

## makra.cpp

```

#ifndef LOCAL
#pragma GCC optimize ("O3")
#endif
#include <bits/stdc++.h>
using namespace std;
//st, nd, mp, pb, eb, endl, siz, all
#define sim template < class c
#define ris return * this
#define mor > muu & operator << (
#define R22(r) sim>typename enable_if<1 r sizeof dud<c>(0),muu&>::type operator<<(c g){
sim > struct rge { c b, e; };
sim > rge<c> range(c b, c e) { return {b, e}; }
sim > auto dud(c* r) -> decltype(cerr << *r);
sim > char dud(...);
struct muu {
#ifdef LOCAL
stringstream a;
~muu() { cerr << a.str() << endl; }
R22(<) a << boolalpha << g; ris; }
R22(==) ris << range(begin(g), end(g)); }
sim, class b mor pair < b, c > r) { ris << "(" << r.st << ", " << r.nd << ")"; }
sim mor rge<c> u) {
a << "[";
for (c i = u.b; i != u.e; ++i) *this << ", " + 2 * (i == u.b) << *i;
ris << "]" ;
}
template <class...c mor tuple<c...> x) {
int q = 0;
a << "(" ;
apply([&](c...y){
(( *this << ", " + 2 * !q++ << y), ...);
}, x);
ris << ")";
}
#define qel(t) sim, class d, class...e mor t<c,d,e...> x){ris << *(d*)&x;}
qel(stack) qel(queue) qel(priority_queue)
#else
sim mor const c& { ris; }
#endif
};
#define imie(r...) "[" #r ": " << (r) << "]"
#define range(b, e) "[" #b ", " #e "): " << range(b, e) << "]"
#define debug muu() << __FUNCTION__ << "#" << __LINE__ << ": "
//using ll,ld,pii,vi,vpii,ull,pdd,
sim> void mini(c &a, const c &b) {if (a > b) a = b;}
sim> void maxi(c &a, const c &b) {if (a < b) a = b;}
int32_t main() {ios_base::sync_with_stdio(0); cin.tie(0);}

```

## geany-config

```

Interface:
Message window: right
Keybindings:
Build: Compile: F8, Build: F9
Focus: Switch to Editor: F1, Switch to VTE: F2
Set Build Commands:
compile: g++ -Wall -Wshadow -o "%e" "%f" -g -O3 -std=c++1z
build: g++ -Wall -Wshadow -o "%e" "%f" -fsanitize=undefined -fsanitize=address -DLOCAL
-D GLIBCXX_DEBUG -g -std=c++1z

```

## Rzeczy\_Na\_Dzien\_Probny.cpp

```

// 1. Sprawdzić int128.
// Oczekiwany output: 61231558446921906466935685523974676212.
int Main() {
__int128_t x = (1llu << 62);
x *= x;
while (x) {

```

```

printf("%d", (int) (x % 10));
x /= 10;
}
printf("\n");
return 0;
}
// 2. Czy rand() działa tak samo na sprawdzacze.
int Main() {
srand(0x12345);
const int a = rand();
printf("a = %d\n", a);
printf("RAND_MAX w zbiorze: = [0, %d]\n", (int) RAND_MAX);
assert(a == 1206605802 /* Tu wkleić wartość po uruchomieniu u siebie. */);
return 0;
}
// 3. Sprawdzanie czasu.
int Main() {
while (clock() <= 0.690 * CLOCKS_PER_SEC);
// Sprawdzić czy uruchomienie zajęło dokładnie 690ms. Jeśli nie da się
// sprawdzić dokładnego czasu wykonania, to można zbinsearchować time limit.
return 0;
}
// 4. Sprawdzić czy time() zwraca różne wartości pomiędzy uruchomieniami.
int Main() {
switch (time(NULL) % 3) {
case 0:
while (true); // Daj TLE.
case 1:
assert(false); // Daj RE.
case 2:
printf("Daj WA.\n");
}
return 0;
}
// 5. Przetestować czy kompresja działa.
// (...) <- Przepisać kompresję.
int Main() {
// Ustawić false do wygenerowania kodu, a true do zweryfikowania kompresji.
bool czy_uruchomienie = false;
Compress compress;
for (int i = 1; i <= 100; i++) {
const uint64_t result = compress.Data(62, i * i);
if (czy_uruchomienie) {
assert(result == i * i);
}
}
}
// 6. Sprawdzić float128.
// Oczekiwany output: 47.610000000.
int Main() {
__float128 x = 6.9;
printf("%.9lf\n", (double)(x*x));
return 0;
}
// 7. Sprawdzić ordered_seta.
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
template <typename T>
using ordered_set =
tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
ordered_set<int> s;
int Main() {

```

```
s.insert(1);
s.insert(2);
cout << s.order_of_key(1) << endl; // Out: 0.
cout << *s.find_by_order(1) << endl; // Out: 2.
}
```

**Data Structures/OrderedSet.cpp**

```
#include<ext/pb_ds/assoc_container.hpp> // ordered set
#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>;
ordered_set<int> s; s.insert(1); s.insert(2);
s.order_of_key(1); // Out: 0.
*s.find_by_order(1); // Out: 2.
```

**Data Structures/Treap.cpp**

```
struct node {
    node *L, *R;
    int ind, prior, sub, lazy; // sub opcjonalne
    node(int ind) : L(0), R(0), ind(ind), prior(rand()), sub(1), lazy(0) {}
};
void rev(node* v) { // przykładowy update, odwraca kolejność w poddrzewie
    v->lazy ^= 1;
    swap(v->L, v->R);
}
void push(node* v) { // opcjonalne
    if (v->lazy) {
        if (v->L) rev(v->L);
        if (v->R) rev(v->R);
        v->lazy = 0;
    }
}
node* attach(node* v, node* l, node* r) {
    v->L = l; // jeśli chcemy trzymać ojca to update w tej funkcji
    v->R = r;
    v->sub = 1 + (l ? l->sub : 0) + (r ? r->sub : 0); // opcjonalne
    return v;
}
node* merge(node* v, node* u) {
    if (!u) return v;
    if (!v) return u;
    push(v);
    push(u);
    if (v->prior > u->prior) return attach(v, v->L, merge(v->R, u));
    else return attach(u, merge(v, u->L), u->R);
}
pair<node*, node*> split_size(node* v, int k) { // (prefiks rozmiaru k, reszta)
    if (!v) return mp(v, v);
    int lewo = v->L ? v->L->sub : 0;
    push(v);
    if (lewo >= k) {
        auto s = split_size(v->L, k);
        return mp(s.st, attach(v, s.nd, v->R));
    } else {
        auto s = split_size(v->R, k - lewo - 1);
        return mp(attach(v, v->L, s.st), s.nd);
    }
}
pair<node*, node*> split_lex(node* v, int k) { // (ind <= k, reszta)
    if (!v) return mp(v, v);
    if (k < v->ind) {
        auto s = split_lex(v->L, k);
        return mp(s.st, attach(v, s.nd, v->R));
    } else {
        auto s = split_lex(v->R, k);
        return mp(attach(v, v->L, s.st), s.nd);
    }
}
int find_pos(node* v, int val) { // -1 jeśli nie ma
    if (!v) return -1;
```

```
int lewo = v->L ? v->L->sub : 0;
if (v->ind == val) return 1 + lewo;
if (val < v->ind) return find_pos(v->L, val);
else return 1 + lewo + find_pos(v->R, val);
}
```

**Flows And Matchings/Dinic.cpp**

```
using T = long long;
struct Flow {
    struct E {
        int dest;
        T orig, *lim, *rev;
    };
    int zr, uj, n = 0;
    vector<unique_ptr<T>> ts;
    vector<vector<E>> graf;
    vector<int> ptr, odl;
    void vert(int v) {
        n = max(n, v + 1);
        graf.resize(n);
        ptr.resize(n);
        odl.resize(n);
    }
    bool iszero(T v) {
        return !v; // Zmienić dla doubli.
    }
    void bfs() {
        fill(odl.begin(), odl.end(), 0);
        vector<int> kol = {zr};
        odl[zr] = 1;
        for (int i = 0; i < (int) kol.size(); i++) {
            for (E& e : graf[kol[i]]) {
                if (!odl[e.dest] and !iszero(*e.lim)) {
                    odl[e.dest] = odl[kol[i]] + 1;
                    kol.push_back(e.dest);
                }
            }
        }
    }
    T dfs(int v, T lim) {
        if (v == uj) return lim;
        T ret = 0, wez;
        for (int& i = ptr[v]; i < (int) graf[v].size(); i++) {
            E& e = graf[v][i];
            if (odl[e.dest] == odl[v] + 1 and !iszero(*e.lim) and
                !iszero(wez = dfs(e.dest, min(*e.lim, lim)))) {
                ret += wez;
                *e.lim -= wez;
                *e.rev += wez;
                lim -= wez;
                if (iszero(lim)) break;
            }
        }
        return ret;
    }
    void add_edge(int u, int v, T lim, bool bi = false /* bidirectional? */) {
        vert(max(u, v));
        T *a = new T(lim), *b = new T(lim * bi);
        ts.emplace_back(a);
        ts.emplace_back(b);
        graf[u].push_back(E{v, lim, a, b});
        graf[v].push_back(E{u, lim * bi, b, a});
    }
    T dinic(int zr_, int uj_) {
        zr = zr_; uj = uj_;
        vert(max(zr, uj));
```

```

T ret = 0;
while (true) {
    bfs();
    fill(ptr.begin(), ptr.end(), 0);
    const T sta = dfs(zr, numeric_limits<T>::max()); // Dla dubli można dać
    if (iszero(sta)) break; // infinity() zamiast
    ret += sta; // max().
}
return ret;
}
vector<int> cut() {
    vector<int> ret;
    bfs();
    for (int i = 0; i < n; i++)
        if (odl[i])
            ret.push_back(i);
    return ret;
}
map<pair<int, int>, T> get_flow() { // Tam gdzie płynie 0 może nie być
    map<pair<int, int>, T> ret; // krawędzi.
    for (int i = 0; i < n; i++)
        for (E& e : graf[i])
            if (*e.lim < e.orig)
                ret[make_pair(i, e.dest)] += e.orig - *e.lim;
    for (auto& i : ret) {
        const pair<int, int> rev{i.first.second, i.first.first};
        const T x = min(i.second, ret[rev]);
        i.second -= x;
        ret[rev] -= x;
    }
    return ret;
}
};

