][k]);
            }
            ans = md - ans;
        }
```

```
        ans = 1LL * ans * a[i][i] % md;
    }
    return ans;
}
```

3.6 LGV 引理

4 组合数学

4.1 组合数预处理

```
namespace BinomialCoefficient {
    vector<int> fac, ifac, iv;
    // 组合数预处理 option=1则还会预处理线性逆元
    void prepareFactorials(int maximum = 1000000, int option = 0) {
        fac.assign(maximum + 1, 0);
        ifac.assign(maximum + 1, 0);
        fac[0] = ifac[0] = 1;
        if (option) { // O(3n)
            iv.assign(maximum + 1, 1);
            for (int p = 2; p * p <= MOD; p++)
                assert(MOD % p != 0);
            for (int i = 2; i <= maximum; i++)
                iv[i] = mul(iv[MOD % i], (MOD - MOD / i));
            for (int i = 1; i <= maximum; i++) {
                fac[i] = mul(i, fac[i - 1]);
                ifac[i] = mul(iv[i], ifac[i - 1]);
            }
        } else { // O(2n + log(MOD))
            for (int i = 1; i <= maximum; i++)
                fac[i] = mul(fac[i - 1], i);
            ifac[maximum] = inv(fac[maximum]);
            for (int i = maximum; i; i--)
                ifac[i - 1] = mul(ifac[i], i);
        }
    }
    inline int binom(int n, int m) {
        if (n < m || n < 0 || m < 0) return 0;
        return mul(fac[n], mul(ifac[m], ifac[n - m]));
    }
}
```

4.2 卢卡斯定理

4.3 小球盒子模型

设有 n 个球，k 个盒子：

1. 球之间互不相同，盒子之间互不相同，可以空盒

   根据乘法原理，答案就是 k^n 。

2. 球之间互不相同，盒子之间互不相同，每个盒子至多装一个球

   相当于每个球找一个没有被选过的盒子放进去，答案是 $k^{\underline{n}}$ ，即 $k(k-1)\cdots(k-n+1)$ 。

3. 球之间互不相同，盒子之间互不相同，每个盒子至少装一个球

   可以先把盒子视为相同：n 个球放进 k 个相同盒子、不能空盒，这就是第二类斯特林数 S_n^k 的定义。最后由于盒子不同，再乘上一个排列数，因此答案就是 $k!S_n^k$ 。

4. 球之间互不相同，盒子全部相同，可以空盒

   枚举非空盒子数量，相当于第二类斯特林数一行求和：$\sum_{i=1}^{k} S_n^i$ 。

5. 球之间互不相同，盒子全部相同，每个盒子至多装一个球

   因为盒子相同，不论怎么放都是一样的，答案是 $[n \le k]$（这是一个布尔运算式，若 $n \le k$ 成立则取 1 ，否则 0）。

6. 球之间互不相同，盒子全部相同，每个盒子至少装一个球

   就是第二类斯特林数 S_n^k 。

7. 球全部相同，盒子之间互不相同，可以空盒

   隔板法经典应用，$n+k-1$ 个球选 $k-1$ 个板，因此答案是 $\binom{n+k-1}{k-1}$ 。

8. 球全部相同，盒子之间互不相同，每个盒子至多装一个球

   盒子不同，相当于要选出 n 个盒子装球，因此答案是 $\binom{n}{k}$ 。

9. 球全部相同，盒子之间互不相同，每个盒子至少装一个球

   隔板法经典应用，$n-1$ 个空隙选 $k-1$ 个插板（可以看作是情况 7 时每个盒子里都预先加入一个球），因此答案是 $\binom{n-1}{k-1}$ 。

10. 球全部相同，盒子全部相同，可以空盒

    定义划分数 $p_{n,k}$ 表示将自然数 n 拆成 k 份的方案数，那么本例的结论就是 $p_{n,k}$ 。

    这个问题有一个经典递推式：$p(n,k) = p(n,k-1) + p(n-k,k)$ 。意义是将 j 个自然数 +1 或者加入一个 0 。下面给出一个代码实现：

```cpp
p[0][0] = 1;
for (int i = 1; i <= n; i++) {
    p[0][i] = 1;
    for (int j = 1; j <= m; j++) {
        if (i >= j) {
            p[i][j] = add(p[i][j - 1], p[i - j][j]);
        } else {
```

```
                p[i][j] = p[i][j - 1];
            }
        }
}
```

11. 球全部相同，盒子全部相同，每个盒子至多装一个球

    和情况 5 一致，就是 $[n \le k]$ 。

12. 球全部相同，盒子全部相同，每个盒子至少装一个球

    显然也是一个划分数：$p_{n-k,k}$ 。

4.4 斯特林数

4.4.1 第一类斯特林数

第一类斯特林数 $\begin{bmatrix} n \\ k \end{bmatrix}$ 表示将 n 个不同元素划分入 k 个非空圆排列的方案数。

$$\begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} + (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix}$$

边界是 $\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1$ 。

第一类斯特林数三角形，从 s(1, 1) 开始：

1									
1	1								
2	3	1							
6	11	6	1						
24	50	35	10	1					
120	274	225	85	15	1				
720	1764	1624	735	175	21	1			
5040	13068	13132	6769	1960	322	28	1		
40320	109584	118124	67284	22449	4536	546	36	1	
362880	1026576	1172700	723680	269325	63273	9450	870	45	1

4.4.2 第二类斯特林数

第二类斯特林数 $\begin{Bmatrix} n \\ k \end{Bmatrix}$ 表示将 n 个不同元素划分为 k 个非空子集的方案数。

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k \begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

边界是 $\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = 1$ 。

基于容斥原理的递推方法：

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k-i)^n$$

第二类斯特林数三角形，从 S(1, 1) 开始：

1									
1	1								
1	3	1							
1	7	6	1						
1	15	25	10	1					
1	31	90	65	15	1				
1	63	301	350	140	21	1			
1	127	966	1701	1050	266	28	1		
1	255	3025	7770	6951	2646	462	36	1	
1	511	9330	34105	42525	22827	5880	750	45	1

5 博弈论

6 图论

6.1 并查集

```cpp
struct dsu {
private:
    // number of nodes
    int n;
    // root node: -1 * component size
    // otherwise: parent
    std::vector<int> pa;
public:
    dsu(int n_ = 0) : n(n_), pa(n_, -1) {}
    // find node x's parent
    int find(int x) {
        return pa[x] < 0 ? x : pa[x] = find(pa[x]);
    }
    // merge node x and node y
    // if x and y had already in the same component, return false, otherwise
        return true
    // Implement (union by size) + (path compression)
    bool unite(int x, int y) {
        int xr = find(x), yr = find(y);
        if (xr != yr) {
            if (-pa[xr] < -pa[yr]) std::swap(xr, yr);
            pa[xr] += pa[yr];
            pa[yr] = xr; // y -> x
            return true;
        }
```

```
        return false;
    }
    // size of the connected component that contains the vertex x
    int size(int x) {
        return -pa[find(x)];
    }
};
```

6.2 最小树形图

```
namespace ZL {
    // a 尽量开大，之后的边都塞在这个里面
    const int N = 100010, M = 100010, inf = 1e9;
    struct edge {
        int u, v, w, use, id;
        edge(int u_ = 0, int v_ = 0, int w_ = 0, int use_ = 0, int id_ = 0)
            : u(u_), v(v_), w(w_), use(use_), id(id_) {}
    }b[M], a[2000100];
    int n, m, ans, pre[N], id[N], vis[N], root, In[N], h[N], len, way[M];
    // 从root 出发能到达每一个点的最小树形图
    // 先调用init 然后把边add 进去，需要方案就getway, way[i] 为1 表示使用
    // 最小值保存在ans
    void init(int _n, int _root) { // 点数 根节点
        n = _n; m = 0; b[0].w = inf; root = _root;
    }
    void add(int u, int v, int w) {
        m++;
        b[m] = edge(u, v, w, 0, m);
        a[m] = b[m];
    }
    int work() {
        len = m;
        for (;;) {
            for (int i = 1; i <= n; i++) { pre[i] = 0; In[i] = inf; id[i] =
                0; vis[i] = 0; h[i] = 0; }
            for (int i = 1; i <= m; i++) {
                if (b[i].u != b[i].v && b[i].w < In[b[i].v]) {
                    pre[b[i].v] = b[i].u; In[b[i].v] = b[i].w; h[b[i].v] = b
                        [i].id;
                }
            }
            for (int i = 1; i <= n; i++) if (pre[i] == 0 && i != root)
                return 0;
```

