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'
```

Struktury danych (2)

OrderedSet.h

Opis: `s.find_by_order(k)` i `s.order_of_key(k)`.
Czas: $\mathcal{O}(\log n)$

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template<class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
                         tree_order_statistics_node_update>;
```

Grafy (3)

3.1 Przepływy

Dinic.h

Opis: Dinic ze skalowaniem. Należy ustawić zakres `it` w `flow` zgodnie z U.
Czas: $\mathcal{O}(nm \log U)$

```
struct dinic {
  struct edge {
    int to, rev;
    ll cap;
  };
  vi lvl, ptr, q;
  vector<vector<edge>> adj;
  dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
  void add_edge(int u, int v, ll cap, ll rcap = 0) {
    int i = sz(adj[u]), j = sz(adj[v]);
    adj[u].push_back({v, j + (u == v), cap});
    adj[v].push_back({u, i, rcap});
  }
  ll dfs(int v, int t, ll f) {
    if (v == t || !f) return f;
    for (int& i = ptr[v]; i < sz(adj[v]); i++) {
      edge& e = adj[v][i];
      if (lvl[e.to] == lvl[v] + 1)
        if (ll p = dfs(e.to, t, min(f, e.cap))) {
          e.cap -= p, adj[e.to][e.rev].cap += p;
          return p;
        }
    }
    return 0;
  }
  ll flow(int s, int t) {
    ll f = 0; q[0] = s;
    for (int it = 29; it >= 0; it--) do {
      lvl = ptr = vi(sz(q));
      int qi = 0, qe = lvl[s] = 1;
      while (qi < qe && !lvl[t]) {
        int v = q[qi++];
        for (edge e : adj[v])
          if (!lvl[e.to] && e.cap >> it)
            q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
      }
      while (ll p = dfs(s, t, LLONG_MAX)) f += p;
    } while (lvl[t]);
    return f;
  }
};
```

GomoryHu.h

Opis: Tworzy drzewo gdzie min cut to minimum na ścieżce.
Czas: $\mathcal{O}(n)$ przepływów

```
struct edge { int u, v; ll w; };
vector<edge> gomory_hu(int n, const vector<edge>& ed) {
  vector<edge> t; vi p(n);
  rep(i, 1, n) {
    dinic d(n);
    for (edge e : ed) d.add_edge(e.u, e.v, e.w, e.w);
    t.push_back({i, p[i], d.flow(i, p[i])});
    rep(j, i + 1, n) if (p[j] == p[i] && d.lvl[j]) p[j] = i;
  }
  return t;
}
```

MCMF.h

Opis: MCMF z Dijkstrą. Jeżeli są ujemne krawędzie to przed puszczeniem `flow` w `pi` trzeba policzyć najkrótsze ścieżki z `s`.
Czas: $\mathcal{O}(Fm \log n)$

```
#include <ext/pb_ds/priority_queue.hpp>
const ll INF = 2e18;
struct MCMF {
  struct edge {
    int from, to, rev;
    ll cap, cost;
  };
  int n;
  vector<vector<edge>> adj;
  vector<ll> dst, pi;
  __gnu_pbds::priority_queue<pair<ll, int>> q;
  vector<decltype(q)::point_iterator> it;
  vector<edge*> p;
  MCMF(int _n) : n(_n), adj(n), pi(n), p(n) {}
  void add_edge(int u, int v, ll cap, ll cost) {
    int i = sz(adj[u]), j = sz(adj[v]);
    adj[u].push_back({u, v, j + (u == v), cap, cost});
    adj[v].push_back({v, u, i, 0, -cost});
  }
  bool path(int s, int t) {
    dst.assign(n, INF); it.assign(n, q.end());
    q.push({dst[s] = 0, s});
    while (!q.empty()) {
```

```
      int u = q.top().second; q.pop();
      for (edge& e : adj[u]) {
        ll d = dst[u] + pi[u] + e.cost - pi[e.to];
        if (e.cap && d < dst[e.to]) {
          dst[e.to] = d, p[e.to] = &e;
          if (it[e.to] == q.end())
            it[e.to] = q.push({-dst[e.to], e.to});
          else
            q.modify(it[e.to], {-dst[e.to], e.to});
        }
      }
    }
    rep(i, 0, n) pi[i] = min(pi[i] + dst[i], INF);
    return pi[t] != INF;
  }
  pair<ll, ll> flow(int s, int t, ll cap) {
    ll f = 0, c = 0;
    while (f < cap && path(s, t)) {
      ll d = cap - f;
      for (edge* e = p[t]; e; e = p[e->from])
        d = min(d, e->cap);
      for (edge* e = p[t]; e; e = p[e->from])
        e->cap -= d, adj[e->to][e->rev].cap += d;
      f += d, c += d * pi[t];
    }
    return {f, c};
  }
};
```