```

#### Flows And Matchings/Gomory Hu.cpp

```

#define int ll //jeśli long longi potrzebne
struct GomoryHu {
    vector<vector< pair<int,int> >> graph, tree;
    vector<vector<int>> nodes;
    vector<bool> visited; //wymaga naszego dinica
    vector<int> groupId, contrId; //numeracja od zera
    int wnode, n;
    GomoryHu(int N) : graph(N), visited(N), groupId(N), contrId(N), tree(N), n(N) {}
    void addEdge(int u, int v, int cap) {
        graph[u].emplace_back(v, cap);
        graph[v].emplace_back(u, cap);
    }
    void dfs(int v, int type) {
        visited[v] = true; contrId[v] = type;
        for (auto P : tree[v]) { if (!visited[P.first]) { dfs(P.first, type); } }
    }
    vector<pair<pair<int,int>,int>> run() {
        vector<int> allNodes(n);
        iota(allNodes.begin(), allNodes.end(), 0);
        nodes = vector<vector<int>>(allNodes);
        tree = vector<vector<pair<int,int>>>(n);
        fill(groupId.begin(), groupId.end(), 0);
        for (int step = 1; step < n; step++) {
            Flow flow;
            for (int i = 0; i < (int)nodes.size(); i++) {
                if ((int)nodes[i].size() > 1) { wnode = i; break; }
            }
            fill(visited.begin(), visited.end(), false);
            visited[wnode] = true;
            for (auto P : tree[wnode]) { dfs(P.first, nodes[P.first][0]); }
            for (int v = 0; v < n; v++) {

```

```

int a = groupId[v] == wnode ? v : contrId[groupId[v]];
for (auto& P : graph[v]) {
    int b = groupId[P.first] == wnode ? P.first : contrId[groupId[P.first]];
    if (a != b) { flow.add_edge(a, b, P.second); }
}
}
int a = nodes[wnode][0], b = nodes[wnode][1], f = flow.dinic(a, b);
auto pom = flow.cut();
vector<bool> cut(n, false);
for (int i : pom)
    cut[i]=1;
for (int v = 0; v < step; v++) {
    if (v == wnode) { continue; }
    for (auto& P : tree[v]) {
        if (P.first == wnode && !cut[contrId[v]]) { P.first = step; }
    }
}
vector<pair<int,int>> PA, PB;
for (auto& P : tree[wnode]) { (cut[contrId[P.first]] ? PA : PB).push_back(P); }
tree[wnode] = PA; tree[step] = PB;
tree[wnode].emplace_back(step, f);
tree[step].emplace_back(wnode, f);
vector<int> A, B;
for (int v : nodes[wnode]) {
    (cut[v] ? A : B).push_back(v);
    if (!cut[v]) { groupId[v] = step; }
}
nodes[wnode] = A;
nodes.push_back(B);
}
vector<pair<pair<int,int>,int>> res;
for (int i = 0; i < n; i++)
    for (auto P : tree[i])
        if (nodes[i][0]<nodes[P.first][0])
            res.push_back({nodes[i][0], nodes[P.first][0], P.second});
return res;
}
};

```

#undef int

#### Flows And Matchings/Highest Label Push Relabel.cpp

```

class LinkedList {
public:
    LinkedList(int N) : N(N), next(N) { clear(); }
    void clear() { head.assign(N, -1); }
    int front(int h) { return head[h]; }
    void pop(int h) { head[h] = next[head[h]]; }
    void push(int h, int u) { next[u] = head[h], head[h] = u; }
private:
    int N;
    vector<int> next, head;
};

class DoublyLinkedList {
private:
    struct Node { int prev, next; };
public:
    DoublyLinkedList(int N) : N(N), nodes(N) { clear(); }
    void clear() { head.assign(N, -1); }
    void erase(int h, int u) {
        int pv = nodes[u].prev, nx = nodes[u].next;
        if (nx >= 0) nodes[nx].prev = pv;
        if (pv >= 0) nodes[pv].next = nx;
        else head[h] = nx;
    }
    void insert(int h, int u) {
        nodes[u] = { -1, head[h] };

```

```

    if (head[h] >= 0) nodes[head[h]].prev = u;
    head[h] = u;
}
template <typename Func>
void erase_all(int first, int last, Func f) {
    for (int i = first; i <= last; ++i) {
        for (int h = head[i]; h >= 0; h = nodes[h].next) f(h);
        head[i] = -1;
    }
}
private:
    int N;
    vector<int> head;
    vector<Node> nodes;
};
template <
    typename CapType, typename TotalCapType,
    bool UseGlobal = true, bool UseGap = true
>
class HighestLabelPushRelabel {
private:
    TotalCapType inf = pow(10., sizeof(TotalCapType) / 4 * 9);
    struct Edge { int to, rev; CapType cap; };
public:
    HighestLabelPushRelabel(int N) : N(N), E(0), G(N), que(N), list(N), dlist(N) {}
    TotalCapType maximum_flow(int s, int t) {
        if (s == t) return 0;
        highest_active = 0; // highest label (active)
        highest = 0; // highest label (active and inactive)
        height.assign(N, 0); height[s] = N;
        for (int i = 0; i < N; ++i) if (i != s) dlist.insert(height[i], i);
        count.assign(N, 0); count[0] = N - 1;
        excess.assign(N, 0); excess[s] = inf; excess[t] = -inf;
        for (auto& e : G[s]) push(s, e);
        global_relabel(t);
        for (int u = -1, rest = N; highest_active >= 0; ) {
            if ((u = list.front(highest_active)) < 0) { --highest_active; continue; }
            list.pop(highest_active);
            discharge(u);
            if (--rest == 0) rest = N, global_relabel(t);
        }
        return excess[t] + inf;
    }
    inline void add_directed_edge(int u, int v, CapType cap) {
        E++;
        G[u].push_back({v, (int) G[v].size(), cap});
        G[v].push_back({u, (int) G[u].size() - 1, 0});
    }
    inline void add_undirected_edge(int u, int v, CapType cap) {
        G[u].push_back({v, (int) G[v].size(), cap});
        G[v].push_back({u, (int) G[u].size() - 1, cap});
    }
private:
    void global_relabel(int t) {
        if (!UseGlobal) return;
        height.assign(N, N); height[t] = 0;
        count.assign(N, 0);
        int qh = 0, qt = 0;
        for (que[qt++] = t; qh < qt; ) {
            int u = que[qh++], h = height[u] + 1;
            for (auto& e : G[u]) if (height[e.to] == N && G[e.to][e.rev].cap > 0) {
                count[height[e.to] = h] += 1;
                que[qt++] = e.to;
            }
        }
    }
}

```

```

list.clear(); dlist.clear();
for (int u = 0; u < N; ++u) if (height[u] < N) {
    dlist.insert(height[u], u);
    if (excess[u] > 0) list.push(height[u], u);
}
highest = highest_active = height[que[qt - 1]];
}
void push(int u, Edge& e) {
    int v = e.to;
    CapType df = min(excess[u], TotalCapType(e.cap));
    e.cap -= df, G[v][e.rev].cap += df;
    excess[u] -= df, excess[v] += df;
    if (0 < excess[v] && excess[v] <= df) list.push(height[v], v);
}
void discharge(int u) {
    int nh = N;
    for (auto& e : G[u]) if (e.cap > 0) {
        if (height[u] == height[e.to] + 1) {
            push(u, e);
            if (excess[u] == 0) return;
        } else nh = min(nh, height[e.to] + 1);
    }
    int h = height[u];
    if (UseGap && count[h] == 1) {
        auto f = [&](int u) { count[height[u]]--, height[u] = N; };
        dlist.erase_all(h, highest, f);
        highest = h - 1;
    } else {
        count[h]--; dlist.erase(h, u);
        height[u] = nh;
        if (nh == N) return;
        count[nh]++; dlist.insert(nh, u);
        highest = max(highest, highest_active = nh);
        list.push(nh, u);
    }
}
int N, E, highest_active, highest;
vector< vector<Edge> > G;
vector<int> height, count, que;
vector<TotalCapType> excess;
LinkedList list;
DoublyLinkedList dlist;
};
using HIPR GP = HighestLabelPushRelabel<int, int, true, true>;