```
            int cnt = 0; In[root] = 0;
            for (int i = 1; i <= n; i++) {
                if (i != root) a[h[i]].use++; int now = i; ans += In[i];
                while (vis[now] == 0 && now != root) { vis[now] = i; now = pre[now]; }
                if (now != root && vis[now] == i) {
                    cnt++; int kk = now;
                    while (1) {
                        id[now] = cnt; now = pre[now];
                        if (now == kk) break;
                    }
                }
            }
            if (cnt == 0) return 1;
            for (int i = 1; i <= n; i++) if (id[i] == 0) id[i] = ++cnt;
            // 缩环，每一条接入的边都会茶包原来接入的那条边，所以要调整边权
            // 新加的边是u，茶包的边是v
            for (int i = 1; i <= m; i++) {
                int k1 = In[b[i].v], k2 = b[i].v;
                b[i].u = id[b[i].u];
                b[i].v = id[b[i].v];
                if (b[i].u != b[i].v) {
                    b[i].w -= k1; a[++len].u = b[i].id; a[len].v = h[k2]; b[i].id = len;
                }
            }
            n = cnt; root = id[root];
        }
        return 1;
    }
    void getway() {
        for (int i = 1; i <= m; i++) way[i] = 0;
        for (int i = len; i > m; i--) { a[a[i].u].use += a[i].use; a[a[i].v].use -= a[i].use; }
        for (int i = 1; i <= m; i++) way[i] = a[i].use;
    }
}
```

6.3 最近公共祖先

```
constexpr int SIZE = 200010;
constexpr int DEPTH = 21; // 最大深度 2^DEPTH - 1
int pa[SIZE][DEPTH], dep[SIZE];
```

```
vector<int> g[SIZE]; //邻接表
void dfs(int rt, int fin) { //预处理深度和祖先
    pa[rt][0] = fin;
    dep[rt] = dep[pa[rt][0]] + 1; //深度
    for (int i = 1; i < DEPTH; i++) { // rt 的 2^i 祖先等价于 rt 的 2^(i-1) 祖先 的 2^(i-1) 祖先
        pa[rt][i] = pa[pa[rt][i - 1]][i - 1];
    }
    int sz = g[rt].size();
    for (int i = 0; i < sz; ++i) {
        if (g[rt][i] == fin) continue;
        dfs(g[rt][i], rt);
    }
}

int LCA(int x, int y) {
    if (dep[x] > dep[y]) swap(x, y);
    int dif = dep[y] - dep[x];
    for (int j = 0; dif; ++j, dif >>= 1) {
        if (dif & 1) {
            y = pa[y][j]; //先跳到同一高度
        }
    }
    if (y == x) return x;
    for (int j = DEPTH - 1; j >= 0 && y != x; j--) { //从底往上跳
        if (pa[x][j] != pa[y][j]) { //如果当前祖先不相等 我们就需要更新
            x = pa[x][j];
            y = pa[y][j];
        }
    }
    return pa[x][0];
}
```

6.4 强连通分量

```
namespace SCC {
    // Compressed Sparse Row
    template <class E> struct csr {
        std::vector<int> start;
        std::vector<E> elist;
        explicit csr(int n, const std::vector<std::pair<int, E>>& edges)
            : start(n + 1), elist(edges.size()) {
            for (auto e : edges) {
```

```cpp
            start[e.first + 1]++;
        }
        for (int i = 1; i <= n; i++) {
            start[i] += start[i - 1];
        }
        auto counter = start;
        for (auto e : edges) {
            elist[counter[e.first]++] = e.second;
        }
    }
};

struct scc_graph {
public:
    explicit scc_graph(int n) : _n(n) {}

    int num_vertices() { return _n; }

    void add_edge(int from, int to) { edges.push_back({ from, {to} }); }

    // @return pair of (# of scc, scc id)
    std::pair<int, std::vector<int>> scc_ids() {
        auto g = csr<edge>(_n, edges);
        int now_ord = 0, group_num = 0;
        std::vector<int> visited, low(_n), ord(_n, -1), ids(_n);
        visited.reserve(_n);
        auto dfs = [&](auto self, int v) -> void {
            low[v] = ord[v] = now_ord++;
            visited.push_back(v);
            for (int i = g.start[v]; i < g.start[v + 1]; i++) {
                auto to = g.elist[i].to;
                if (ord[to] == -1) {
                    self(self, to);
                    low[v] = std::min(low[v], low[to]);
                } else {
                    low[v] = std::min(low[v], ord[to]);
                }
            }
            if (low[v] == ord[v]) {
                while (true) {
                    int u = visited.back();
                    visited.pop_back();
                    ord[u] = _n;
```

```
                        ids[u] = group_num;
                        if (u == v) break;
                    }
                    group_num++;
                }
            };
            for (int i = 0; i < _n; i++) {
                if (ord[i] == -1) dfs(dfs, i);
            }
            for (auto& x : ids) {
                x = group_num - 1 - x;
            }
            return { group_num, ids };
        }

        // O(N + M)
        // It returns the list of the SCC in topological order.
        std::vector<std::vector<int>> scc() {
            auto ids = scc_ids();
            int group_num = ids.first;
            std::vector<int> counts(group_num);
            for (auto x : ids.second) counts[x]++;
            std::vector<std::vector<int>> groups(ids.first);
            for (int i = 0; i < group_num; i++) {
                groups[i].reserve(counts[i]);
            }
            for (int i = 0; i < _n; i++) {
                groups[ids.second[i]].push_back(i);
            }
            return groups;
        }

    private:
        int _n;
        struct edge {
            int to;
        };
        std::vector<std::pair<int, edge>> edges;
    };
}
```

6.5 最大流

```cpp
template <class T> struct simple_queue {
    std::vector<T> payload;
    int pos = 0;
    void reserve(int n) { payload.reserve(n); }
    int size() const { return int(payload.size()) - pos; }
    bool empty() const { return pos == int(payload.size()); }
    void push(const T& t) { payload.push_back(t); }
    T& front() { return payload[pos]; }
    void clear() {
        payload.clear();
        pos = 0;
    }
    void pop() { pos++; }
};

template <class Cap> struct mf_graph {
public:
    mf_graph() : _n(0) {}
    mf_graph(int n) : _n(n), g(n) {}

    // returns an integer k such that this is the k-th edge that is added.
    int add_edge(int from, int to, Cap cap) {
        assert(0 <= from && from < _n);
        assert(0 <= to && to < _n);
        assert(0 <= cap);
        int m = int(pos.size());
        pos.push_back({ from, int(g[from].size()) });
        int from_id = int(g[from].size());
        int to_id = int(g[to].size());
        if (from == to) to_id++;
        g[from].push_back(_edge{ to, to_id, cap });
        g[to].push_back(_edge{ from, from_id, 0 });
        return m;
    }

    struct edge {
        int from, to;
        Cap cap, flow;
    };

    edge get_edge(int i) {
        int m = int(pos.size());
        assert(0 <= i && i < m);
```

```
            auto _e = g[pos[i].first][pos[i].second];
            auto _re = g[_e.to][_e.rev];
            return edge{ pos[i].first, _e.to, _e.cap + _re.cap, _re.cap };
        }
        std::vector<edge> edges() {
            int m = int(pos.size());
            std::vector<edge> result;
            for (int i = 0; i < m; i++) {
                result.push_back(get_edge(i));
            }
            return result;
        }
        void change_edge(int i, Cap new_cap, Cap new_flow) {
            int m = int(pos.size());
            assert(0 <= i && i < m);
            assert(0 <= new_flow && new_flow <= new_cap);
            auto& _e = g[pos[i].first][pos[i].second];
            auto& _re = g[_e.to][_e.rev];
            _e.cap = new_cap - new_flow;
            _re.cap = new_flow;
        }

        // max flow from s to t
        // O(M*N^2) general
        // O(min(M*N^2/3, M^3/2)) if capacities of edges are 1
        Cap flow(int s, int t) {
            return flow(s, t, std::numeric_limits<Cap>::max());
        }
        Cap flow(int s, int t, Cap flow_limit) {
            assert(0 <= s && s < _n);
            assert(0 <= t && t < _n);
            assert(s != t);

            std::vector<int> level(_n), iter(_n);
            simple_queue<int> que;

            auto bfs = [&]() {
                std::fill(level.begin(), level.end(), -1);
                level[s] = 0;
                que.clear();
                que.push(s);
                while (!que.empty()) {
                    int v = que.front();
```

```
                que.pop();
                for (auto e : g[v]) {
                    if (e.cap == 0 || level[e.to] >= 0) continue;
                    level[e.to] = level[v] + 1;
                    if (e.to == t) return;
                    que.push(e.to);
                }
            }
        };
        auto dfs = [&](auto self, int v, Cap up) {
            if (v == s) return up;
            Cap res = 0;
            int level_v = level[v];
            for (int& i = iter[v]; i < int(g[v].size()); i++) {
                _edge& e = g[v][i];
                if (level_v <= level[e.to] || g[e.to][e.rev].cap == 0)
                    continue;
                Cap d =
                    self(self, e.to, std::min(up - res, g[e.to][e.rev].cap))
                        ;
                if (d <= 0) continue;
                g[v][i].cap += d;
                g[e.to][e.rev].cap -= d;
                res += d;
                if (res == up) break;
            }
            return res;
        };

