目录

1 数学

1.1 FWT

矩阵表示

or 形式 (子集卷积):

$$T_{ij} = [i|j = i] = [j \in i]$$

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

and 形式 (超集卷积):

$$T_{ij} = [i\&j = i] = [i \in j]$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

xor 形式 (T与自身互为逆矩阵):

$$T_{ij} = (-1)^{parity(i\&j)}$$

$$\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

```cpp
using ll = long long;

// or
void FWT(ll *a, int len, int inv) {
    for (int h = 1; h < len; h <<= 1) {
        for (int i = 0; i < len; i += (h << 1)) {
            for (int j = 0; j < h; ++j) {
                a[i + j + h] += a[i + j] * inv;
            }
        }
    }
}
// and
void FWT(ll *a, int len, int inv) {
    for (int h = 1; h < len; h <<= 1) {
        for (int i = 0; i < len; i += (h << 1)) {
            for (int j = 0; j < h; ++j) {
                a[i + j] += a[i + j + h] * inv;
            }
        }
    }
}
// xor
void FWT(ll *a, int len, int inv) {
    for (int h = 1; h < len; h <<= 1) {
        for (int i = 0; i < len; i += (h << 1)) {
            for (int j = 0; j < h; ++j) {
                ll x = a[i + j], y = a[i + j + h];
                a[i + j] = x + y, a[i + j + h] = x - y;
                if (inv == -1)
                    a[i + j] /= 2, a[i + j + h] /= 2;
            }
        }
    }
}
```

1.2 多项式

1.2.1 FFT

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

using db = double;
using ll = long long;

int mod;

namespace FFT {
    const db pi = acos(-1.0);
    struct Comp {
        db x, y;
        Comp() { x = 0.0, y = 0.0; }
        Comp(db _x,db _y):x(_x),y(_y) {}
        Comp operator + (const Comp&rhs)const {
            return Comp(x+rhs.x, y+rhs.y);
        }
        Comp operator - (const Comp&rhs)const {
            return Comp(x-rhs.x, y-rhs.y);
        }
        Comp operator * (const Comp&rhs)const {
            return Comp(x*rhs.x-rhs.y*y, x*rhs.y+y*rhs.x);
        }
    };
    Comp conj(const Comp&rhs) {
        return Comp(rhs.x, -rhs.y);
    }
    Comp exp_i(const db &x) {
        return Comp(cos(x), sin(x));
    }

    const int L = 21, N = 1 << L;
    Comp roots[N];
    int rev[N];
    struct _init {
        _init() {
            for(int i = 0; i < N; i++) {
                rev[i] = (rev[i>>1]>>1)|((i&1)<<L-1);
            }
            roots[1] = {1, 0};
            for(int i = 1; i < L; i++) {
                db angle = 2*pi/(1<<i+1);
                for(int j = (1<<i-1); j < (1<<i); j++) {
                    roots[j<<1] = roots[j];
                    roots[j<<1|1] =
    ↪ exp_i((j*2+1-(1<<i))*angle);
                }
            }
        }
    } init;
    inline void trans(Comp &a, Comp &b, const Comp &c) {
        Comp d = b * c;
        b = a - d;
        a = a + d;
    }
    void fft(vector<Comp> &a, int n) {
        assert((n & (n - 1)) == 0);
        int zeros = __builtin_ctz(n), shift = L - zeros;
        for(int i = 0; i < n; i++) {
            if(i < (rev[i]>>shift)) {
                swap(a[i], a[rev[i]>>shift]);
            }
        }
        for(int i = 1; i < n; i <<= 1)
            for(int j = 0; j < n; j += i * 2)
                for(int k = 0; k < i; k++)
                    trans(a[j+k], a[i+j+k], roots[i+k]);
    }

    vector<Comp> fa, fb;
    vector<int> multiply(const vector<int> &a, const
    ↪ vector<int> &b) {
```

```
    // 三次变两次
    int la = a.size(), lb = b.size();
    int need = la + lb - 1, n = 1 << (32 - __builtin_clz(need - 1));
    if(n > fa.size()) fa.resize(n);
    for(int i = 0; i < n; i++) {
        fa[i] = Comp(i < la ? a[i] : 0, i < lb ? b[i] : 0);
    }
    fft(fa, n);
    Comp r(0, -0.25/n);
    for(int i = 0, j = 0; i <= (n>>1); i++, j = n - i) {
        Comp x = fa[i] * fa[i], y = fa[j] * fa[j];
        if(i != j) fa[j] = (x - conj(y)) * r;
        fa[i] = (y - conj(x)) * r;
    }
    fft(fa, n);
    vector<int> c(need);
    for(int i = 0; i < need; i++) c[i]=fa[i].x+0.5;
    return c;
  }

  vector<ll> multiply(const vector<ll> &a, const vector<ll> &b) {
    // 朴素三次
    int la = a.size(), lb = b.size();
    int need = la + lb - 1, n = 1 << (32 - __builtin_clz(need - 1));
    if(n > fa.size()) fa.resize(n), fb.resize(n);
    for(int i = 0; i < n; i++) {
        fa[i] = Comp(i < la ? a[i] : 0, 0);
        fb[i] = Comp(i < lb ? b[i] : 0, 0);
    }
    fft(fa, n);
    fft(fb, n);
    for(int i = 0; i < n; i++) {
        fa[i] = fa[i] * fb[i];
    }
    reverse(fa.begin() + 1, fa.begin() + n);
    fft(fa, n);
    Comp r(1.0 / n, 0);
    for(int i = 0; i < n; i++) fa[i] = fa[i] * r;
    vector<ll> c(need);
    for(int i = 0; i < need; i++) c[i]=ll(fa[i].x+0.5);
    return c;
  }

  const int M = (1 << 15) - 1;
  vector<int> multiply_mod(const vector<int> &a, const vector<int> &b, bool eq = 0) {
    // (a, b) * (c, d)
    int la = a.size(), lb = b.size();
    int need = la + lb - 1, n = need > 1 ? 1 << (32 - __builtin_clz(need - 1)) : 1;
    if(fa.size() < n) fa.resize(n);
    if(fb.size() < n) fb.resize(n);
    for(int i = 0; i < n; i++) {
        fa[i] = i < la ? Comp(a[i]>>15, a[i]&M) : Comp(0, 0);
    }
    fft(fa, n);
    if(eq) copy(fa.begin(),fa.begin()+n, fb.begin());
    else {
        for(int i = 0; i < n; i++) {
            fb[i] = i < lb ? Comp(b[i]>>15, b[i]&M) : Comp(0, 0);
        }
        fft(fb, n);
    }
    Comp r(0.5/n,0);
    for(int i = 0, j = 0; i <= (n>>1); i++, j = n - i) {
        Comp x = (fa[i]+conj(fa[j]))*fb[i]*r; // (a, 0)*(c, d)
        Comp y = (fa[i]-conj(fa[j]))*conj(fb[j])*r; // (0, b)*(c, d)
        if(i != j) {
            Comp _x = (fa[j]+conj(fa[i]))*fb[j]*r;
            Comp _y = (fa[j]-conj(fa[i]))*conj(fb[i])*r;
            fa[i] = _x, fb[i] = _y;
        }
        fa[j] = x, fb[j] = y;
    }
    fft(fa, n);
    fft(fb, n);
    vector<int> c(need);
    for(int i = 0; i < need; i++) {
        ll x = ll(fa[i].x + 0.5) % mod, y = ll(fa[i].y + 0.5) % mod;
        ll z = ll(fb[i].x + 0.5) % mod, w = ll(fb[i].y + 0.5) % mod;
        c[i] = ((x<<30) + z + (y + w << 15)) % mod;
    }
    return c;
  }
  vector<int> sqr_mod(const vector<int> &a) {
    return multiply_mod(a, a, 1);
  }
}
using FFT::multiply;
using FFT::multiply_mod;
using FFT::sqr_mod;


int main() {
  using namespace std;
  ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
  int n, m;
  cin >> n >> m >> mod;
  vector<int> f(n + 1), g(m + 1);
  for(auto &x : f) cin >> x;
  for(auto &x : g) cin >> x;
  vector<int> h = multiply_mod(f, g);
  for(auto &x : h) cout << x << " ";
  cout << endl;
}
```

1.2.2 NTT

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

using ll = unsigned long long;
using Int = unsigned;

namespace Polynomial {

using Poly = std::vector<Int>;
constexpr Int P(998244353), G(3);
inline void inc(Int &x, Int y, Int mod = P) {
    (x += y) >= mod ? x -= mod : 0;
}

inline Int fpow(Int x, Int k = P - 2, Int mod = P) {
    Int r = 1;
    for (; k; k >>= 1, x = (ll) x * x % mod)
        if (k & 1) r = (ll) r * x % mod;
    return r;
```

```cpp
}

const int MAXW = 1 << 22;
Int w[MAXW];
struct omegaGen {
    omegaGen() {
        Int x = fpow(G, (P - 1) / MAXW);
        for(int i = MAXW >> 1; i; i >>= 1) {
            w[i] = 1;
            for(int j = 1; j < i; ++j)
                w[i + j] = (ll) w[i + j - 1] * x % P;
            x = (ll) x * x % P;
        }
    }
} gen;

Poly &operator *= (Poly &a, Int b) {
    for(auto &x : a)
        x = (ll) x * b % P;
    return a;
}
Poly operator * (Poly a, Int b) { return a *= b; }
Poly operator * (Int a, Poly b) { return b * a; }
Poly &operator /= (Poly &a, Int b) { return a *= fpow(b); }
Poly operator / (Poly a, Int b) { return a /= b; }
Poly &operator += (Poly &a, Poly b) {
    a.resize(std::max(a.size(), b.size()));
    for(size_t i = 0; i < b.size(); i++)
        inc(a[i], b[i]);
    return a;
}
Poly operator + (Poly a, Poly b) { return a += b; }
Poly &operator -= (Poly &a, Poly b) {
    a.resize(std::max(a.size(), b.size()));
    for(size_t i = 0; i < b.size(); i++)
        inc(a[i], P - b[i]);
    return a;
}
Poly operator - (Poly a, Poly b) { return a - b; }
Poly operator - (Poly a) {
    for(auto &x : a)
        if(x) x = P - x;
    return a;
}
Poly &operator >>= (Poly &a, Int x) {
    if (x >= (Int)a.size()) {
        a.clear();
    } else {
        a.erase(a.begin(), a.begin() + x);
    }
    return a;
}
Poly &operator <<= (Poly &a, Int x) {
    a.insert(a.begin(), x, 0);
    return a;
}
Poly operator >> (Poly a, Int x) { return a >>= x; }
Poly operator << (Poly a, Int x) { return a <<= x; }

Poly &cdot(Poly &a, Poly b) {
    assert(a.size() == b.size());
    for (size_t i = 0; i < a.size(); i++)
        a[i] = (ll) a[i] * b[i] % P;
    return a;
}
Poly dot(Poly a, Poly b) { return cdot(a, b); }
void norm(Poly &a) {
    if (!a.empty()) {
        a.resize(1 << std::__lg(a.size() * 2 - 1));
    }
}

void dft(Int *a, int n) {
    //assert((n & n - 1) == 0);
    for(int k = n >> 1; k; k >>= 1) {
        for(int i = 0; i < n; i += k << 1) {
            for(int j = 0; j < k; j++) {
                Int x = a[i + j], y = a[i + j + k], ww =
    ↪ w[k + j];
                a[i + j] = x + y >= P ? x + y - P : x + y;
                a[i + j + k] = (ll) (x - y + P) * ww % P;
            }
        }
    }
}

void idft(Int *a, int n) {
    //assert((n & n - 1) == 0);
    for(int k = 1; k < n; k <<= 1) {
        for(int i = 0; i < n; i += k << 1) {
            for(int j = 0; j < k; j++) {
                Int x = a[i + j], y = (ll) a[i + j + k] *
    ↪ w[k + j] % P;
                if(x >= P) x -= P;
                a[i + j + k] = x - y + P;
                a[i + j] = x + y;
            }
        }
    }
    for (Int i = 0, inv = P - (P - 1) / n; i < n; i++)
        a[i] = (ll) a[i] * inv % P;
    std::reverse(a + 1, a + n);
}


void dft(Poly &a) { dft(a.data(), a.size()); }
void idft(Poly &a) { idft(a.data(), a.size()); }

Poly operator* (Poly a, Poly b) {
    if(a.empty() || b.empty()) return {};
    int len = a.size() + b.size() - 1;
    if(a.size() <= 16 || b.size() <= 16) {
        Poly c(len);
        for(size_t i = 0; i < a.size(); i++)
            for(size_t j = 0; j < b.size(); j++)
                c[i + j] = (c[i + j] + ll(a[i]) * b[j]) %
    ↪ P;
        return c;
    }
    int n = 1 << std::__lg(len - 1) + 1;
    if (a != b) {
        a.resize(n), b.resize(n);
        dft(a), dft(b);
        cdot(a, b);
    } else {
        a.resize(n), dft(a);
        cdot(a, a);
    }
    idft(a);
    a.resize(len);
    return a;
}

// return: inv(a) mod x ^ n, n = a.size()
// require n = 2 ^ k
// resize: n' = 1 << lg(2 * n - 1) (n >= 1)
Poly inverse(Poly a) {
    int n = a.size();
    assert((n & n - 1) == 0);
    if (n == 1) return {fpow(a[0])};
    int m = n >> 1;
    Poly b = inverse(Poly(a.begin(), a.begin() + m)), c =
    ↪ b;
    b.resize(n);
```

```cpp
    dft(a), dft(b), cdot(a, b), idft(a);
    for (int i = 0; i < m; i++) a[i] = 0;
    for (int i = m; i < n; i++) a[i] = P - a[i];
    dft(a), cdot(a, b), idft(a);
    for (int i = 0; i < m; i++) a[i] = c[i];
    return a;
}

// return x s.t. x^2 = a mod x^n, n = a.size()
// not hold x is lexcicalgraphically smallest
// no need (n & (n - 1)) == 0
// require a[0] != 0
Poly polysqrt(Poly a) {
    int n = a.size();
    if (n == 1) return {Modroot::calc(2, a[0], P)};

    int m = ((n + 1) >> 1);
    Poly b = polysqrt(Poly(a.begin(), a.begin() + m));
    b.resize(n);

    Poly c = b * b;
    c.resize(n);

    b *= 2;
    c += a;

    norm(b);

    c = c * inverse(b);
    c.resize(n);

    return c;
}

// return: q(len = n - m + 1), a = b * q + r
Poly operator/ (Poly a, Poly b) {
    int n = a.size(), m = b.size();
    if (n < m) return {0};
    int k = 1 << std::__lg(n - m << 1 | 1);
    std::reverse(a.begin(), a.end());
    std::reverse(b.begin(), b.end());
    a.resize(k), b.resize(k), b = inverse(b);
    a = a * b;
    a.resize(n - m + 1);
    std::reverse(a.begin(), a.end());
    return a;
}

// return: {q(len = n - m + 1), r(len = m - 1)}
// require: b.size() > 0
std::pair<Poly, Poly> operator% (Poly a, Poly b) {
    int m = b.size();
    Poly q = a / b;
    b = b * q;
    a.resize(m - 1);
    for (int i = 0; i < m - 1; i++) inc(a[i], P - b[i]);
    return {q, a};
}

Poly der(Poly a) {
    int sz = a.size();
    for(int i = 0; i + 1 < sz; i++)
        a[i] = (ll) (i + 1) * a[i + 1] % P;
    a.pop_back();
    return a;
}

std::vector<Int> inv = {1, 1};
void updateInv(Int n) {
    if (inv.size() <= n) {
        Int p = inv.size();
        inv.resize(n + 1);
        for (Int i = p; i <= n; i++)
            inv[i] = (ll) (P - P / i) * inv[P % i] % P;
    }
}

Poly integ(Poly a, Int c = 0) {
    int n = a.size();
    updateInv(n);
    a.resize(n + 1);
    for (int i = n - 1; i >= 0; i--)
        a[i + 1] = (ll) inv[i + 1] * a[i] % P;
    a[0] = c;
    return a;
}

// return: ln(a) mod x ^ n, n = a.size()
Poly ln(Poly a) {
    int n = a.size();
    norm(a);
    assert(a[0] == 1);
    a = inverse(a) * der(a);
    a.resize(n - 1);
    return integ(a);
}

// un-checked
Poly exp(Poly a) {
    int n = a.size();
    assert((n & n - 1) == 0);
    assert(a[0] == 0);
    if (n == 1) return {1};
    int m = n >> 1;
    Poly b = exp(Poly(a.begin(), a.begin() + m)), c;
    b.resize(n), c = ln(b);
    a.resize(n << 1), b.resize(n << 1), c.resize(n << 1);
    dft(a), dft(b), dft(c);
    for (int i = 0; i < n << 1; i++) a[i] = (ll(1) + P +
    ↪ a[i] - c[i]) * b[i] % P;
    idft(a);
    a.resize(n);
    return a;
}

// checked
Poly power(Poly a, Int k, Int kreal) {
    int n = a.size();
    long long d = 0;
    while (d < n && !a[d]) d++;
    if (d == n) return a;
    a >>= d;
    Int b = fpow(a[0]);
    norm(a *= b);
    a = exp(ln(a) * k) * fpow(b, P - 1 - k % (P - 1));
    a.resize(n);
    d *= kreal;
    for (int i = n - 1; i >= d; i--) a[i] = a[i - d];
    d = std::min(d, (long long) n);
    for(int i = d - 1; i >= 0; --i) a[i] = 0;
    return a;
}

// k1 = k % (P - 1), k2 = k % P
// kreal = min(k, P)
Poly power(Poly a, Int k1, Int k2, Int kreal) {
    int n = a.size();
    long long d = 0;
    while (d < n && !a[d]) d++;
    if (d == n) return a;
    a >>= d;
    Int b = fpow(a[0]);
    norm(a *= b);
    a = exp(ln(a) * k2) * fpow(b, P - 1 - k1 % (P - 1));
```

```cpp
    a.resize(n);
    d *= kreal;
    for (int i = n - 1; i >= d; i--) a[i] = a[i - d];
    d = std::min(d, (long long) n);
    for(int i = d - 1; i >= 0; --i) a[i] = 0;
    return a;
}

// [x^n] f / g
Int divAt(Poly f, Poly g, ll n) {
    int len = std::max(f.size(), g.size());
    assert(len > 0);
    int m = 1 << std::__lg(2 * len - 1);
    len = m << 1;
    f.resize(len);
    for (; n; n >>= 1) {
        g.resize(len);
        dft(f), dft(g);
        for (int i = 0; i < len; ++i)
            f[i] = (ll) f[i] * g[i ^ 1] % P;
        for (int i = 0; i < m; ++i)
            g[i] = (ll) g[i << 1] * g[i << 1 | 1] % P;
        g.resize(m);
        idft(f), idft(g);
        for (int i = 0; i < m; ++i) f[i] = f[i * 2 + (n & 1)];
        fill(f.begin() + m, f.end(), 0);
    }
    return (ll) f[0] * fpow(g[0]) % P;
}

}  // namespace Polynomial

using namespace Polynomial;
using namespace std;

/*
int main() {
    ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
    int n;
    cin >> n;
    string s;
    cin >> s;
    Int k2 = 0, k1 = 0, kreal = 0;
    for(int i = 0, w = s.size(); i < s.size(); i++) {
        --w;
        inc(k2, (LL) (s[i] - '0') * fpow(10, w) % P);
        inc(k1, (LL) (s[i] - '0') * fpow(10, w, P - 1) % (P - 1), P - 1);
        if(kreal * 10 + (s[i] - '0') <= P) kreal = kreal * 10 + (s[i] - '0');
    }
    Poly a(n);
    for(int i = 0; i < n; ++i) cin >> a[i];
    Poly b = power(a, k1, k2, kreal);
    for(int i = 0; i < n; ++i) cout << b[i] << " \n"[i == n - 1];
} */

Int read() {
    int x;
    cin >> x;
    x %= 998244353;
    x += 998244353;
    x %= 998244353;
    return x;
}

int main() {
    ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
    ll n;
    int k;
    cin >> n >> k;
    Poly f(k + 1), a(k);
    f[0] = 1;
    for(int i = 1; i <= k; ++i) {
        f[i] = read();
        if(f[i]) f[i] = P - f[i];
    }
    for(auto &x : a) {
        x = read();
    }
    Poly b = a * f;
    b.resize(k);
    Int ans = divAt(b, f, n);
    cout << ans << endl;
}
```