```

## Flows And Matchings/Hungarian.cpp

```

//a[1..n][1..m] - wagi, n<=m, 0(n^2*m), znajduje minimalny koszt (maszynowy: *-1)
//u[0..n], v[0..m] - funkcja potencjalu, p[0..m], ans[0..n] - skojarzenie
struct hungarian { // cost = -v[0]
    int n, m;
    vector<vi> a;
    vi u, v, p, way, ans;
    hungarian(int _n, int _m) : n(_n), m(_m) {
        #define re(x, y) (x).resize(y + 1)
        re(a, n); re(u, n); re(v, m); re(p, m); re(way, m); re(ans, n);
        for (auto &x : a) re(x, m);
    }
    void addcost(int v1, int v2, int c) {
        a[v1][v2] = c;
    }
    void solve() {
        for(int i = 1; i <= n; ++i) {
            p[0] = i;
            int j0 = 0;
            vi minv(m + 1, INT_MAX); // uwaga, moze byc potrzebny LL!
            vector<char> used(m + 1, false);

```

```

do {
    used[j0] = true;
    int i0 = p[j0], delta = INT_MAX, j1 = 0;
    for (int j = 1; j <= m; ++j) {
        if (!used[j]) {
            int cur = a[i0][j] - u[i0] - v[j];
            if (cur < minv[j]) {
                minv[j] = cur;
                way[j] = j0;
            }
            if (minv[j] < delta) {
                delta = minv[j];
                j1 = j;
            }
        }
    }
    for (int j = 0; j <= m; ++j) {
        if (used[j]) {
            u[p[j]] += delta;
            v[j] -= delta;
        }
        else {
            minv[j] -= delta;
        }
    }
    j0 = j1;
} while (p[j0] != 0);
do {
    int j1 = way[j0];
    p[j0] = p[j1];
    j0 = j1;
} while (j0);
}
for (int j = 1; j <= m; ++j)
    ans[p[j]] = j; // odzyskiwanie wyniku
};

```

## Flows\_And\_Matchings/Matching.cpp

```

struct Matching {
    int n, tim = 0, top = 0;
    vector<int> mat, fa, s, q, pre, vis, head;
    vector<pair<int, int>> e;
    Matching(int N) : n(N + 1), mat(n, -1), fa(n), s(n), q(n), pre(n), vis(n), head(n, -1) {}
    void edge_impl(int x, int y) {e.emplace_back(y, head[x]);head[x] = e.size() - 1;}
    void add_edge(int x, int y) {edge_impl(x, y), edge_impl(y, x);}
    int find(int x) {return x == fa[x] ? x : fa[x] = find(fa[x]);}
    int lca(int x, int y) {
        for (++tim, x = find(x), y = find(y); ; swap(x, y)) {
            if (~x) {
                if (vis[x] == tim) return x;
                vis[x] = tim;
                x = ~mat[x] ? find(pre[mat[x]]) : -1;
            }
        }
    }
    void blossom(int x, int y, int l) {
        while (find(x) != l) {
            pre[x] = y;
            if (s[mat[x]] == 1) s[q[top++] = mat[x]] = 0;
            if (fa[x] == x) fa[x] = l;
            if (fa[mat[x]] == mat[x]) fa[mat[x]] = l;
            y = mat[x];
            x = pre[y];
        }
    }
    bool match(int x) {
        iota(fa.begin(), fa.end(), 0);
    }
};

```

```

fill(s.begin(), s.end(), -1);
s[q[0]] = x;
top = 1;
for (int i = 0; i < top; ++i) {
    for (int t = head[q[i]]; ~t; t = e[t].second) {
        int y = e[t].first;
        if (s[y] == -1) {
            pre[y] = q[i];
            s[y] = 1;
            if (mat[y] == -1) {
                for (int u = y, v = q[i], lst; ~v; u = lst, v = ~u ? pre[u] : -1)
                    lst = mat[v], mat[v] = u, mat[u] = v;
                return true;
            }
            s[q[top++] = mat[y]] = 0;
        }
        else if (!s[y] && find(y) != find(q[i])) {
            int l = lca(y, q[i]);
            blossom(y, q[i], l);
            blossom(q[i], y, l);
        }
    }
}
return false;
}
int run() {
    int size = 0;
    for (int i = 0; i < n; ++i)
        if (mat[i] == -1 && match(i))
            size++;
    return size;
}
};

```

## Flows\_And\_Matchings/Min\_Cost.cpp

```

struct MinCost {
    struct kra {
        int cel, *prze1, *prze2;
        ll koszt;
    };
    int n=0, zr, uj;
    const ll inf=1e9;
    vector<vector<kra>> graf;
    vector<int> bylo, aktu;
    vector<ll> odl, pamodl;
    void vert(int v) {
        if (v>n) {
            n=v;
            graf.resize(n);
            bylo.resize(n);
            aktu.resize(n);
            odl.resize(n);
            pamodl.resize(n);
        }
    }
    void add_edge(int v, int u, int prze, ll koszt) {
        vert(v+1); vert(u+1);
        kra ret1{u, new int(prze), new int(0), koszt};
        kra ret2{v, ret1.prze2, ret1.prze1, -koszt};
        graf[v].push_back(ret1);
        graf[u].push_back(ret2);
    }
    void spfa() {
        for (int i=0; i<n; i++) {
            aktu[i]=1;
            pamodl[i]=inf;
        }
    }
};

```

```

}
aktu[zr]=pamodl[zr]=0;
queue <int> kol;
kol.push(zr);
while(!kol.empty()) {
    int v=kol.front();
    kol.pop();
    if (aktu[v])
        continue;
    aktu[v]=1;
    for (kra i : graf[v]) {
        if (*i.przel && pamodl[v]+i.koszt<pamodl[i.cel]) {
            pamodl[i.cel]=pamodl[v]+i.koszt;
            aktu[i.cel]=0;
            kol.push(i.cel);
        }
    }
}
}
void dij() {
    for (int i=0; i<n; i++)
        odl[i]=inf;
    priority_queue < pair <ll,int> > dijks;
    dijks.push({0, zr});
    while(!dijks.empty()) {
        ll dis=-dijks.top().first;
        int v=dijks.top().second;
        dijks.pop();
        if (odl[v]!=inf)
            continue;
        odl[v]=pamodl[v]+dis;
        for (auto j : graf[v])
            if ((*j.przel && odl[j.cel]==inf)
                dijks.push({-(dis+pamodl[v]-pamodl[j.cel]+j.koszt), j.cel}));
    }
}
int dfs(int v) {
    if (v==uj)
        return 1;
    bylo[v]=1;
    for (int i=0; i<(int)graf[v].size(); i++) {
        if (!bylo[graf[v][i].cel] && (*graf[v][i].przel) &&
            odl[v]+graf[v][i].koszt==odl[graf[v][i].cel] && dfs(graf[v][i].cel)) {
            (*graf[v][i].przel)--;
            (*graf[v][i].prze2)++;
            return 1;
        }
    }
    return 0;
}
pair <int,ll> flow(int zr,zr, int ujuj) {
    zr=zr,zr; uj=uj,uj;
    vert(zr+1); vert(uj+1);
    // spfa();
    pair <int,ll> ret{0, 0};
    while(1) {
        spfa();
        odl = pamodl;
        // dij();
        for (int i=0; i<n; i++)
            bylo[i]=0;
        if (!dfs(zr))
            break;
        ret.first++;
        ret.second+=odl[uj];
    }
}

```

```

}
return ret;
};
Flows_And_Matchings/Turbo_Matching.cpp
struct matching {
    int n;
    vector<vi> V;
    vector<bool> odw;
    vi para, strona;
    matching(int _n) : n(_n) { // zakladam numeracje od 1, nie moze byc 0
        V.resize(n + 1);
        odw.resize(n + 1);
        para.resize(n + 1);
    }
    void addedge(int a, int b) { // zakladam ze a jest z lewej, b z prawej
        V[a].pb(b);
        strona.pb(a);
    }
    bool skojarz(int x) { // x nalezy do strona
        odw[x] = 1;
        for (auto v : V[x]) {
            if (!para[v] || (!odw[para[v]] && skojarz(para[v]))) {
                para[v] = x;
                para[x] = v;
                return 1;
            }
        }
        return 0;
    }
    vpil go() {
        sort(all(strona));
        strona.resize(unique(all(strona)) - strona.begin());
        for (int i = 1; i <= n; ++i)
            shuffle(all(V[i]), default_random_engine(2137));
        sort(all(strona), [&](int a, int b){return siz(V[a]) < siz(V[b]);});
        bool ok = 0;
        do {
            ok = 0;
            for (auto i : strona) odw[i] = 0;
            for (auto i : strona)
                if (!para[i] && skojarz(i))
                    ok = 1;
        } while (ok);
        vpil res;
        for (auto i : strona)
            if (para[i])
                res.pb(mp(i, para[i]));
        return res;
    }
};
Flows_And_Matchings/Weighted_Matching.cpp
static const int INF = INT_MAX;
static const int N = 1017;
struct edge {
    int u, v, w;
    edge() {} edge(int ui, int vi, int wi) : u(ui), v(vi), w(wi) {}
};
int n, n_x;
edge g[N * 2][N * 2];
int lab[N * 2], match[N * 2], slack[N * 2], st[N * 2], pa[N * 2];
int flo_from[N * 2][N * 2], S[N * 2], vis[N * 2];
vector<int> flo[N * 2];
queue<int> q;
int e_delta(const edge &e) { return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2; }
void update_slack(int u, int x) {
    if (!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x])) slack[x] = u;
}