        Cap flow = 0;
        while (flow < flow_limit) {
            bfs();
            if (level[t] == -1) break;
            std::fill(iter.begin(), iter.end(), 0);
            while (flow < flow_limit) {
                Cap f = dfs(dfs, t, flow_limit - flow);
                if (!f) break;
                flow += f;
            }
        }
        return flow;
    }

```

```cpp
    std::vector<bool> min_cut(int s) {
        std::vector<bool> visited(_n);
        simple_queue<int> que;
        que.push(s);
        while (!que.empty()) {
            int p = que.front();
            que.pop();
            visited[p] = true;
            for (auto e : g[p]) {
                if (e.cap && !visited[e.to]) {
                    visited[e.to] = true;
                    que.push(e.to);
                }
            }
        }
        return visited;
    }

private:
    int _n;
    struct _edge {
        int to, rev;
        Cap cap;
    };
    std::vector<std::pair<int, int>> pos;
    std::vector<std::vector<_edge>> g;
};
```

6.6 最小费用最大流

```cpp
/*
 * 费用流Cost常用类型的上限：int范围内 0 <= nx <= 2e9 + 1000,  long long范围内：0 <= nx <= 8e18 + 1000
 *
 * min_cost_slope() 函数返回的是一个分段函数F(x)（其中x代表流量上界，F(x)代表当前最大流量的最小费用）
 * 返回的vector是所有F(x)改变的点
 * 时间复杂度 O(f(N + M))log(N + M) f(N + M) 代表图的流量总和
 * */
namespace MCMF {
    template <class T> struct simple_queue {
        std::vector<T> payload;
        int pos = 0;
```

```cpp
        void reserve(int n) { payload.reserve(n); }
        int size() const { return int(payload.size()) - pos; }
        bool empty() const { return pos == int(payload.size()); }
        void push(const T& t) { payload.push_back(t); }
        T& front() { return payload[pos]; }
        void clear() {
            payload.clear();
            pos = 0;
        }
        void pop() { pos++; }
    };

    template <class E> struct csr {
        std::vector<int> start;
        std::vector<E> elist;
        explicit csr(int n, const std::vector<std::pair<int, E>>& edges)
            : start(n + 1), elist(edges.size()) {
            for (auto e : edges) {
                start[e.first + 1]++;
            }
            for (int i = 1; i <= n; i++) {
                start[i] += start[i - 1];
            }
            auto counter = start;
            for (auto e : edges) {
                elist[counter[e.first]++] = e.second;
            }
        }
    };

    template <class Cap, class Cost> struct mcf_graph {
    public:
        mcf_graph() {}
        explicit mcf_graph(int n) : _n(n) {}

        int add_edge(int from, int to, Cap cap, Cost cost) {
            assert(0 <= from && from < _n);
            assert(0 <= to && to < _n);
            assert(0 <= cap);
            assert(0 <= cost);
            int m = int(_edges.size());
            _edges.push_back({ from, to, cap, 0, cost });
            return m;
```

```
        }

        struct edge {
            int from, to;
            Cap cap, flow;
            Cost cost;
        };

        edge get_edge(int i) {
            int m = int(_edges.size());
            assert(0 <= i && i < m);
            return _edges[i];
        }
        std::vector<edge> edges() { return _edges; }

        std::pair<Cap, Cost> flow(int s, int t) {
            return flow(s, t, std::numeric_limits<Cap>::max());
        }
        std::pair<Cap, Cost> flow(int s, int t, Cap flow_limit) {
            return slope(s, t, flow_limit).back();
        }
        std::vector<std::pair<Cap, Cost>> slope(int s, int t) {
            return slope(s, t, std::numeric_limits<Cap>::max());
        }
        std::vector<std::pair<Cap, Cost>> slope(int s, int t, Cap flow_limit
            ) {
            assert(0 <= s && s < _n);
            assert(0 <= t && t < _n);
            assert(s != t);

            int m = int(_edges.size());
            std::vector<int> edge_idx(m);

            auto g = [&]() {
                std::vector<int> degree(_n), redge_idx(m);
                std::vector<std::pair<int, _edge>> elist;
                elist.reserve(2 * m);
                for (int i = 0; i < m; i++) {
                    auto e = _edges[i];
                    edge_idx[i] = degree[e.from]++;
                    redge_idx[i] = degree[e.to]++;
                    elist.push_back({ e.from, {e.to, -1, e.cap - e.flow, e.
                        cost} });
```

```
                    elist.push_back({ e.to, {e.from, -1, e.flow, -e.cost} })
                        ;
                }
                auto _g = csr<_edge>(_n, elist);
                for (int i = 0; i < m; i++) {
                    auto e = _edges[i];
                    edge_idx[i] += _g.start[e.from];
                    redge_idx[i] += _g.start[e.to];
                    _g.elist[edge_idx[i]].rev = redge_idx[i];
                    _g.elist[redge_idx[i]].rev = edge_idx[i];
                }
                return _g;
            }();

            auto result = slope(g, s, t, flow_limit);

            for (int i = 0; i < m; i++) {
                auto e = g.elist[edge_idx[i]];
                _edges[i].flow = _edges[i].cap - e.cap;
            }

            return result;
        }

    private:
        int _n;
        std::vector<edge> _edges;

        // inside edge
        struct _edge {
            int to, rev;
            Cap cap;
            Cost cost;
        };

        std::vector<std::pair<Cap, Cost>> slope(csr<_edge>& g,
            int s,
            int t,
            Cap flow_limit) {
            // variants (C = maxcost):
            // -(n-1)C <= dual[s] <= dual[i] <= dual[t] = 0
            // reduced cost (= e.cost + dual[e.from] - dual[e.to]) >= 0 for
                all edge
```

```cpp

        // dual_dist[i] = (dual[i], dist[i])
        std::vector<std::pair<Cost, Cost>> dual_dist(_n);
        std::vector<int> prev_e(_n);
        std::vector<bool> vis(_n);
        struct Q {
            Cost key;
            int to;
            bool operator<(Q r) const { return key > r.key; }
        };
        std::vector<int> que_min;
        std::vector<Q> que;
        auto dual_ref = [&]() {
            for (int i = 0; i < _n; i++) {
                dual_dist[i].second = std::numeric_limits<Cost>::max();
            }
            std::fill(vis.begin(), vis.end(), false);
            que_min.clear();
            que.clear();

            // que[0..heap_r) was heapified
            size_t heap_r = 0;

            dual_dist[s].second = 0;
            que_min.push_back(s);
            while (!que_min.empty() || !que.empty()) {
                int v;
                if (!que_min.empty()) {
                    v = que_min.back();
                    que_min.pop_back();
                } else {
                    while (heap_r < que.size()) {
                        heap_r++;
                        std::push_heap(que.begin(), que.begin() + heap_r
                            );
                    }
                    v = que.front().to;
                    std::pop_heap(que.begin(), que.end());
                    que.pop_back();
                    heap_r--;
                }
                if (vis[v]) continue;
                vis[v] = true;
```

```cpp
                if (v == t) break;
                // dist[v] = shortest(s, v) + dual[s] - dual[v]
                // dist[v] >= 0 (all reduced cost are positive)
                // dist[v] <= (n-1)C
                Cost dual_v = dual_dist[v].first, dist_v = dual_dist[v].second;
                for (int i = g.start[v]; i < g.start[v + 1]; i++) {
                    auto e = g.elist[i];
                    if (!e.cap) continue;
                    // |-dual[e.to] + dual[v]| <= (n-1)C
                    // cost <= C - -(n-1)C + 0 = nC
                    Cost cost = e.cost - dual_dist[e.to].first + dual_v;
                    if (dual_dist[e.to].second - dist_v > cost) {
                        Cost dist_to = dist_v + cost;
                        dual_dist[e.to].second = dist_to;
                        prev_e[e.to] = e.rev;
                        if (dist_to == dist_v) {
                            que_min.push_back(e.to);
                        } else {
                            que.push_back(Q{ dist_to, e.to });
                        }
                    }
                }
            }
            if (!vis[t]) {
                return false;
            }

            for (int v = 0; v < _n; v++) {
                if (!vis[v]) continue;
                // dual[v] = dual[v] - dist[t] + dist[v]
                //         = dual[v] - (shortest(s, t) + dual[s] - dual[t]) +
                //         (shortest(s, v) + dual[s] - dual[v]) = - shortest(s,
                //         t) + dual[t] + shortest(s, v) = shortest(s, v) -
                //         shortest(s, t) >= 0 - (n-1)C
                dual_dist[v].first -= dual_dist[t].second - dual_dist[v].second;
            }
            return true;
        };
```