3.2 DFS

SCC.h

Opis: Znajduje SCC w kolejności topologicznej.
Czas: $\mathcal{O}(n+m)$

```
struct SCC {
  int n, t = 0, cnt = 0;
  vector<vi> adj;
  vi val, p, st;
  SCC(int _n) : n(_n), adj(n), val(n), p(n, -1) {}
  void add_edge(int u, int v) { adj[u].push_back(v); }
  int dfs(int u) {
    int low = val[u] = ++t; st.push_back(u);
    for (int v : adj[u]) if (p[v] == -1)
      low = min(low, val[v] ?: dfs(v));
    if (low == val[u]) {
      for (int x = -1; x != u;)
        p[x = st.back()] = cnt, st.pop_back();
      cnt++;
    }
    return low;
  }
  void build() {
    rep(i, 0, n) if (!val[i]) dfs(i);
    rep(i, 0, n) p[i] = cnt - 1 - p[i];
  }
};
```

TwoSat.h

Opis: 2-SAT.
Czas: $\mathcal{O}(n+m)$

```
struct two_sat {
  int n;
  vector<pii> ed;
  vector<bool> b;
  two_sat(int _n) : n(_n) {}
  int add_var() { return n++; }
  void either(int x, int y) {
    x = max(2 * x, -1 - 2 * x), y = max(2 * y, -1 - 2 * y);
    ed.push_back({x, y}); }
  void implies(int x, int y) { either(~x, y); }
  void must(int x) { either(x, x); }
  void at_most_one(const vi& v) {
    if (sz(v) <= 1) return;
    int cur = ~v[0];
    rep(i, 2, sz(v)) {
      int nxt = add_var();
      either(cur, ~v[i]); either(cur, nxt);
      either(~v[i], nxt); cur = ~nxt;
    }
    either(cur, ~v[1]);
  }
  bool solve() {
    SCC scc(2 * n);
    for (auto [u, v] : ed)
      scc.add_edge(u ^ 1, v), scc.add_edge(v ^ 1, u);
    scc.build(); b.resize(n, 1);
    rep(i, 0, n) {
      if (scc.p[2 * i] == scc.p[2 * i + 1]) return 0;
      if (scc.p[2 * i] < scc.p[2 * i + 1]) b[i] = 0;
    }
    return 1;
  }
};
```

Matma (4)

4.1 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;
}
```

ModInt.h

```
template<int M, int R>
struct mod {
  static const int MOD = M, ROOT = R;
  int x;
  mod(ll y = 0) : x(y % M) { x += (x < 0) * M; }
  mod operator+=(mod o) {
    if ((x += o.x) >= M) x -= M;
    return *this; }
  mod operator-=(mod o) {
    if ((x -= o.x) < 0) x += M;
    return *this; }
  mod operator*=(mod o) {
    x = 1ll * x * o.x % M;
    return *this; }
  mod operator/=(mod o) { return (*this) *= o.inv(); }
  friend mod operator+(mod a, mod b) { return a += b; }
  friend mod operator-(mod a, mod b) { return a -= b; }
  friend mod operator*(mod a, mod b) { return a *= b; }
  friend mod operator/(mod a, mod b) { return a /= b; }
  auto operator<=>(const mod&) const = default;
  mod pow(ll n) const {
    mod a = x, b = 1;
    for (; n; n /= 2, a *= a) if (n & 1) b *= a;
    return b;
  }
  mod inv() const { return pow(M - 2); }
};
using mint = mod<998244353, 3>;
```

4.2 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);
}
```

4.3 Wielomiany

NTT.h
Czas: $\mathcal{O}((n+m)\log(n+m))$

```
template<class T>
void ntt(vector<T>& a, bool inv) {
  int n = sz(a); vector<T> b(n);
  for (int i = n / 2; i; i /= 2, swap(a, b)) {
    T w = T(T::ROOT).pow((T::MOD - 1) / n * i), m = 1;
    for (int j = 0; j < n; j += 2 * i, m *= w) rep(k, 0, i) {
      T u = a[j + k], v = a[j + k + i] * m;
      b[j / 2 + k] = u + v, b[j / 2 + k + n / 2] = u - v;
    }
  }
  if (inv) {
    reverse(1 + all(a));
    T z = T(n).inv(); rep(i, 0, n) a[i] *= z;
  }
}
template<class T>
vector<T> conv(vector<T> a, vector<T> b) {
  int s = sz(a) + sz(b) - 1, n = 1 << __lg(2 * s - 1);
  a.resize(n); ntt(a, 0); b.resize(n); ntt(b, 0);
  rep(i, 0, n) a[i] *= b[i];
  ntt(a, 1); a.resize(s);
  return a;
}
```