```

```

}
void set_slack(int x) {
    slack[x] = 0;
    for (int u = 1; u <= n; u++)
        if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
            update_slack(u, x);
}
void q_push(int x) {
    if (x <= n) q.push(x);
    else for (size_t i = 0; i < flo[x].size(); i++) q_push(flo[x][i]);
}
void set_st(int x, int b) {
    st[x] = b;
    if (x > n) for (size_t i = 0; i < flo[x].size(); i++) set_st(flo[x][i], b);
}
int get_pr(int b, int xr) {
    int pr = find(flo[b].begin(), flo[b].end(), xr) - flo[b].begin();
    if (pr % 2 == 1) {
        reverse(flo[b].begin() + 1, flo[b].end());
        return (int)flo[b].size() - pr;
    } else return pr;
}
void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n) return;
    edge e = g[u][v];
    int xr = flo_from[u][e.u], pr = get_pr(u, xr);
    for (int i = 0; i < pr; i++) set_match(flo[u][i], flo[u][i ^ 1]);
    set_match(xr, v);
    rotate(flo[u].begin(), flo[u].begin() + pr, flo[u].end());
}
void augment(int u, int v) {
    for (;;) {
        int xnv = st[match[u]];
        set_match(u, v);
        if (!xnv) return;
        set_match(xnv, st[pa[xnv]]);
        u = st[pa[xnv]], v = xnv;
    }
}
int get_lca(int u, int v) {
    static int t = 0;
    for (++t; u || v; swap(u, v)) {
        if (u == 0) continue;
        if (vis[u] == t) return u;
        vis[u] = t; u = st[match[u]];
        if (u) u = st[pa[u]];
    }
    return 0;
}
void add_blossom(int u, int lca, int v) {
    int b = n + 1; while (b <= n_x && st[b]) ++b;
    if (b > n_x) ++n_x;
    lab[b] = 0, S[b] = 0;
    match[b] = match[lca];
    flo[b].clear(); flo[b].push_back(lca);
    for (int x = u, y; x != lca; x = st[pa[y]])
        flo[b].push_back(x), flo[b].push_back(y = st[match[x]]), q_push(y);
    reverse(flo[b].begin() + 1, flo[b].end());
    for (int x = v, y; x != lca; x = st[pa[y]])
        flo[b].push_back(x), flo[b].push_back(y = st[match[x]]), q_push(y);
    set_st(b, b);
    for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
    for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
    for (size_t i = 0; i < flo[b].size(); i++) {

```

```

        int xs = flo[b][i];
        for (int x = 1; x <= n_x; ++x)
            if (g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][x]))
                g[b][x] = g[xs][x], g[x][b] = g[x][xs];
        for (int x = 1; x <= n; ++x) if (flo_from[xs][x]) flo_from[b][x] = xs;
    }
    set_slack(b);
}
void expand_blossom(int b) {
    for (size_t i = 0; i < flo[b].size(); ++i)
        set_st(flo[b][i], flo[b][i]);
    int xr = flo_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
    for (int i = 0; i < pr; i += 2) {
        int xs = flo[b][i], xns = flo[b][i + 1];
        pa[xs] = g[xns][xs].u;
        S[xs] = 1, S[xns] = 0;
        slack[xs] = 0, set_slack(xns);
        q_push(xns);
    }
    S[xr] = 1, pa[xr] = pa[b];
    for (size_t i = pr + 1; i < flo[b].size(); i++) {
        int xs = flo[b][i];
        S[xs] = -1, set_slack(xs);
    }
    st[b] = 0;
}
bool on_found_edge(const edge &e) {
    int u = st[e.u], v = st[e.v];
    if (S[v] == -1) {
        pa[v] = e.u, S[v] = 1; int nu = st[match[v]];
        slack[v] = slack[nu] = 0; S[nu] = 0, q_push(nu);
    } else if (S[v] == 0) {
        int lca = get_lca(u, v);
        if (!lca) return augment(u, v), augment(v, u), true;
        else add_blossom(u, lca, v);
    }
    return false;
}
bool matching() {
    memset(S + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) * n_x);
    q = queue<int>();
    for (int x = 1; x <= n_x; x++) {
        if (st[x] == x && !match[x]) pa[x] = 0, S[x] = 0, q_push(x);
    }
    if (q.empty()) return false;
    for (;;) {
        while (q.size()) {
            int u = q.front(); q.pop();
            if (S[st[u]] == 1) continue;
            for (int v = 1; v <= n; v++) {
                if (g[u][v].w > 0 && st[u] != st[v]) {
                    if (e_delta(g[u][v]) == 0) {
                        if (on_found_edge(g[u][v])) return true;
                    } else update_slack(u, st[v]);
                }
            }
        }
        int d = INF;
        for (int b = n + 1; b <= n_x; b++) {
            if (st[b] == b && S[b] == 1) d = min(d, lab[b] / 2);
        }
        for (int x = 1; x <= n_x; x++) {
            if (st[x] == x && slack[x]) {
                if (S[x] == -1) d = min(d, e_delta(g[slack[x]][x]));
            }

```



```

    else if (S[x] == 0) d = min(d, e_delta(g[slack[x]][x]) / 2);
}
for (int u = 1; u <= n; u++) {
    if (S[st[u]] == 0) {
        if (lab[u] <= d) return 0;
        lab[u] -= d;
    } else if (S[st[u]] == 1) lab[u] += d;
}
for (int b = n + 1; b <= n_x; b++) {
    if (st[b] == b) {
        if (S[st[b]] == 0) lab[b] += d * 2;
        else if (S[st[b]] == 1) lab[b] -= d * 2;
    }
}
q = queue<int>();
for (int x = 1; x <= n_x; x++)
    if (st[x] == x && slack[x] && st[slack[x]] != x && e_delta(g[slack[x]][x]) == 0)
        if (on_found_edge(g[slack[x]][x])) return true;
for (int b = n + 1; b <= n_x; b++)
    if (st[b] == b && S[b] == 1 && lab[b] == 0) expand_blossom(b);
}
return false;
}
pair<long long, int> solve() {
    memset(match + 1, 0, sizeof(int) * n);
    n_x = n;
    int n_matches = 0;
    long long tot_weight = 0;
    for (int u = 0; u <= n; u++) st[u] = u, flo[u].clear();
    int w_max = 0;
    for (int u = 1; u <= n; u++)
        for (int v = 1; v <= n; v++) {
            flo_from[u][v] = (u == v ? u : 0);
            w_max = max(w_max, g[u][v].w);
        }
    for (int u = 1; u <= n; u++) lab[u] = w_max;
    while (matching()) ++n_matches;
    for (int u = 1; u <= n; u++) {
        if (match[u] && match[u] < u) {
            tot_weight += g[u][match[u]].w;
        }
    }
    return make_pair(tot_weight, n_matches);
}
void add_edge(int u, int v, int w) {
    g[u][v].w = g[v][u].w = max(g[u][v].w, w);
}
void init(int _n) {
    n = _n;
    for (int u = 1; u <= n; u++)
        for (int v = 1; v <= n; v++) {
            g[u][v] = edge(u, v, 0);
        }
}

```

#### Geometry/Dewolaj.cpp

```

typedef long long T;
struct P {
    T x, y;
    int id;
    P operator-(P b) { return P{x - b.x, y - b.y}; }
    T cross(P b) { return x * b.y - y * b.x; }
    T cross(P b, P c) const { return (b - *this).cross(c - *this); }
    T dot(P b) { return x * b.x + y * b.y; }
    bool inTriangle(const P &a, const P &b, const P &c) const {