```
            Cap flow = 0;
            Cost cost = 0, prev_cost_per_flow = -1;
            std::vector<std::pair<Cap, Cost>> result = { {Cap(0), Cost(0)}
                };
            while (flow < flow_limit) {
                if (!dual_ref()) break;
                Cap c = flow_limit - flow;
                for (int v = t; v != s; v = g.elist[prev_e[v]].to) {
                    c = std::min(c, g.elist[g.elist[prev_e[v]].rev].cap);
                }
                for (int v = t; v != s; v = g.elist[prev_e[v]].to) {
                    auto& e = g.elist[prev_e[v]];
                    e.cap += c;
                    g.elist[e.rev].cap -= c;
                }
                Cost d = -dual_dist[s].first;
                flow += c;
                cost += c * d;
                if (prev_cost_per_flow == d) {
                    result.pop_back();
                }
                result.push_back({ flow, cost });
                prev_cost_per_flow = d;
            }
            return result;
        }
    };
}
```

6.7 全局最小割

```
constexpr int N = 601;
constexpr int inf = 0x3f3f3f3f;
int edge[N][N]; // 边权存这里
int dis[N], vis[N], bin[N];
int n, m;
int contract(int& s, int& t) {  // Find s,t
    memset(dis, 0, sizeof(dis));
    memset(vis, false, sizeof(vis));
    int i, j, k, mincut, maxc;
    for (i = 1; i <= n; i++) {
        k = -1;
        maxc = -1;
```

```
        for (j = 1; j <= n; j++) {
            if (!bin[j] && !vis[j] && dis[j] > maxc) {
                k = j;
                maxc = dis[j];
            }
        }
        if (k == -1) return mincut;
        s = t; t = k;
        mincut = maxc;
        vis[k] = true;
        for (j = 1; j <= n; j++) {
            if (!bin[j] && !vis[j]) {
                dis[j] += edge[k][j];
            }
        }
    }
    return mincut;
}

int stoerWagner() { // O(NM + N^2logN) <=> O(N^3)
    int mincut, i, j, s, t, ans;
    for (mincut = inf, i = 1; i < n; i++) {
        ans = contract(s, t);
        bin[t] = true;
        if (mincut > ans) mincut = ans;
        if (mincut == 0) return 0;
        for (j = 1; j <= n; j++) {
            if (!bin[j]) {
                edge[s][j] = (edge[j][s] += edge[j][t]);
            }
        }
    }
    return mincut;
}
```

6.8 二分图最大权匹配

```
namespace KM {
    typedef long long ll;
    const int maxn = 510;
    const int inf = 1e9;
    int vx[maxn], vy[maxn], lx[maxn], ly[maxn], slack[maxn];
    int w[maxn][maxn]; // 以上为权值类型
```

```
    int pre[maxn], left[maxn], right[maxn], NL, NR, N;
    void match(int& u) {
        for (; u; std::swap(u, right[pre[u]]))
            left[u] = pre[u];
    }
    void bfs(int u) {
        static int q[maxn], front, rear;
        front = 0; vx[q[rear = 1] = u] = true;
        while (true) {
            while (front < rear) {
                int u = q[++front];
                for (int v = 1; v <= N; ++v) {
                    int tmp;
                    if (vy[v] || (tmp = lx[u] + ly[v] - w[u][v]) > slack[v])
                        continue;
                    pre[v] = u;
                    if (!tmp) {
                        if (!left[v]) return match(v);
                        vy[v] = vx[q[++rear] = left[v]] = true;
                    } else slack[v] = tmp;
                }
            }
            int a = inf;
            for (int i = 1; i <= N; ++i)
                if (!vy[i] && a > slack[i]) a = slack[u = i];
            for (int i = 1; i <= N; ++i) {
                if (vx[i]) lx[i] -= a;
                if (vy[i]) ly[i] += a;
                else slack[i] -= a;
            }
            if (!left[u]) return match(u);
            vy[u] = vx[q[++rear] = left[u]] = true;

        }

    }
    void exec() {
        for (int i = 1; i <= N; ++i) {
            for (int j = 1; j <= N; ++j) {
                slack[j] = inf;
                vx[j] = vy[j] = false;
            }
            bfs(i);
```

```
            }
        }
        ll work(int nl, int nr) { // NL , NR 为左右点数, 返回最大权匹配的权值和
            NL = nl; NR = nr;
            N = std::max(NL, NR);
            for (int u = 1; u <= N; ++u)
                for (int v = 1; v <= N; ++v)
                    lx[u] = std::max(lx[u], w[u][v]);
            exec();
            ll ans = 0;
            for (int i = 1; i <= N; ++i)
                ans += lx[i] + ly[i];
            return ans;
        }
        void output() { // 输出左边点与右边哪个点匹配, 没有匹配输出0
            for (int i = 1; i <= NL; ++i)
                printf("%d ", (w[i][right[i]] ? right[i] : 0));
            printf("\n");
        }
}
```

6.9 一般图最大匹配

```
// UOJ79 copy from jiangly
#include <bits/stdc++.h>
struct Graph {
    int n;
    std::vector<std::vector<int>> e;
    Graph(int n) : n(n), e(n) {}
    void addEdge(int u, int v) {
        e[u].push_back(v);
        e[v].push_back(u);
    }
    std::vector<int> findMatching() {
        std::vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);
        // disjoint set union
        auto find = [&](int u) {
            while (f[u] != u)
                u = f[u] = f[f[u]];
            return u;
        };
        auto lca = [&](int u, int v) {
            u = find(u);
```

```
            v = find(v);
            while (u != v) {
                if (dep[u] < dep[v])
                    std::swap(u, v);
                u = find(link[match[u]]);
            }
            return u;
        };

        std::queue<int> que;
        auto blossom = [&](int u, int v, int p) {
            while (find(u) != p) {
                link[u] = v;
                v = match[u];
                if (vis[v] == 0) {
                    vis[v] = 1;
                    que.push(v);
                }
                f[u] = f[v] = p;
                u = link[v];
            }
        };

        // find an augmenting path starting from u and augment (if exist)
        auto augment = [&](int u) {
            while (!que.empty())
                que.pop();
            std::iota(f.begin(), f.end(), 0);
            // vis = 0 corresponds to inner vertices, vis = 1 corresponds to
                outer vertices
            std::fill(vis.begin(), vis.end(), -1);
            que.push(u);
            vis[u] = 1, dep[u] = 0;
            while (!que.empty()){
                int u = que.front();
                que.pop();
                for (auto v : e[u]) {
                    if (vis[v] == -1) {
                        vis[v] = 0;
                        link[v] = u;
                        dep[v] = dep[u] + 1;
                        // found an augmenting path
                        if (match[v] == -1) {
```

```
                        for (int x = v, y = u, temp; y != -1; x = temp,
                            y = x == -1 ? -1 : link[x]) {
                            temp = match[y];
                            match[x] = y;
                            match[y] = x;
                        }
                        return;
                    }
                    vis[match[v]] = 1;
                    dep[match[v]] = dep[u] + 2;
                    que.push(match[v]);
                } else if (vis[v] == 1 && find(v) != find(u)) {
                    // found a blossom
                    int p = lca(u, v);
                    blossom(u, v, p);
                    blossom(v, u, p);
                }
            }
        }
    };

    // find a maximal matching greedily (decrease constant)
    auto greedy = [&]() {
        for (int u = 0; u < n; ++u) {
            if (match[u] != -1)
                continue;
            for (auto v : e[u]) {
                if (match[v] == -1) {
                    match[u] = v;
                    match[v] = u;
                    break;
                }
            }
        }
    };
    greedy();
    for (int u = 0; u < n; ++u)
        if (match[u] == -1)
            augment(u);
    return match;
}
};

```

```
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    Graph g(n);
    for (int i = 0; i < m; ++i) {
        int u, v;
        std::cin >> u >> v;
        --u, --v;
        g.addEdge(u, v);
    }
    auto match = g.findMatching();
    int ans = 0;
    for (int u = 0; u < n; ++u)
        if (match[u] != -1)
            ++ans;
    std::cout << ans / 2 << "\n";
    for (int u = 0; u < n; ++u) // 输出每个人匹配的对象，如果没有则输出0
        std::cout << match[u] + 1 << " \n"[u == n - 1];
    return 0;
}
```

6.10 2-sat

6.11 最大团

```
/*
 * 最大团 Bron-Kerbosch algorithm
 * 最劣复杂度 O(3^(n/3))
 * 采用位运算形式实现
 * */
namespace Max_clique {
#define ll long long
#define TWOL(x) (1ll<<(x))
    const int N = 60;
    int n, m;          // 点数 边数
    int r = 0;         // 最大团大小
    ll G[N];           // 以二进制形式存图
    ll clique = 0;     // 最大团 以二进制形式存储
    void BronK(int S, ll P, ll X, ll R) { // 调用时参数这样设置: 0, TWOL(n)-1, 0, 0
        if (P == 0 && X == 0) {
            if (r < S) {
```

