

Contents

1	Maths	1
1.1	Modular Arithmetic	1
1.2	Modnum	2
1.3	Primality Test	3
1.4	Sieve of Eratosthenes	3
1.5	Euclidean Algorithm	3
1.6	Extended Euclidean Algorithm	3
2	Geometry	4
2.1	Points	4

1 Maths

1.1 Modular Arithmetic

```
1 // **Really important note**: inputs of the modAdd, modSub, and modMul
2 // functions must all be normalized (within the range [0..mod - 1]) before use
3
4 /// @status:
5 /// - modMul and modPow tested on:
6 ///     https://cses.fi/problemset/task/1095
7 ///     https://cses.fi/problemset/task/1712
8 /// - modAdd, modSub, modMul, extGcd, modInv tested on:
9 ///     https://cses.fi/problemset/task/1082
10 /// - C (binary coefficient) tested on:
11 ///     https://cses.fi/problemset/task/1079
12
13 #pragma once
14
15 #include <bits/stdc++.h>
16
17 using namespace std;
18
19 int modAdd(int a, int b, int mod) {
20     a += b;
21     if (a >= mod) a -= mod;
22     return a;
23 }
24
25 int modSub(int a, int b, int mod) {
26     a -= b;
27     if (a < 0) a += mod;
28     return a;
29 }
30
31 int modMul(int a, int b, int mod) {
32     int64_t res = (int64_t) a * b;
```

```
33     return (int) (res % mod);
34 }
35
36 int64_t binPow(int64_t a, int64_t x) {
37     int64_t res = 1;
38     while (x) {
39         if (x & 1) res *= a;
40         a *= a;
41         x >>= 1;
42     }
43     return res;
44 }
45
46 int64_t modPow(int64_t a, int64_t x, int mod) {
47     int res = 1;
48     while (x) {
49         if (x & 1) res = modMul(res, a, mod);
50         a = modMul(a, a, mod);
51         x >>= 1;
52     }
53     return res;
54 }
55
56 /// This solves the equation ax + by = gcd(a, b);
57 /// Input: a, b
58 /// Output: g (returned), x, y (passed by ref)
59 int64_t extGcd(int64_t a, int64_t b, int64_t& x, int64_t& y) {
60     if (b == 0) {
61         x = 1;
62         y = 0;
63         return a;
64     }
65     int64_t x1, y1;
66     int64_t g = extGcd(b, a % b, x1, y1);
67     x = y1;
68     y = x1 - y1 * (a / b);
69     assert(g == 1);
70     return g;
71 }
72
73 /// This mod inverse function applies Fermat's little theorem
74 /// Can only be used if m is prime, and a and m are coprime
75 int64_t modInvPrimeMod(int64_t a, int64_t m, int mod) {
76     return modPow(a, m - 2, mod);
77 }
78
79 int64_t modInv(int64_t a, int mod) {
```

```

80     int64_t x, y;
81     int64_t g = extGcd(a, mod, x, y);
82     assert(g == 1);
83     return (x % mod + mod) % mod;
84 }
85
86 vector<int> calcInv(int n, int mod) {
87     vector<int> inv(n + 1);
88     inv[1] = 1;
89
90     for(int i = 2; i <= n; ++i) {
91         inv[i] = mod - (mod / i) * inv[mod % i] % mod;
92     }
93
94     return inv;
95 }
96
97 struct BinomialCoefficient {
98     const int N;
99     int mod;
100    vector<int> fact;
101    vector<int> inv_fact;
102
103    BinomialCoefficient(int N_, int mod_) : N(N_), mod(mod_), fact(N), inv_fact(N) {
104        fact[0] = 1;
105        for (int i = 1; i <= N; i++) {
106            fact[i] = modMul(fact[i - 1], i, mod);
107        }
108        inv_fact[N] = modInv(fact[N], mod);
109        for (int i = N - 1; i >= 0; i--) {
110            inv_fact[i] = modMul(inv_fact[i + 1], i + 1, mod);
111        }
112    }
113
114    int C(int n, int k) {
115        int res = modMul(fact[n], modMul(inv_fact[k], inv_fact[n - k], mod), mod);
116        return res;
117    }
118 };