```

```

#define tmp(a, b) (cross(a, b) > 0)
return tmp(a, b) == tmp(b, c) && tmp(b, c) == tmp(c, a);
#undef tmp
};
int cmpCircle(P a, P b, P c, P d) {
    P v1 = b - a, v2 = d - a, v3 = b - c, v4 = d - c;
    ld tmp=(ld)abs(v1.cross(v2))*v3.dot(v4)+(ld)v1.dot(v2)*abs(v3.cross(v4));
    if (abs(tmp) < 1e-8) return 0;if (tmp == 0) return 0;
    if (tmp > 0) return 1;return -1;
}
struct pair_hash {
    template <class T1, class T2>
    std::size_t operator()(const std::pair<T1, T2> &p) const {
        return p.first * 10000 + p.second;
    }
};
unordered_map<pair<int, int>, pair<int, int>, pair_hash> mt;
set<pair<int, int>> edges;
vector<vector<int>> triangles;
void rec(int a, int c, const vector<P> &points);
void trim(int a, int b) {
    assert(a < b);
    auto it = mt.find({a, b});
    if ((it != mt.end()) && (it->second == make_pair(-1, -1))) mt.erase(it);
}
void change(int a, int b, int from, int to) {
    if (a > b) swap(a, b);
    if (!mt.count({a, b})) mt[{a, b}] = {-1, -1};
    for (int *x : vector<int>*>{{mt[{a, b}].first, &mt[{a, b}].second}}
        if (*x == from) {
            *x = to;
            trim(a, b);
            return;
        }
    }
    assert(false);
}
void rec(int a, int c, const vector<P> &points) {
    if (a > c) swap(a, c);
    if (!mt.count({a, c})) return;
    int b = mt[{a, c}].first;
    int d = mt[{a, c}].second;
    if (b > d) swap(b, d);
    if (b == -1 || d == -1) return;
    for (int rep = 0; rep < 2; ++rep) {
        if (points[a].inTriangle(points[b], points[c], points[d])) return;
        swap(a, c);
    }
    if (cmpCircle(points[a], points[b], points[c], points[d]) != 1) return;
    assert(!mt.count({b, d})) || (mt[{b, d}] == make_pair(-1, -1));
    mt[{b, d}] = {a, c};
    trim(b, d);
    mt.erase(make_pair(a, c));
    change(a, b, c, d);change(b, c, a, d);change(a, d, c, b);change(c, d, a, b);
    rec(a, b, points);rec(b, c, points);rec(c, d, points);rec(d, a, points);
}
void addTriangle(int a, int b, int c) {
    change(a, b, -1, c);change(a, c, -1, b);change(b, c, -1, a);
}
void anyTriangulation(vector<P> points) {
    sort(points.begin(), points.end(), [](const P &a, const P &b) {
        return make_pair(a.x, a.y) < make_pair(b.x, b.y);});
    vector<P> upper, lower;
    for (P C : points) {
#define backback(w) w[(int)w.size() - 2]

```



```

while ((int)upper.size() >= 2 && backback(upper).cross(upper.back(), C)>0){
    addTriangle(C.id, backback(upper).id, upper.back().id);
    upper.pop_back();
}
upper.push_back(C);
while ((int)lower.size() >= 2 && lower[(int)lower.size() - 2].cross(lower.back(), C)
< 0) {
    addTriangle(C.id, backback(lower).id, lower.back().id);
    lower.pop_back();
}
lower.push_back(C);
#undef backback
} //all collinear
if (lower.size()==upper.size() && lower.size()==points.size()) assert(false);
}
const int nax = 1e6 + 5;
int memo_x[nax], memo_y[nax];
void dewolaj() {
    mt.clear(); edges.clear(); triangles.clear();
    int n;
    scanf("%d", &n);
    vector<P> points(n);
    for (int i = 0; i < n; ++i) {
        scanf("%d%d", &memo_x[i], &memo_y[i]);
        points[i] = P{memo_x[i], memo_y[i], i};
    }
    mt.reserve(4123123);
    anyTriangulation(points);
    vector<pair<int, int>> init;
    for (auto ppp : mt) init.push_back(ppp.first);
    for (pair<int, int> p : init)
        if (mt.count(p) && mt[p] != make_pair(-1, -1)) rec(p.first, p.second, points);
    for (auto ppp : mt)
        if (ppp.second != make_pair(-1, -1)) {
            int i = ppp.first.first, j = ppp.first.second;
            assert(i != j);
            edges.insert({i, j});
            if (mt[{i, j}].first > j) triangles.push_back(vector<int>{i, j, mt[{i, j}].first});
            if (mt[{i, j}].second > j) triangles.push_back(vector<int>{i, j, mt[{i, j}].second});
        }
}
// edges zawiera krawędzie triangulacji

```

### Geometry/Halfplanes.cpp

```

pii operator-(pii a, pii b) {return {a.first - b.first, a.second - b.second};}
pii rot(pii x) { return {x.second, -x.first}; }
using hpl = pair<pii, ll>; //(v, m) = {x : sc(x, v) >= m}
ll sc(pii a, pii b) {return a.first*ll*b.first + a.second*ll*b.second;}
ll ve(pii a, pii b) {return a.first*ll*b.second - a.second*ll*b.first;}
//wartości rzędu współrzędne ^ 4
__int128 det(hpl a, hpl b, hpl c) {
    return a.second * (__int128)ve(b.first, c.first) +
        b.second * (__int128)ve(c.first, a.first) +
        c.second * (__int128)ve(a.first, b.first);
}
bool subset(hpl a, hpl b) {
    if (ve(a.first, b.first) || sc(a.first, b.first) < 0) return false;
    return a.second * (__int128) abs(b.first.first) >= b.second * (__int128) abs(a.first.fi
rst) &&
        a.second * (__int128) abs(b.first.second) >= b.second * (__int128) abs(a.first.second);
}
bool disjoint(hpl a, hpl b) {
    return subset(a, {-b.first.first, -b.first.second}, -b.second); }
bool around(hpl a, hpl b, hpl c) {

```

```

ll ab = ve(a.first, b.first);
ll bc = ve(b.first, c.first);
ll ca = ve(c.first, a.first);
assert(ab > 0 || bc > 0 || ca > 0);
return (ab >= 0 && bc >= 0 && ca >= 0) || (ab <= 0 && bc <= 0 && ca <= 0);
}
void ang_sort(vector<hpl> &a) {
    if (a.empty()) return;
    hpl mid = a.back();
    a.pop_back();
    vector<hpl> left, right;
    for (hpl c : a) {
        ll v = ve(c.first, mid.first), s = sc(c.first, mid.first);
        if (make_pair(v, s) > pair<ll, ll>(0, 0))
            left.push_back(c);
        else
            right.push_back(c);
    }
    sort(right.begin(), right.end(), [](hpl x, hpl y){return ve(x.first, y.first) > 0;});
    sort(left.begin(), left.end(), [](hpl x, hpl y){return ve(x.first, y.first) > 0;});
    left.push_back(mid);
    left.insert(left.end(), right.begin(), right.end());
    swap(a, left);
}
//Jeśli przecięcie jest nieograniczone, to półproste idące do nieskończoności są wyznaczo
ne
//przez pierwszy i ostatni element zwróconego wektora
//Jeśli przecięcie jest puste, zwraca pusty wektor
vector<hpl> find_hull(vector<hpl> vec) {
    vector<hpl> hull;
    int first = 0;
    ang_sort(vec);
    for (hpl curr : vec) {
        if (!hull.empty() && disjoint(curr, hull.back()))
            return {}; //Przecięcie jest puste
        if (!hull.empty() && disjoint(curr, hull[first]))
            return {}; //Przecięcie jest puste
        if (!hull.empty() && subset(hull.back(), curr)) //Case kiedy jedna półpłaszczyzna zaw
iera się w drugiej, na ogół można wywalić
            continue;
        if (!hull.empty() && subset(curr, hull.back())) //J.w.
            hull.pop_back();
        while (hull.size() - first >= 2u && det(hull.back(), *(hull.end() - 2), curr) <= 0) {
            if (around(hull[hull.size() - 2], hull.back(), curr))
                return {}; //Przecięcie jest puste
            else
                hull.pop_back();
        }
        while (hull.size() - first >= 2u && det(curr, hull[first], hull[first + 1]) >= 0) {
            if (around(curr, hull[first], hull[first + 1]))
                return {}; //Przecięcie jest puste
            else
                first++;
        }
        if (hull.size() - first < 2u || det(hull.back(), curr, hull[first]) < 0)
            hull.push_back(curr);
    }
    return vector<hpl>(hull.begin() + first, hull.end());
}
//Półpłaszczyzna wyznaczona przez prostą xy
hpl make_hpl(pii x, pii y) {
    pii v = rot(y - x);
    assert(sc(v, x) == sc(v, y));
    return {v, sc(x, v)};
}

```

## Geometry/Rotor.cpp

```

pii in[nax];
int wh[nax];
pair <pii, pii> dir[nax * nax / 2];
pii operator-(pii a, pii b) {return {a.first - b.first, a.second - b.second;}}
ll pro(pii a, pii b) {return a.first*1ll*b.second - a.second*1ll*b.first;}
bool cmp(pair <pii, pii> a, pair <pii, pii> b) {
    ll p = pro(a.first, b.first);
    if (p > 0) return 1; if (p < 0) return 0; return a.second < b.second;
    //Jak nie ma trzech współliniowych to po prostu: return pro(a.first, b.first) > 0;
}
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        scanf("%d%d", &in[i].first, &in[i].second);
    sort(in, in + n);
    for (int i = 0; i < n; ++i)
        wh[i] = i;
    int cou = 0;
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            dir[cou++] = {max(in[i].first - in[j], in[j] - in[i]), {i, j}};
    sort(dir, dir + cou, cmp);
    for (int i = 0; i < cou; ++i) {
        debug << imie(i) imie(dir[i]);
        auto c = dir[i].second;
        pii x = in[wh[c.first]], y = in[wh[c.second]];
        swap(wh[c.first], wh[c.second]);
        swap(in[wh[c.first]], in[wh[c.second]]);
        //Policz wynik dla posortowanych prostopadłe do prostej xy
    }
}

