Contest (1)

sol.cpp

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

#define rep(i, a, b) for (int i = (a); i < (b); i++)
#define all(x) begin(x), end(x)
#define sz(x) int((x).size())
using ll = long long;
using pii = pair<int, int>;
using vi = vector<int>;

#ifdef LOCAL
auto& operator<<(auto&, pair<auto, auto>);
auto operator<<(auto& o, auto x) -> decltype(x.end(), o) {
  o << '{';
  for (int i = 0; auto y : x) o << ", " + !i++ * 2 << y;
  return o << '}';
}
auto& operator<<(auto& o, pair<auto, auto> x) {
  return o << '(' << x.first << ", " << x.second << ')';
}
void __print(auto... x) { ((cerr << ' ' << x), ...) << endl; }
#define debug(x...) cerr << "[" #x "]:", __print(x)
#else
#define debug(...) 2137
#endif

int main() {
  cin.tie(0)->sync_with_stdio(0);
}
```

.vimrc

```
set nu et ts=2 sw=2
filetype indent on
syntax on
colorscheme habamax
hi MatchParen ctermfg=66 ctermbg=234 cterm=underline
nnoremap ; :
nnoremap : ;
inoremap {<cr> {<cr>}<esc>O <bs>
```

Makefile

```
CXXFLAGS=-std=c++20 -Wall -Wextra -Wshadow
sol: sol.cpp
  g++ $(CXXFLAGS) -fsanitize=address,undefined -g -DLOCAL \
      sol.cpp -o sol
fast: sol.cpp
  g++ $(CXXFLAGS) -O2 sol.cpp -o fast
```

test.sh

```
#!/bin/bash
for((i=1;i>0;i++)) do
  echo "$i"
  echo "$i" | ./gen > int
  diff -w <(./sol < int) <(./slow < int) || break
done
```

hash.sh

```
#!/bin/bash
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

.bashrc

```
alias rm='trash'
alias mv='mv -i'
alias cp='cp -i'
```

Matma (2)

2.1 Arytmetyka modularna

GCD.h

Opis: Rozszerzony algorytm Euklidesa.
Czas: $\mathcal{O}(\log \min(a, b))$

```
ll gcd(ll a, ll b, ll &x, ll &y) {
  if (!b) return x = 1, y = 0, a;
  ll d = gcd(b, a % b, y, x);
  return y -= a / b * x, d;
}
```

CRT.h

Opis: Chińskie twierdzenie o resztach.
Czas: $\mathcal{O}(\log \min(m, n))$

```
ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  ll x, y, g = gcd(m, n, x, y);
  assert((a - b) % g == 0); // no solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m * n / g : x;
}
```

ModMul.h

Opis: Mnożenie i potęgowanie dwóch long longów modulo. Jest to wyraźnie szybsze niż zamiana na __int128.

```
using ull = uint64_t;
ull modmul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
  ull ans = 1;
  for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
  return ans;
}
```

2.2 Liczby pierwsze

MillerRabin.h

Opis: Test pierwszości Millera-Rabina.

```
bool prime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
      s = __builtin_ctzll(n - 1), d = n >> s;
  for (ull a : A) {
    ull p = modpow(a % n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--)
      p = modmul(p, p, n);
    if (p != n - 1 && i != s) return 0;
  }
  return 1;
}
```

PollardRho.h

Opis: Algorytm faktoryzacji rho Pollarda.
Czas: $\mathcal{O}(n^{1/4})$

```
ull pollard(ull n) {
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  auto f = [&](ull x) { return modmul(x, x, n) + i; };
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x, y) - min(x, y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
void factor(ull n, map<ull, int>& cnt) {
  if (n == 1) return;
  if (prime(n)) { cnt[n]++; return; }
  ull x = pollard(n);
  factor(x, cnt); factor(n / x, cnt);
}
```

Geometria (3)

3.1 Podstawy

Point.h

Opis: Podstawowy szablon do geometrii.

```
template<class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct pt {
  T x, y;
  pt operator+(pt o) const { return {x + o.x, y + o.y}; }
  pt operator-(pt o) const { return {x - o.x, y - o.y}; }
  pt operator*(T a) const { return {x * a, y * a}; }
  pt operator/(T a) const { return {x / a, y / a}; }
  friend T cross(pt a, pt b) { return a.x * b.y - a.y * b.x; }
  friend T cross(pt p, pt a, pt b) {
    return cross(a - p, b - p); }
  friend T dot(pt a, pt b) { return a.x * b.x + a.y * b.y; }
  friend T dot(pt p, pt a, pt b) {
    return dot(a - p, b - p); }
  friend T abs2(pt a) { return a.x * a.x + a.y * a.y; }
  friend T abs(pt a) { return sqrt(abs2(a)); }
  auto operator<=>(pt o) const {
    return pair(sgn(x - o.x), sgn(y - o.y)) <=> pair(0, 0); }
  bool operator==(pt o) const {
    return sgn(x - o.x) == 0 && sgn(y - o.y) == 0; }
  friend auto& operator<<(auto& o, pt a) {
    return o << '(' << a.x << ", " << a.y << ')'; }
};
using P = pt<ll>;
```

AngleCmp.h

Opis: Sortuje punkty rosnąco po kącie z przedziału $(-\pi, \pi]$. Punkt $(0, 0)$ ma kąt 0.

```
bool angle_cmp(P a, P b) {
  auto half = [](P p) { return sgn(p.y) ?: -sgn(p.x); };
  int A = half(a), B = half(b);
  return A == B ? sgn(cross(a, b)) > 0 : A < B;
}
```

LineDist.h

Opis: Najkrótsza odległość między punktem i prostą/odcinkiem.

```
auto line_dist(P p, P a, P b) {
  return abs(cross(p, a, b)) / abs(b - a);
}
auto seg_dist(P p, P a, P b) {
  if (sgn(dot(a, p, b)) <= 0) return abs(p - a);
  if (sgn(dot(b, p, a)) <= 0) return abs(p - b);
  return line_dist(p, a, b);
}
```