

1	Contest
2	Grafy
3	Matma
4	Teksty
5	Geometria

Contest (1)

sol.cpp
<pre>#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); }</pre>
.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
```

1	test.sh
1	<pre>#!/bin/bash for ((i=1;i>0;i++)) do echo "\$i" echo "\$i" ./gen > int diff -w <./sol <int) <./slow <int) break done</pre>
2	hash.sh
2	<pre>#!/bin/bash cpp -dD -P -fpreprocessed tr -d '[:space:]' md5sum cut -c-6</pre>
	.bashrc
2	<pre>alias rm='trash' alias mv='mv -i' alias cp='cp -i'</pre>

Grafy (2)

2.1 Przepływy

Dinic.h
Opis: Dinic ze skalowaniem. Należy ustawić zakres it w flow zgodnie z U . Czas: $\mathcal{O}(nm \log U)$
<pre>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; } } };</pre>

<pre> } while (lvl[t]); return f; } };</pre>
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};
}
};
```

2.2 Grafy skierowane

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];
    }
};
```

Matma (3)

3.1 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 >= (1ll)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;
}
```

3.2 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);
}
```

Teksty (4)

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 - 1]);
        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;
}
```

Geometria (5)

5.1 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);
}
```

5.2 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])};
}
```