```

## Graphs/2sat.cpp

```

const int nax = 100000;
vector <int> implies[2*nax]; //wymuszenia, 2*x to zmienna 2*x+1 to jej negacja
int sat_val[2*nax], sat_vis[2*nax], sat_sort[2*nax], sat_ile;
inline void sat_or(int a, int b) {
    implies[a^1].push_back(b);
    implies[b^1].push_back(a);
}
void sat_dfs_mark(int x) {
    sat_vis[x]=1;
    sat_val[x]=sat_val[x^1]==-1;
    for (int i : implies[x]) if (!sat_vis[i]) sat_dfs_mark(i);
}
void sat_dfs(int x) {
    sat_vis[x]=1;
    for (int i : implies[x]) if (!sat_vis[i]) sat_dfs(i);
    sat_sort[--sat_ile]=x^1;
}
bool sat2(int n) { //n - liczba zmiennych, zmienne numerujemy od 0
#define REP for (int i = 0; i < 2 * n; ++i)
    sat_ile=2 * n;
    REP sat_vis[i]=0, sat_val[i]=-1;
    REP if (!sat_vis[i]) sat_dfs(i);
    REP sat_vis[i]=0;
    REP if (!sat_vis[sat_sort[i]]) sat_dfs_mark(sat_sort[i]);
    REP if (sat_val[i]) for (int j : implies[i]) if (!sat_val[j]) return 0;
    return 1;
}

```

## Graphs/Centroid.cpp

```

struct centro { // indeksowane od 0, par to drzewo centroidow
    vector <vi> edges;
    vector <bool> vis;
    vi par, sz;

```

```

    int n;
    centro(int n) : n(n) {
        edges.resize(n);
        vis.resize(n);
        par.resize(n);
        sz.resize(n);
    }
    void edge(int a, int b) {
        edges[a].pb(b);
        edges[b].pb(a);
    }
    int find_size(int v, int p = -1) {
        if (vis[v]) return 0;
        sz[v] = 1;
        for (int x: edges[v]) {
            if (x != p) {
                sz[v] += find_size(x, v);
            }
        }
        return sz[v];
    }
    int find_centroid(int v, int p, int n) {
        for (int x: edges[v]) {
            if (x != p) {
                if (!vis[x] && sz[x] > n / 2) {
                    return find_centroid(x, v, n);
                }
            }
        }
        return v;
    }
    void init_centroid(int v = 0, int p = -1) {
        find_size(v);
        int c = find_centroid(v, -1, sz[v]);
        vis[c] = true;
        par[c] = p;
        for (int x: edges[c]) {
            if (!vis[x]) {
                init_centroid(x, c);
            }
        }
    }
};

```

## Graphs/DMST.cpp

```

struct RollbackUF {
    vi e;
    vprii st;
    RollbackUF(int n) : e(n, -1) {}
    int size(int x) { return -e[find(x)]; }
    int find(int x) { return e[x] < 0 ? x : find(e[x]); }
    int time() { return siz(st); }
    void rollback(int t) {
        for (int i = time(); i-- > t;) e[st[i].first] = st[i].nd;
        st.resize(t);
    }
    bool unite(int a, int b) {
        a = find(a), b = find(b);
        if (a == b) return false;
        if (e[a] > e[b]) swap(a, b);
        st.pb({a, e[a]});
        st.pb({b, e[b]});
        e[a] += e[b];
        e[b] = a;
        return true;
    }
};

```

```

};
struct Edge {
    int a, b;
    ll w;
};
struct Node { // lazy skew heap node
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() {
        prop();
        return key;
    }
};
Node* merge(Node* a, Node* b) {
    if (!a || !b) return a ? b;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}
void pop(Node*& a) {
    a->prop();
    a = merge(a->l, a->r);
} // wierzchołki numerujemy od 0, r to korzen dmst
pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
    for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
    ll res = 0;
    vector<int> seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<Edge> Q(n), in(n, {-1, -1}), comp;
    deque<tuple<int, int, vector<Edge>>> cycs;
    for (int s = 0; s < n; ++s) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1, {}};
            Edge e = heap[u]->top();
            heap[u]->delta -= e.w, pop(heap[u]);
            Q[qi] = e, path[qi++] = u, seen[u] = s;
            res += e.w, u = uf.find(e.a);
            if (seen[u] == s) { // found cycle, contract
                Node* cyc = 0;
                int end = qi, time = uf.time();
                do cyc = merge(cyc, heap[w = path[--qi]]);
                while (uf.unite(u, w));
                u = uf.find(u), heap[u] = cyc, seen[u] = -1;
                cycs.push_front({u, time, {&Q[qi], &Q[end]}});
            }
        }
        for (int i = 0; i < qi; ++i) in[uf.find(Q[i].b)] = Q[i];
    }
    for (auto& [u, t, comp] : cycs) { // restore sol (optional)
        uf.rollback(t);
        Edge inEdge = in[u];
        for (auto& e : comp) in[uf.find(e.b)] = e;
        in[uf.find(inEdge.b)] = inEdge;
    }
}

```

```

for (int i = 0; i < n; ++i) par[i] = in[i].a;
return {res, par};
}

```

### Graphs/Dominatory.cpp

```

struct Dominators {
    int n_orig, n;
    vector<int> parent, semi, vertex, dom, ancestor, label;
    vector<vector<int>> succ, pred, bucket;
    Dominators(int _n): n_orig(_n), n(_n + 1), parent(n), semi(n), vertex(n), dom(n), ances
    tor(n), label(n), succ(n), pred(n), bucket(n) {
        n = n_orig;
    }
    void add_edge(int a, int b) {
        a++; b++; succ[a].push_back(b);
    }
    void COMPRESS(int v) {
        if (ancestor[ancestor[v]] != 0) {
            COMPRESS(ancestor[v]);
            if (semi[label[ancestor[v]]] < semi[label[v]]) label[v] = label[ancestor[v]];
            ancestor[v] = ancestor[ancestor[v]];
        }
    }
    void LINK(int v, int w) { ancestor[w] = v; }
    int EVAL(int v) {
        if (ancestor[v] == 0) return v;
        else {
            COMPRESS(v);
            return label[v];
        }
    }
    void DFS(int v) {
        semi[v] = ++n; vertex[n] = v;
        for (auto ng : succ[v]) {
            if (semi[ng] == 0) {
                parent[ng] = v;
                DFS(ng);
            }
            pred[ng].push_back(v);
        }
    }
    // Zwraca vector dominatorow (-1 dla 0). // Wszystkie numerowane od 0,
    vector<int> doit() { // wszystko musi być osiągalne z 0.
        iota(label.begin(), label.end(), 0);
        n = 0; DFS(1);
        for (int i = n; i >= 2; --i) {
            int w = vertex[i];
            for (auto ng : pred[w]) {
                int u = EVAL(ng);
                if (semi[u] < semi[w]) { semi[w] = semi[u]; }
            }
            bucket[vertex[semi[w]]].push_back(w);
            LINK(parent[w], w);
            while (!bucket[parent[w]].empty()) {
                int v = bucket[parent[w]].back();
                bucket[parent[w]].pop_back();
                int u = EVAL(v);
                if (semi[u] < semi[v]) dom[v] = u;
                else dom[v] = parent[w];
            }
        }
        for (int i = 2; i <= n; ++i) {
            int w = vertex[i];
            if (dom[w] != vertex[semi[w]]) { dom[w] = dom[dom[w]]; }
        }
        dom[1] = 0;
        vector<int> res(n_orig);
        for (int i = 0; i < n_orig; ++i) res[i] = dom[i + 1] - 1;
        return res;
    }
};

```

### Graphs/HLD.cpp

```

vi drz[nax];
int roz[nax], jump[nax], pre[nax], post[nax], fad[nax], czas;

```

```

void dfs_roz(int v) {
    roz[v]=1; // drz[] ma nie zawierać krawędzi
    for (int &i : drz[v]) { // do ojca.
        fad[i]=v; // Init:
        dfs_roz(i); // dfs_roz(root);
        roz[v]+=roz[i]; // dfs_pre(root);
        if (roz[i]>roz[drz[v][0]]) // Użycie get_path(v, u) zwróci
            swap(i, drz[v][0]); // przedziały odpowiadające ścieżce
    } // z v do u. Przedziały odpowiadające
    // ścieżce z v do lca mają
    void dfs_pre(int v) // first>=second, zaś te dla ścieżki
    { // z lca do u mają first<=second.
        if (!jump[v]) // Przedziały są po kolei.
            jump[v]=v; // Lca występuje w nich dwa razy,
        pre[v]=(++czas); // najpierw jako second,
        if (!drz[v].empty()) // a zaraz potem jako first.
            jump[drz[v][0]]=jump[v];
        for (int i : drz[v])
            dfs_pre(i);
        post[v]=czas;
    }
    int lca(int v, int u) {
        while(jump[v]!=jump[u]) {
            if (pre[v]<pre[u])
                swap(v, u);
            v=fad[jump[v]];
        }
        return (pre[v]<pre[u] ? v : u);
    }
    vpil path_up(int v, int u) {
        vpil ret;
        while(jump[v]!=jump[u]) {
            ret.pb({pre[jump[v]], pre[v]});
            v=fad[jump[v]];
        }
        ret.pb({pre[u], pre[v]});
        return ret;
    }
    vpil get_path(int v, int u) {
        int w=lca(v, u);
        auto ret=path_up(v, w);
        auto pom=path_up(u, w);
        for (auto &i : ret)
            swap(i.st, i.nd);
        while(!pom.empty()) {
            ret.pb(pom.back());
            pom.pop_back();
        }
        return ret;
    }
}