1.2.3 拉格朗日插值

$$f(x)=\sum_i f\left(x_i\right)\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}$$

1.2.4 连通图计数

设大小为n的满足一个限制P的简单无向图数量为g_n，满足限制P且连通的简单无向图数量为f_n，如果已知$g_{1\dots n}$求f_n，可以得到递推式

$$f_n=g_n-\sum_{k=1}^{n-1}\binom{n-1}{k-1}f_k g_{n-k}$$

这个递推式的意义就是用任意图的数量减掉不连通的数量，而不连通的数量可以通过枚举1号点所在连通块大小来计算.

注意，由于$f_0=0$，因此递推式的枚举下界取0和1都是可以的.

推一推式子会发现得到一个多项式求逆，再仔细看看，其实就是一个多项式ln.

1.3 线性代数

1.3.1 高斯消元

辗转相除高斯消元

对于定义在环上而非域上的矩阵，利用辗转相除进行消元

下面例子中 mod 未必为素数，通过辗转相除消元求行列式

```cpp
using ll = long long;

int gauss(ll a[][N], int n) {
    int ans = 1;
    for (int j = 0; j < n; ++j) {
        for (int i = j + 1; i < n; ++i) {
            while (a[i][j]) {
                int t = a[j][j] / a[i][j];
                for (int k = j; k < m; ++k)
                    a[j][k] = (a[j][k] + (mod - t) * a[i][k]) % mod;
                swap(a[i], a[j]);
                ans = mod - ans;
            }
        }
    }
    for (int i = 0; i < n; ++i)
        ans = ans * a[i][i] % mod;
    return ans;
}
```

求秩与解线性方程组

分别维护行列指针，维护每一列所对应的有效行，有效行的总数即为秩

线性方程组有唯一解当且仅当 列满秩 且 系数矩阵的秩 等于 增广矩阵的秩

```cpp
// 消增广矩阵，注意第 m 行存放目标向量
// 若无解返回空 vector
// 若有无穷多解，则非主元置 0

vector<ll> gauss(ll a[][N], int n, int m) {
    vector<int> row(m, -1);  // 每个变元所对应的有效行
    vector<ll> ans(m, 0);

    int r = 0;
    for (int c = 0; c < m; ++c) { // 扫描每一列，用 r 维护
        int sig = -1;
        for (int i = r; i < n; ++i)
            if(a[i][c]) {
                sig = i; break;
            }
        if (sig == -1) continue; // 无效列

        row[c] = r;
        if (sig != r)
            swap(a[sig], a[r]);

        ll inv = fpow(a[r][c], mod - 2);
        for(int i = 0; i < n; ++i) {
            if (i == r) continue;
            ll del = inv * a[i][c] % mod;
            for (int j = c; j <= m; ++j)
                a[i][j] = (a[i][j] + (mod - del) * a[r][j]) % mod;
        }
        ++r;
    }

    if (r < n && a[r][m]) {
        cerr << "no solution!" << endl;
        return {};
    }

    for (int i = 0; i < m; ++i) { // ax = b
        if (row[i] != -1) {
            ans[i] = fpow(a[row[i]][i]) * a[row[i]][m] % mod;
        } else {
            ans[i] = 0; // 非主元置 0 即为合法解
        }
    }

    return ans;
}
```

矩阵求逆

维护一个矩阵 B，初始设为 n 阶单位矩阵，在消元的同时对 B 进行一样的操作，当把 A 消成单位矩阵时 B 就是逆矩阵.

1.3.2 矩阵树定理

无向图：设图G的基尔霍夫矩阵$L(G)$等于度数矩阵减去邻接矩阵，则G的生成树个数等于$L(G)$的任意一个代数余子式的值.
有向图：类似地定义$L_{in}(G)$等于**入度**矩阵减去邻接矩阵(i指向j有边，则$A_{i,j}=1$)，$L_{out}(G)$等于**出度**矩阵减去邻接矩阵.
则以i为根的内向树个数即为L_{out}的第i个主子式(即关于第i行第i列的余子式)，外向树个数即为L_{in}的第i个主子式.
(可以看出，只有无向图才满足$L(G)$的所有代数余子式都相等.)
BEST定理(有向图欧拉回路计数)：如果G是有向欧拉图，则G的欧拉回路的个数等于以一个任意点为根的内/外向树个数乘以$\prod_v(\deg(v)-1)!$.
并且在欧拉图里，无论以哪个结点为根，也无论内向树还是外向树，个数都是一样的.
另外无向图欧拉回路计数是NP问题.

1.3.3 线性基

```cpp
using ll = long long;

const int N = 70;
const int Lg = 60;

ll bs[N];

// find x in S s.t. k ^ x >= low and k ^ x is minimum,
// !!! return k ^ x, not x
// if not exist, return inf
ll lower(ll k, ll low) {

    // ll x = k ^ low; // expected value in S
    // ll res = 0; // value represented in S
    ll x = 0;
    ll ex = k ^ low;
    int lb = -1;

    // 在前缀可表示的范围内的寻找:
    // 对于范围内 low[i] = 0 的位,
    // 考虑另一分支的可行性
    for (int i = Lg - 1; i >= 0; i--) {
        if (((low >> i) & 1ll) == 0) {      // another branch
            int d = ((ex ^ x) >> i) & 1ll; // cur branch
            if (d || bs[i]) {
                lb = i;
            }
        }

        if (((ex ^ x) >> i) & 1ll) {
            if (bs[i]) {
                x ^= bs[i];
            } else {
                break;
            }
        }
    }

    if ((ex ^ x) == 0) // 可表示
        return k ^ x;

    if (lb == -1) // 不可表示，且不可更大
        return inf;

    if ((((k ^ x ^ low) >> lb) & 1ll) == 0)
        x ^= bs[lb];

    for (int i = lb - 1; i >= 0; i--) {
        if (((k ^ x) >> i) & 1ll) {
            if (bs[i]) {
                x ^= bs[i];
            }
        }
    }

    return k ^ x;
}
```

1.4 反演与容斥

1.4.1 二项式反演

形式一：

$$f(n)=\sum_{i=m}^{n}\binom{n}{i} \iff g(i)g(n)=\sum_{i=m}^{n}(-1)^{n-i}\binom{n}{i}f(i)$$

形式二: (常用)

$$f(n)=\sum_{i=n}^{m}\binom{i}{n}g(i) \iff g(n)=\sum_{i=n}^{m}(-1)^{i-n}\binom{i}{n}f(i)$$

常见用法
钦定 (至少) k 个与恰好 k 个之间的转化
记号:
记 $f(n)$ 表示先钦定至少选 n 个, 再统计钦定情况如此的方案数之和, 其中会包含重复的方案数.
记 $g(n)$ 表示恰好选 n 个的方案数, 不会重复.
那么, 对于 $i\ge n$, $g(i)$ 在 $f(n)$ 中被重复计算了 $\binom{i}{n}$ 次, 故 $f(n)=\sum_{i=n}^{m}\binom{i}{n}g(i)$, 其中 m 为数目上限
通常, f 较易求, 再通过反演即可求 g

1.4.2 单位根反演

$$[n|k]=\frac{1}{n}\sum_{i=0}^{n-1}\omega_n^{ik}$$

$$\sum_{i\ge 0}[x^{ik}]f(x)=\frac{1}{k}\sum_{j=0}^{k-1}f(\omega_k^j)$$

1.4.3 Min-Max容斥

设有数集 S, 有:

$$\max(S)=\sum_{\varnothing\ne T\subseteq S}(-1)^{|T|-1}\min(T)$$

$$\min(S)=\sum_{\varnothing\ne T\subseteq S}(-1)^{|T|-1}\max(T)$$

第 k 大数的 Min-Max 容斥

$$\max_k(S)=\sum_{T\subseteq S,|T|\ge k}(-1)^{|T|-k}\times\binom{|T|-1}{k-1}\times\min(T)$$

1.5 组合数

1.5.1 常用组合数

卡特兰数

$$C_n=\frac{1}{n+1}\binom{2n}{n}=\binom{2n}{n}-\binom{2n}{n-1}$$

- n个元素按顺序入栈, 出栈序列方案数
- 长为$2n$的合法括号序列数
- $n+1$个叶子的满二叉树个数

递推式:

$$C_n=\sum_{i=0}^{n-1}C_iC_{n-i-1}$$

$$C_n=C_{n-1}\frac{4n-2}{n+1}$$

普通生成函数:

$$C(x)=\frac{1-\sqrt{1-4x}}{2x}$$

扩展: 如果有n个左括号和m个右括号, 方案数为

$$\binom{n+m}{n}-\binom{n+m}{m-1}$$

施罗德数

$$S_n=S_{n-1}+\sum_{i=0}^{n-1}S_iS_{n-i-1}$$

$$(n+1)s_n=(6n-3)s_{n-1}-(n-2)s_{n-2}$$

其中S_n是(大)施罗德数, s_n是小施罗德数(也叫超级卡特兰数).
除了$S_0=s_0=1$以外, 都有$S_i=2s_i$.
施罗德数的组合意义:

- 从$(0,0)$走到(n,n), 每次可以走右, 上, 或者右上一步, 并且不能超过$y=x$这条线的方案数
- 长为n的括号序列, 每个位置也可以为空, 并且括号对数和空位置数加起来等于n的方案数
- 凸n边形的**任意**剖分方案数

(有些人会把大(而不是小)施罗德数叫做超级卡特兰数.)
默慈金数

$$M_{n+1}=M_n+\sum_{i=0}^{n-1}M_iM_{n-1-i}=\frac{(2n+3)M_n+3nM_{n-1}}{n+3}$$

$$M_n=\sum_{i=0}^{\frac{n}{2}}\binom{n}{2i}C_i$$

在圆上的n个**不同的**点之间画任意条不相交(包括端点)的弦的方案数.
也等价于在网格图上, 每次可以走右上, 右下, 正右方一步, 且不能走到$y<0$的位置, 在此前提下从$(0,0)$走到$(n,0)$的方案数.
扩展: 默慈金数画的弦不可以共享端点. 如果可以共享端点的话是A054726, 后面的表里可以查到.

1.5.2 斯特林数

1. **第一类斯特林数**
$\begin{bmatrix}n\\k\end{bmatrix}$表示$n$个元素划分成$k$个轮换的方案数.
递推式: $\begin{bmatrix}n\\k\end{bmatrix}=\begin{bmatrix}n-1\\k-1\end{bmatrix}+(n-1)\begin{bmatrix}n-1\\k\end{bmatrix}$.
求同一行: 分治FFT $O(n\log^2 n)$, 或者倍增$O(n\log n)$(每次都是$f(x)=g(x)g(x+d)$的形式, 可以用$g(x)$反转之后做一个卷积求出后者).

$$\sum_{k=0}^{n}\begin{bmatrix}n\\k\end{bmatrix}x^k=\prod_{i=0}^{n-1}(x+i)$$

求同一列: 用一个轮换的指数生成函数做k次幂

$$\sum_{n=0}^{\infty}\begin{bmatrix}n\\k\end{bmatrix}\frac{x^n}{n!}=\frac{(\ln(1-x))^k}{k!}=\frac{x^k}{k!}\left(\frac{\ln(1-x)}{x}\right)^k$$

2. **第二类斯特林数**
$\begin{Bmatrix}n\\k\end{Bmatrix}$表示$n$个元素划分成$k$个子集的方案数.
递推式: $\begin{Bmatrix}n\\k\end{Bmatrix}=\begin{Bmatrix}n-1\\k-1\end{Bmatrix}+k\begin{Bmatrix}n-1\\k\end{Bmatrix}$.
求一个: 容斥, 狗都会做

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

求同一行: FFT, 狗都会做
求同一列: 指数生成函数

$$\sum_{n=0}^{\infty}\begin{Bmatrix}n\\k\end{Bmatrix}\frac{x^n}{n!}=\frac{(e^x-1)^k}{k!}=\frac{x^k}{k!}\left(\frac{e^x-1}{x}\right)^k$$

普通生成函数

$$\sum_{n=0}^{\infty}\begin{Bmatrix}n\\k\end{Bmatrix}x^n=x^k\left(\prod_{i=1}^{k}(1-ix)\right)^{-1}$$

3. **幂的转换**

上升幂与普通幂的转换

$$x^{\overline{n}} = \sum_k \left[{n \atop k}\right] x^k$$

$$x^n = \sum_k \left\{{n \atop k}\right\} (-1)^{n-k} x^{\overline{k}}$$

下降幂与普通幂的转换

$$x^n = \sum_k \left\{{n \atop k}\right\} x^{\underline{k}} = \sum_k \binom{x}{k} \left\{{n \atop k}\right\} k!$$

$$x^{\underline{n}} = \sum_k \left[{n \atop k}\right] (-1)^{n-k} x^k$$

另外，多项式的**点值**表示的每项除以阶乘之后卷上e^{-x}乘上阶乘之后是牛顿插值表示，或者不乘阶乘就是**下降幂**系数表示。反过来的转换当然卷上e^x就行了。原理是每次差分等价于乘以$(1-x)$，展开之后用一次卷积取代多次差分。