Conv3.h
Opis: NTT z Garnerem. Działa dla $n+m \leq 2^{24}$ i $c_k \leq 5 \cdot 10^{25}$.
Czas: $\mathcal{O}((n+m)\log(n+m))$

```
template<class T>
vector<T> mconv(const auto& x, const auto& y) {
  auto con = [&](const auto& v) {
    vector<T> w(sz(v)); rep(i, 0, sz(v)) w[i] = v[i].x;
    return w; };
  return conv(con(x), con(y));
}
template<class T>
vector<T> conv3(const vector<T>& a, const vector<T>& b) {
  using m0 = mod<754974721, 11>; auto c0 = mconv<m0>(a, b);
  using m1 = mod<167772161, 3>;  auto c1 = mconv<m1>(a, b);
  using m2 = mod<469762049, 3>;  auto c2 = mconv<m2>(a, b);
  int n = sz(c0); vector<T> d(n); m1 r01 = m1(m0::MOD).inv();
  m2 r02 = m2(m0::MOD).inv(), r12 = m2(m1::MOD).inv();
  rep(i, 0, n) {
    int x = c0[i].x, y = ((c1[i] - x) * r01).x,
        z = (((c2[i] - x) * r02 - y) * r12).x;
    d[i] = (T(z) * m1::MOD + y) * m0::MOD + x;
  }
  return d;
}
```

Teksty (5)

5.1 Podstawy

KMP.h
Czas: $\mathcal{O}(n)$

```
vi kmp(const string& s) {
  vi p(sz(s));
  rep(i, 1, sz(s)) {
    int g = p[i - 1];
    while (g && s[i] != s[g]) g = p[g - 1];
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}
```

Z.h
Czas: $\mathcal{O}(n)$

```
vi z(const string& s) {
  int n = sz(s), l = -1, r = -1;
  vi f(n); f[0] = n;
  rep(i, 1, sz(s)) {
    if (i < r) f[i] = min(r - i, f[i - l]);
    while (i + f[i] < n && s[i + f[i]] == s[f[i]]) f[i]++;
    if (i + f[i] > r) l = i, r = i + f[i];
  }
  return f;
}
```

Manacher.h
Opis: `p[1][k]` – środek w k, `p[0][k]` – środek między $k-1$ a k.
Czas: $\mathcal{O}(n)$

```
array<vi, 2> manacher(const string& s) {
  int n = sz(s);
  array<vi, 2> p = {vi(n + 1), vi(n)};
  rep(z, 0, 2) for (int i = 0, l = 0, r = 0; i < n; i++) {
    int t = r - i + !z;
    if (i < r) p[z][i] = min(t, p[z][l + t]);
    int L = i - p[z][i], R = i + p[z][i] - !z;
    while (L >= 1 && R + 1 < n && s[L - 1] == s[R + 1])
      p[z][i]++, L--, R++;
    if (R > r) l = L, r = R;
  }
  return p;
}
```

Duval.h
Opis: Rozkłada słowo na nierosnący ciąg podsłów mniejszych od swoich wszystkich nietrywialnych sufiksów.
Czas: $\mathcal{O}(n)$

```
vi duval(const string& s) {
  int n = sz(s); vi f;
  for (int i = 0; i < n;) {
    int j = i + 1, k = i;
    for (; j < n && s[k] <= s[j]; j++) {
      if (s[k] < s[j]) k = i;
      else ++k;
    }
    for (; i <= k; i += j - k) f.push_back(i);
  }
  return f.push_back(n), f;
}
```

5.2 Struktury sufiksowe

SuffixArray.h
Opis: Zawiera pusty sufiks. `lcp[k]` – najdłuższy wspólny prefiks $k-1$ i k.
Czas: $\mathcal{O}(n\log n)$

```
struct suffix_array {
  vi sa, lcp;
  suffix_array(string s, int lim = 128) {
    int n = sz(s) + 1, k = 0, a, b;
    vi x(all(s) + 1), y(n), ws(max(n, lim)), rank(n);
    sa = lcp = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
      p = j, iota(all(y), n - j);
      rep(i, 0, n) if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      rep(i, 0, n) ws[x[i]]++;
      rep(i, 1, lim) ws[i] += ws[i - 1];
      for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      rep(i, 1, n) a = sa[i - 1], b = sa[i], x[b] =
        (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
    }
    rep(i, 1, n) rank[sa[i]] = i;
    for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
      for (k && k--, j = sa[rank[i] - 1];
          s[i + k] == s[j + k]; k++);
  }
};
```

Geometria (6)

6.1 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);
}
```

6.2 Wielokąty

ConvexHull.h

Opis: Otoczka wypukła w kierunku CCW.
Czas: $\mathcal{O}(n \log n)$

```
vector<P> convex_hull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts) + 1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 &&
             sgn(cross(h[t - 2], h[t - 1], p)) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```