```

## Graphs/Link\_Cut.cpp

```

struct Splay {
    Splay *l = 0, *r = 0, *p = 0;
    bool flip = false; // Wywal jak nie używasz make_root.
    void update() {
        //update anything stored for nodes
        assert(!flip and (!l or !l->flip) and (!r or !r->flip));
    }
    void touch() { //Do any lazy prop here
        if (flip) {
            swap(l, r);
            if (l) l->flip = !l->flip;
            if (r) r->flip = !r->flip;
            flip = false;
        }
    }
}

```

```

bool sroot() { return !p or (p->l != this and p->r != this); }
void connect(Splay* c, bool left) { (left ? l : r) = c; if (c) c->p = this; }
void rotate() {
    Splay* f = p; Splay* t = f->p;
    const bool isr = f->sroot();
    const bool left = (this == f->l);
    f->connect(left ? r : l, left);
    connect(f, !left);
    if (isr) p = t;
    else t->connect(this, f == t->l);
    f->update();
}
void push(){sroot()?touch():p->push();if (l) l->touch();if (r) r->touch();}
void splay() {
    push();
    while (!sroot()) {
        Splay* x = p->p;
        if (!p->sroot()) (((p->l == this) == (x->l == p)) ? p : this)->rotate();
        rotate();
    }
    update();
}
Splay* expose(){//v będzie korzeniem splaya zawierającego ścieżkę do korzenia
    Splay *q = this, *x = 0; //prawe dziecko będzie nullem. Jak zejdziemy w dół,
    while (q) { // to potem trzeba zrobić splay().
        q->splay(); // LCA(u, v): u->expose(); return v->expose();
        q->r = x; q->update();
        x = q; q = q->p;
    }
    splay();
    return x;
}
Splay* root() { // Zwraca roota drzewowego (nie splejowego!).
    expose();
    Splay* s = this;
    while (s->touch(), s->l) s = s->l;
    s->splay();
    return s;
}
void cut() { // Usuwa krawędź znajdującą się nad danym wierzchołkiem.
    expose(); assert(!l /* Nie jest rootem. */);
    Splay* s = l;
    while (s->touch(), s->r) s = s->r;
    s->splay(); s->r->p = 0; s->r = 0;
}
void link(Splay* to) {
    expose(); assert(!l /* Jest rootem. */);
    p = to;
}
// Sprawia, że wierzchołek jest rootem w logicznym i w splayowym drzewie.
void make_root() { expose(); flip = !flip; touch(); }
};

```

## Graphs/Low.cpp

```

struct Low { // działa dla multikrawędzi, petli, niespojnego grafu
    int n, cnt, edges;
    vector<vector<PII>> G;
    VI low, pre, par, par_nr, used, most, root, vis_s, add_s;
    Low(int n) : n(n), edges(0) {
        G.resize(n + 1);
        low.resize(n + 1);
        pre.resize(n + 1);
        par.resize(n + 1);
        par_nr.resize(n + 1);
        root.resize(n + 1);
    }
}

```

```

void edge(int a, int b) {
    ++edges;
    G[a].eb(b, edges);
    G[b].eb(a, edges);
}
void dfs(int v) {
    pre[v] = ++cnt;
    low[v] = pre[v];
    for (auto it : G[v]) {
        int u = it.st;
        int nr = it.nd;
        if (used[nr]) continue;
        used[nr] = 1;
        if (!pre[u]) {
            par[u] = v;
            par_nr[u] = nr;
            dfs(u);
            mini(low[v], low[u]);
        } else
            mini(low[v], pre[u]);
    }
}
void go() { // trzeba wywolac na poczatku!
    used.resize(edges + 1);
    FOR(i, 1, n) if (!pre[i]) root[i] = 1, dfs(i);
}
vector<PII> mosty() {
    most.resize(edges + 1);
    vector<PII> ans;
    FOR(i, 1, n) {
        if (!root[i] && low[i] == pre[i]) {
            ans.eb(i, par[i]);
            most[par_nr[i]] = 1;
        }
    }
    return ans;
}
VI pkt_art() { // tylko jesli potrzebujemy
    VI ans, take(n + 1), root_sons(n + 1);
    FOR(i, 1, n) {
        if (par[i] && root[par[i]]) {
            ++root_sons[par[i]];
        }
    }
    FOR(i, 1, n) {
        if (root[i] && root_sons[i] >= 2) take[i] = 1;
        if (!root[i] && !root[par[i]] && low[i] >= pre[par[i]]) take[par[i]] = 1;
    }
    FOR(i, 1, n) if (take[i]) ans.pb(i);
    return ans;
}
// kod nizej tylko jesli potrzebujemy 2spojnych
using comps = vector<vector<PII>>;
void dfs_s(int v, vector<PII> &moja) {
    vis_s[v] = 1;
    for (auto it : G[v]) {
        int u = it.st;
        int nr = it.nd;
        if (most[nr]) continue;
        if (!add_s[nr]) {
            moja.eb(v, u);
            add_s[nr] = 1;
        }
        if (!vis_s[u]) dfs_s(u, moja);
    }
}

```

```

}
comps bico() {
    vis_s.resize(n + 1);
    add_s.resize(edges + 1);
    comps ans;
    for (auto it : mosty()) ans.pb({it});
    FOR(i, 1, n) {
        if (!vis_s[i]) {
            vector<PII> curr;
            dfs_s(i, curr);
            if (!curr.empty()) ans.pb(curr);
        }
    }
    return ans;
}
};

```

### Graphs/Planarity\_Check.cpp

```

#define PB push_back
#define SZ(x) ((int)(x).size())
#define FOR(i,a,b) for (int i = (a); i <= (b); ++i)
#define VI vector<int> //numeracja od zera
//nie dawać multikrawędzi ani pętli
//nie odpalać IsPlanar() ponad raz
struct FU {
    VI p; vector<bool> b;
    FU() {}
    FU(int n) : p(n), b(n) { iota(p.begin(), p.end(), 0); }
    pair<int, bool> Find(int v) {
        if (p[v] == v) { return {v, 0}; }
        auto res = Find(p[v]); res.second ^= b[v];
        p[v] = res.first; b[v] = res.second; return res;
    }
    bool Union(int x, int y, bool flip) {
        bool xb, yb; tie(x, xb) = Find(x); tie(y, yb) = Find(y);
        if (x == y) { return !(xb ^ yb ^ flip); }
        p[x] = y; b[x] = xb ^ yb ^ flip; return true;
    }
};
struct PlanarityTest {
    int N, M, tm;
    vector<VI> adj, dn, up; vector<pair<int, int>> e_up;
    vector<bool> vis; VI low, pre;
    FU fu;
    void DfsLow(int v, int p) {
        vis[v] = true; low[v] = pre[v] = tm++;
        for (int s : adj[v]) {
            if (s == p) { continue; }
            if (!vis[s]) {
                dn[v].PB(s); DfsLow(s, v); low[v] = min(low[v], low[s]);
            } else if (pre[s] < pre[v]) {
                up[v].PB(SZ(e_up)); e_up.PB({v, s});
                low[v] = min(low[v], pre[s]);
            }
        }
    }
}
VI Interlace(const VI &ids, int lo) {
    VI ans;
    for (int e : ids) if (pre[e_up[e].second] > lo) ans.PB(e);
    return ans;
}
bool AddFU(const VI &a, const VI &b) {
    FOR (k, 1, SZ(a) - 1) if (!fu.Union(a[k - 1], a[k], 0)) { return false; }
    FOR (k, 1, SZ(b) - 1) if (!fu.Union(b[k - 1], b[k], 0)) { return false; }
    if (SZ(a) && SZ(b) && !fu.Union(a[0], b[0], 1)) { return false; }
    return true;
}
bool DfsPlanar(int v, int p) {