4. **斯特林多项式(斯特林数关于斜线的性质)**

定义：

$$\sigma_n(x) = \frac{\left[{x \atop n}\right]}{x(x-1)\dots(x-n)}$$

$\sigma_n(x)$的最高次数是x^{n-1}。(所以作为唯一的特例，$\sigma_0(x) = \frac{1}{x}$不是多项式。)

斯特林多项式实际上非常神奇，它与两类斯特林数都有关系。

$$\left[{n \atop n-k}\right] = n^{\underline{k+1}} \sigma_k(n)$$

$$\left\{{n \atop n-k}\right\} = (-1)^{k+1} n^{\underline{k+1}} \sigma_k(-(n-k))$$

不过它并不好求。可以$O(k^2)$直接计算前几个点值然后插值，或者如果要推式子的话可以用后面提到的二阶欧拉数。

1.5.3 欧拉数

1. **欧拉数**

$\left\langle{n \atop k}\right\rangle$: n个数的排列，有k个上升的方案数。

$$\left\langle{n \atop k}\right\rangle = (n-k)\left\langle{n-1 \atop k-1}\right\rangle + (k+1)\left\langle{n-1 \atop k}\right\rangle$$

$$\left\langle{n \atop k}\right\rangle = \sum_{i=0}^{k+1} (-1)^i \binom{n+1}{i} (k+1-i)^n$$

$$\sum_{k=0}^{n-1} \left\langle{n \atop k}\right\rangle = n!$$

$$x^n = \sum_{k=0}^{n-1} \left\langle{n \atop k}\right\rangle \binom{x+k}{n}$$

$$k!\left\{{n \atop k}\right\} = \sum_{i=0}^{n-1} \left\langle{n \atop i}\right\rangle \binom{i}{n-k}$$

2. **二阶欧拉数**

$\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle$: 每个数都出现两次的多重排列，并且每个数两次出现之间的数都比它要大。在此前提下有k个上升的方案数。

$$\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = (2n-k-1)\left\langle\!\left\langle{n-1 \atop k-1}\right\rangle\!\right\rangle + (k+1)\left\langle\!\left\langle{n-1 \atop k}\right\rangle\!\right\rangle$$

$$\sum_{k=0}^{n-1} \left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = (2n-1)!! = \frac{(2n)^{\underline{n}}}{2^n}$$

3. **二阶欧拉数与斯特林数的关系**

$$\left\{{x \atop x-n}\right\} = \sum_{k=0}^{n-1} \left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle \binom{x+n-k-1}{2n}$$

$$\left[{x \atop x-n}\right] = \sum_{k=0}^{n-1} \left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle \binom{x+k}{2n}$$

1.6 线性规划

1.6.1 对偶原理

给定一个原始线性规划：

$$\begin{aligned} \text{Minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{Subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \\ & x_j \geq 0 \end{aligned}$$

定义它的对偶线性规划为：

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{m} b_i y_i \\ \text{Subject to} \quad & \sum_{i=1}^{m} a_{ij} y_i \leq c_j, \\ & y_i \geq 0 \end{aligned}$$

用矩阵可以更形象地表示为：

$$\begin{array}{rr} \text{Minimize} & \mathbf{c}^T\mathbf{x} \\ \text{Subject to} & A\mathbf{x} \geq \mathbf{b}, \\ & \mathbf{x} \geq 0 \end{array} \iff \begin{array}{rr} \text{Maximize} & \mathbf{b}^T\mathbf{y} \\ \text{Subject to} & A^T\mathbf{y} \leq \mathbf{c}, \\ & \mathbf{y} \geq 0 \end{array}$$

1.7 杂项

1.7.1 约瑟夫环

```
typedef long long ll;

ll k;

// 约瑟夫环
int main() {
    int T;
    cin >> T;
    ll n, m;
    for (int cas = 1; cas <= T; ++cas) {
        scanf("%lld %lld %lld", &n, &m, &k);
        ll res = (k-1) % (n-m+1);
        if (k == 1) res = (m - 1) % n;
        else for (ll i = n - m + 1, j, t; i < n; i = j) {
            if (i < k) {
                j = i + 1;
                res = (res + k) % j;
            } else {
                j = i + i / (k - 1);
                if (j > n) j = n;
                t = j - i;
                res -= j - t * k;
                if (res < 0) res += j;
                else res += res / (k - 1);
            }
        }
        printf("Case #%d: %lld\n", cas, res + 1);
```

```
    }
}
```

1.7.2 杨辉三角行区间和

```
using ll = long long;

// 边界从 (s, x) 移动到 (s + 1, nx)
ll move(int x, int nx, int s, ll sum) {
    assert(x >= -1);
    ll res = (2 * sum + mod - C(s, x)) % mod;
    while (x + 1 <= nx) {
        x++;
        res = (res + C(s + 1, x)) % mod;
    }
    while (x > nx) {
        res = (res + mod - C(s + 1, x)) % mod;
        x--;
    }
    return res;
};

void proc(int k) {
    // 第 s 行的 左右边界
    auto le = [&](int s) -> int {
        // ...
    };
    auto ri = [&](int s) -> int {
        // ...
    };
    // 这里应该暴力计算首行和当首行为 0 时也可以这样写
    ll lsum = move(-1, le(0), -1, 0);
    ll rsum = move(-1, ri(0), -1, 0);

    for (int s = 0; s <= k; s++) {
        if (le(s) < ri(s)) {
            // ... 第 s 行对答案的贡献
        }
        lsum = move(le(s), le(s + 1), s, lsum);
        rsum = move(ri(s), ri(s + 1), s, rsum);
    }
}
```

1.7.3 辛普森积分

```
// Adaptive Simpson's method : double simpson::solve (double (*f) (double), double l, double r, double eps) : integrates f over (l, r) with error eps.

double area (double (*f)(double), double l, double r) {
    double m = l + (r - l) / 2;
    return (f(l) + 4 * f(m) + f(r)) * (r - l) / 6;
}

double solve (double (*f) (double), double l, double r, double eps, double a) {
    double m = l + (r - l) / 2;
    double left = area(f, l, m), right = area(f, m, r);
    if (fabs(left + right - a) <= 15 * eps)
        return left + right + (left + right - a) / 15.0;
    return solve(f, l, m, eps / 2, left) + solve(f, m, r, eps / 2, right);
}

double solve (double (*f) (double), double l, double r, double eps) {
    return solve(f, l, r, eps, area (f, l, r));
}
```

1.7.4 纳什均衡

首先定义纯策略和混合策略：纯策略是指你一定会选择某个选项，混合策略是指你对每个选项都有一个概率分布p_i，你会以相应的概率选择这个选项.

考虑这样的游戏：有几个人(当然也可以是两个)各自独立地做决定，然后同时公布每个人的决定，而每个人的收益和所有人的选择有关.

那么纳什均衡就是每个人都决定一个混合策略，使得在其他人都是纯策略的情况下，这个人最坏情况下(也就是说其他人的纯策略最针对他的时候)的收益是最大的. 也就是说，收益函数对这个人的混合策略求一个偏导，结果是0(因为是极大值).

纳什均衡点可能存在多个，不过在一个双人**零和**游戏中，纳什均衡点一定唯一存在.

1.7.5 康托展开

求排列的排名：先对每个数都求出它后面有几个数比它小(可以用树状数组预处理)，记为c_i，则排列的排名就是

$$\sum_{i=1}^{n} c_i(n-i)!$$

已知排名构造排列：从前到后先分别求出c_i，有了c_i之后再用一个平衡树(需要维护排名)倒序处理即可.

1.7.6 常用NTT素数及原根

$p = r \times 2^k + 1$	r	k	最小原根
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
985661441	235	22	3
998244353	119	23	3
1004535809	479	21	3
1005060097*	1917	19	*5*
2013265921	15	27	*31*
2281701377	17	27	3
31525197391593473	7	52	3
180143985094819841	5	55	*6*
1945555039024054273	27	56	*5*
4179340454199820289	29	57	3

*注：1005060097有点危险，在变化长度大于$524288 = 2^{19}$时不可用.

2 数论

2.1 常见预处理与快速幂

```cpp
// 预处理组合数
const int N = 2e5 + 7;
const ll mod = 998244353;

ll fac[N], ifac[N];
void init() {
    fac[0] = 1;
    for (int i = 1; i < N; ++i) fac[i] = i * fac[i-1] % mod;
    ifac[N - 1] = fpow(fac[N - 1], mod - 2);
    for (int i = N - 1; i; --i) ifac[i - 1] = i * ifac[i] % mod;
}

ll C(int n, int k) {
    if (k < 0 || k > n) return 0;
    return (fac[n] * ifac[k] % mod) * ifac[n - k] % mod;
}

// 线性求逆元□注意有效的 i < mod
ll inv[maxn];
void init() {
    inv[0] = 0, inv[1] = 1;
    for (int i = 2; i < N; ++i)
        inv[i] = inv[mod % i] * (mod - mod / i) % mod;
}

// 快速幂
ll fpow(ll a, ll k = mod - 2, ll p = mod) {
    ll res = 1; a %= p;
    for (; k; k >>= 1, a = a * a % p) {
        if (k & 1)
            res = res * a % p;
    }
    return res;
}
```

2.2 因数分解与素性判定

2.2.1 朴素因数分解

```cpp
// 素因数分解
int p[maxn], l[maxn], cnt2 = 0;
void Fact(int x) {
    cnt2 = 0;
    for (int i = 2; 1ll * i * i <= x; i++) {
        if(x % i == 0) {
            p[++cnt2] = i; l[cnt2] = 0;
            while(x % i == 0) {
                x /= i; ++l[cnt2];
            }
        }
    }
    if (x != 1) {  // 则此时x一定是素数□且为原本x的大于根号x的唯一素因子
        p[++cnt2] = x; l[cnt2] = 1;
    }
}

// vector ver. 无次数
void Fact(ll x, vector<int>& fact) {
    for (ll i = 2; 1ll * i * i <= x; ++i) {
        if(x % i == 0) {
            fact.push_back(i);
            while(x % i == 0) x /= i;
        }
    }
    if (x != 1) fact.push_back(x);
}
```

2.2.2 Miller-Rabin 与 Pollard-Rho

```cpp
typedef long long ll;

// 注意在 MR 里的 fpow 模数超过 int 范围□需要开 __int128

ll mul(ll a, ll b, ll p) {
    return __int128(a) * b % p;
}

ll fpow(ll a, ll k, ll p) {
    ll res = 1; a %= p;
    for (; k; k >>= 1, a = mul(a, a, p)) {
        if (k & 1)
            res = mul(res, a, p);
    }
    return res;
}


ll randint(ll l, ll r) {
    static mt19937 eng(time(0));
    uniform_int_distribution<ll> dis(l, r);
    return dis(eng);
}

bool is_prime(ll x) {
    int s = 0; ll t = x - 1;
    if (x == 2) return true;
    if (x < 2 || !(x & 1)) return false;
    while (!(t & 1)) {   //将x分解成(2^s)*t的样子
        s++; t >>= 1;
    }
    ll lst[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    for(ll a : lst) { //随便选一个素数进行测试
        if(a >= x) break;
        ll b = fpow(a, t, x); //先算出a^t
        for (int j = 1; j <= s; ++j) { //然后进行s次平方
            ll k = mul(b, b, x);        //求b的平方
            if (k == 1 && b != 1 && b != x - 1) //用二次探测判断
                return false;
            b = k;
        }
        if (b != 1)
            return false; //用费马小定律判断
    }
    return true; //如果进行多次测试都是对的□那么x就很有可能是素数
}

ll gcd(ll a, ll b) { return b == 0 ? a : gcd(b, a % b); }

// @author: Pecco
ll Pollard_Rho(ll n) {
    if (n == 4) return 2;
    if (is_prime(n)) return n;
    while (1) {
        ll c = randint(1, n - 1); // 生成随机的c
        auto f = [=](ll x) { return ((__int128)x * x + c) % n; }; // lll表示__int128□防溢出
        ll t = f(0), r = f(f(0));
        while (t != r) {
            ll d = __gcd(abs(t - r), n);
            if (d > 1)
                return d;
```

```cpp
            t = f(t), r = f(f(r));
        }
    }
}


// 优化掉一个log
ll Pollard_Rho(ll n) {
    if (n == 4) return 2;
    if (is_prime(n)) return n;
    while (1) {
        ll c = randint(1, n - 1);
        auto f = [=](ll x) { return ((__int128)x * x + c) % n; };
        ll t = 0, r = 0, p = 1, q;
        do {
            for (int i = 0; i < 128; ++i) { // 令固定距离C=128
                t = f(t), r = f(f(r));
                if (t == r || (q = (lll)p * abs(t - r) % n) == 0) // 如果发现环或者积即将为0退出
                    break;
                p = q;
            }
            ll d = gcd(p, n);
            if (d > 1)
                return d;
        } while (t != r);
    }
}

vector<ll> factors;

// 将 n 进行素因子分解factors: 存放 n 的所有不去重素因子
void getfactors(ll n) {
    if (n == 1) return;
    if (is_prime(n)) { factors.push_back(n); return; } // 如果是质因子
    ll p = n;
    while (p == n)
        p = Pollard_Rho(n);
    getfactors(n / p), getfactors(p); //递归处理
}
```

2.3 筛法

2.3.1 线性筛

```cpp
const int maxn = 1000000 + 5;
bool isnt[maxn];
int prime[maxn];
int cnt = 0;

// 线性筛法 [1, n] 内素数
void Prime(int n) {
    isnt[1] = true;
    cnt = 0;
    for (int i = 2; i <= n; i++) {
        if (!isnt[i]) prime[++cnt] = i;
        for (int j = 1; j <= cnt; j++) {
            if (1ll * i * prime[j] > n) break;
            isnt[i * prime[j]] = 1;
            if (i % prime[j] == 0) break;
        }
    }
}

// 线性筛求积性函数
int phi[maxn], mu[maxn], d[maxn], D[maxn], q[maxn];
void Sieve(int n) {
    isnt[1] = true;
    phi[1] = 1;
    //mu[1] = 1;
    cnt = 0;
    for(int i = 2; i <= n; i++) {
        if (!isnt[i]) {
            prime[++cnt] = i;
            phi[i] = i - 1;
            //mu[i] = -1;
            // d[i] = 2; q[i] = 1;
            // D[i] = i + 1; q[i] = 1;
        }
        for (int j = 1; j <= cnt; j++) {
            int x = i * prime[j];
            if (x > n) break;
            isnt[x] = 1;
            if (i % prime[j] == 0) {
                phi[x] = phi[i] * prime[j];
                // mu[x] = 0;
                // d[x] = d[i] / (q[i] + 1) * (q[i] + 2), q[x] = q[i] + 1;
                // D[x] = D[i] / (prime[j] ^ (q[i] + 1) - 1) * (prime[j] ^ (q[i] + 2) - 1), q[x] = q[i] + 1;
                break;
            } else {
                phi[x] = phi[i] * (prime[j] - 1); // mu[x] = -mu[i]
                // d[x] = 2 * d[i], q[x] = 1;
                // D[x] = (prime[j] + 1) * D[i], q[x] = 1;
            }
        }
    }
}
```