```

## 1.2 Modnum

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4  #include "mod.hpp"

```

```

5
6  using namespace std;
7
8  template <typename T, int md>
9  struct Modnum {
10     using M = Modnum;
11     T v;
12     Modnum(T _v=0) : v(fix(_v)) {}
13
14     T fix(int64_t x) {
15         if (x < -md || x > 2 * md) x %= md;
16         if (x >= md) x -= md;
17         if (x < 0) x += md;
18         return x;
19     }
20
21     M operator+(M o) { return M(v + o.v); }
22     M operator-(M o) { return M(v - o.v); }
23     M operator*(M o) { return M(fix((int64_t) v * o.v)); }
24     M operator/(M o) {
25         return *this * modInv(o.v, md);
26     }
27     M pow(int64_t x) {
28         M a(v);
29         M res(1);
30         while (x) {
31             if (x & 1) res = res * a;
32             a = a * a;
33             x >>= 1;
34         }
35         return res;
36     }
37     friend istream& operator>>(istream& is, M& o) {
38         is >> o.v; o.v = o.fix(o.v); return is;
39     }
40     friend ostream& operator<<(ostream& os, const M& o) {
41         return os << o.v;
42     }
43 };

```

## 1.3 Primality Test

```

1  // Simple primality test
2
3  #pragma once
4  #include <bits/stdc++.h>

```

```

5
6 bool isPrime(int x) {
7     for (int d = 2; d * d <= x; d++) {
8         if (x % d == 0)
9             return false;
10    }
11    return true;
12 }

```

---

## 1.4 Sieve of Eratosthenes

```

1 // Computes primes in the range [2, n] in  $n \ln \ln \sqrt{n} + o(n)$  time.
2 // https://cp-algorithms.com/algebra/sieve-of-eratosthenes.html
3
4 #pragma once
5
6 #include <bits/stdc++.h>
7
8 using namespace std;
9
10 int main() {
11     int n = 100;
12     vector<char> is_prime(n+1, true);
13     is_prime[0] = is_prime[1] = false;
14     for (int i = 2; i * i <= n; i++) {
15         if (is_prime[i]) {
16             for (int j = i * i; j <= n; j += i)
17                 is_prime[j] = false;
18         }
19     }
20     for (int i = 0; i < n; i++) {
21         if (is_prime[i]) {
22             cout << i << ' ';
23         }
24     }
25 }

```

---

## 1.5 Euclidean Algorithm

```

1 // Euclidean algorithm for GCD & LCM
2 // https://cp-algorithms.com/algebra/euclid-algorithm.html
3
4 #pragma once
5 #include <bits/stdc++.h>
6

```

---

```

7 int gcd (int a, int b) {
8     return b ? gcd (b, a % b) : a;
9 }
10
11 // lcm(a,b) = a*b/(gcd(a,b))
12 int lcm (int a, int b) {
13     return a / gcd(a, b) * b;
14 }

```

---

## 1.6 Extended Euclidean Algorithm

```

1 // Extended Euclidean algorithm
2 // Solves for coefficients x,y such that ax + by = gcd(a,b)
3
4 #pragma once
5 #include <bits/stdc++.h>
6
7 int gcd(int a, int b, int& x, int& y) {
8     if (b == 0) {
9         x = 1;
10        y = 0;
11        return a;
12    }
13    int x1, y1;
14    int d = gcd(b, a % b, x1, y1);
15    x = y1;
16    y = x1 - y1 * (a / b);
17    return d;
18 }
19
20 // Iterative version
21 int gcd(int a, int b, int& x, int& y) {
22     x = 1, y = 0;
23     int x1 = 0, y1 = 1, a1 = a, b1 = b;
24     while (b1) {
25         int q = a1 / b1;
26         tie(x, x1) = make_tuple(x1, x - q * x1);
27         tie(y, y1) = make_tuple(y1, y - q * y1);
28         tie(a1, b1) = make_tuple(b1, a1 - q * b1);
29     }
30     return a1;
31 }

```

---

## 2 Geometry

### 2.1 Points