```
                r = S;
                clique = R;
            }
        }
        if (P == 0) return;
        int u = __builtin_ctzll(P | X);
        ll c = P & ~G[u];
        while (c) {
            int v = __builtin_ctzll(c);
            ll pv = TWOL(v);
            BronK(S + 1, P & G[v], X & G[v], R | pv);
            P ^= pv; X |= pv; c ^= pv;
        }
    }
    void init() {
        cin >> n >> m;
        for (int i = 0; i < m; i++) {
            int u, v;
            cin >> u >> v;
            --u, --v;
            G[u] |= TWOL(v);
            G[v] |= TWOL(u);
        }
        BronK(0, TWOL(n)-1, 0, 0);
        cout << r << ' ' << clique << '\n';
    }
}
```

7 数据结构

7.1 树状数组

```
template<typename T> struct fenwickTree {
    int n, hbit;
    vector<T> tree;
    fenwickTree(int n_ = 0) : n(n_), tree(n_ + 1), hbit(log2(n_) + 1) {}
    int lowbit(int x) { return x & (-x); }
    int size() { return n; }
    void add(int pos, int x) { // pos位置加上x
        for (; pos <= n; pos += lowbit(pos)) {
            tree[pos] += x;
        }
    }
```

```
    T query(int pos) { // 查询pos位置的前缀和 即a[1] + a[2] + ... + a[pos]
        T res = 0;
        for (; pos > 0; pos -= lowbit(pos)) {
            res += tree[pos];
        }
        return res;
    }
    T sum(int l, int r) { // [l, r]区间查询
        return query(r) - query(l - 1);
    }
    int kth(int k) { // 第k大元素
        int ans = 0, cnt = 0;
        for (int i = hbit; i >= 0; i--) {
            ans += (1 << i);
            if (ans > n || cnt + tree[ans] >= k) ans -= (1 << i);
            else cnt += tree[ans];
        }
        return ++ans;
    }
};
```

7.2 线段树

8 字符串

8.1 KMP

```
namespace KMP {
    vector<int> getPrefixTable(string s) { // 求前缀表
        int n = s.length();
        vector<int> nxt(n, 0);
        for (int i = 1; i < n; i++) {
            int j = nxt[i - 1];
            while (j > 0 && s[i] != s[j]) {
                j = nxt[j - 1];
            }
            if (s[i] == s[j]) j++;
            nxt[i] = j;
        }
        return nxt;
    }

    vector<int> kmp(string s, string t) { // 返回所有匹配位置的集合
        int n = s.length(), m = t.length();
```

```
        vector<int> res;
        vector<int> nxt = getPrefixTable(t);
        for (int i = 0, j = 0; i < n; i++) {
            while (j > 0 && j < m && s[i] != t[j]) {
                j = nxt[j - 1];
            }
            if (s[i] == t[j]) j++;
            if (j == m) {
                res.push_back(i + 1 - m);
                j = nxt[m - 1];
            }
        }
        return res;
    }
}
```

8.2 Z-Function

```
// O(N) 查询字符串s每一位开始的LCP
vector<int> z_function(string s) {
    int n = (int)s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r && z[i - l] < r - i + 1) {
            z[i] = z[i - l];
        } else {
            z[i] = max(0, r - i + 1);
            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;
}
```

8.3 Manacher

```
namespace Manacher {
    static constexpr int SIZE = 1e5 + 5; // 预设为原串长度
    int len = 1; // manacher 预处理后字符串的长度
    char stk[SIZE << 1]; // manacher预处理字符串 需要2倍空间+1
    void init(string s) { // 初始化stk
        stk[0] = '*'; len = 1;
        for (int i = 0; i < s.length(); ++i) {
```

```
            stk[len++] = s[i];
            stk[len++] = '*';
        }
    }
    int manacher() { // 返回最长回文子串长度
        vector<int> rad(len << 1); // 存储每个点作为对称中心可拓展的最大半径
        int md = 0; // 最远回文串对称中心下标
        for (int i = 1; i < len; ++i) {
            int& r = rad[i] = 0;
            if (i <= md + rad[md]) {
                r = min(rad[2 * md - i], md + rad[md] - i);
            }
            while (i - r - 1 >= 0 && i + r + 1 < len &&
                stk[i - r - 1] == stk[i + r + 1]) ++r;
            if (i + r >= md + rad[md]) md = i;
        }
        int res = 0;
        for (int i = 0; i < len; ++i) {
            if (rad[i] > res) {
                res = rad[i];
            }
        }
        return res;
    }
}
```

8.4 Trie

```
struct trie {
    int cnt;
    vector<vector<int>> nxt;
    vector<bool> vis;
    /* 初始化的时候size需要设置为字符串总长之和 26是字符集大小 */
    trie(int size_ = 0) :cnt(0), vis(size_, false), nxt(size_, vector<int
        >(26, 0)) {}
    void insert(string s) {  // 插入字符串
        int p = 0;
        for (int i = 0; i < (int)s.length(); i++) {
            int c = s[i] - 'a';
            if (!nxt[p][c]) nxt[p][c] = ++cnt;
            p = nxt[p][c];
        }
        vis[p] = true;
```

```cpp
    }
    bool find(string s) {  // 查找字符串
        int p = 0;
        for (int i = 0; i < (int)s.length(); i++) {
            int c = s[i] - 'a';
            if (!nxt[p][c]) return false;
            p = nxt[p][c];
        }
        return vis[p];
    }
};
```

8.5 01-Trie

```cpp
template<typename T> struct xorTrie {
    int HIGHBIT, cnt;
    vector<vector<int>> nxt;
    vector<bool> vis;
    xorTrie(int n_ = 0, int highbit_ = 30) : HIGHBIT(highbit_), cnt(0) {
        int size_ = upperBoundEstimate(n_);
        nxt.resize(size_, vector<int>(2, 0));
        vis.resize(size_, false);
    }
    int upperBoundEstimate(int n) { // 求内存上界
        int hbit = log2(n);
        return n * (HIGHBIT - hbit + 1) + (1 << (hbit + 1)) - 1;
    }
    void insert(T x) { // 插入
        int p = 0;
        for (int i = HIGHBIT; ~i; i--) {
            int s = ((x >> i) & 1);
            if (!nxt[p][s]) nxt[p][s] = ++cnt;
            p = nxt[p][s];
        }
        vis[p] = true;
    }
    bool find(T x) { // 查询
        int p = 0;
        for (int i = HIGHBIT; ~i; i--) {
            int s = ((x >> i) & 1);
            if (!nxt[p][s]) return false;
            p = nxt[p][s];
        }
```

```
            return vis[p];
        }
};
```

9 计算几何

```
namespace Geometry {
#define db long double
#define pi acos(-1.0)
    constexpr db eps = 1e-7;
    int sign(db k) {
        if (k > eps) return 1;
        else if (k < -eps) return -1;
        return 0;
    }
    int cmp(db k1, db k2) { // k1 < k2 : -1, k1 == k2 : 0, k1 > k2 : 1
        return sign(k1 - k2);
    }
    int inmid(db k1, db k2, db k3) { // k3 在 [k1, k2] 内
        return sign(k1 - k3) * sign(k2 - k3) <= 0;
    }

    struct point { // 点类
        db x, y;
        point() {}
        point(db x_, db y_) :x(x_), y(y_) {}
        point operator + (const point& k) const { return point(k.x + x, k.y + y); }
        point operator - (const point& k) const { return point(x - k.x, y - k.y); }
        point operator * (db k) const { return point(x * k, y * k); }
        point operator / (db k1) const { return point(x / k1, y / k1); }
        point turn(db k1) { return point(x * cos(k1) - y * sin(k1), x * sin(k1) + y * cos(k1)); } // 逆时针旋转
        point turn90() { return point(-y, x); } // 逆时针方向旋转 90 度
        db len() { return sqrt(x * x + y * y); } // 向量长度
        db len2() { return x * x + y * y; } // 向量长度的平方
        db getPolarAngle() { return atan2(y, x); } // 向量极角
        db dis(point k) { return ((*this) - k).len(); } // 到点k的距离
        point unit() { db d = len(); return point(x / d, y / d); } // 单位向量
        point getdel() { // 将向量的方向调整为指向第一/四象限 包括y轴正方向
            if (sign(x) == -1 || (sign(x) == 0 && sign(y) == -1))
```