2.3.2 Min25 筛

```cpp
using ll = long long;
using i128 = __int128;
//using i128 = int64_t;

const ll mod = 998244353;

namespace min25 {
    const int N = 1e6 + 10;
    ll po[40][N];
    inline ll fpow(ll e, ll k) {
        return po[e][k];
    }
    void precalc() {
        for(int e = 0; e < 40; ++e) {
            po[e][0] = 1;
            for(int i = 1; i < N; i++)
                po[e][i] = e * po[e][i - 1] % mod;
        }
    }
    ll n;
    int B;
    int _id[N * 2];
    inline int id(ll x) {
        return x <= B ? x : n / x + B;
    }
    inline int Id(ll x) {
        return _id[id(x)];
    }
    // f(p) = p - 1 = fh(p) - fg(p);
    inline ll fg(ll x) {
        // assert(x <= sqrt(n));
        return 1;
    }
    inline ll fh(ll x) {
        // assert(x <= sqrt(n));
```

```cpp
        return x;
    }
    // \sum_{i=2}^n fg(i)
    inline ll sg(ll x) {
        return (x - 1) % mod;
    }
    // \sum_{i=2}^n fh(i)
    inline ll sh(ll x) {
        return ((i128) x * (x + 1) / 2 + mod - 1) % mod;
    }
    // f(p^e)
    inline ll f(ll p, ll e) {
        //return (pe - pe / p) % mod;
        return (p + (mod - 2) * fpow(e, p)) % mod;
    }
    bitset<N> np;
    ll p[N>>2], pn;
    ll pg[N>>2], ph[N>>2];
    void sieve(ll sz) {
        for(int i = 2; i <= sz; i++) {
            if(!np[i]) {
                p[++pn] = i;
                pg[pn] = (pg[pn - 1] + fg(i)) % mod;
                ph[pn] = (ph[pn - 1] + fh(i)) % mod;
            }
            for(int j = 1; j <= pn && i * p[j] <= sz; j++) {
                np[i * p[j]] = 1;
                if(i % p[j] == 0) {
                    break;
                }
            }
        }
    }
    ll m;
    ll g[N * 2], h[N * 2];
    ll w[N * 2];

    void compress() {
        for (int i = 1; i <= m; i++) {
            g[i] = (h[i] + mod - g[i] + mod - g[i]) % mod;
        }
        for (int i = 1; i <= pn; i++) {
            pg[i] = (ph[i] + mod - pg[i] + mod - pg[i]) % mod;
        }
    }

    ll dfs_F(int k, ll n) {
        if (n < p[k] || n <= 1) return 0;
        ll res = g[Id(n)] + mod - pg[k - 1], pw2;
        for (int i = k; i <= pn && (pw2 = (ll) p[i] * p[i]) <= n; ++i) {
            ll pw = p[i];
            for (int c = 1; pw2 <= n; ++c, pw = pw2, pw2 *= p[i])
                res = (res + ((ll) f(p[i], c) * dfs_F(i + 1, n / pw) + f(p[i], c + 1))) % mod;
        }
        return res;
    }

    void init(ll _n) {
        n = _n;
        B = sqrt(n) + 100;
        pn = 0;
        sieve(B);
        m = 0;
        for(ll i = 1, j; i <= n; i = j + 1) {
            j = n / (n / i);
            ll t = n / i;
            _id[id(t)] = ++m;
            w[m] = t;
            g[m] = sg(t);
            h[m] = sh(t);
            //printf("id: %lld, w: %lld, g: %lld, h: %lld\n", m, t, g[m], h[m]);
        }
        for (int j = 1; j <= pn; j++) {
            ll z = (ll) p[j] * p[j];
            for(int i = 1; i <= m && z <= w[i]; i++) {
                int k = Id(w[i] / p[j]);
                g[i] = (g[i] + (ll) (mod - fg(p[j])) * (g[k] - pg[j - 1] + mod)) % mod;
                h[i] = (h[i] + (ll) (mod - fh(p[j])) * (h[k] - ph[j - 1] + mod)) % mod;
            }
        }
        compress();

        /* 递推 min25
        for(int j = pn; j > 0; j--) {
            ll z = (ll) p[j] * p[j];
            for(int i = 1; i <= m && z <= w[i]; i++) {
                ll pe = p[j];
                for(int e = 1; pe * p[j] <= w[i]; e++, pe *= p[j]) {
                    g[i] = (g[i] + (ll) f(p[j], e) * (g[Id(w[i] / pe)] - pg[j] + mod) + f(p[j], e + 1)) % mod;
                }
            }
        } */
    }
    ll get(ll x) { // x == n / i
        if(x < 1) return 0;
        return (dfs_F(1, x) + 1) % mod;
    }
    ll get(ll l, ll r) {
        return get(r) - get(l - 1);
    }
}

void Solve() {
    long long n;
    scanf("%lld", &n);
    min25::init(n);
    long long res = min25::get(n);
    printf("%lld\n", res);
}

int main() {
    min25::precalc();
    int T;
    scanf("%d", &T);
    while(T--) {
        Solve();
    }
}
```

2.4 扩展欧几里得

```cpp
using i128 = __int128;

// ax + by = c
// 有解当且仅当 gcd(a, b) | c
// 要求 a, b 不全为 0
// 无合法性检查
```

```cpp
void exgcd(i128 a, i128 b, i128 &x, i128 &y, i128 c = 1) {
    if (b == 0) {
        x = c / a;
        y = 0;
    } else {
        exgcd(b, a % b, x, y, c);
        i128 tmp = x;
        x = y;
        y = tmp - (a / b) * y;
    }
}
```

2.5 扩展欧拉定理

$$a^b \equiv a^{b \bmod \phi(p)},\ (a,b)=1$$

$$a^b \equiv a^{b \bmod \phi(p)+\phi(p)},\ (a,b)\neq 1$$

2.6 中国剩余定理

2.6.1 两个数的 crt

```cpp
// mul 表示慢速乘
ll crt(ll a1, ll p, ll a2, ll q) {
    ll ip = fpow(p, q-2, q);
    ll iq = fpow(q, p-2, p);
    ll n = p * q;
    return (mul(mul(a1, q, n), iq, n) + mul(mul(a2, p, n),
    ↪ ip, n)) % n;
}
```

2.6.2 excrt

```cpp
using ll = long long;

ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}

// x === a1 (mod b1), x === a2 (mod b2)
// 合法性检查：返回 -1 则为无解
pair<ll, ll> excrt(ll a1, ll b1, ll a2, ll b2) {
    ll g = gcd(b1, b2);
    ll lcm = (b1 / g) * b2;

    if ((a1 - a2) % g) return {-1, -1};

    i128 x, y;
    exgcd(b1, b2, x, y, a1 - a2);
    ll res = (a1 - b1 * x) % lcm;
    if (res < 0) res += lcm;
    return {res, lcm};
}
```

2.7 卢卡斯定理

2.7.1 模素数卢卡斯

```cpp
// 卢卡斯定理，要求 p 为素数
ll lucas(ll n, ll m, ll p) {
    if (!m) return 1;
    return C(n % p, m % p, p) * lucas(n / p, m / p, p) % p;
}
```

2.7.2 扩展卢卡斯

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;


// -p: 素因子
// -l: 次数
// return: 本质不同素因子个数
// 1-base
int Fact(int x, int *p, int *l);

ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}

void exgcd(ll a, ll b, ll &x, ll &y, ll c = 1);

// x === a1 (mod b1), x === a2 (mod b2)
// 合法性检查：返回 -1 则为无解
pair<ll, ll> excrt(ll a1, ll b1, ll a2, ll b2);


// --------------------------


// 扩展卢卡斯定理

// 扩欧求逆元
ll INV(ll a, ll p) {
    ll x, y;
    exgcd(a, p, x, y);
    return (x % p + p) % p;
}

// 递归求解(n! / px) mod pk
ll F(ll n, ll p, ll pk) {
    if (n == 0) return 1;
    ll rou = 1; // 循环节
    ll rem = 1; // 余项
    for (ll i = 1; i <= pk; ++i) {
        if (i % p)
            rou = rou * i % pk;
    }
    rou = fpow(rou, n / pk, pk);
    for (ll i = pk * (n / pk); i <= n; ++i) {
        if (i % p)
            rem = rem * (i % pk) % pk; // 小心i炸int
    }
    return F(n / p, p, pk) * rou % pk * rem % pk;
}

// 素数p在n!中的次数
ll G(ll n, ll p) {
    if (n < p) return 0;
    return G(n / p, p) + (n / p);
}

ll C_pk(ll n, ll m, ll p, ll pk) {
    ll fz = F(n, p, pk), fm1 = INV(F(m, p, pk), pk),
       fm2 = INV(F(n - m, p, pk), pk);
    ll mi = fpow(p, G(n, p) - G(m, p) - G(n - m, p), pk);
    return fz * fm1 % pk * fm2 % pk * mi % pk;
}

const int N = 107;

int ps[N], l[N];

ll exlucas(ll n, ll m, ll P) {
    int num = Fact(P, ps, l); // 素因子分解见素因子分
    ↪ 解.cpp
```

```cpp

    ll res = 0, mo = 1;
    for (int i = 1; i <= num; ++i) {
        ll pk = 1;
        for (int j = 0; j < l[i]; ++j) {
            pk *= ps[i];
        }

        ll x = C_pk(n, m, ps[i], pk);
        pair<ll, ll> pr = excrt(x, pk, res, mo);

        mo = pr.second;
        res = pr.first;
    }

    return res % P;
}

int main() {
    ll n, m, p; cin >> n >> m >> p;
    ll ans = exlucas(n, m, p);

    cout << ans << endl;
}
```

2.8 原根与离散对数

2.8.1 原根

```cpp
// 得到 p 的原根
ll generator(ll p) {
    static ll rec, ans;
    if (p == rec)
        return ans;
    rec = p;
    vector<ll> fact;
    ll phi = p - 1, n = phi;
    for (ll i = 2; 1ll * i * i <= n; ++i) {
        if (n % i == 0) {
            fact.push_back(i);
            while (n % i == 0)
                n /= i;
        }
    }
    if (n > 1)
        fact.push_back(n);
    for (ll res = 2; res <= p; ++res) {
        bool ok = 1;
        for (ll factor : fact) {
            if (fpow(res, phi / factor, p) == 1) {
                ok = false;
                break;
            }
        }
        if (ok)
            return ans = res;
    }
    return ans = -1;
}
```

2.8.2 BSGS

```cpp
// a ^ k == b mod p
ll BSGS(ll a, ll b, ll p) { // p <= 1e9
    static ll rec;
    static map<ll, ll> mp;

    ll sq = (ll)ceil(sqrt(p));
    if (rec != p) {
        rec = p;
        mp.clear();
        ll le = 1, bs = fpow(a, sq, p);
        for (ll i = 1; i <= sq; ++i) {
            le = le * bs % p;
            if (le < 0)
                le += p;
            mp[le] = i * sq;
        }
    }

    ll ri = (b % p);
    if (ri < 0)
        ri += p;
    for (ll j = 0; j <= sq; ++j) {
        if (mp.count(ri)) {
            return mp[ri] - j;
        }
        ri = ri * a % p;
        if (ri < 0)
            ri += p;
    }
    return -1;
}

// x ^ a == b mod p
ll calc(ll a, ll b, ll p) {
    ll g = generator(p); // 求原根-见原根.cpp
    ll ga = fpow(g, a, p);
    ll c = BSGS(ga, b, p);
    ll res = fpow(g, c, p);
    return res;
}
```

2.9 杂项

2.9.1 大数整除小数取模

计算 $\frac{a}{b} \bmod p$:
当 a 的本值太大无法表示时，可以计算 a 对 b * p 取模的结果，再除 b，模 p

$$\frac{a}{b} \bmod p = \frac{a \bmod b*p}{b} \bmod p$$

2.9.2 立方根复杂度求 mobius 函数

```cpp
int getmu(int x) {
    int pr, cur = 0;
    for (int i = 1; i <= cnt; ++i) {
        cur = 0;
        while (x % prime[i] == 0) {
            ++cur; x /= prime[i];
        }
        if (cur > 1) return 0;
    }
    if (x == 1) return 1;
    int sq = sqrt(x) + 0.5;
    if (1ll * sq * sq == x) return 0;
    return 1;
}
```

2.9.3 直接求 euler 函数

```cpp
// primes 为预处理的素数表
ll getPhi(ll x) {
    ll phi = x, num = x;
    for (int p : primes) {
        if (p > x / p) break;
        if (x % p == 0)
```

```
            phi = (phi / p) * (p - 1), x /= p;
        while (x % p == 0)
            x /= p;
    }
    if (x > 1)
        phi = phi / x * (x - 1);
    return phi;
}
```

3 数据结构

3.1 点分治

```
vector<int> vec[N];
bool vis[N];

namespace DFZ {

    int sz[N], maxx[N];

    int rt, sum;
    void calcsz(int x, int dad) {
        //cerr << rt << "   " << sum << endl;
        maxx[x] = 0; sz[x] = 1;
        for (int y : vec[x]) {
            if(y == dad || vis[y]) continue;
            calcsz(y, x);
            sz[x] += sz[y];
            maxx[x] = max(sz[y], maxx[x]);
        }
        assert(sum - sz[x] >= 0);
        maxx[x] = max(maxx[x], sum - sz[x]);
        if (maxx[x] < maxx[rt]) rt = x;
    }

    void dfz(int x) {
        calcsz(x, 0); vis[rt] = 1;
        TREE::init();
        TREE::dfs(rt, 0, 0, 0, 1);
        ALL += TREE::gao();
        // calc mx, dis, build seg treedp

        int cen = rt, psum = sum;
        for (int y : vec[cen]) {
            if (vis[y]) continue;
            rt = cen;
            sum = (sz[y] > sz[cen] ? psum - sz[cen] :
  ↪ sz[y]);
            dfz(y);
        }
    }
}

DFZ::rt = 0; DFZ::maxx[0] = inf; DFZ::sum = n;
DFZ::dfz(1);
```

3.2 树链剖分

```
vector<int> vec[N];

struct cutTree {
    int dep[N], sz[N];
    int nd[N], fn[N], gn[N], clc;
    int top[N], son[N], fa[N];

    void init(int n) {
        clc = 0;
        for(int x = 1; x <= n; x++) son[x] = 0;
        for(int x = 1; x <= n; x++) fa[x] = 0;
    }
    // dfs1(rt, 1);
    void dfs1(int x, int d) {
        dep[x] = d, sz[x] = 1;
        for(int y : vec[x]) {
            if(y == fa[x]) continue;
            fa[y] = x; dfs1(y, d + 1); sz[x] += sz[y];
            if(!son[x] || sz[y] > sz[son[x]]) son[x] = y;
        }
    }
    // dfs2(rt, rt);
    void dfs2(int x, int an) {
        top[x] = an;
        nd[fn[x] = ++clc] = x;
        if(son[x]) dfs2(son[x], an);
        for(int y : vec[x]) {
            if(y == fa[x] || y == son[x]) continue;
            dfs2(y, y);
        }
        gn[x] = clc;
    }
    int lca(int x, int y) {
        while(top[x] != top[y]) {
            if(dep[top[x]] > dep[top[y]])
                swap(x, y);
            y = fa[top[y]];
        }
        return dep[x] < dep[y] ? x : y;
    }
    // anc[x][0] = x, !! 默认根为 1, 若 rt 非 1, 需要传参
    int up(int x, int k, int rt = 1) {
        if(k < 0) return -1;
        if(k >= dep[x]) return 0;
        while(k) {
            if(fn[x] - fn[top[x]] < k)
                k -= fn[x] - fn[top[x]] + 1, x =
  ↪ fa[top[x]];
            else x = nd[fn[x] - k], k = 0;
        }
        return x;
    }
    int query(int x, int y) {
        int res = 0;
        while(top[x] != top[y]) {
            if(dep[top[x]] > dep[top[y]]) swap(x, y);
            int l = fn[top[y]], r = fn[y];
            res += bit.query(r) - bit.query(l - 1);
            y = fa[top[y]];
        }
        if(dep[x] > dep[y]) swap(x, y);
        return res + bit.query(fn[y]) - bit.query(fn[x] -
  ↪ 1);
    }
} CT;
```

3.3 主席树

```
const int N = 100010;
const int M = N * 40;

struct node {
    int l, r;
    ll val;
} tr[M];

int rt[N], tot;

int clone(int k) {
    ++tot;
    tr[tot] = tr[k];
    return tot;
}

void up(int k) {
    tr[k].val = tr[tr[k].l].val ^ tr[tr[k].r].val;
}

void upd(int &k, int x, int l, int r, ll val) {
    k = clone(k);
    if(l == r) {
        tr[k].val ^= val;
```

```cpp
    } else {
        int m = ((l + r) >> 1);
        if (x <= m) upd(tr[k].l, x, l, m, val);
        else upd(tr[k].r, x, m + 1, r, val);
        up(k);
    }
}

/* 树形建主席树 */
void dfs(int x, int dad) {
    rt[x] = rt[dad];
    upd(rt[x], a[x], 1, n, val);
    for (auto y : ch[x]) {
        if(y == dad) continue;
        dfs(y, x);
    }
}
```

3.4 线段树合并

```cpp
// merge v to u

void merge(int &u, int v, int l, int r) {
    if (!v) return;
    else if (!u) u = v;
    else if (l == r) {
        // merge at leaf
    } else {
        int mid = ((l + r) >> 1);
        merge(tr[u].l, tr[v].l, l, mid);
        merge(tr[u].r, tr[v].r, mid + 1, r);
        up(u);
    }
}
```

3.5 zkw 线段树

```cpp
// 0-base

struct SegT {
    typedef pair<int, int> T;
    T ini = {0, 0};
    T combine(T u, T v) {
        return max(u, v);
    }
    int n; vector<T> arr;

    // local
    SegT(int sz): n{1} { while(n < sz) n <<= 1;
    ↪ arr.resize(n * 2, ini); };

