

UW
1 Contest
2 Wzory
3 Struktury danych
4 Grafy
5 Matma
6 Teksty
7 Geometria
8 Inne

Contest (1)

```
sol.cpp

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

#ifdef LOCAL
auto& operator<<(auto&, pair<auto, auto>);
auto& operator<<(auto& o, auto x) {
    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() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
}
```

```
.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
```

```
sol.cpp .vimrc Makefile test.sh WaveletTree.h

1      g++ $(CXXFLAGS) -fsanitize=address,undefined -g -DLOCAL \
2      sol.cpp -o sol

1 fast: sol.cpp
2      g++ $(CXXFLAGS) -O2 sol.cpp -o fast

1 test.sh

2 #!/bin/bash

3 for((i=1;i>0;i++)) do
4     echo "$i"
5     echo "$i" | ./gen > int
6     diff -w <(. /sol < int) <(. /slow < int) || break
7 done

5

5 Wzory (2)
```

2.1 Grafy

2.1.1 Twierdzenie Königa

W grafie dwudzielnym zachodzi

- $n_k = p_w$
- $n_k + p_k = n$
- $p_w + n_w = n$

oraz

- $p_w$  to zbiór wierzchołków na brzegu min-cut
- $n_w$  to dopełnienie  $p_w$
- $p_k$  to  $n_k$  z dodanymi pojedynczymi krawędziami każdego nieskojarzonego wierzchołka

2.1.2 Twierdzenie Erdősa-Gallaia

Ciąg stopni  $d_1 \geq \dots \geq d_n$  opisuje prosty graf wtw gdy  $\sum d_i$  jest parzysta oraz dla każdego  $1 \leq k \leq n$  zachodzi

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k).$$

2.1.3 Twierdzenie Gale’a-Rysera

Ciągi stopni  $a_1 \geq \dots \geq a_n$  oraz  $b_1, \dots, b_n$  opisują prosty graf dwudzielnym wtw gdy  $\sum a_i = \sum b_i$  oraz dla każdego  $1 \leq k \leq n$  zachodzi

$$\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k).$$

2.1.4 Przepływy z wymaganiami

Szukamy przepływu  $\leq F$  takiego, że  $f_i \geq d_i$  dla każdej krawędzi. Tworzymy nowe źródło  $s'$  i ujście  $t'$ . Następnie dodajemy krawędzie

1

- $(u_i, t', d_i), (s', v_i, d_i), (u_i, v_i, c_i - d_i)$  zamiast  $(u_i, v_i, c_i, d_i)$
- $(t, s, F)$

Przepływ spełnia wymagania jeżeli maksymalnie wypełnia wszystkie krawędzie  $s'$ .

2.2 Optymalizacja

2.2.1 Mnożniki Lagrange’a

Jeżeli optymalizujemy  $f(x_1, \dots, x_n)$  przy ograniczeniach typu  $g_k(x_1, \dots, x_n) = 0$  to  $x_1, \dots, x_n$  jest ekstremum lokalnym tylko jeżeli gradient  $\nabla f(x_1, \dots, x_n)$  jest kombinacją liniową gradientów  $\nabla g_k(x_1, \dots, x_n)$ .

Struktury danych (3)

```
WaveletTree.h
Stosowanie: st – początek, ed – koniec, sst – posortowany początek.
Czas:  $\mathcal{O}((n + q) \log n)$ 

struct node {
    int lo, hi;
    vector<int> s;
    node *l = 0, *r = 0;
    node(auto st, auto ed, auto sst) {
        int n = ed - st;
        lo = sst[0];
        hi = sst[n - 1] + 1;
        if (lo + 1 < hi) {
            int mid = sst[n / 2];
            if (mid == sst[0]) mid = *upper_bound(sst, sst + n, mid);
            s.reserve(n + 1);
            s.push_back(0);
            for (auto it = st; it != ed; it++) {
                s.push_back(s.back() + (*it < mid));
            }
            auto k = stable_partition(st, ed, [&](int x) {
                return x < mid;
            });
            auto sm = lower_bound(sst, sst + n, mid);
            if (k != st) l = new node(st, k, sst);
            if (k != ed) r = new node(k, ed, sm);
        }
    }
    int kth(int a, int b, int k) {
        if (lo + 1 == hi) return lo;
        int x = s[a], y = s[b];
        return k < y - x ? l->kth(x, y, k)
            : r->kth(a - x, b - y, k - (y - x));
    }
    int count(int a, int b, int k) {
        if (lo >= k) return 0;
        if (hi <= k) return b - a;
        int x = s[a], y = s[b];
        return (l ? l->count(x, y, k) : 0) +
            (r ? r->count(a - x, b - y, k) : 0);
    }
    int freq(int a, int b, int k) {
        if (k < lo || hi <= k) return 0;
        if (lo + 1 == hi) return b - a;
        int x = s[a], y = s[b];
        return (l ? l->freq(x, y, k) : 0) +
```

```
        (r ? r->freq(a - x, b - y, k) : 0);
    }
};
```