```
                return (*this) * (-1);
            else return (*this);
        }
        bool operator < (const point& k) const { // 水平序排序 x坐标为第一关键字,y坐标第二关键字
            return x == k.x ? y < k.y : x < k.x;
        }
        bool operator == (const point& k) const { return cmp(x, k.x) == 0 && cmp(y, k.y) == 0; }
        bool getP() const { // 判断点是否在上半平面 含x负半轴 不含x正半轴及零点
            return sign(y) == 1 || (sign(y) == 0 && sign(x) == -1);
        }
        void input() { cin >> x >> y; }
    };
    db cross(point k1, point k2) { return k1.x * k2.y - k1.y * k2.x; } // 向量 k1,k2 的叉积
    db dot(point k1, point k2) { return k1.x * k2.x + k1.y * k2.y; }   // 向量 k1,k2 的点积
    db rad(point k1, point k2) { // 向量 k1,k2 之间的有向夹角
        return atan2(cross(k1, k2), dot(k1, k2));
    }
    int inmid(point k1, point k2, point k3) { // k1 k2 k3共线时 判断点 k3 是否在线段 k1k2 上
        return inmid(k1.x, k2.x, k3.x) && inmid(k1.y, k2.y, k3.y);
    }
    int compareAngle(point k1, point k2) { // 比较向量 k1,k2 的角度大小 角度按照atan2()函数定义
        // k1 < k2 返回 1, k1 >= k2 返回 0
        return k1.getP() < k2.getP() || (k1.getP() == k2.getP() && sign(cross(k1, k2)) > 0);
    }
    point proj(point k1, point k2, point q) { // q 到直线 k1,k2 的投影
        point k = k2 - k1; return k1 + k * (dot(q - k1, k) / k.len2());
    }
    point reflect(point k1, point k2, point q) { return proj(k1, k2, q) * 2 - q; } // q 关于直线 k1,k2 的对称点
    int counterclockwise(point k1, point k2, point k3) { // k1 k2 k3 逆时针1 顺时针-1 否则0
        return sign(cross(k2 - k1, k3 - k1));
    }
    int checkLL(point k1, point k2, point k3, point k4) { // 判断直线 k1k2 和直线k3k4 是否相交
```

```
        // 即判断直线 k1k2 和 k3k4 是否平行 平行返回0 不平行返回1
        return sign(cross(k2 - k1, k4 - k3)) != 0;
    }
    point getLL(point k1, point k2, point k3, point k4) { // 求 k1k2 k3k4 两直线交点
        db w1 = cross(k1 - k3, k4 - k3), w2 = cross(k4 - k3, k2 - k3);
        return (k1 * w2 + k2 * w1) / (w1 + w2);
    }
    int intersect(db l1, db r1, db l2, db r2) { // 判断 [l1,r1] 和 [l2, r2] 是否相交
        if (l1 > r1) swap(l1, r1);
        if (l2 > r2) swap(l2, r2);
        return cmp(r1, l2) != -1 && cmp(r2, l1) != -1;
    }
    int checkSS(point k1, point k2, point k3, point k4) { // 判断线段 k1k2 和线段 k3k4 是否相交
        return intersect(k1.x, k2.x, k3.x, k4.x) && intersect(k1.y, k2.y, k3.y, k4.y) &&
            sign(cross(k3 - k1, k4 - k1)) * sign(cross(k3 - k2, k4 - k2)) <= 0 &&
            sign(cross(k1 - k3, k2 - k3)) * sign(cross(k1 - k4, k2 - k4)) <= 0;
    }
    db disSP(point k1, point k2, point q) { // 点 q 到线段 k1k2 的最短距离
        point k3 = proj(k1, k2, q);
        if (inmid(k1, k2, k3)) return q.dis(k3);
        else return min(q.dis(k1), q.dis(k2));
    }
    db disLP(point k1, point k2, point q) { // 点 q 到直线 k1k2 的最短距离
        point k3 = proj(k1, k2, q);
        return q.dis(k3);
    }
    db disSS(point k1, point k2, point k3, point k4) { // 线段 k1k2 和线段 k3k4 的最短距离
        if (checkSS(k1, k2, k3, k4)) return 0;
        else return min(min(disSP(k1, k2, k3), disSP(k1, k2, k4)),
            min(disSP(k3, k4, k1), disSP(k3, k4, k2)));
    }
    bool onLine(point k1, point k2, point q) { // 判断点 q 是否在直线 k1k2 上
        return sign(cross(k1 - q, k2 - q)) == 0;
    }
    bool onSegment(point k1, point k2, point q) { // 判断点 q 是否在线段
```

```
    k1k2 上
    if (!onLine(k1, k2, q)) return false; // 如果确定共线 要删除这个特判
    return inmid(k1, k2, q);
}
void polarAngleSort(vector<point>& p, point t) { // p为待排序点集 t为极角排序中心
    sort(p.begin(), p.end(), [&](const point& k1, const point& k2) {
        return compareAngle(k1 - t, k2 - t);
    });
}

struct line { // 直线 / 线段类
    point p[2];
    line() {}
    line(point k1, point k2) { p[0] = k1, p[1] = k2; }
    point& operator [] (int k) { return p[k]; }
    point dir() { return p[1] - p[0]; } // 向量 p[0] -> p[1]
    bool include(point k) { // 判断点是否在直线上
        return sign(cross(p[1] - p[0], k - p[0])) > 0;
    }
    bool includeS(point k) { // 判断点是否在线段上
        return onSegment(p[0], p[1], k);
    }
    line push(db len) { // 向外 (左手边) 平移 len 个单位
        point delta = (p[1] - p[0]).turn90().unit() * len;
        return line(p[0] - delta, p[1] - delta);
    }
};
bool parallel(line k1, line k2) { // 判断是否平行
    return sign(cross(k1.dir(), k2.dir())) == 0;
}
bool sameLine(line k1, line k2) { // 判断是否共线
    return parallel(k1, k2) && parallel(k1, line(k2.p[0], k1.p[0]));
}
bool sameDir(line k1, line k2) { // 判断向量 k1 k2 是否同向
    return parallel(k1, k2) && sign(dot(k1.dir(), k2.dir())) == 1;
}
bool operator < (line k1, line k2) {
    if (sameDir(k1, k2)) return k2.include(k1[0]);
    return compareAngle(k1.dir(), k2.dir());
}
bool checkLL(line k1, line k2) {
    return checkLL(k1[0], k1[1], k2[0], k2[1]);
```

```cpp
    }
    point getLL(line k1, line k2) {  // 求 k1 k2 两直线交点 不要忘了判平行!
        return getLL(k1[0], k1[1], k2[0], k2[1]);
    }
    bool checkpos(line k1, line k2, line k3) {  // 判断是否三线共点
        return k3.include(getLL(k1, k2));
    }

    struct circle { // 圆类
        point o;
        double r;
        circle() {}
        circle(point o_, double r_) : o(o_), r(r_) {}
        int inside(point k) {  // 判断点 k 和圆的位置关系
            return cmp(r, o.dis(k)); // 圆外:-1, 圆上:0, 圆内:1
        }
    };
    int checkposCC(circle k1, circle k2) { // 返回两个圆的公切线数量
        if (cmp(k1.r, k2.r) == -1) swap(k1, k2);
        db dis = k1.o.dis(k2.o);
        int w1 = cmp(dis, k1.r + k2.r), w2 = cmp(dis, k1.r - k2.r);
        if (w1 > 0) return 4; // 外离
        else if (w1 == 0) return 3; // 外切
        else if (w2 > 0) return 2;  // 相交
        else if (w2 == 0) return 1; // 内切
        else return 0; // 内离(包含)
    }
    vector<point> getCL(circle k1, point k2, point k3) { // 求直线 k2k3 和圆 k1 的交点
        // 沿着 k2->k3 方向给出 相切给出两个
        point k = proj(k2, k3, k1.o);
        db d = k1.r * k1.r - (k - k1.o).len2();
        if (sign(d) == -1) return {};
        point del = (k3 - k2).unit() * sqrt(max((db)0.0, d));
        return { k - del,k + del };
    }
    vector<point> getCC(circle k1, circle k2) { // 求圆 k1 和圆 k2 的交点
        // 沿圆 k1 逆时针给出, 相切给出两个
        int pd = checkposCC(k1, k2); if (pd == 0 || pd == 4) return {};
        db a = (k2.o - k1.o).len2(), cosA = (k1.r * k1.r + a -
            k2.r * k2.r) / (2 * k1.r * sqrt(max(a, (db)0.0)));
        db b = k1.r * cosA, c = sqrt(max((db)0.0, k1.r * k1.r - b * b));
        point k = (k2.o - k1.o).unit(), m = k1.o + k * b, del = k.turn90() *
```