    // global
    void init(int sz) {
        arr.clear(); for (n = 1; n < sz; n <<= 1);
    ↪ arr.resize(n * 2, ini);
    }

    void update(int i, T v) {
        for (arr[i += n] = v; i >>= 1; )
            arr[i] = combine(arr[i<<1], arr[i<<1|1]);
    }
    T query (int l, int r) {
        T resl = ini, resr = ini;
        for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
            if (l & 1) resl = combine(resl, arr[l++]);
            if (r & 1) resr = combine(resr, arr[--r]);
        }
        return combine(resl, resr);
    }
};
```

3.6 矩形面积并

```cpp
// 矩形面积并，标记永久化，注意要开八倍空间

typedef long long ll;

const int maxn = 100050;

struct node {
    int sum, lazy;
    void clear() {
        sum = 0; lazy = 0;
    }
} tr[maxn << 3];

void up(int k, int l, int r) {
    if (tr[k].lazy) {
        tr[k].sum = r - l + 1;
    } else {
        tr[k].sum = tr[k<<1].sum + tr[k<<1|1].sum;
    }
}

void build(int k, int l, int r) {
    tr[k].clear();

    if (l == r) {
        tr[k<<1].clear();
        tr[k<<1|1].clear();
        return;
    }

    int mid = ((l + r) >> 1);
    build(k<<1, l, mid);
    build(k<<1|1, mid + 1, r);
    up(k, l, r);
}

void upd(int k, int cl, int cr, int tag, int l, int r) {
    if (cl <= l && r <= cr) {
        tr[k].lazy += tag;
        up(k, l, r);
    } else {
        int mid = ((l + r) >> 1);
        if(cl <= mid) upd(k<<1, cl, cr, tag, l, mid);
        if(cr > mid) upd(k<<1|1, cl, cr, tag, mid + 1, r);
        up(k, l, r);
    }
}

// 查询的时候记一下祖先结点是否有 lazy > 0
```

4 图论

4.1 2-sat

4.2 Dinic 网络流

```
using ll = long long;

// 前向链表多测注意清空□tot 初值要设为 1
struct edge {
    int to, pre;
    ll w;
} e[M];
int last[N], tot = 1;

void ine(int a, int b, ll w) {
    e[++tot] = (edge){b, last[a], w};
    last[a] = tot;
}

void add(int a, int b, ll w) {
    ine(a, b, w);
    ine(b, a, 0);
}

inline int sgn(ll v) {
    if (v == 0) return 0;
    return v > 0 ? 1 : -1;
}

namespace Dinic {

    int n, s, t;
    int lv[N], cur[M];  // lv□层数□cur□当前弧

    bool bfs()  {
        fill(lv + 1, lv + 1 + n, -1);
        lv[s] = 0;
        copy(last + 1, last + 1 + n, cur + 1);
        queue<int> q;
        q.push(s);
        while(!q.empty()) {
            int u = q.front(); q.pop();
            for(int i = cur[u]; i; i = e[i].pre) {
                int to = e[i].to;
                ll vol = e[i].w;
                if(vol > 0 && lv[to] == -1)
                    lv[to] = lv[u] + 1, q.push(to);
            }
        }
        return lv[t] != -1; // 如果汇点未访问过则不可达
    }

    ll dfs(int u = s, ll f = inf) {
        if(u == t)
            return f;
        for(int &i = cur[u]; i; i = e[i].pre) {
            int to = e[i].to;
            ll vol = e[i].w;
            if(vol > 0 && lv[to] == lv[u] + 1) {
                ll c = dfs(to, min(vol, f));
                if(sgn(c)) {
                    e[i].w -= c;
                    e[i ^ 1].w += c;    // 反向边
                    return c;
                }
            }
        }
        return 0; // 输出流量大小
    }

    ll dinic(int _n, int _s, int _t) {
        n = _n; s = _s; t = _t;
        ll ans = 0;
        while(bfs()) {
            ll f;
            while((f = dfs()) > 0)
                ans += f;
        }
        return ans;
    }
} // namespace Dinic

using Dinic::dinic;
```

4.3 Dijkstra 费用流

```
#include <bits/stdc++.h>
#include <ext/pb_ds/priority_queue.hpp>
using ll = long long;
using namespace std;
typedef pair<ll, ll> P;

const int N = 5e3 + 7, M = 1e6 + 7;
const ll inf = 0x3f3f3f3f3f3f3f3f;

struct edge {
    int to, pre;
    ll w, c; // w: 流量 weight□c: 费用 cost
} e[M * 2];
int head[N], ecnt = 1; // 加入的第一条边编号为 2
void ine(int u, int v, ll w, ll c) {
    e[++ecnt] = {v, head[u], w, c};
    head[u] = ecnt;
}
void add(int u, int v, ll w, ll c) {
    ine(u, v, w, c); ine(v, u, 0, -c);
}
void init(int n) {
    fill(head + 1, head + 1 + n, 0);
    ecnt = 1;
}

ll h[N], dis[N]; // bool vis[N];
int pe[N]; // 父边

// spfa 只用来预处理 h□不跑流□也可以用来跑流□见注释□
// h 在 flow_dij 调用时清空
void spfa(int s, int n) {
    static bool inq[N];

    fill(dis + 1, dis + 1 + n, inf);
    fill(inq + 1, inq + 1 + n, false);

    queue<int> q;

    q.push(s);
    dis[s] = 0;

    while (!q.empty()) {
        int x = q.front();
        q.pop();
        inq[x] = false;

        for (int i = head[x]; i; i = e[i].pre)
            if (e[i].w > 0) {
                int y = e[i].to;

                if (dis[y] > dis[x] + e[i].c) {
                    // pe[y] = i; 若要跑流□则加上这一行
                    dis[y] = dis[x] + e[i].c;
                    if (!inq[y]) {
```

```cpp
                        q.push(y);
                        inq[y] = true;
                    }
                }
            }
        }
}

void dij(int s, int n) {
    fill(dis + 1, dis + 1 + n, inf);
    dis[s] = 0;
    // fill(vis + 1, vis + 1 + n, false);

    using pq_t = __gnu_pbds::priority_queue<P, greater<P>, __gnu_pbds::thin_heap_tag>;
    pq_t q;
    static pq_t::point_iterator it[N];

    for (int i = 1; i <= n; i++)
        it[i] = q.push({dis[i], i});

    while(!q.empty()) {
        auto tp = q.top();
        int x = tp.second;
        ll w = tp.first;
        q.pop();

        if(w != dis[x]) continue;
        // if(vis[x]) continue;
        // vis[x] = true;

        for(int i = head[x]; i; i = e[i].pre) {
            int y = e[i].to;
            if(e[i].w > 0 && dis[y] > dis[x] + h[x] - h[y] + e[i].c) {
                pe[y] = i;
                dis[y] = dis[x] + h[x] - h[y] + e[i].c;
                q.modify(it[y], {dis[y], y});
            }
        }
    }
}

P flow_dij(int s, int t, int n) {
    fill(h + 1, h + 1 + n, 0);
    /*
    spfa(s, n);
    for(int i = 1; i <= n; i++)
        h[i] = dis[i];
    // 如果初始有负权就像这样跑一遍 SPFA 预处理 h 数组
    */

    ll cost = 0, flow = 0;

    while (1) {
        dij(s, n);
        if(dis[t] >= inf) break;

        for (int i = 1; i <= n; i++) // 先更新 h 数组
            h[i] += dis[i];

        ll nowflow = inf;
        for (int x = t; x != s; x = e[pe[x] ^ 1].to)
            nowflow = min(nowflow, e[pe[x]].w);

        flow += nowflow;
        cost += nowflow * h[t]; // 计算流量和费用

        for (int x = t; x != s; x = e[pe[x] ^ 1].to) {
            e[pe[x]].w -= nowflow;
            e[pe[x] ^ 1].w += nowflow;
        } // 更新边容量
    }
    return {flow, cost};
}

// https://ac.nowcoder.com/acm/problem/222408
int main() {
    ios::sync_with_stdio(0), cin.tie(nullptr), cout.tie(nullptr);
    int n; cin >> n;
    vector<int> z(n + 1), v(n + 1);
    ll ans = 0;
    for(int i = 1; i <= n; i++) {
        int x, y; cin >> x >> y >> z[i] >> v[i];
        ans += x * x;
        ans += y * y;
        ans += z[i] * z[i];
    }
    int num = n * 2 + 2, s = num - 1, t = num;
    init(num);
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            add(i, j + n, 1, 1ll * v[i] * v[i] * (j - 1) * (j - 1) + 2ll * v[i] * z[i] * (j - 1));
    for(int i = 1; i <= n; i++)
        add(s, i, 1, 0);
    for(int i = 1; i <= n; i++)
        add(n + i, t, 1, 0);
    ans += flow_dij(s, t, num).second;
    cout << ans << '\n';
    return 0;
}
```

4.4 二分图最大带权匹配

```cpp
// 1-base
// 主过程: 设置好 cost[N][N] 即可调用
// cost 可以为负数

using ll = long long;
#define prev prevv

const int N = 305;
const ll INF = 0x3f3f3f3f3f3f3f3f;

int n;
ll cost[N][N];
ll lx[N], ly[N];
int match[N];
ll slack[N];
int prev[N];
bool vy[N];

void augment(int root) {
    fill(vy + 1, vy + n + 1, false);
    fill(slack + 1, slack + n + 1, INF);
    int py;
    match[py = 0] = root;

    do {
        vy[py] = true;
        int x = match[py];
        ll delta = INF;
        int yy;
        for (int y = 1; y <= n; y++) {
            if (!vy[y]) {
                if (lx[x] + ly[y] - cost[x][y] < slack[y]) {
                    slack[y] = lx[x] + ly[y] - cost[x][y];
                    prev[y] = py;
                }
                if (slack[y] < delta) {
```

```
                    delta = slack[y];
                    yy = y;
                }
            }
        }
        for (int y = 0; y <= n; y++) {
            if (vy[y]) {
                lx[match[y]] -= delta;
                ly[y] += delta;
            } else {
                slack[y] -= delta;
            }
        }
        py = yy;
    } while (match[py] != -1);

    do {
        int pre = prev[py];
        match[py] = match[pre];
        py = pre;
    } while (py);
}

ll KM() {
    for (int i = 1; i <= n; i++) {
        lx[i] = ly[i] = -INF;
        match[i] = -1;
        for (int j = 1; j <= n; j++) {
            lx[i] = std::max(lx[i], cost[i][j]);
        }
    }
    ll answer = 0;
    for (int root = 1; root <= n; root++) {
        augment(root);
    }
    for (int i = 1; i <= n; i++) {
        answer += lx[i];
        answer += ly[i];
        // printf("%d %d\n", match[i], i);
    }
    return answer;
}
```

4.5 带花树

```
// 带花树：一般图最大匹配
const int maxn = 505;

struct Match {
    int n, father[maxn], vst[maxn], match[maxn], pre[maxn], Type[maxn], times;
    vector<int> edges[maxn];
    queue<int> Q;

    void ine(int x, int y) { edges[x].push_back(y); }
    void ine2(int x, int y) {
        ine(x, y);
        ine(y, x);
    }

    void init(int num) {
        times = 0;
        n = num;
        for (int i = 0; i <= n; ++i)
            edges[i].clear(), vst[i] = 0, match[i] = 0, pre[i] = 0;
    }

    int LCA(int x, int y) {
        times++;
        x = father[x], y = father[y]; //已知环位置
        while (vst[x] != times) {
            if (x) {
                vst[x] = times;
                x = father[pre[match[x]]];
            }
            swap(x, y);
        }
        return x;
    }

    void blossom(int x, int y, int lca) {
        while (father[x] != lca) {
            pre[x] = y;
            y = match[x];
            if (Type[y] == 1) {
                Type[y] = 0;
                Q.push(y);
            }
            father[x] = father[y] = father[lca];
            x = pre[y];
        }
    }

    int Augument(int s) {
        for (int i = 0; i <= n; ++i)
            father[i] = i, Type[i] = -1;
        Q = queue<int>();
        Type[s] = 0;
        Q.push(s); //仅入队o型点
        while (!Q.empty()) {
            int Now = Q.front();
            Q.pop();
            for (int Next : edges[Now]) {
                if (Type[Next] == -1) {
                    pre[Next] = Now;
                    Type[Next] = 1; //标记为i型点
                    if (!match[Next]) {
                        for (int to = Next, from = Now; to; from = pre[to]) {
                            match[to] = from;
                            swap(match[from], to);
                        }
                        return true;
                    }
                    Type[match[Next]] = 0;
                    Q.push(match[Next]);
                } else if (Type[Next] == 0 && father[Now] != father[Next]) {
                    int lca = LCA(Now, Next);
                    blossom(Now, Next, lca);
                    blossom(Next, Now, lca);
                }
            }
        }
        return false;
    }

    void gao() {
        int res = 0; // 最大匹配数
        for (int i = n; i >= 1; --i)
            if (!match[i])
                res += Augument(i);
        printf("%d\n", res);
        for (int i = 1; i <= n; ++i)
            printf("%d ", match[i]);
        printf("\n");
    }
} G;

int main() {
    int n, m;
    cin >> n >> m;
```

```
    G.init(n);
    for (int i = 1, x, y; i <= m; i++) {
        scanf("%d %d", &x, &y);
        G.ine2(x, y);
    }
    G.gao();
    return 0;
}
```

4.6 Hall 定理

霍尔定理

二分图中, 左侧点集 X 存在最大匹配的充要条件是:
X 中的任意 k 个点至少与 Y 中 k 个点相邻.
推论: 正则二分图存在完美匹配 (正则图 指每个点度数相等的图)

5 字符串

5.1 后缀自动机

```
// 在字符集比较小的时候可以直接开go数组，否则需要用map或者
  ↪ 哈希表替换
// 注意!!!结点数要开成串长的两倍

// 全局变量与数组定义
int last, len[maxn], fa[maxn], go[maxn][26], sam_cnt;
int c[maxn], q[maxn]; // 用来桶排序

// 在主函数开头加上这句初始化
last = sam_cnt = 1;

// 以下是按val进行桶排序的代码
for (int i = 1; i <= sam_cnt; i++)
    c[len[i] + 1]++;
for (int i = 1; i <= n; i++)
    c[i] += c[i - 1]; // 这里n是串长
for (int i = 1; i <= sam_cnt; i++)
    q[++c[len[i]]] = i;

//加入一个字符 均摊O(1)
void extend(int c) {
    int p = last, np = ++sam_cnt;
    len[np] = len[p] + 1;

    while (p && !go[p][c]) {
        go[p][c] = np;
        p = fa[p];
    }

    if (!p)
        fa[np] = 1;
    else {
        int q = go[p][c];

        if (len[q] == len[p] + 1)
            fa[np] = q;
        else {
            int nq = ++sam_cnt;
            len[nq] = len[p] + 1;
            memcpy(go[nq], go[q], sizeof(go[q]));

            fa[nq] = fa[q];
            fa[np] = fa[q] = nq;

            while (p && go[p][c] == q){
                go[p][c] = nq;
                p = fa[p];
            }
        }
    }

    last = np;
}
```

5.2 广义后缀自动机

5.2.1 对 Trie 建自动机

以 bfs 遍历 Trie 上的每一个结点，以 父结点 在 SAM 中对应的结点为 last 调用 insert 即可.

5.2.2 对多模式串建自动机

```
// 多串创建 sam 的方法:
// 在插入每个串前, 设置 last = 1, 然后一路 last =
  ↪ extend(last, c)

int extend(int p, int c) {
    int np = 0;

    if (!go[p][c]) {
        np = ++sam_cnt;
        len[np] = len[p] + 1;
        while (p && !go[p][c]) {
            go[p][c] = np;
            p = fa[p];
        }
    }

    if (!p)
        fa[np] = 1;
    else {
        int q = go[p][c];

        if (len[q] == len[p] + 1) {
            if (np)
                fa[np] = q;
            else
                return q;
        }
        else {
            int nq = ++sam_cnt;
            len[nq] = len[p] + 1;
            memcpy(go[nq], go[q], sizeof(go[q]));

            fa[nq] = fa[q];
            fa[q] = nq;
            if (np)
                fa[np] = nq;

            while (p && go[p][c] == q){
                go[p][c] = nq;
                p = fa[p];
            }

            if (!np)
                return nq;
        }
    }

    return np;
}
```

5.3 AC 自动机

```
using namespace std;
const int maxn = 500000 + 5;

const int rt = 0; // 默认 trie 树根为 0

int tr[maxn][26], tot = rt; // 初始化时要设置 tot = rt
int e[maxn];                // 标记字符串结尾
int fail[maxn];

void insert(char *s) {
    int p = rt; // from root
    for (int i = 0; s[i]; i++) {
        int k = s[i] - 'a';
        if (!tr[p][k])
            tr[p][k] = ++tot;   // 根节点为0
```