### OrderedSet.h

**Stosowanie:** 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 <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
                        tree_order_statistics_node_update>;
```

### Treap.h

**Czas:**  $\mathcal{O}(\log n)$

```
mt19937_64 rng(2137);
struct node {
    int val, sz = 1;
    uint64_t pr;
    node *l = 0, *r = 0;
    node(int x) {
        val = x;
        pr = rng();
    }
    void pull() {
        sz = 1 + size(l) + size(r);
    }
    friend int size(node* a) {
        return a ? a->sz : 0;
    }
    friend pair<node*, node*> split(node* a, int k) {
        if (!a) return {0, 0};
        if (k <= size(a->l)) {
            auto [la, lb] = split(a->l, k);
            a->l = lb;
            a->pull();
            return {la, a};
        } else {
            auto [ra, rb] = split(a->r, k - size(a->l) - 1);
            a->r = ra;
            a->pull();
            return {a, rb};
        }
    }
    friend node* merge(node* a, node* b) {
        if (!a || !b) return a ? a : b;
        if (a->pr > b->pr) {
            a->r = merge(a->r, b);
            a->pull();
            return a;
        } else {
            b->l = merge(a, b->l);
            b->pull();
            return b;
        }
    }
};
```

### LineSet.h

**Opis:** Znajduje maksimum funkcji liniowych online. Dla doubli  $\text{div}(a,b) = a/b$  oraz  $\text{INF} = 1/.0$ .  
**Czas:**  $\mathcal{O}(\log n)$

```
struct line {
    mutable ll a, b, p;
```

### OrderedSet.h Treap.h LineSet.h Dinic.h MCMF.h

```
    bool operator<(const line& o) const { return a < o.a; }
    bool operator<(ll x) const { return p < x; }
};
struct line_set : multiset<line, less<>> {
    static const ll INF = LLONG_MAX;
    ll div(ll a, ll b) {
        return a / b - ((a ^ b) < 0 && a % b);
    }
    bool inter(iterator x, iterator y) {
        if (y == end()) return x->p = INF, false;
        if (x->a == y->a) x->p = x->b > y->b ? INF : -INF;
        else x->p = div(y->b - x->b, x->a - y->a);
        return x->p >= y->p;
    }
    void add(ll a, ll b) {
        auto z = insert({a, b, 0}), y = z++, x = y;
        while (inter(y, z)) z = erase(z);
        if (x != begin() && inter(--x, y)) inter(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p) {
            inter(x, erase(y));
        }
    }
    ll get(ll x) {
        line l = *lower_bound(x);
        return l.a * x + l.b;
    }
};
```

## Grafy (4)

### 4.1 Przepływy

#### Dinic.h

**Czas:**  $\mathcal{O}(nm \log U)$

```
struct dinic {
    struct edge {
        int to, rev;
        int cap;
    };
    int n;
    vector<vector<edge>> adj;
    vector<int> q, lvl, it;
    dinic(int _n) {
        n = _n;
        adj.resize(n);
        q.reserve(n);
        lvl.resize(n);
        it.resize(n);
    }
    void add_edge(int u, int v, int cap) {
        int i = ssize(adj[u]), j = ssize(adj[v]) + (u == v);
        adj[u].push_back({v, j, cap});
        adj[v].push_back({u, i, 0});
    }
    bool bfs(int s, int t, int r) {
        q.clear();
        lvl.assign(n, -1);
        lvl[s] = 0;
        q.push_back(s);
        for (int i = 0; i < ssize(q); i++) {
            int u = q[i];
            for (edge& e : adj[u]) {
                if (e.cap >= r && lvl[e.to] == -1) {
                    lvl[e.to] = lvl[u] + 1;
```