```
            c;
        return { m - del ,m + del };
    }
    vector<point> tangentCP(circle k1, point k2) { // 点 k2 到圆 k1 的切点沿圆 k1 逆时针给出
        db a = (k2 - k1.o).len(), b = k1.r * k1.r / a, c = sqrt(max((db)0.0, k1.r * k1.r - b * b));
        point k = (k2 - k1.o).unit(), m = k1.o + k * b, del = k.turn90() * c;
        return { m - del ,m + del };
    }
    vector<line> tangentOutCC(circle k1, circle k2) {
        int pd = checkposCC(k1, k2);
        if (pd == 0) return {};
        if (pd == 1) {
            point k = getCC(k1, k2)[0];
            return { line(k,k) };
        }
        if (cmp(k1.r, k2.r) == 0) {
            point del = (k2.o - k1.o).unit().turn90().getdel();
            return { line(k1.o - del * k1.r,k2.o - del * k2.r),
                line(k1.o + del * k1.r,k2.o + del * k2.r) };
        } else {
            point p = (k2.o * k1.r - k1.o * k2.r) / (k1.r - k2.r);
            vector<point> A = tangentCP(k1, p), B = tangentCP(k2, p);
            vector<line> ans; for (int i = 0; i < A.size(); i++)
                ans.push_back(line(A[i], B[i]));
            return ans;
        }
    }
    vector<line> tangentInCC(circle k1, circle k2) {
        int pd = checkposCC(k1, k2);
        if (pd <= 2) return {};
        if (pd == 3) {
            point k = getCC(k1, k2)[0];
            return { line(k, k) };
        }
        point p = (k2.o * k1.r + k1.o * k2.r) / (k1.r + k2.r);
        vector<point> A = tangentCP(k1, p), B = tangentCP(k2, p);
        vector<line> ans;
        for (int i = 0; i < (int)A.size(); i++) ans.push_back(line(A[i], B[i]));
        return ans;
```

```
    }
    vector<line> tangentCC(circle k1, circle k2) { // 求两圆公切线
        int flag = 0;
        if (k1.r < k2.r) swap(k1, k2), flag = 1;
        vector<line> A = tangentOutCC(k1, k2), B = tangentInCC(k1, k2);
        for (line k : B) A.push_back(k);
        if (flag) for (line& k : A) swap(k[0], k[1]);
        return A;
    }
    db getAreaCT(circle k1, point k2, point k3) { // 圆 k1 与三角形 k2k3k1.o
        的有向面积交
        point k = k1.o; k1.o = k1.o - k; k2 = k2 - k; k3 = k3 - k;
        int pd1 = k1.inside(k2), pd2 = k1.inside(k3);
        vector<point> A = getCL(k1, k2, k3);
        if (pd1 >= 0) {
            if (pd2 >= 0) return cross(k2, k3) / 2;
            return k1.r * k1.r * rad(A[1], k3) / 2 + cross(k2, A[1]) / 2;
        } else if (pd2 >= 0) {
            return k1.r * k1.r * rad(k2, A[0]) / 2 + cross(A[0], k3) / 2;
        } else {
            int pd = cmp(k1.r, disSP(k2, k3, k1.o));
            if (pd <= 0) return k1.r * k1.r * rad(k2, k3) / 2;
            return cross(A[0], A[1]) / 2 + k1.r * k1.r * (rad(k2, A[0]) +
                rad(A[1], k3)) / 2;
        }
    }
    db getAreaCC(circle k1, circle k2) { // 两圆面积交
        db d = k1.o.dis(k2.o);
        if (cmp(d, k1.r + k2.r) >= 0) return 0; // 两圆相离
        if (cmp(k1.r, k2.r) == -1) swap(k1, k2);
        if (cmp(k1.r - k2.r, d) >= 0) return pi * k2.r * k2.r; // 圆k1包含k2
        db g1 = acos((k1.r * k1.r + d * d - k2.r * k2.r) / (2 * k1.r * d));
        db g2 = acos((k2.r * k2.r + d * d - k1.r * k1.r) / (2 * k2.r * d));
        return g1 * k1.r * k1.r + g2 * k2.r * k2.r - k1.r * d * sin(g1);
    }
    circle getCircleOut(point k1, point k2, point k3) { // 三角形外接圆
        db a1 = k2.x - k1.x, b1 = k2.y - k1.y, c1 = (a1 * a1 + b1 * b1) / 2;
        db a2 = k3.x - k1.x, b2 = k3.y - k1.y, c2 = (a2 * a2 + b2 * b2) / 2;
        db d = a1 * b2 - a2 * b1;
        point o(k1.x + (c1 * b2 - c2 * b1) / d, k1.y + (a1 * c2 - a2 * c1) /
            d);
        return circle(o, k1.dis(o));
    }
```

```
    circle getCircleIn(point k1, point k2, point k3) { // 三角形内切圆
        db a = k1.dis(k2), b = k2.dis(k3), c = k3.dis(k1);
        db len = a + b + c;
        db r = abs(cross(k1 - k2, k1 - k3)) / len;
        point o((k1.x * b + k2.x * c + k3.x * a) / len, (k1.y * b + k2.y * c
            + k3.y * a) / len);
        return circle(o, r);
    }
    circle minCircleCovering(vector<point> A) { // 最小圆覆盖 O(n)随机增量法
        // random_shuffle(A.begin(), A.end()); // <= C++14
        auto seed = chrono::steady_clock::now().time_since_epoch().count();
        default_random_engine e(seed);
        shuffle(A.begin(), A.end(), e); // >= C++11
        circle ans = circle(A[0], 0);
        for (int i = 1; i < A.size(); i++) {
            if (ans.inside(A[i]) == -1) {
                ans = circle(A[i], 0);
                for (int j = 0; j < i; j++) {
                    if (ans.inside(A[j]) == -1) {
                        ans.o = (A[i] + A[j]) / 2;
                        ans.r = ans.o.dis(A[i]);
                        for (int k = 0; k < j; k++) {
                            if (ans.inside(A[k]) == -1)
                                ans = getCircleOut(A[i], A[j], A[k]);
                        }
                    }
                }
            }
        }
        return ans;
    }

    typedef vector<point> polygon;
    db area(polygon p) { // 多边形有向面积
        if (p.size() < 3) return 0;
        db ans = 0;
        for (int i = 1; i < p.size() - 1; i++)
            ans += cross(p[i] - p[0], p[i + 1] - p[0]);
        return 0.5L * ans;
    }

    int checkConvexP(polygon p, point a) { // O(logn)判断点是否在凸包内 2内
        部 1边界 0外部
```

```
        // 必须保证凸多边形是一个水平序凸包且不能退化
        // 退化情况 比如凸包退化成线段 可使用 onSegment() 函数特判
        auto check = [&](int x) {
            int ccw1 = counterclockwise(p[0], a, p[x]),
                ccw2 = counterclockwise(p[0], a, p[x + 1]);
            if (ccw1 == -1 && ccw2 == -1) return 2;
            else if (ccw1 == 1 && ccw2 == 1) return 0;
            else if (ccw1 == -1 && ccw2 == 1) return 1;
            else return 1;
        };
        if (counterclockwise(p[0], a, p[1]) <= 0 && counterclockwise(p[0], a
            , p.back()) >= 0) {
            int l = 1, r = p.size() - 2, mid;
            while (l <= r) {
                mid = (l + r) >> 1;
                int chk = check(mid);
                if (chk == 1) l = mid + 1;
                else if (chk == -1) r = mid;
                else break;
            }
            int res = counterclockwise(p[mid], a, p[mid + 1]);
            if (res < 0) return 2;
            else if (res == 0) return 1;
            else return 0;
        } else {
            return 0;
        }
    }
    int checkPolyP(vector<point> p, point q) { // O(n)判断点是否在一般多边形
        内
        // 必须保证简单多边形的点按逆时针给出 返回 2 内部 1 边界 0 外部
        int pd = 0, n = p.size();
        for (int i = 0; i < n; i++) {
            point u = p[i], v = p[(i + 1) % n];
            if (onSegment(u, v, q)) return 1;
            if (cmp(u.y, v.y) > 0) swap(u, v);
            if (cmp(u.y, q.y) >= 0 || cmp(v.y, q.y) < 0) continue;
            if (sign(cross(u - v, q - v)) < 0) pd ^= 1;
        }
        return pd << 1;
    }
    db convexDiameter(polygon p) { // O(n)旋转卡壳求凸包直径 / 平面最远点对
        的平方
```

```
        int n = p.size(); // 请保证多边形是凸包
        db ans = 0;
        for (int i = 0, j = n < 2 ? 0 : 1; i < j; i++) {
            for (;; j = (j + 1) % n) {
                ans = max(ans, (p[i] - p[j]).len2());
                if (sign(cross(p[i + 1] - p[i], p[(j + 1) % n] - p[j])) <= 0) break;
            }
        }
        return ans;
    }
    polygon convexHull(polygon A, int flag = 1) { // 凸包 flag=0 不严格 flag=1 严格
        int n = A.size(); polygon ans(n + n);
        sort(A.begin(), A.end()); int now = -1;
        for (int i = 0; i < A.size(); i++) {
            while (now > 0 && sign(cross(ans[now] - ans[now - 1], A[i] - ans[now - 1])) < flag)
                now--;
            ans[++now] = A[i];
        }
        int pre = now;
        for (int i = n - 2; i >= 0; i--) {
            while (now > pre && sign(cross(ans[now] - ans[now - 1], A[i] - ans[now - 1])) < flag)
                now--;
            ans[++now] = A[i];
        }
        ans.resize(now);
        return ans;
    }
    bool checkConvexHull(polygon p) { // 检测多边形是否是凸包（可以有三点共线）
        int sgn, n = p.size(), i = 0; // 如果三点共线不算凸包 去掉ccw=0的情况
        for (;; i++) { // 这一步是为了防止第一步遇到共线的三个点
            sgn = counterclockwise(p[i], p[(i + 1) % n], p[(i + 2) % n]);
            if (sgn) break;
        }
        for (; i < n; i++) {
            int ccw = counterclockwise(p[i], p[(i + 1) % n], p[(i + 2) % n]);
            if (ccw && ccw != sgn) {
```