```cpp
            p = tr[p][k];
        }
        e[p]++;  // 尾部标记
}

void build() {
    queue<int> q;
    fill(fail, fail + 1 + tot, 0);

    for (int i = 0; i < 26; i++)
        if (tr[rt][i])
            q.push(tr[rt][i]);

    while (!q.empty()) {
        int k = q.front(); q.pop();
        for (int i = 0; i < 26; i++) {
            if (tr[k][i]) {
                fail[tr[k][i]] = tr[fail[k]][i];
                q.push(tr[k][i]); //入队
            } else
                tr[k][i] = tr[fail[k]][i]; // trie图
        }
    }
}

int query(char *t) {
    int p = rt, ans = 0;
    for (int i = 0; t[i]; i++) {
        p = tr[p][t[i] - 'a'];
        for (int j = p; j && (e[j] != -1); j = fail[j]) {
            ans += e[j];
            e[j] = -1;  // 防止重复遍历
        }
    }
    return ans;
}
```

5.4 后缀数组

5.4.1 倍增

```cpp
// h[i] = lcp(sa[i], sa[i - 1])
// sa[i] = 排名为 i 的后缀□下标□
// rk[i] = 后缀 i 的排名
// 需要将 s[n] 设为一个比一切字符大的数□才能正确地输出 height
// 不需要清空

template<int MAXN> class SA {
    public:
    int n, sa[MAXN], rk[MAXN], h[MAXN];

    void init() {
        // 不需要 init
    }

    void compute(int *s, int n, int m) {
        int i, p, w, j, k;
        this->n = n;
        if (n == 1) {
            sa[0] = rk[0] = h[0] = 0; return;
        }
        memset(cnt, 0, m * sizeof(int));
        for (i = 0; i < n; ++i) ++cnt[rk[i] = s[i]];
        for (i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
        for (i = n - 1; ~i; --i) sa[--cnt[rk[i]]] = i;
        for (w = 1; w < n; w <<= 1, m = p) {
            for (p = 0, i = n - 1; i >= n - w; --i) id[p++] = i;
            for (i = 0; i < n; ++i)
                if (sa[i] >= w) id[p++] = sa[i] - w;
            memset(cnt, 0, m * sizeof(int));
            for (i = 0; i < n; ++i) ++cnt[px[i] = rk[id[i]]];
            for (i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
            for (i = n - 1; ~i; --i) sa[--cnt[px[i]]] = id[i];
            memcpy(old_rk, rk, n * sizeof(int));
            for (i = p = 1, rk[sa[0]] = 0; i < n; ++i)
                rk[sa[i]] = cmp(sa[i], sa[i-1], w) ? p - 1 : p++;
        }
        for (i = 0; i < n; ++i) rk[sa[i]] = i;
        for (i = k = h[rk[0]] = 0; i < n; h[rk[i++]] = k)
            if (rk[i])
                for (k > 0 ? --k : 0, j = sa[rk[i] - 1]; s[i + k] == s[j + k]; ++k) {}
    }
    private:
    int old_rk[MAXN], id[MAXN], px[MAXN], cnt[MAXN];
    bool cmp(int x, int y, int w) {
        return old_rk[x] == old_rk[y] && old_rk[x + w] == old_rk[y + w];
    }
};
```

5.4.2 SAIS

```cpp
const int BUFFER_SIZE = 1u << 26 | 1;
char buf[BUFFER_SIZE], *buf_ptr = buf;

#define alloc(x, type, len)      \
    type *x = (type *)buf_ptr;   \
    buf_ptr += (len) * sizeof(type);

#define clear_buf() \
    memset(buf, 0, buf_ptr - buf), buf_ptr = buf;

template <int MAXN> class SuffixArray {
#define ltype true
#define stype false

public:
    int sa[MAXN], rk[MAXN], hei[MAXN];

    void compute(int n, int m, int *s) {
        sais(n, m, s, sa);
        for (int i = 0; i < n; ++i) rk[sa[i]] = i;
        for (int i = 0, h = 0; i < n; i++) {
            if (rk[i]) {
                int j = sa[rk[i] - 1];
                while (s[i + h] == s[j + h]) ++h;
                hei[rk[i]] = h;
            } else {
                h = 0;
            }
            if (h) --h;
        }
    }

private:
    int lbuc[MAXN], sbuc[MAXN];

    void induce(int n, int m, int *s, bool *type, int *sa, int *buc,
                int *lbuc, int *sbuc)
    {
        memcpy(lbuc + 1, buc, m * sizeof(int));
        memcpy(sbuc + 1, buc + 1, m * sizeof(int));

        sa[lbuc[s[n - 1]]++] = n - 1;
```

```cpp
        for (int i = 0; i < n; i++) {
            int t = sa[i] - 1;
            if (t >= 0 && type[t] == ltype)
                sa[lbuc[s[t]]++] = t;
        }

        for (int i = n - 1; i >= 0; i--) {
            int t = sa[i] - 1;
            if (t >= 0 && type[t] == stype)
                sa[--sbuc[s[t]]] = t;
        }
    }

    void sais(int n, int m, int *s, int *sa) {
        alloc(type, bool, n + 1);
        alloc(buc, int , m + 1);

        type[n] = false;

        for (int i = n - 1; i >= 0; i--) {
            ++buc[s[i]];
            type[i] = s[i] > s[i + 1] || (s[i] == s[i + 1] && type[i + 1] == ltype);
        }
        for (int i = 1; i <= m; i++) {
            buc[i] += buc[i - 1];
            sbuc[i] = buc[i];
        }
        memset(rk, -1, n * sizeof(int));

        alloc(lms, int, n + 1);

        int n1 = 0;
        for (int i = 0; i < n; i++) {
            if (!type[i] && (i == 0 || type[i - 1]))
                lms[rk[i] = n1++] = i;
        }

        lms[n1] = n;
        memset(sa, -1, n * sizeof(int));

        for (int i = 0; i < n1; i++) sa[--sbuc[s[lms[i]]]] = lms[i];
        induce(n, m, s, type, sa, buc, lbuc, sbuc);

        int m1 = 0;
        alloc(s1, int, n + 1);

        for (int i = 0, t = -1; i < n; i++) {
            int r = rk[sa[i]];
            if (r != -1) {
                int len = lms[r + 1] - sa[i] + 1;
                m1 += t == -1 || len != lms[rk[t] + 1] - t + 1 ||
                    memcmp(s + t, s + sa[i], len * sizeof(int)) != 0;
                s1[r] = m1;
                t = sa[i];
            }
        }

        alloc(sa1, int, n + 1);

        if (n1 == m1) {
            for (int i = 0; i < n1; i++)
                sa1[s1[i] - 1] = i;
        } else {
            sais(n1, m1, s1, sa1);
        }

        memset(sa, -1, n * sizeof(int));
        memcpy(sbuc + 1, buc + 1, m * sizeof(int));

        for (int i = n1 - 1; i >= 0; i--) {
            int t = lms[sa1[i]];
            sa[--sbuc[s[t]]] = t;
        }
        induce(n, m, s, type, sa, buc, lbuc, sbuc);
    }
#undef stype
#undef rtype
};
```

5.5 马拉车

```cpp
/* manacher */
const int maxn = 300007;
// s[0]: 特殊值□s[1, 3 .. tot]: 分隔值
int s[maxn], p[maxn]; // p 为半径
//int L[maxn], R[maxn]; // 包含点i的回文串□回文中心最左□右
int n, tot;

void manacher() {
    int right = 0, idx = 0;
    for(int i = 1; i <= tot; i++) {
        if(i < right)
            p[i] = min(p[2 * idx - i], right - i);
        else
            p[i] = 1;

        while(i + p[i] <= tot && s[i + p[i]] == s[i - p[i]])
            p[i]++;

        p[i]--;

        if(i + p[i] > right) {
            right = i + p[i];
            idx = i;
        }
    }
}
```

6 动态规划

6.1 数位 DP

```
// 记忆化搜索
struct cmp {
    bool operator()(const state& a, const state& b) const {
        // ...
    }
};

struct _hash {
    size_t operator()(const state& st) const {
        size_t res = st.cnt[0];
        for(int i = 1; i < 10; ++i) {
            res *= 19260817;
            res += st.cnt[i];
        }
        return res;
    }
};
unordered_map <state, ll, _hash, cmp> dp[20][20];

// -pos: 搜到的位置
// -st: 当前状态
// -lead: 是否有前导 0
// -limit: 是否有最高位限制
ll dfs(int pos, state st, int lead, int limit){
    // 边界情况
    if(pos < 0 /* && ... */) return 0;
    // 记忆化搜索
    int wd = st.w[st.d];
    if((!limit) && (!lead) && dp[pos][wd].count(st)) return
    ↪ dp[pos][wd][st];

    ll res = 0;
    // 最高位最大值
    int cur = limit ? a[pos] : 9;
    for(int i = 0; i <= cur; ++i) {
        // 有前导0且当前位也是0
        if((!i) && lead) res += dfs(pos-1, st, 1,
    ↪ limit&&(i==cur));
        // 有前导0且当前位非0 (出现最高位)
        else if(i && lead) res += dfs(pos-1, st.add(i), 0,
    ↪ limit&&(i==cur));
        else res += dfs(pos-1, st.add(i), 0,
    ↪ limit&&(i==cur));
    }
    // 没有前导0和最高限制时可以直接记录当前dp值以便下次搜
    ↪ 到同样的情况可以直接使用
    if(!limit&&!lead) dp[pos][wd][st] = res;
    return res;
}

ll gao(ll x) {
    memset(a, 0, sizeof(a));
    int len=0;
    while(x) a[len++]=x%10,x/=10;
    // init st
    return dfs(len-1, st, 1, 1);
}

int main()
{
    int T;
    ll l, r; int d;
    cin >> T;
    while(T--) {
        scanf("%lld %lld %d", &l, &r, &d);
        ll ans = gao(r, d) - gao(l - 1, d);
        printf("%lld\n", ans);
    }
}
```

7 几何

```cpp
#include <bits/stdc++.h>
#include <vector>
#define mp make_pair
#define fi first
#define se second
#define pb push_back
using namespace std;
using db = double;

mt19937 eng(time(0));

const db eps = 1e-6;
const db pi = acos(-1);
int sgn(db k) {
    if (k > eps)
        return 1;
    else if (k < -eps)
        return -1;
    return 0;
}
// -1: < | 0: == | 1: >
int cmp(db k1, db k2) { return sgn(k1 - k2); }
// k3 in [k1, k2]
int inmid(db k1, db k2, db k3) { return sgn(k1 - k3) * sgn(k2 - k3) <= 0; }
// 点 (x, y)
struct point {
    db x, y;
    point operator+(const point &k1) const {
        return (point){k1.x + x, k1.y + y};
    }
    point operator-(const point &k1) const {
        return (point){x - k1.x, y - k1.y};
    }
    point operator*(db k1) const { return (point){x * k1, y * k1}; }
    point operator/(db k1) const { return (point){x / k1, y / k1}; }
    int operator==(const point &k1) const {
        return cmp(x, k1.x) == 0 && cmp(y, k1.y) == 0;
    }
    // 逆时针旋转 k1 弧度
    point rotate(db k1) {
        return (point){x * cos(k1) - y * sin(k1), x * sin(k1) + y * cos(k1)};
    }
    // 逆时针旋转 90 度
    point rotleft() { return (point){-y, x}; }
    // 优先比较 x 坐标
    bool operator<(const point k1) const {
        int a = cmp(x, k1.x);
        if (a == -1)
            return 1;
        else if (a == 1)
            return 0;
        else
            return cmp(y, k1.y) == -1;
    }
    // 模长
    db abs() { return sqrt(x * x + y * y); }
    // 模长的平方
    db abs2() { return x * x + y * y; }
    // 与点 k1 的距离
    db dis(point k1) { return ((*this) - k1).abs(); }
    // 化为单位向量, require: abs() > 0
    point unit() {
        db w = abs();
        return (point){x / w, y / w};
    }
    // 读入
    void scan() {
        double k1, k2;
        scanf("%lf%lf", &k1, &k2);
        x = k1;
        y = k2;
    }
    // 输出
    void print() { printf("%.11lf %.11lf\n", x, y); }
    // 方向角 atan2(y, x)
    db getw() { return atan2(y, x); }
    // 将向量对称到 (-pi, pi] 半平面中
    point getdel() {
        if (sgn(x) == -1 || (sgn(x) == 0 && sgn(y) == -1))
            return (*this) * (-1);
        else
            return (*this);
    }
    // (-pi, 0] -> 0, (0, pi] -> 1
    int getP() const { return sgn(y) == 1 || (sgn(y) == 0 && sgn(x) == -1); }
};

/* 点与线段的位置关系及交点 */

// k3 在 矩形 [k1, k2] 中
int inmid(point k1, point k2, point k3) {
    return inmid(k1.x, k2.x, k3.x) && inmid(k1.y, k2.y, k3.y);
}
db cross(point k1, point k2) { return k1.x * k2.y - k1.y * k2.x; }
db dot(point k1, point k2) { return k1.x * k2.x + k1.y * k2.y; }
// 从 k1 转到 k2 的方向角
db rad(point k1, point k2) { return atan2(cross(k1, k2), dot(k1, k2)); }
// k1 k2 k3 逆时针 1 顺时针 -1 否则 0
int clockwise(point k1, point k2, point k3) {
    return sgn(cross(k2 - k1, k3 - k1));
}
// 按 (-pi, pi] 顺序进行极角排序
int cmpangle(point k1, point k2) {
    return k1.getP() < k2.getP() ||
           (k1.getP() == k2.getP() && sgn(cross(k1, k2)) > 0);
}
// 点 q 在线段 k1, k2 上
int onS(point k1, point k2, point q) {
    return inmid(k1, k2, q) && sgn(cross(k1 - q, k2 - k1)) == 0;
}
// q 到直线 k1,k2 的投影
point proj(point k1, point k2, point q) {
    point k = k2 - k1;
    return k1 + k * (dot(q - k1, k) / k.abs2());
}
// q 关于直线 k1, k2 的镜像
point reflect(point k1, point k2, point q) { return proj(k1, k2, q) * 2 - q; }
// 判断 直线 (k1, k2) 和 直线 (k3, k4) 是否相交
int checkLL(point k1, point k2, point k3, point k4) {
    return cmp(cross(k3 - k1, k4 - k1), cross(k3 - k2, k4 - k2)) != 0;
}
// 求直线 (k1, k2) 和 直线 (k3, k4) 的交点
point getLL(point k1, point k2, point k3, point k4) {
    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) {
```

```cpp
    if (l1 > r1)
        swap(l1, r1);
    if (l2 > r2)
        swap(l2, r2);
    return cmp(r1, l2) != -1 && cmp(r2, l1) != -1;
}
// 线段与线段相交判断（非严格相交）
int checkSS(point k1, point k2, point k3, point k4) {
    return intersect(k1.x, k2.x, k3.x, k4.x) &&
           intersect(k1.y, k2.y, k3.y, k4.y) &&
           sgn(cross(k3 - k1, k4 - k1)) * sgn(cross(k3 - k2, k4 - k2)) <= 0 &&
           sgn(cross(k1 - k3, k2 - k3)) * sgn(cross(k1 - k4, k2 - k4)) <= 0;
}
// 点 q 到 直线 (k1, k2) 的距离
db disLP(point k1, point k2, point q) {
    return fabs(cross(k1 - q, k2 - q)) / k1.dis(k2);
}
// 点 q 到 线段 (k1, k2) 的距离
db disSP(point k1, point k2, point q) {
    point k3 = proj(k1, k2, q);
    if (inmid(k1, k2, k3))
        return q.dis(k3);
    else
        return min(q.dis(k1), q.dis(k2));
}
// 线段 (k1, k2) 到 线段 (k3, k4) 的距离
db disSS(point k1, point k2, point k3, point k4) {
    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)));
}

/* 直线与半平面交 */

// 直线 p[0] -> p[1]
struct line {
    point p[2];
    line(point k1, point k2) {
        p[0] = k1;
        p[1] = k2;
    }
    point &operator[](int k) { return p[k]; }
    // k 严格位于直线左侧 / 半平面 p[0] -> p[1]
    int include(point k) { return sgn(cross(p[1] - p[0], k - p[0])) > 0; }
    // 方向向量
    point dir() { return p[1] - p[0]; }
    // 向左平移 d, 默认为 eps
    line push(db d = eps) {
        point delta = (p[1] - p[0]).rotleft().unit() * d;
        return {p[0] + delta, p[1] + delta};
    }
};
// 直线与直线交点
point getLL(line k1, line k2) { return getLL(k1[0], k1[1], k2[0], k2[1]); }
// 两直线平行
int parallel(line k1, line k2) { return sgn(cross(k1.dir(), k2.dir())) == 0; }
// 平行且同向
int sameDir(line k1, line k2) {
    return parallel(k1, k2) && sgn(dot(k1.dir(), k2.dir())) == 1;
}
// 同向则左侧优先，否则按极角排序，用于半平面交
int operator<(line k1, line k2) {
    if (sameDir(k1, k2))
        return k2.include(k1[0]);
    return cmpangle(k1.dir(), k2.dir());
}
// k3 (半平面) 包含 k1, k2 的交点，用于半平面交
int checkpos(line k1, line k2, line k3) { return k3.include(getLL(k1, k2)); }
// 求半平面交，半平面是逆时针方向，输出按照逆时针
vector<line> getHL(vector<line> L) {
    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 closepoint(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(closepoint(A, l, mid), closepoint(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(), [](point k1, 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;
}