```
                    q.push_back(e.to);
                    if (e.to == t) return true;
                }
            }
        }
        return false;
    }
    ll dfs(int u, int t, ll cap) {
        if (u == t) return cap;
        ll f = 0;
        for (int& i = it[u]; i < ssize(adj[u]); i++) {
            edge& e = adj[u][i];
            if (e.cap > 0 && lvl[u] + 1 == lvl[e.to]) {
                ll add = dfs(e.to, t, min(cap - f, (ll)e.cap));
                e.cap -= add;
                adj[e.to][e.rev].cap += add;
                f += add;
            }
            if (f == cap) return f;
        }
        lvl[u] = -1;
        return f;
    }
    ll flow(int s, int t, ll cap) {
        ll f = 0;
        for (int i = 29; i >= 0; i--) {
            while (f < cap && bfs(s, t, 1 << i)) {
                it.assign(n, 0);
                f += dfs(s, t, cap - f);
            }
        }
        return f;
    }
};
```

#### MCMF.h

**Stosowanie:** Jeżeli są ujemne krawędzie, przed pusczeniem flow w dst trzeba policzyć najkrótsze ścieżki z s i puścić reduce(t).  
**Czas:**  $\mathcal{O}(Fm \log n)$

```
#include <ext/pb_ds/priority_queue.hpp>
ll INF64 = 2e18;
struct MCMF {
    struct edge {
        int to, rev;
        int cap;
        ll cost;
    };
    struct cmp {
        bool operator()(const auto& l, const auto& r) const {
            return l.second > r.second;
        }
    };
    int n;
    vector<vector<edge>> adj;
    vector<ll> dst;
    ll c = 0;
    __gnu_pbds::priority_queue<pair<int, ll>, cmp> q;
    vector<decltype(q)::point_iterator> its;
    vector<int> id;
    MCMF(int _n) {
        n = _n;
        adj.resize(n);
        id.resize(n);
    }
    void add_edge(int u, int v, int cap, int cost) {
        int i = ssize(adj[u]), j = ssize(adj[v]) + (u == v);
        adj[u].push_back({v, j, cap, cost});
        adj[v].push_back({u, i, 0, -cost});
```

```

}
void reduce(int t) {
    for (int i = 0; i < n; i++) {
        for (edge& e : adj[i]) {
            if (dst[i] != INF64 && dst[e.to] != INF64) {
                e.cost += dst[i] - dst[e.to];
            }
        }
    }
    c += dst[t];
}
bool dijkstra(int s, int t) {
    dst.assign(n, INF64);
    its.assign(n, q.end());
    dst[s] = 0;
    q.push({s, 0});
    while (!q.empty()) {
        int u = q.top().first;
        q.pop();
        for (edge& e : adj[u]) {
            if (e.cap > 0) {
                ll d = dst[u] + e.cost;
                if (d < dst[e.to]) {
                    dst[e.to] = d;
                    if (its[e.to] == q.end()) {
                        its[e.to] = q.push({e.to, dst[e.to]});
                    } else {
                        q.modify(its[e.to], {e.to, dst[e.to]});
                    }
                    id[e.to] = e.rev;
                }
            }
        }
    }
    reduce(t);
    return dst[t] != INF64;
}
pair<ll, ll> flow(int s, int t, ll cap) {
    ll ff = 0;
    ll cc = 0;
    while (ff < cap && dijkstra(s, t)) {
        ll f = cap - ff;
        for (int i = t; i != s;) {
            edge& e = adj[i][id[i]];
            f = min(f, (ll)adj[e.to][e.rev].cap);
            i = e.to;
        }
        for (int i = t; i != s;) {
            edge& e = adj[i][id[i]];
            e.cap -= f;
            adj[e.to][e.rev].cap += f;
            i = e.to;
        }
        ff += f;
        cc += f * c;
    }
    return {ff, cc};
}
};
```

4.2 Skojarzenia

Matching.h  
Czas:  $\mathcal{O}(m\sqrt{n})$