```
1  #pragma once
2
3  #include <bits/stdc++.h>
4  #include "geoutil.hpp"
5
6  using namespace std;
7
8
9  template<typename T>
10 struct Point {
11     using P = Point;
12     T x, y;
13
14     Point(T x_ = 0, T y_ = 0) : x(x_), y(y_) {}
15
16     P operator+(const P &o) const { return P(x + o.x, y + o.y); }
17
18     P operator-(const P &o) const { return P(x - o.x, y - o.y); }
19
20     P operator*(T d) const { return P(x * d, y * d); }
21
22     P operator/(T d) const { return P(x / d, y / d); }
23
24     T dot(P o) const { return x * o.x + y * o.y; }
25
26     T cross(P o) const { return x * o.y - y * o.x; }
27
28     T abs2() const { return x * x + y * y; }
29
30     long double abs() const { return sqrt((long double) abs2()); }
31
32     double angle() const { return atan2(y, x); } //  $[-\pi, \pi]$ 
33     P unit() const { return *this / abs(); } // makes  $abs()=1$ 
34     P perp() const { return P(-y, x); } // rotates  $+\pi/2$ 
35
36     P rotate(double a) const { // ccw
37         return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
38     }
39
40     friend istream &operator>>(istream &is, P &p) {
41         return is >> p.x >> p.y;
42     }
43 }
```

```
44 friend ostream &operator<<(ostream &os, P &p) {
45     return os << "(" << p.x << ", " << p.y << ")";
46 }
47
48 // position of c relative to a->b
49 // > 0: c is on the left of a->b
50 friend T orient(P a, P b, P c) {
51     return (b - a).cross(c - a);
52 }
53
54 // Check if  $\vec{u}$  and  $\vec{v}$  are parallel
55 // ( $\vec{u} = c\vec{v}$ ) where  $c \in \mathbb{R}$ )
56 friend bool parallel(P u, P v) {
57     return u.cross(v) == 0;
58 }
59
60 // Check if point p lies on the segment ab
61 friend bool onSegment(P a, P b, P p) {
62     return orient(a, b, p) == 0 &&
63         min(a.x, b.x) <= p.x &&
64         max(a.x, b.x) >= p.x &&
65         min(a.y, b.y) <= p.y &&
66         max(a.y, b.y) >= p.y;
67 }
68
69 friend bool boundingBox(P p1, P q1, P p2, P q2) {
70     if (max(p1.x, q1.x) < min(p2.x, q2.x)) return true;
71     if (max(p1.y, q1.y) < min(p2.y, q2.y)) return true;
72     if (max(p2.x, q2.x) < min(p1.x, q1.x)) return true;
73     if (max(p2.y, q2.y) < min(p1.y, q1.y)) return true;
74     return false;
75 }
76
77 friend bool intersect(P p1, P p2, P p3, P p4) {
78     // Check if two segments are parallel
79     if (parallel(p2 - p1, p4 - p3)) {
80         // Check if 4 ps are colinear
81         if (!parallel(p2 - p1, p3 - p1)) return false;
82         if (boundingBox(p1, p2, p3, p4)) return false;
83         return true;
84     }
85
86     // check if one line is completely on one side of the other
87     for (int i = 0; i < 2; i++) {
88         if (sgn((p2 - p1).cross(p3 - p1)) == sgn((p2 - p1).cross(p4 - p1))
89             && sgn((p2 - p1).cross(p3 - p1)) != 0) {
90             return false;
91         }
92     }
93 }
```

```
91     }
92     swap(p1, p3);
93     swap(p2, p4);
94 }
95 return true;
96 }
97
98 // Check if p is in ∠bac (including the rays)
99 friend bool inAngle(P a, P b, P c, P p) {
100     assert(orient(a, b, c) != 0);
101     if (orient(a, b, c) < 0) swap(b, c);
102     return orient(a, b, p) >= 0 && orient(a, c, p) <= 0;
103 }
104
105 // Angle ∠bac (+/-)
106 friend double directedAngle(P a, P b, P c) {
107     if (orient(a, b, c) >= 0) {
108         return (b - a).angle(c - a);
109     }
110     return 2 * PI - (b - a).angle(c - a);
111 }
112 };
```

---