/* 圆基础操作 */

// 圆 (o, r)
struct circle {
    point o;
    db r;
    void scan() {
        o.scan();
        scanf("%lf", &r);
    }
    int inside(point k) { return cmp(r, o.dis(k)); }
};

// 两圆位置关系（两圆公切线数量）
```

```cpp
int checkposCC(circle k1, circle k2) {
    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;
    else if (w1 == 0)
        return 3;
    else if (w2 > 0)
        return 2;
    else if (w2 == 0)
        return 1;
    else
        return 0;
}
// 直线与圆交点，沿 k2->k3 方向给出，相切给出两个
vector<point> getCL(circle k1, point k2, point k3) {
    point k = proj(k2, k3, k1.o);
    db d = k1.r * k1.r - (k - k1.o).abs2();
    if (sgn(d) == -1)
        return {};
    point del = (k3 - k2).unit() * sqrt(max((db)0.0, d));
    return {k - del, k + del};
}
// 两圆交点，沿圆 k1 逆时针给出，相切给出两个
vector<point> getCC(circle k1, circle k2) {
    int pd = checkposCC(k1, k2);
    if (pd == 0 || pd == 4)
        return {};
    db a = (k2.o - k1.o).abs2(), 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.rotleft() * c;
    return {m - del, m + del};
}
// 点到圆的切点，沿圆 k1 逆时针给出，注意未判位置关系
vector<point> TangentCP(circle k1, point k2) {
    db a = (k2 - k1.o).abs(), 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.rotleft() * c;
    return {m - del, m + del};
}
// 外公切线
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().rotleft().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;
    }
}
// 内公切线
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 < A.size(); i++)
        ans.push_back((line){A[i], B[i]});
    return ans;
}
// 所有公切线
vector<line> TangentCC(circle k1, circle k2) {
    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;
}
// 圆 k1 与三角形 k2 k3 k1.o 的有向面积交
db getarea(circle k1, point k2, point k3) {
    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;
    }
}
// 多边形与圆面积交
db getarea(vector<point> A, circle c) {
    int n = A.size();
    if (n <= 2)
        return 0.0;
    A.push_back(A[0]);
    db res = 0.0;
    for (int i = 0; i < n; i++) {
        point k1 = A[i], k2 = A[i + 1];
        res += getarea(c, k1, k2);
    }
    return fabs(res);
```

```cpp
}
// 三角形外接圆
circle getcircle(point k1, point k2, point k3) {
    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 =
        (point){k1.x + (c1 * b2 - c2 * b1) / d, k1.y + (a1 * c2 - a2 * c1) / d};
    return (circle){o, k1.dis(o)};
}
// 最小圆覆盖
circle getScircle(vector<point> A) {
    shuffle(A.begin(), A.end(), eng);
    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 = getcircle(A[i], A[j], A[k]);
                }
        }
    return ans;
}

/* 多边形 */

// 多边形有向面积
db area(vector<point> A) {
    db ans = 0;
    for (int i = 0; i < A.size(); i++)
        ans += cross(A[i], A[(i + 1) % A.size()]);
    return ans / 2;
}
// 判断是否为逆时针凸包
int checkconvex(vector<point> A) {
    int n = A.size();
    A.push_back(A[0]);
    A.push_back(A[1]);
    for (int i = 0; i < n; i++)
        if (sgn(cross(A[i + 1] - A[i], A[i + 2] - A[i])) == -1)
            return 0;
    return 1;
}
// 点与简单多边形位置关系: 2 内部 1 边界 0 外部
int contain(vector<point> A, point q) {
    int pd = 0;
    A.push_back(A[0]);
    for (int i = 1; i < A.size(); i++) {
        point u = A[i - 1], v = A[i];
        if (onS(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 (sgn(cross(u - v, q - v)) < 0)
            pd ^= 1;
    }
    return pd << 1;
}
// flag=0 不严格 flag=1 严格
vector<point> ConvexHull(vector<point> A, int flag = 1) {
    int n = A.size();
    if (n == 1)
        return A;
    if (n == 2) {
        if (A[0] == A[1])
            return {A[0]};
        else
            return A;
    }
    vector<point> ans(n * 2);
    sort(A.begin(), A.end());
    int now = -1;
    for (int i = 0; i < A.size(); i++) {
        while (now > 0 &&
               sgn(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 &&
               sgn(cross(ans[now] - ans[now - 1], A[i] - ans[now - 1])) < flag)
            now--;
        ans[++now] = A[i];
    }
    ans.resize(now);
    return ans;
}
// 凸包直径
db convexDiameter(vector<point> A) {
    int now = 0, n = A.size();
    db ans = 0;
    for (int i = 0; i < A.size(); i++) {
        now = max(now, i);
        while (1) {
            db k1 = A[i].dis(A[now % n]), k2 = A[i].dis(A[(now + 1) % n]);
            ans = max(ans, max(k1, k2));
            if (k2 > k1)
                now++;
            else
                break;
        }
    }
    return ans;
}
// 直线切凸包, 保留 k1,k2,p 逆时针的所有点
vector<point> convexcut(vector<point> A, point k1, point k2) {
    int n = A.size();
    A.push_back(A[0]);
    vector<point> ans;
    for (int i = 0; i < n; i++) {
        int w1 = clockwise(k1, k2, A[i]), w2 = clockwise(k1, k2, A[i + 1]);
        if (w1 >= 0)
            ans.push_back(A[i]);
        if (w1 * w2 < 0)
            ans.push_back(getLL(k1, k2, A[i], A[i + 1]));
    }
    return ans;
}
// 多边形 A 和 直线 (线段) k1->k2 严格相交, 注释部分为线段
int checkPoS(vector<point> A, point k1, point k2) {
    struct ins {
        point m, u, v;
        int operator<(const ins &k) const { return m < k.m; }
    };
    vector<ins> B;
```

```cpp
    // if (contain(A,k1)==2||contain(A,k2)==2) return 1;
    vector<point> poly = A;
    A.push_back(A[0]);
    for (int i = 1; i < A.size(); i++)
        if (checkLL(A[i - 1], A[i], k1, k2)) {
            point m = getLL(A[i - 1], A[i], k1, k2);
            if (inmid(A[i - 1], A[i], m) /
*&&inmid(k1,k2,m)*/)
                B.push_back((ins){m, A[i - 1], A[i]});
        }
    if (B.size() == 0)
        return 0;
    sort(B.begin(), B.end());
    int now = 1;
    while (now < B.size() && B[now].m == B[0].m)
        now++;
    if (now == B.size())
        return 0;
    int flag = contain(poly, (B[0].m + B[now].m) / 2);
    if (flag == 2)
        return 1;
    point d = B[now].m - B[0].m;
    for (int i = now; i < B.size(); i++) {
        if (!(B[i].m == B[i - 1].m) && flag == 2)
            return 1;
        int tag = sgn(cross(B[i].v - B[i].u, B[i].m + d -
B[i].u));
        if (B[i].m == B[i].u || B[i].m == B[i].v)
            flag += tag;
        else
            flag += tag * 2;
    }
    // return 0;
    return flag == 2;
}
int checkinp(point r, point l, point m) {
    if (cmpangle(l, r)) {
        return cmpangle(l, m) && cmpangle(m, r);
    }
    return cmpangle(l, m) || cmpangle(m, r);
}
// 快速检查线段是否和多边形严格相交
int checkPosFast(vector<point> A, point k1, point k2) {
    if (contain(A, k1) == 2 || contain(A, k2) == 2)
        return 1;
    if (k1 == k2)
        return 0;
    A.push_back(A[0]);
    A.push_back(A[1]);
    for (int i = 1; i + 1 < A.size(); i++)
        if (checkLL(A[i - 1], A[i], k1, k2)) {
            point now = getLL(A[i - 1], A[i], k1, k2);
            if (inmid(A[i - 1], A[i], now) == 0 ||
inmid(k1, k2, now) == 0)
                continue;
            if (now == A[i]) {
                if (A[i] == k2)
                    continue;
                point pre = A[i - 1], ne = A[i + 1];
                if (checkinp(pre - now, ne - now, k2 -
now))
                    return 1;
            } else if (now == k1) {
                if (k1 == A[i - 1] || k1 == A[i])
                    continue;
                if (checkinp(A[i - 1] - k1, A[i] - k1, k2 -
k1))
                    return 1;
            } else if (now == k2 || now == A[i - 1])
                continue;
            else
                return 1;
        }
    return 0;
}

/* 普通凸包中的二分 */

// 求经过点 x 切凸包 A 的两个切点，返回下标. 方向：A 上
[fi, se] 为点 x
// 能看到的区域. 需要保证 x 严格在凸包 A 外侧，A 的点数 >=
3 需要保证 A
// 是严格凸包，即无三点共线
pair<int, int> getTangentCoP(const vector<point> &A, point
x) {
    int sz = A.size();
    assert(sz >= 3);
    int res[2];
    int flag = 1;
    if (clockwise(A[sz - 1], A[0], x) == -1)
        flag = -1;
    int l = 0, r = sz - 1, ans = 0;
    while (l < r) {
        int mid = ((l + r) >> 1);
        if (clockwise(A[mid], A[mid + 1], x) == flag &&
            clockwise(A[0], A[mid + 1], x) == flag)
            ans = mid + 1, l = mid + 1;
        else
            r = mid;
    }
    res[0] = ans;
    l = ans, r = sz - 1, ans = sz - 1;
    while (l < r) {
        int mid = ((l + r) >> 1);
        if (clockwise(A[mid], A[mid + 1], x) == flag)
            ans = mid, r = mid;
        else
            l = mid + 1;
    }
    res[1] = ans;
    if (flag == -1)
        swap(res[0], res[1]);
    return {res[0], res[1]};
}

// 判断点是否在凸多边形 A 内部，flag = 1 严格，0 不严格
bool containCoP(const vector<point> &A, point x, int flag =
1) {
    int sz = A.size();
    assert(sz >= 3);
    if (!flag && (onS(A[0], A[1], x) || onS(A[sz - 1],
A[0], x)))
        return 1;
    if (!(clockwise(A[0], A[1], x) == 1 && clockwise(A[sz -
1], A[0], x) == 1))
        return 0;
    int l = 1, r = sz - 1, ans = 1;
    while (l < r) {
        int mid = l + r >> 1;
        if (clockwise(A[0], A[mid], x) == 1)
            ans = mid, l = mid + 1;
        else
            r = mid;
    }
    return clockwise(A[ans], A[ans + 1], x) >= flag;
}

/* 上下凸包中的二分 */

// 拆分凸包成上下凸壳 凸包尽量都随机旋转一个角度来避免出现
相同横坐标
// 尽量特判只有一个点的情况 凸包逆时针
void getUDP(vector<point> A, vector<point> &U,
vector<point> &D) {
```

```cpp
    db l = 1e100, r = -1e100;
    for (int i = 0; i < A.size(); i++)
        l = min(l, A[i].x), r = max(r, A[i].x);
    int wherel, wherer;
    for (int i = 0; i < A.size(); i++)
        if (cmp(A[i].x, l) == 0)
            wherel = i;
    for (int i = A.size(); i; i--)
        if (cmp(A[i - 1].x, r) == 0)
            wherer = i - 1;
    U.clear();
    D.clear();
    int now = wherel;
    while (1) {
        D.push_back(A[now]);
        if (now == wherer)
            break;
        now++;
        if (now >= A.size())
            now = 0;
    }
    now = wherel;
    while (1) {
        U.push_back(A[now]);
        if (now == wherer)
            break;
        now--;
        if (now < 0)
            now = A.size() - 1;
    }
}
// 需要保证凸包点数大于等于 3, 2 内部 ,1 边界 ,0 外部
int containCoP(const vector<point> &U, const vector<point>
  ↪ &D, point k) {
    db lx = U[0].x, rx = U[U.size() - 1].x;
    if (k == U[0] || k == U[U.size() - 1])
        return 1;
    if (cmp(k.x, lx) == -1 || cmp(k.x, rx) == 1)
        return 0;
    int where1 =
        lower_bound(U.begin(), U.end(), (point){k.x,
  ↪ -1e100}) - U.begin();
    int where2 =
        lower_bound(D.begin(), D.end(), (point){k.x,
  ↪ -1e100}) - D.begin();
    int w1 = clockwise(U[where1 - 1], U[where1], k),
        w2 = clockwise(D[where2 - 1], D[where2], k);
    if (w1 == 1 || w2 == -1)
        return 0;
    else if (w1 == 0 || w2 == 0)
        return 1;
    return 2;
}
// d 是方向 , 输出上方切点和下方切点
pair<point, point> getTangentCow(const vector<point> &U,
  ↪ const vector<point> &D,
                                 point d) {
    if (sgn(d.x) < 0 || (sgn(d.x) == 0 && sgn(d.y) < 0))
        d = d * (-1);
    point whereU, whereD;
    if (sgn(d.x) == 0)
        return mp(U[0], U[U.size() - 1]);
    int l = 0, r = U.size() - 1, ans = 0;
    while (l < r) {
        int mid = l + r >> 1;
        if (sgn(cross(U[mid + 1] - U[mid], d)) <= 0)
            l = mid + 1, ans = mid + 1;
        else
            r = mid;
    }
    whereU = U[ans];
    l = 0, r = D.size() - 1, ans = 0;
    while (l < r) {
        int mid = l + r >> 1;
        if (sgn(cross(D[mid + 1] - D[mid], d)) >= 0)
            l = mid + 1, ans = mid + 1;
        else
            r = mid;
    }
    whereD = D[ans];
    return mp(whereU, whereD);
}
// 先检查 contain, 逆时针给出
pair<point, point> getTangentCoP(const vector<point> &U,
  ↪ const vector<point> &D,
                                 point k) {
    db lx = U[0].x, rx = U[U.size() - 1].x;
    if (k.x < lx) {
        int l = 0, r = U.size() - 1, ans = U.size() - 1;
        while (l < r) {
            int mid = l + r >> 1;
            if (clockwise(k, U[mid], U[mid + 1]) == 1)
                l = mid + 1;
            else
                ans = mid, r = mid;
        }
        point w1 = U[ans];
        l = 0, r = D.size() - 1, ans = D.size() - 1;
        while (l < r) {
            int mid = l + r >> 1;
            if (clockwise(k, D[mid], D[mid + 1]) == -1)
                l = mid + 1;
            else
                ans = mid, r = mid;
        }
        point w2 = D[ans];
        return mp(w1, w2);
    } else if (k.x > rx) {
        int l = 1, r = U.size(), ans = 0;
        while (l < r) {
            int mid = l + r >> 1;
            if (clockwise(k, U[mid], U[mid - 1]) == -1)
                r = mid;
            else
                ans = mid, l = mid + 1;
        }
        point w1 = U[ans];
        l = 1, r = D.size(), ans = 0;
        while (l < r) {
            int mid = l + r >> 1;
            if (clockwise(k, D[mid], D[mid - 1]) == 1)
                r = mid;
            else
                ans = mid, l = mid + 1;
        }
        point w2 = D[ans];
        return mp(w2, w1);
    } else {
        int where1 =
            lower_bound(U.begin(), U.end(), (point){k.x,
  ↪ -1e100}) - U.begin();
        int where2 =
            lower_bound(D.begin(), D.end(), (point){k.x,
  ↪ -1e100}) - D.begin();
        if ((k.x == lx && k.y > U[0].y) ||
            (where1 && clockwise(U[where1 - 1], U[where1],
  ↪ k) == 1)) {
            int l = 1, r = where1 + 1, ans = 0;
            while (l < r) {
                int mid = l + r >> 1;
                if (clockwise(k, U[mid], U[mid - 1]) == 1)
                    ans = mid, l = mid + 1;
                else
                    r = mid;
```