```

struct matching {
    int n, m;
```

```

vector<vector<int>>> adj;
vector<int> pb, pa;
vector<int> lvl, it;
matching(int _n, int _m) {
    n = _n;
    m = _m;
    adj.resize(n);
    pb.resize(n, -1);
    pa.resize(m, -1);
    it.resize(n);
}
void add_edge(int u, int v) {
    adj[u].push_back(v);
}
bool bfs() {
    bool res = false;
    lvl.assign(n, -1);
    queue<int> q;
    for (int i = 0; i < n; i++) {
        if (pb[i] == -1) {
            q.push(i);
            lvl[i] = 0;
        }
    }
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int j : adj[u]) {
            if (pa[j] == -1) {
                res = true;
            } else if (lvl[pa[j]] == -1) {
                lvl[pa[j]] = lvl[u] + 1;
                q.push(pa[j]);
            }
        }
    }
    return res;
}
bool dfs(int u) {
    for (auto& i = it[u]; i < ssize(adj[u]); i++) {
        int v = adj[u][i];
        if (pa[v] == -1 ||
            (lvl[pa[v]] == lvl[u] + 1 && dfs(pa[v]))) {
            pb[u] = v;
            pa[v] = u;
            return true;
        }
    }
    return false;
}
int match() {
    int ans = 0;
    while (bfs()) {
        it.assign(n, 0);
        for (int i = 0; i < n; i++) {
            if (pb[i] == -1 && dfs(i)) ans++;
        }
    }
    return ans;
}
};
```

4.3 Grafy skierowane

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

```

struct SCC {
    int n, cnt = 0;
    vector<vector<int>>> adj;
    vector<int> p, low, in;
    stack<int> st;
    int tour = 0;
    SCC(int _n) {
        n = _n;
        adj.resize(n);
        p.resize(n, -1);
        low.resize(n);
        in.resize(n, -1);
    }
    void add_edge(int u, int v) {
        adj[u].push_back(v);
    }
    void dfs(int u) {
        low[u] = in[u] = tour++;
        st.push(u);
        for (int v : adj[u]) {
            if (in[v] == -1) {
                dfs(v);
                low[u] = min(low[u], low[v]);
            } else {
                low[u] = min(low[u], in[v]);
            }
        }
        if (low[u] == in[u]) {
            int v = -1;
            do {
                v = st.top();
                st.pop();
                in[v] = n;
                p[v] = cnt;
            } while (v != u);
            cnt++;
        }
    }
    void build() {
        for (int i = 0; i < n; ++i) {
            if (in[i] == -1) dfs(i);
        }
        for (int i = 0; i < n; i++) p[i] = cnt - 1 - p[i];
    }
};
```

Matma (5)

5.1 Arytmetyka modularna

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+=(const mod& o) {
        if ((x += o.x) >= M) x -= M;
        return *this;
    }
    mod operator-=(const mod& o) {
        if ((x -= o.x) < 0) x += M;
        return *this;
    }
};
```

```
mod operator*=(const mod& o) {
    x = 1ll * x * o.x % M;
    return *this;
}
mod operator/=(const mod& o) {
    return (*this) *= o.inv();
}
friend mod operator+(mod a, const mod& b) { return a += b; }
friend mod operator-(mod a, const mod& b) { return a -= b; }
friend mod operator*(mod a, const mod& b) { return a *= b; }
friend mod operator/(mod a, const mod& b) { return a /= b; }
auto operator<=>(const mod& const = default;
mod pow(1ll n) const {
    mod a = x, b = 1;
    while (n > 0) {
        if (n % 2 == 1) b *= a;
        a *= a;
        n /= 2;
    }
    return b;
}
mod inv() const {
    return pow(M - 2);
}
};
using mint = mod<998244353, 3>;
```

GCD.h

Opis: Znajduje  $x$  i  $y$  takie, że  $ax + by = \gcd(a, b)$ .  
Czas:  $\mathcal{O}(\log(\min(a, b)))$

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

5.2 Sploty

NTT.h

Stosowanie: Jeżeli  $\text{MOD} = 998244353$  to  $n + m \leq 2^{23}$ .  
Czas:  $\mathcal{O}((n + m) \log(n + m))$

```
template<typename T>
void ntt(vector<T>& a, bool inv) {
    int n = ssize(a);
    vector<T> b(n);
    for (int i = n / 2; i > 0; 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) {
            for (int k = 0; k < i; k++) {
                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(a.begin() + 1, a.end());
        T ni = T(n).inv();
        for (int i = 0; i < n; i++) a[i] *= ni;
    }
}
template<typename T>
vector<T> conv(vector<T> a, vector<T> b) {
    int s = ssize(a) + ssize(b) - 1;
    int n = 1 << (___lg(2 * s - 1));
```

```
a.resize(n); b.resize(n);
ntt(a, false); ntt(b, false);
for (int i = 0; i < n; i++) a[i] *= b[i];
ntt(a, true);
a.resize(s);
return a;
}
```

