Contents

1 Maths

1.1 Modular Arithmetic

```
// **Really important note**: inputs of the modAdd, modSub, and modMul
// functions must all be normalized (within the range [0..mod - 1]) before use

/// @status:
///   - modMul and modPow tested on:
///      https://cses.fi/problemset/task/1095
///      https://cses.fi/problemset/task/1712
///   - modAdd, modSub, modMul, extGcd, modInv tested on:
///      https://cses.fi/problemset/task/1082
///   - C (binary coefficient) tested on:
///      https://cses.fi/problemset/task/1079

#pragma once

#include <bits/stdc++.h>

using namespace std;

int modAdd(int a, int b, int mod) {
    a += b;
    if (a >= mod) a -= mod;
    return a;
}

int modSub(int a, int b, int mod) {
    a -= b;
    if (a < 0) a += mod;
    return a;
}

int modMul(int a, int b, int mod) {
    int64_t res = (int64_t) a * b;
    return (int) (res % mod);
}

int64_t binPow(int64_t a, int64_t x) {
    int64_t res = 1;
    while (x) {
        if (x & 1) res *= a;
        a *= a;
        x >>= 1;
    }
    return res;
}

int64_t modPow(int64_t a, int64_t x, int mod) {
    int res = 1;
    while (x) {
        if (x & 1) res = modMul(res, a, mod);
        a = modMul(a, a, mod);
        x >>= 1;
    }
    return res;
}

/// This solves the equation ax + by = gcd(a, b);
/// Input: a, b
//  Output: g (returned), x, y (passed by ref)
int64_t extGcd(int64_t a, int64_t b, int64_t& x, int64_t& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int64_t x1, y1;
    int64_t g = extGcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    assert(g == 1);
    return g;
}

/// This mod inverse function applies Fermat's little theorem
//  Can only be used if m is prime, and a and m are coprime
int64_t modInvPrimeMod(int64_t a, int64_t m, int mod) {
    return modPow(a, m - 2, mod);
}

int64_t modInv(int64_t a, int mod) {
    int64_t x, y;
    int64_t g = extGcd(a, mod, x, y);
    assert(g == 1);
    return (x % mod + mod) % mod;
}

vector<int> calcInv(int n, int mod) {
    vector<int> inv(n + 1);
    inv[1] = 1;

    for(int i = 2; i <= n; ++i) {
        inv[i] = mod - (mod / i) * inv[mod % i] % mod;
    }

    return inv;
}

struct BinomialCoefficient {
    const int N;
    int mod;
    vector<int> fact;
    vector<int> inv_fact;

    BinomialCoefficient(int N_, int mod_) : N(N_), mod(mod_), fact(N), inv_fact(N) {
        fact[0] = 1;
        for (int i = 1; i <= N; i++) {
            fact[i] = modMul(fact[i - 1], i, mod);
        }
        inv_fact[N] = modInv(fact[N], mod);
        for (int i = N - 1; i >= 0; i--) {
            inv_fact[i] = modMul(inv_fact[i + 1], i + 1, mod);
        }
    }

    int C(int n, int k) {
        int res = modMul(fact[n], modMul(inv_fact[k], inv_fact[n - k], mod), mod);
        return res;
    }
};
```

1.2 Modnum

```
#pragma once

#include <bits/stdc++.h>
#include "mod.hpp"

using namespace std;

template <typename T, int md>
struct Modnum {
    using M = Modnum;
    T v;
    Modnum(T _v=0) : v(fix(_v)) {}

    T fix(int64_t x) {
        if (x < -md || x > 2 * md) x %= md;
        if (x >= md) x -= md;
        if (x < 0) x += md;
        return x;
    }

    M operator+(M o) { return M(v + o.v); }
    M operator-(M o) { return M(v - o.v); }
    M operator*(M o) { return M(fix((int64_t) v * o.v)); }
    M operator/(M o) {
        return *this * modInv(o.v, md);
    }
    M pow(int64_t x) {
        M a(v);
        M res(1);
        while (x) {
            if (x & 1) res = res * a;
            a = a * a;
            x >>= 1;
        }
        return res;
    }
    friend istream& operator>>(istream& is, M& o) {
        is >> o.v; o.v = o.fix(o.v); return is;
    }
    friend ostream& operator<<(ostream& os, const M& o) {
        return os << o.v;
    }
};
```

1.3 Primality Test

```
// Simple primality test

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

```cpp
bool isPrime(int x) {
    for (int d = 2; d * d <= x; d++) {
        if (x % d == 0)
            return false;
    }
    return true;
}
```

1.4 Sieve of Eratosthenes

```cpp
// Computes primes in the range [2, n] in n ln ln sqrt(n) + o(n) time.
// https://cp-algorithms.com/algebra/sieve-of-eratosthenes.html