```
                return false;
            }
        }
        return true;
    }
    polygon convexCut(polygon A, point k1, point k2) { // 半平面 k1k2 切凸包 A
        int n = A.size();  // 保留所有满足 k1 -> p -> k2 为逆时针方向的点
        A.push_back(A[0]); // 保留的点可能有重点
        polygon ans;
        line cut(k1, k2);
        for (int i = 0; i < n; i++) {
            int ccw1 = counterclockwise(k1, k2, A[i]);
            int ccw2 = counterclockwise(k1, k2, A[i + 1]);
            if (ccw1 >= 0) ans.push_back(A[i]);
            if (ccw1 * ccw2 <= 0) {
                if (sameLine(cut, line(A[i], A[i + 1]))) { // 半平面恰好切到凸包上某条边
                    ans.push_back(A[i]);
                    ans.push_back(A[i + 1]);
                } else {
                    ans.push_back(getLL(k1, k2, A[i], A[i + 1]));
                }
            }
        }
        return ans;
    }

    vector<line> getHL(vector<line>& L) { // 求半平面交 逆时针方向存储
        sort(L.begin(), L.end());
        deque<line> q;
        for (int i = 0; i < (int)L.size(); ++i) {
            if (i && sameDir(L[i], L[i - 1])) continue;
            while (q.size() > 1 && !checkpos(q[q.size() - 2], q[q.size() - 1], L[i])) q.pop_back();
            while (q.size() > 1 && !checkpos(q[1], q[0], L[i])) q.pop_front();
            q.push_back(L[i]);
        }
        while (q.size() > 2 && !checkpos(q[q.size() - 2], q[q.size() - 1], q[0])) q.pop_back();
        while (q.size() > 2 && !checkpos(q[1], q[0], q[q.size() - 1])) q.pop_front();
```

```
        vector<line> ans;
        for (int i = 0; i < q.size(); ++i) ans.push_back(q[i]);
        return ans;
    }

    db closestPoint(vector<point>& A, int l, int r) { // 最近点对，先要按照
        x 坐标排序
        if (r - l <= 5) {
            db ans = 1e20;
            for (int i = l; i <= r; ++i)
                for (int j = i + 1; j <= r; j++)
                    ans = min(ans, A[i].dis(A[j]));
            return ans;
        }
        int mid = l + r >> 1;
        db ans = min(closestPoint(A, l, mid), closestPoint(A, mid + 1, r));
        vector<point> B;
        for (int i = l; i <= r; i++)
            if (abs(A[i].x - A[mid].x) <= ans)
                B.push_back(A[i]);
        sort(B.begin(), B.end(), [&](const point& k1, const point& k2) {
            return k1.y < k2.y;
        });
        for (int i = 0; i < B.size(); i++)
            for (int j = i + 1; j < B.size() && B[j].y - B[i].y < ans; j++)
                ans = min(ans, B[i].dis(B[j]));
        return ans;
    }
}
using namespace Geometry;
```

10 杂项

10.1 快速 IO

```
// fast IO by yosupo
struct Scanner {
    FILE* fp = nullptr;
    char line[(1 << 15) + 1];
    size_t st = 0, ed = 0;
    void reread() {
        memmove(line, line + st, ed - st);
        ed -= st;
```

```
        st = 0;
        ed += fread(line + ed, 1, (1 << 15) - ed, fp);
        line[ed] = '\0';
    }
    bool succ() {
        while (true) {
            if (st == ed) {
                reread();
                if (st == ed) return false;
            }
            while (st != ed && isspace(line[st])) st++;
            if (st != ed) break;
        }
        if (ed - st <= 50) reread();
        return true;
    }
    template <class T, enable_if_t<is_same<T, string>::value, int> = 0>
    bool read_single(T& ref) {
        if (!succ()) return false;
        while (true) {
            size_t sz = 0;
            while (st + sz < ed && !isspace(line[st + sz])) sz++;
            ref.append(line + st, sz);
            st += sz;
            if (!sz || st != ed) break;
            reread();
        }
        return true;
    }
    template <class T, enable_if_t<is_integral<T>::value, int> = 0>
    bool read_single(T& ref) {
        if (!succ()) return false;
        bool neg = false;
        if (line[st] == '-') {
            neg = true;
            st++;
        }
        ref = T(0);
        while (isdigit(line[st])) {
            ref = 10 * ref + (line[st++] - '0');
        }
        if (neg) ref = -ref;
        return true;
```

```
    }
    template <class T> bool read_single(vector<T>& ref) {
        for (auto& d : ref) {
            if (!read_single(d)) return false;
        }
        return true;
    }
    void read() {}
    template <class H, class... T> void read(H& h, T&... t) {
        bool f = read_single(h);
        assert(f);
        read(t...);
    }
    Scanner(FILE* _fp) : fp(_fp) {}
};

struct Printer {
public:
    template <bool F = false> void write() {}
    template <bool F = false, class H, class... T>
    void write(const H& h, const T&... t) {
        if (F) write_single(' ');
        write_single(h);
        write<true>(t...);
    }
    template <class... T> void writeln(const T&... t) {
        write(t...);
        write_single('\n');
    }

    Printer(FILE* _fp) : fp(_fp) {}
    ~Printer() { flush(); }

private:
    static constexpr size_t SIZE = 1 << 15;
    FILE* fp;
    char line[SIZE], small[50];
    size_t pos = 0;
    void flush() {
        fwrite(line, 1, pos, fp);
        pos = 0;
    }
    void write_single(const char& val) {
```

```
        if (pos == SIZE) flush();
        line[pos++] = val;
    }
    template <class T, enable_if_t<is_integral<T>::value, int> = 0>
    void write_single(T val) {
        if (pos > (1 << 15) - 50) flush();
        if (val == 0) {
            write_single('0');
            return;
        }
        if (val < 0) {
            write_single('-');
            val = -val;  // todo min
        }
        size_t len = 0;
        while (val) {
            small[len++] = char('0' + (val % 10));
            val /= 10;
        }
        for (size_t i = 0; i < len; i++) {
            line[pos + i] = small[len - 1 - i];
        }
        pos += len;
    }
    void write_single(const string& s) {
        for (char c : s) write_single(c);
    }
    void write_single(const char* s) {
        size_t len = strlen(s);
        for (size_t i = 0; i < len; i++) write_single(s[i]);
    }
    template <class T> void write_single(const vector<T>& val) {
        auto n = val.size();
        for (size_t i = 0; i < n; i++) {
            if (i) write_single(' ');
            write_single(val[i]);
        }
    }
    void write_single(long double d) {
        {
            long long v = d;
            write_single(v);
            d -= v;
```

```
        }
        write_single('.');
        for (int _ = 0; _ < 8; _++) {
            d *= 10;
            long long v = d;
            write_single(v);
            d -= v;
        }
    }
};

Scanner sc(stdin);
Printer pr(stdout);
```

10.2 蔡勒公式

```
int zeller(int y, int m, int d) { // 蔡勒公式 返回星期几
        if (m <= 2) y--, m += 12;
        int c = y / 100; y %= 100;
        int w = ((c >> 2) - (c << 1) + y + (y >> 2) +
                (13 * (m + 1) / 5) + d - 1) % 7;
        if (w < 0) w += 7;
        return (w);
}
int getId(int y, int m, int d) { // 返回到公元1年1月1日的天数
        if (m < 3) { y--; m += 12; }
        return 365 * y + y / 4 - y / 100 + y / 400 +
                (153 * (m - 3) + 2) / 5 + d - 307;
}
```

10.3 枚举子集

10.3.1 暴力遍历

```
/* O(3^n)遍历子集 */
for (int i = 0; i < (1 << n); i++) { // i是当前需要遍历的全集
    for (int l = i;; l = i & (l - 1)) { // l是i的子集
        int r = i - l; // l+r=i
        if (!l) break;
    }
}
```

10.3.2 遍历大小为 k 的子集

```cpp
int n, k;
int s = (1 << k) - 1;
while (s < (1 << n)) { // O(binom(n, k))
    // 每次取出s就是一个大小为k的子集
    int x = s & -s, y = s + x;
    s = (((s & ~y) / x) >> 1) | y;
}
```

10.4 高维前缀和/SoSDP

```cpp
/* 高维前缀和/子集前缀和 */
for (int i = 0; i < n; i++) { // O(n2^n)
    for (int j = 0; j < (1 << n); j++) {
        if (j & (1 << i)) {
            pre[j] += pre[j ^ (1 << i)];
        }
    }
}
```