NTT3.h

Stosowanie:  $n + m \leq 2^{24}$ .  
Czas:  $\mathcal{O}((n + m) \log(n + m))$

```
template<typename T>
vector<T> mconv(const auto& a, const auto& b) {
    auto cp = [&](const auto& v) {
        vector<T> vv(ssize(v));
        for (int i = 0; i < ssize(v); i++) vv[i] = T(v[i].x);
        return vv;
    };
    return conv(cp(a), cp(b));
}
template<typename 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);
    m1 r01 = m1(m0::MOD).inv();
    m2 r02 = m2(m0::MOD).inv(), r12 = m2(m1::MOD).inv();
    vector<T> d(ssize(c0));
    for (int i = 0; i < ssize(c0); i++) {
        int a = c0[i].x;
        int b = ((c1[i] - a) * r01).x;
        int c = (((c2[i] - a) * r02 - b) * r12).x;
        d[i] = (T(c) * m1::MOD + b) * m0::MOD + a;
    }
    return d;
}
```

FST.h

Stosowanie:  $n$  musi być potęgą dwójki.  
Czas:  $\mathcal{O}(n \log n)$

```
void fst(vector<mint>& a, bool inv) {
    int n = ssize(a);
    for (int i = 1; i < n; i *= 2) {
        for (int j = 0; j < n; j += 2 * i) {
            for (int k = 0; k < i; k++) {
                mint u = a[j + k], v = a[j + k + i];
                a[j + k] = u + v, a[j + k + i] = u - v; // XOR
                // a[j + k] = inv ? u - v : u + v; // AND
                // a[j + k + i] = inv ? v - u : u + v; // OR
            }
        }
    }
    // XOR
    if (inv) {
        mint ni = mint(n).inv();
        for (int i = 0; i < n; i++) a[i] = a[i] * ni;
    }
}
vector<mint> conv(vector<mint> a, vector<mint> b) {
    int n = ssize(a);
    fst(a, false); fst(b, false);
    for (int i = 0; i < n; i++) a[i] = a[i] * b[i];
    fst(a, true);
    return a;
}
```

Teksty (6)

KMP.h

Czas:  $\mathcal{O}(n)$

```
vector<int> kmp(const string& s) {
    int n = ssize(s);
    vector<int> p(n);
    for (int i = 1; i < n; i++) {
        int j = p[i - 1];
        while (j > 0 && s[i] != s[j]) j = p[j - 1];
        p[i] = j + (s[i] == s[j]);
    }
    return p;
}
```

Manacher.h

Stosowanie: Zwraca długość najdłuższego palindromu.  $p[2 * i]$  – środek w  $i$ ,  $p[2 * i + 1]$  – środek między  $i$  a  $i + 1$ .  
Czas:  $\mathcal{O}(n)$

```
vector<int> manacher(const string& s) {
    int n = ssize(s);
    string t(2 * n - 1, '#');
    for (int i = 0; i < n; i++) t[2 * i] = s[i];
    vector<int> p(2 * n - 1);
    for (int i = 0, l = -1, r = -1; i < 2 * n - 1; i++) {
        if (i <= r) p[i] = min(r - i + 1, p[l + r - i]);
        while (p[i] < min(i + 1, 2 * n - 1 - i)) {
            if (t[i - p[i]] != t[i + p[i]]) break;
            p[i]++;
        }
        if (i + p[i] - 1 > r) {
            l = i - p[i] + 1;
            r = i + p[i] - 1;
        }
    }
    for (int i = 0; i < 2 * n - 1; i++) {
        p[i] -= t[i - p[i] + 1] == '#';
    }
    return p;
}
```