#pragma once

#include <bits/stdc++.h>

using namespace std;

int main() {
    int n = 100;
    vector<char> is_prime(n+1, true);
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i * i <= n; i++) {
        if (is_prime[i]) {
            for (int j = i * i; j <= n; j += i)
                is_prime[j] = false;
        }
    }
    for (int i = 0; i < n; i++) {
        if (is_prime[i]) {
            cout << i << ' ';
        }
    }
}
```

1.5 Euclidean Algorithm

```cpp
// Euclidean algorithm for GCD & LCM
// https://cp-algorithms.com/algebra/euclid-algorithm.html

#pragma once
#include <bits/stdc++.h>

int gcd (int a, int b) {
    return b ? gcd (b, a % b) : a;
}

// lcm(a,b) = a*b/(gcd(a,b))
int lcm (int a, int b) {
    return a / gcd(a, b) * b;
}
```

1.6 Extended Euclidean Algorithm

```cpp
// Extended Euclidean algorithm
// Solves for coefficients x,y such that ax + by = gcd(a,b)

#pragma once
#include <bits/stdc++.h>

int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

// Iterative version
int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}
```

2 Geometry

2.1 Points

```cpp
#pragma once

#include <bits/stdc++.h>
#include "geoutil.hpp"

using namespace std;


template<typename T>
struct Point {
    using P = Point;
    T x, y;

    Point(T x_ = 0, T y_ = 0) : x(x_), y(y_) {}

    P operator+(const P &o) const { return P(x + o.x, y + o.y); }

    P operator-(const P &o) const { return P(x - o.x, y - o.y); }

    P operator*(T d) const { return P(x * d, y * d); }

    P operator/(T d) const { return P(x / d, y / d); }

    T dot(P o) const { return x * o.x + y * o.y; }

    T cross(P o) const { return x * o.y - y * o.x; }

    T abs2() const { return x * x + y * y; }

    long double abs() const { return sqrt((long double) abs2()); }

    double angle() const { return atan2(y, x); } // [−π, π]
    P unit() const { return *this / abs(); } // makes abs()=1
    P perp() const { return P(-y, x); } // rotates +π/2

    P rotate(double a) const { // ccw
        return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
    }

    friend istream &operator>>(istream &is, P &p) {
        return is >> p.x >> p.y;
    }

    friend ostream &operator<<(ostream &os, P &p) {
        return os << "(" << p.x << ", " << p.y << ")";
    }

    // position of c relative to a->b
    // > 0: c is on the left of a->b
    friend T orient(P a, P b, P c) {
        return (b - a).cross(c - a);
    }

    // Check if u⃗ and v⃗ are parallel
    // (u⃗ = cv⃗) where c ∈ R)
    friend bool parallel(P u, P v) {
        return u.cross(v) == 0;
    }

    // Check if point p lies on the segment ab
    friend bool onSegment(P a, P b, P p) {
        return orient(a, b, p) == 0 &&
               min(a.x, b.x) <= p.x &&
               max(a.x, b.x) >= p.x &&
               min(a.y, b.y) <= p.y &&
               max(a.y, b.y) >= p.y;
    }

    friend bool boundingBox(P p1, P q1, P p2, P q2) {
        if (max(p1.x, q1.x) < min(p2.x, q2.x)) return true;
        if (max(p1.y, q1.y) < min(p2.y, q2.y)) return true;
        if (max(p2.x, q2.x) < min(p1.x, q1.x)) return true;
        if (max(p2.x, q2.x) < min(p1.x, q1.x)) return true;
        return false;
    }

    friend bool intersect(P p1, P p2, P p3, P p4) {
        // Check if two segments are parallel
        if (parallel(p2 - p1, p4 - p3)) {
            // Check if 4 ps are colinear
            if (!parallel(p2 - p1, p3 - p1)) return false;
            if (boundingBox(p1, p2, p3, p4)) return false;
            return true;
        }

        // check if one line is completely on one side of the other
        for (int i = 0; i < 2; i++) {
            if (sgn((p2 - p1).cross(p3 - p1)) == sgn((p2 - p1).cross(p4 - p1))
                && sgn((p2 - p1).cross(p3 - p1)) != 0) {
                return false;
            }
            swap(p1, p3);
            swap(p2, p4);
        }
        return true;
    }

    // Check if p is in ∠bac (including the rays)
    friend bool inAngle(P a, P b, P c, P p) {
        assert(orient(a, b, c) != 0);
        if (orient(a, b, c) < 0) swap(b, c);
        return orient(a, b, p) >= 0 && orient(a, c, p) <= 0;
    }

    // Angle ∠bac (+/-)
    friend double directedAngle(P a, P b, P c) {
        if (orient(a, b, c) >= 0) {
            return (b - a).angle(c - a);
        }
        return 2 * PI - (b - a).angle(c - a);
    }
};
```