```cpp
            }
            point w1 = U[ans];
            l = where1, r = U.size() - 1, ans = U.size() - 1;
            while (l < r) {
                int mid = l + r >> 1;
                if (clockwise(k, U[mid], U[mid + 1]) == 1)
                    l = mid + 1;
                else
                    ans = mid, r = mid;
            }
            point w2 = U[ans];
            return mp(w2, w1);
        } else {
            int l = 1, r = where2 + 1, ans = 0;
            while (l < r) {
                int mid = l + r >> 1;
                if (clockwise(k, D[mid], D[mid - 1]) == -1)
                    ans = mid, l = mid + 1;
                else
                    r = mid;
            }
            point w1 = D[ans];
            l = where2, r = D.size() - 1, ans = D.size() - 1;
            while (l < r) {
                int mid = l + r >> 1;
                if (clockwise(k, D[mid], D[mid + 1]) == -1)
                    l = mid + 1;
                else
                    ans = mid, r = mid;
            }
            point w2 = D[ans];
            return mp(w1, w2);
        }
    }
}

// 三维计算几何

struct P3 {
    db x, y, z;
    P3 operator+(P3 k1) { return (P3){x + k1.x, y + k1.y, z + k1.z}; }
    P3 operator-(P3 k1) { return (P3){x - k1.x, y - k1.y, z - k1.z}; }
    P3 operator*(db k1) { return (P3){x * k1, y * k1, z * k1}; }
    P3 operator/(db k1) { return (P3){x / k1, y / k1, z / k1}; }
    db abs2() { return x * x + y * y + z * z; }
    db abs() { return sqrt(x * x + y * y + z * z); }
    P3 unit() { return (*this) / abs(); }
    int operator<(const P3 k1) const {
        if (cmp(x, k1.x) != 0)
            return x < k1.x;
        if (cmp(y, k1.y) != 0)
            return y < k1.y;
        return cmp(z, k1.z) == -1;
    }
    int operator==(const P3 k1) {
        return cmp(x, k1.x) == 0 && cmp(y, k1.y) == 0 && cmp(z, k1.z) == 0;
    }
    void scan() {
        double k1, k2, k3;
        scanf("%lf%lf%lf", &k1, &k2, &k3);
        x = k1;
        y = k2;
        z = k3;
    }
};
P3 cross(P3 k1, P3 k2) {
    return (P3){k1.y * k2.z - k1.z * k2.y, k1.z * k2.x - k1.x * k2.z,
                k1.x * k2.y - k1.y * k2.x};
}
db dot(P3 k1, P3 k2) { return k1.x * k2.x + k1.y * k2.y + k1.z * k2.z; }
// p=(3,4,5),l=(13,19,21),theta=85 ans=(2.83,4.62,1.77)
P3 turn3D(db k1, P3 l, P3 p) {
    l = l.unit();
    P3 ans;
    db c = cos(k1), s = sin(k1);
    ans.x = p.x * (l.x * l.x * (1 - c) + c) +
            p.y * (l.x * l.y * (1 - c) - l.z * s) +
            p.z * (l.x * l.z * (1 - c) + l.y * s);
    ans.y = p.x * (l.x * l.y * (1 - c) + l.z * s) +
            p.y * (l.y * l.y * (1 - c) + c) +
            p.z * (l.y * l.z * (1 - c) - l.x * s);
    ans.z = p.x * (l.x * l.z * (1 - c) - l.y * s) +
            p.y * (l.y * l.z * (1 - c) + l.x * s) +
            p.z * (l.x * l.x * (1 - c) + c);
    return ans;
}
typedef vector<P3> VP;
typedef vector<VP> VVP;
db Acos(db x) { return acos(max(-(db)1, min(x, (db)1))); }
// 球面距离 , 圆心原点 , 半径 1
db Odist(P3 a, P3 b) {
    db r = Acos(dot(a, b));
    return r;
}
db r;
P3 rnd;
vector<db> solve(db a, db b, db c) {
    db r = sqrt(a * a + b * b), th = atan2(b, a);
    if (cmp(c, -r) == -1)
        return {0};
    else if (cmp(r, c) <= 0)
        return {1};
    else {
        db tr = pi - Acos(c / r);
        return {th + pi - tr, th + pi + tr};
    }
}
vector<db> jiao(P3 a, P3 b) {
    // dot(rd+x*cos(t)+y*sin(t),b) >= cos(r)
    if (cmp(Odist(a, b), 2 * r) > 0)
        return {0};
    P3 rd = a * cos(r), z = a.unit(), y = cross(z, rnd).unit(),
       x = cross(y, z).unit();
    vector<db> ret = solve(-(dot(x, b) * sin(r)), -(dot(y, b) * sin(r)),
                           -(cos(r) - dot(rd, b)));
    return ret;
}
db norm(db x, db l = 0, db r = 2 * pi) { // change x into [l,r)
    while (cmp(x, l) == -1)
        x += (r - l);
    while (cmp(x, r) >= 0)
        x -= (r - l);
    return x;
}
db disLP(P3 k1, P3 k2, P3 q) {
    return (cross(k2 - k1, q - k1)).abs() / (k2 - k1).abs();
}
db disLL(P3 k1, P3 k2, P3 k3, P3 k4) {
    P3 dir = cross(k2 - k1, k4 - k3);
    if (sgn(dir.abs()) == 0)
        return disLP(k1, k2, k3);
```

```cpp
    return fabs(dot(dir.unit(), k1 - k2));
}
VP getFL(P3 p, P3 dir, P3 k1, P3 k2) {
    db a = dot(k2 - p, dir), b = dot(k1 - p, dir), d = a - b;
    if (sgn(fabs(d)) == 0)
        return {};
    return {(k1 * a - k2 * b) / d};
}
VP getFF(P3 p1, P3 dir1, P3 p2, P3 dir2) { // 返回一条线
    P3 e = cross(dir1, dir2), v = cross(dir1, e);
    db d = dot(dir2, v);
    if (sgn(abs(d)) == 0)
        return {};
    P3 q = p1 + v * dot(dir2, p2 - p1) / d;
    return {q, q + e};
}
// 3D Covex Hull Template
db getV(P3 k1, P3 k2, P3 k3, P3 k4) { // get the Volume
    return dot(cross(k2 - k1, k3 - k1), k4 - k1);
}
db rand_db() { return 1.0 * rand() / RAND_MAX; }
VP convexHull2D(VP A, P3 dir) {
    P3 x = {(db)rand(), (db)rand(), (db)rand()};
    x = x.unit();
    x = cross(x, dir).unit();
    P3 y = cross(x, dir).unit();
    P3 vec = dir.unit() * dot(A[0], dir);
    vector<point> B;
    for (int i = 0; i < A.size(); i++)
        B.push_back((point){dot(A[i], x), dot(A[i], y)});
    B = ConvexHull(B);
    A.clear();
    for (int i = 0; i < B.size(); i++)
        A.push_back(x * B[i].x + y * B[i].y + vec);
    return A;
}
namespace CH3 {
VVP ret;
set<pair<int, int>> e;
int n;
VP p, q;
void wrap(int a, int b) {
    if (e.find({a, b}) == e.end()) {
        int c = -1;
        for (int i = 0; i < n; i++)
            if (i != a && i != b) {
                if (c == -1 || sgn(getV(q[c], q[a], q[b], q[i])) > 0)
                    c = i;
            }
        if (c != -1) {
            ret.push_back({p[a], p[b], p[c]});
            e.insert({a, b});
            e.insert({b, c});
            e.insert({c, a});
            wrap(c, b);
            wrap(a, c);
        }
    }
}
VVP ConvexHull3D(VP _p) {
    p = q = _p;
    n = p.size();
    ret.clear();
    e.clear();
    for (auto &i : q)
        i = i + (P3){rand_db() * 1e-4, rand_db() * 1e-4, rand_db() * 1e-4};
    for (int i = 1; i < n; i++)
        if (q[i].x < q[0].x)
            swap(p[0], p[i]), swap(q[0], q[i]);
    for (int i = 2; i < n; i++)
        if ((q[i].x - q[0].x) * (q[1].y - q[0].y) >
            (q[i].y - q[0].y) * (q[1].x - q[0].x))
            swap(q[1], q[i]), swap(p[1], p[i]);
    wrap(0, 1);
    return ret;
}
} // namespace CH3
VVP reduceCH(VVP A) {
    VVP ret;
    map<P3, VP> M;
    for (VP nowF : A) {
        P3 dir = cross(nowF[1] - nowF[0], nowF[2] - nowF[0]).unit();
        for (P3 k1 : nowF)
            M[dir].pb(k1);
    }
    for (pair<P3, VP> nowF : M)
        ret.pb(convexHull2D(nowF.se, nowF.fi));
    return ret;
}
//  把一个面变成 ( 点 , 法向量 ) 的形式
pair<P3, P3> getF(VP F) {
    return mp(F[0], cross(F[1] - F[0], F[2] - F[0]).unit());
}
// 3D Cut 保留 dot(dir,x-p)>=0 的部分
VVP ConvexCut3D(VVP A, P3 p, P3 dir) {
    VVP ret;
    VP sec;
    for (VP nowF : A) {
        int n = nowF.size();
        VP ans;
        int dif = 0;
        for (int i = 0; i < n; i++) {
            int d1 = sgn(dot(dir, nowF[i] - p));
            int d2 = sgn(dot(dir, nowF[(i + 1) % n] - p));
            if (d1 >= 0)
                ans.pb(nowF[i]);
            if (d1 * d2 < 0) {
                P3 q = getFL(p, dir, nowF[i], nowF[(i + 1) % n])[0];
                ans.push_back(q);
                sec.push_back(q);
            }
            if (d1 == 0)
                sec.push_back(nowF[i]);
            else
                dif = 1;
            dif |= (sgn(dot(dir, cross(nowF[(i + 1) % n] - nowF[i],
                                       nowF[(i + 1) % n] - nowF[i]))) == -1);
        }
        if (ans.size() > 0 && dif)
            ret.push_back(ans);
    }
    if (sec.size() > 0)
        ret.push_back(convexHull2D(sec, dir));
    return ret;
}
db vol(VVP A) {
    if (A.size() == 0)
        return 0;
    P3 p = A[0][0];
    db ans = 0;
    for (VP nowF : A)
        for (int i = 2; i < nowF.size(); i++)
            ans += abs(getV(p, nowF[0], nowF[i - 1], nowF[i]));
    return ans / 6;
}
```

```
VVP init(db INF) {
    VVP pss(6, VP(4));
    pss[0][0] = pss[1][0] = pss[2][0] = {-INF, -INF, -INF};
    pss[0][3] = pss[1][1] = pss[5][2] = {-INF, -INF, INF};
    pss[0][1] = pss[2][3] = pss[4][2] = {-INF, INF, -INF};
    pss[0][2] = pss[5][3] = pss[4][1] = {-INF, INF, INF};
    pss[1][3] = pss[2][1] = pss[3][2] = {INF, -INF, -INF};
    pss[1][2] = pss[5][1] = pss[3][3] = {INF, -INF, INF};
    pss[2][2] = pss[4][3] = pss[3][1] = {INF, INF, -INF};
    pss[5][0] = pss[4][0] = pss[3][0] = {INF, INF, INF};
    return pss;
}
```

8 杂项

8.1 java 高精度

```java
import java.math.BigInteger;
import java.util.Scanner;

public class QHD {
    private static Scanner sc = new Scanner(System.in);
    public static void solve() {
        BigInteger n = sc.nextBigInteger();
        BigInteger m = sc.nextBigInteger();
        BigInteger one = BigInteger.valueOf(1);
        BigInteger ans = BigInteger.valueOf(0);
        BigInteger a1 = m.shiftLeft(1); // 左移
        BigInteger a2 = m.shiftRight(1); // 右移
        BigInteger a3 = m.multiply(n); // 乘
        BigInteger a4 = m.divide(n); // 除
        BigInteger a5 = m.subtract(n); // 减
        BigInteger a6 = m.add(n); // 加
        BigInteger a7 = m.mod(n); // 模

        String ns = n.toString();
        String s = "114514";
        s.substring(1, 3); // -> 145

        StringBuilder sb = new StringBuilder(ns);
        sb.append('1');
    }
    public static void main(String[] args) {
        int t = sc.nextInt();
        for(int cas = 1; cas <= t; cas++) {
            solve();
        }
    }
}
```

8.2 重载 umap

```cpp
struct state {
    int w[51];
} st;

struct cmp {
    bool operator()(const state& a, const state& b) const {
        for(int i = 1; i <= 50; ++i) {
            if(a.w[i] != b.w[i]) return false;
        }
        return true;
    }
};

struct _hash {
    size_t operator()(const state& st) const {
        size_t res = st.w[1];
        for(int i = 2; i <= 50; ++i) {
            res *= 19260817;
            res += st.w[i];
        }
        return res;
    }
};

unordered_map <state, ll, _hash, cmp> mp_st;

// mp.reserve(1e7);
```

8.3 bitset

```cpp
using ull = unsigned long long;

const int Lg = 64;

// 寻找 bitset 中任意一个非 0 位
template<size_t MAXN>
int findAny(bitset<MAXN>& a) {
    ull *p = (ull *)&a;

    int len = MAXN / Lg;
    for (int i = 0; i < len; i++) {
        if (p[i]) {
            for (int j = 0; j < Lg; j++) {
                if ((p[i] >> j) & 1ll) {
                    return (i * Lg + j);
                }
            }
        }
    }

    assert(false);
    return -1;
}

// bitset: 将第 i 位设置为 0/1
// s.set(i, val);

// bitset: 清空
// s.reset();

// bitset: 取第 i 位
// s.test(i);

// _Find_first(): 寻找第一位
int x = st[i]._Find_first();

// _Find_next(int i): 寻找下一位
int y = st[i]._Find_next(x);
```

8.4 模拟退火

```cpp
/**
 * cur: 起始位置
 * initT: 初始温度
 * c: 退火速率
 */
db SA(point origin, db initT, db eps=1e-6, db c=0.9999, int
  ↪ times=50000000) {
    point cur = origin;

    db t = initT;
    db ans = calc(cur), fx = ans;
    int steps = 0;
    for (int cnt = 0; cnt < 10; cnt++) {
        int i = 0;
        cur = origin;
        t = initT;
        for (i = 1; i <= times && fabs(t) > eps; i++) {
            point nex = cur.move(t);
            db fy = calc(nex);
            ans = min(ans, fy);
            db delta = fy - fx;
            if (exp(-delta / t) > Rand())
                cur = nex, fx = fy;
            t *= c;
        }
        steps = i;
    }

```

```cpp
    cout << "times = " << steps << endl;
    return ans;
}
```

8.5 对拍

8.5.1 windows 对拍

```bat
:loop
data.exe > in.txt
J-std.exe < in.txt > J-std.txt
J.exe < in.txt > J-ans.txt
fc /A J-std.txt J-ans.txt
if not errorlevel 1 goto loop
pause
:end
```

8.5.2 linux 对拍

```bash
#!/bin/bash

while true; do
    ./data > data.in
    ./C < data.in > ans.out
    ./C-baoli < data.in > baoli.out
    if diff ans.out baoli.out; then
        printf AC
    else
        echo WA
        exit 0
    fi
done
```

8.6 快读

```cpp
// pku

char buf[1 << 20], *p1 = buf, *p2 = buf;

#ifdef ONLINE_JUDGE
char get() {
    if (p1 == p2) {
        p1 = buf;
        p2 = buf + std::fread(buf, 1, 1 << 20, stdin);
    }
    if (p1 == p2) {
        return EOF;
    }
    return *p1++;
}
#else
char get() { return std::getchar(); }
#endif

ll readLong() {
    ll x = 0;
    char c = get();
    while (!std::isdigit(c)) {
        c = get();
    }
    while (std::isdigit(c)) {
        x = x * 10 + c - '0';
        c = get();
    }
    return x;
}

int readInt() {
    int x = 0;
    char c = get();
    while (!std::isdigit(c)) {
        c = get();
    }
    while (std::isdigit(c)) {
        x = x * 10 + c - '0';
        c = get();
    }
    return x;
}

// pku
char gc() {
    static char buf[1 << 20], *p1 = buf, *p2 = buf;
    return p1 == p2 &&
            (p2 = (p1 = buf) + fread(buf, 1, 1 << 20, stdin), p1 == p2) ? EOF : *p1++;
}

inline ll read() {
    ll x = 0;
    char ch = gc();
    bool positive = 1;
    for (; !isdigit(ch); ch = gc())
        if (ch == '-')
            positive = 0;
    for (; isdigit(ch); ch = gc())
        x = x * 10 + ch - '0';
    return positive ? x : -x;
}
```

8.7 随机

```cpp
mt19937 eng(time(0));
int randInt(int a, int b) {
    uniform_int_distribution<int> dis(a, b);
    return dis(eng);
}
```

8.8 python

```python
from functools import cmp_to_key

lst = [(1, 5), (2, 4), (3, 6), (9, 9), (5, 1)]

def cmp(a, b):
    return a[1] * b[0] - a[0] * b[1]

sortedlst = sorted(lst, key=cmp_to_key(cmp))

print(lst)
print(sortedlst)
```