SuffixArray.h

Stosowanie: Jeżeli tekst ma znaki inne niż a-z trzeba zmienić inicjalizację.  
Czas:  $\mathcal{O}(n \log n)$

```
vector<int> suffix_array(const string& s) {
    int n = ssize(s);
    vector<int> p(n), cnt(26);
    for (int i = 0; i < n; i++) cnt[s[i] - 'a']++;
    for (int i = 1; i < 26; i++) cnt[i] += cnt[i - 1];
    for (int i = 0; i < n; i++) p[--cnt[s[i] - 'a']] = i;
    vector<int> rnk(n);
    for (int i = 1; i < n; i++) {
        rnk[p[i]] = s[p[i]] == s[p[i - 1]] ? rnk[p[i - 1]] : i;
    }
    cnt.resize(n);
    vector<int> np(n), nrnk(n);
    for (int len = 1; len < n; len *= 2) {
        iota(cnt.begin(), cnt.end(), 0);
        for (int i = n - len; i < n; i++) np[cnt[rnk[i]]++] = i;
        for (int i = 0; i < n; i++) {
            if (p[i] - len >= 0) {
                np[cnt[rnk[p[i] - len]]++] = p[i] - len;
            }
        }
        nrnk[np[0]] = 0;
```

```
    for (int i = 1; i < n; i++) {
        int a = np[i - 1];
        int b = np[i];
        if (max(a, b) + len < n && rnk[a] == rnk[b] &&
            rnk[a + len] == rnk[b + len]) {
            nrnk[b] = nrnk[a];
        } else {
            nrnk[b] = i;
        }
    }
    swap(p, np);
    swap(rnk, nrnk);
}
return p;
};
vector<int> build_lcp(const string& s, const vector<int>& sa) {
    int n = ssize(s);
    vector<int> pos(n);
    for (int i = 0; i < n; i++) pos[sa[i]] = i;
    vector<int> lcp(n - 1);
    int k = 0;
    for (int i = 0; i < n; i++) {
        if (pos[i] == 0) continue;
        while (i + k < n && s[i + k] == s[sa[pos[i] - 1] + k]) k++;
        lcp[pos[i] - 1] = k;
        k = max(0, k - 1);
    }
    return lcp;
}
```

Z.h

Czas:  $\mathcal{O}(n)$

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

Geometria (7)

Point.h

```
struct pt {
    ll 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*(ll a) const { return {x * a, y * a}; }
    pt operator/(ll a) const { return {x / a, y / a}; }
    auto operator<=>(const pt&) const = default;
    friend ll cross(pt a, pt b) { return a.x * b.y - a.y * b.x; }
    friend ll dot(pt a, pt b) { return a.x * b.x + a.y * b.y; }
    friend ll norm(pt a) { return a.x * a.x + a.y * a.y; }
    friend int half(pt a) {
        if (a.y < 0) return -1;
        if (a.y == 0 && a.x >= 0) return 0;
        return 1;
    }
};
```

```
    }
    friend auto& operator<<(auto& o, pt a) {
        return o << '(' << a.x << ", " << a.y << ')';
    }
};
```

ConvexHull.h

Stosowanie: Usuwa punkty współliniowe.

Czas:  $\mathcal{O}(n \log n)$

```
vector<pt> convex_hull(vector<pt> p) {
    if (ssize(p) <= 1) return p;
    sort(p.begin(), p.end());
    vector<pt> h(ssize(p) + 1);
    int s = 0, t = 0;
    for (int it = 0; it < 2; it++) {
        for (pt a : p) {
            while (t >= s + 2) {
                pt u = h[t - 2], v = h[t - 1];
                if (cross(v - u, a - v) <= 0) t--;
                else break;
            }
            h[t++] = a;
        }
        reverse(p.begin(), p.end());
        s = --t;
    }
    h.resize(t - (t == 2 && h[0] == h[1]));
    return h;
}
```

PolygonTangents.h

Stosowanie: Wielokąt musi być CCW i  $n \geq 3$ . Zwraca najbliższe punkty styczne różne od  $a$ .

Czas:  $\mathcal{O}(\log n)$

```
pair<pt, pt> tangents(const vector<pt>& p, pt a) {
    int n = ssize(p);
    pt t[2];
    for (int it = 0; it < 2; it++) {
        auto dir = [&](int i) {
            pt u = p[i] - a;
            pt v = p[i < n - 1 ? i + 1 : 0] - a;
            ll c = cross(u, v);
            if (c != 0) return c < 0;
            if (dot(u, v) > 0) return norm(u) > norm(v);
            return true;
        };
        auto dirx = [&](int i) { return dir(i) ^ it; };
        if (dirx(0) == 1 && dirx(n - 1) == 0) {
            t[it] = p[0];
            continue;
        }
        int s[2] = {0, n - 1};
        while (s[1] - s[0] > 2) {
            int mid = (s[0] + s[1]) / 2;
            int x = dirx(mid);
            if (dirx(s[x ^ 1]) == (x ^ 1)) {
                s[x] = mid;
            } else {
                ((cross(p[mid] - a, p[s[1]] - a) < 0) ^ it
                 ? s[x]
                 : s[x ^ 1]) = mid;
            }
        }
        t[it] = dirx(s[0] + 1) == 0 ? p[s[0] + 2] : p[s[0] + 1];
    }
    return {t[0], t[1]};
}
```

Inne (8)

GCC.h

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
```