```

```

for (int s : dn[v]) if (!DfsPlanar(s, v)) { return false; }
auto sz = SZ(dn[v]);
FOR (i, 0, sz - 1) {
    FOR (j, i + 1, sz - 1) {
        auto a = Interlace(up[dn[v][i]], low[dn[v][j]]);
        auto b = Interlace(up[dn[v][j]], low[dn[v][i]]);
        if (!AddFU(a, b)) { return false; }
    }
    for (int j : up[v]) {
        if (e_up[j].first != v) { continue; }
        auto a = Interlace(up[dn[v][i]], pre[e_up[j].second]);
        auto b = Interlace({j}, low[dn[v][i]]);
        if (!AddFU(a, b)) { return false; }
    }
}
for (int s : dn[v]) {
    for (int idx : up[s]) {
        if (pre[e_up[idx].second] < pre[p]) { up[v].PB(idx); }
    }
    up[s].clear(); up[s].shrink_to_fit();
}
return true;
}
PlanarityTest(int n) : N(n), M(0), adj(n) {}
void AddEdge(int u, int v) { adj[u].PB(v); adj[v].PB(u); ++M; }
bool IsPlanar() {
    if (N <= 3) { return true; }
    if (M > 3 * N - 6) { return false; }
    vis = vector<bool>(N);
    up = dn = vector<VI>(N);
    low = pre = VI(N);
    FOR (v, 0, N - 1) if (!vis[v]) {
        e_up.clear(); tm = 0; DfsLow(v, -1);
        fu = FU(SZ(e_up));
        if (!DfsPlanar(v, -1)) { return false; }
    }
    return true;
}
};

```

## Graphs/Smulewicz.cpp

```

#define int ll//jeśli longi potrzebne
struct SPFA{
    int n; vector<int> odl, oj, czok;
    vector<vector<pair<int,int>>> d; vector<vector<int>> d2;
    const int inf = 1e9;
    SPFA(int _n):n(_n+1){
        odl.resize(n, inf); oj.resize(n); czok.resize(n);
        d.resize(n); d2.resize(n);
    }
    vector<int> cykl; int root;
    bool us(int nr){
        if(nr == root) return 1;
        czok[nr] = 0;
        for(int ak:d2[nr]){
            if(oj[ak] == nr){
                if(us(ak)){
                    cykl.push_back(nr);
                    return 1;
                }
            }
        }
        d2[nr].clear();
        return 0;
    }
    bool licz_sciezki(int s){ // false, gdy z s da sie dojsc do ujemnego cyklu

```

```

vector<int> st, st2; // znaleziony cykl jest w wektorze cykl
odl[s] = 0; czok[s] = 1; st.push_back(s);
while(st.size()){
    for(int i=0; i<(int)st.size(); i++){
        int ak = st[i];
        if(czok[ak]) for(pair<int,int> x:d[ak]){
            int nei, cost; tie(nei, cost) = x;
            if(odl[ak] + cost < odl[nei]){
                root = ak;
                if(us(nei)){
                    cykl.push_back(ak); reverse(cykl.begin(), cykl.end());
                    return 0;
                }
                odl[nei] = odl[ak] + cost; oj[nei] = ak;
                d2[ak].push_back(nei); czok[nei] = 1;
                st2.push_back(nei);
            }
        }
        st.clear(); swap(st, st2);
    }
    return 1;
}
vector<int> ujemny cykl(){
    for (int i=0; i<n-1; i++) add_edge(n-1, i, 0);
    if (licz_sciezki(n-1)){
        return {};
    } else {
        return cykl;
    }
}
void add_edge(int a, int b, int cost){
    d[a].push_back({b, cost});
}
};

```

#undef int

## Graphs/SSS.cpp

```

const int nax=100*1007;
vector<int> graf[nax], farg[nax];
int ost[nax], bylo[nax], post[nax], spo[nax], counter, coudfs;
vector<vector<pair<int,int>>> mer;
void dfs1(int v) {
    if (bylo[v])
        return;
    bylo[v]=1;
    for (int i : graf[v]) dfs1(i);
    coudfs--;
    post[coudfs]=v;
}
void dfs2(int v, int s) {
    if (spo[v]==0)
        return;
    spo[v]=s;
    for (int i : farg[v]) dfs2(i, s);
}
void rek(int l, int r, vector<pair<pair<int,int>,int>> &kra) {
    if (l>r) return;
    counter++;
    vector<int> ver;
    for (auto i : kra) {
        if (ost[i.first.first]<counter) {
            ver.push_back(i.first.first);
            ost[i.first.first]=counter;
        }
        if (ost[i.first.second]<counter) {

```

```

        ver.push_back(i.first.second);
        ost[i.first.second]=counter;
    }
}
for (int i : ver) {
    bylo[i]=0; spo[i]=-1;
    graf[i].clear(); farg[i].clear();
}
int s=(l+r)>>1;
for (auto i : kra) {
    if (i.second<=s) {
        graf[i.first.first].push_back(i.first.second);
        farg[i.first.second].push_back(i.first.first);
    }
}
coudfs=ver.size();
for (int i : ver)
    dfs1(i);
for (int i=0; i<(int)ver.size(); i++)
    dfs2(post[i], post[i]);
for (int i : ver)
    if (i!=spo[i])
        mer[s].push_back({i, spo[i]});
vector <pair<pair<int,int>,int>> lew, pra;
for (auto i : kra) {
    if (spo[i.first.first]==spo[i.first.second])
        lew.push_back(i);
    else
        pra.push_back({spo[i.first.first],spo[i.first.second]}, i.second);
}
rek(l, s-1, lew); rek(s+1, r, pra);
}
void sss(vector <pair<int,int>> kra) {
    mer.clear();
    mer.resize(kra.size());
    vector <pair<pair<int,int>,int>>daj;
    for (int i=0; i<(int)kra.size(); i++) {
        daj.push_back({kra[i], i});
        ost[kra[i].first]=-1;
        ost[kra[i].second]=-1;
    }
    counter=0;
    rek(0, (int)kra.size()-1, daj);
}

```

### Graphs/Vizing.cpp

```

#define int ll//jeśli long longi potrzebne
struct Vizining {
    vector<vector<pair<int,int>>> adj;
    map<pair<int,int>, int> edges;
    vector<int> edge_colors;
    vector<vector<int>> color_to;
    vector<vector<int>> color_queue;
    vector<int> unused_color;
    int N, M, K;
    Vizining(int size) : adj(size), N(size), M(0) {}
    void AddEdge(int u, int v) {
        adj[u].emplace_back(u, M);
        adj[v].emplace_back(v, M);
        edges[make_pair(min(u, v), max(u, v))] = M;
        ++M;
    }
    //numeracja od zera
    //multikrawędzie niebezpieczne
    //ogólnie nietestowane
    int MaxDegree() const {
        int answer = 0;
        for (int i = 0; i < N; ++i)

```

```

        answer=max(answer, (int)adj[i].size());
    }
    return answer;
}
void FindFree(int v) {
    while (!color_queue[v].empty()) {
        const int c = color_queue[v].back();
        if (color_to[v][c] == -1) {
            unused_color[v] = c;
            return;
        } else {
            color_queue[v].pop_back();
        }
    }
    assert(false);
}
void FlipPath(int from, int a, int b) {
    const int to = color_to[from][a];
    color_queue[from].push_back(a);
    color_queue[from].push_back(b);
    color_to[from][b] = -1;
    if (to == -1) { return; }
    FlipPath(to, b, a);
    color_to[from][b] = to;
    color_to[to][b] = from;
    color_to[from][a] = -1;
    FindFree(from);
    FindFree(to);
}
void ColorEdge(int x, int y) {
    for (int col = 0; col <= K; ++col)
        if (color_to[x][col] == -1 && color_to[y][col] == -1) {
            color_to[x][col] = y;
            color_to[y][col] = x;
            FindFree(x); FindFree(y);
            return;
        }
    vector<int> fan{y}, fan_colors{-1};
    vector<bool> has_on(N);
    const int c = unused_color[x];
    int d;
    while (true) {
        d = unused_color[fan.back()];
        if (color_to[x][d] == -1 || d == c) { break; }
        if (has_on[d]) {
            FlipPath(x, d, c);
            FindFree(x);
            return ColorEdge(x, y);
        }
        fan.push_back(color_to[x][d]);
        fan_colors.push_back(d);
        has_on[d] = true;
    }
    fan_colors.push_back(-1);
    for (int i = 0; i < (int)fan.size(); ++i) {
        const int from = fan_colors[i], to = fan_colors[i + 1];
        if (from != -1) {
            color_queue[fan[i]].push_back(from);
            color_queue[x].push_back(from);
            color_to[fan[i]][from] = -1;
        }
        if (to != -1) {
            color_to[fan[i]][to] = x;
            color_to[x][to] = fan[i];
        }
    }
}

```



```

assert(color_to[x][d] == -1);
color_to[x][d] = fan.back();
color_to[fan.back()][d] = x;
for (int v : fan) { FindFree(v); }
FindFree(x);
}
vector<int> ColorGraph() {
    K = MaxDegree();
    edge_colors = vector<int>(M, -1);
    color_to = vector<vector<int>>(N, vector<int>(K + 1, -1));
    vector<int> all_colors(K + 1);
    iota(all_colors.begin(), all_colors.end(), 0);
    color_queue = vector<vector<int>>(N, all_colors);
    unused_color = vector<int>(N);
    for (auto &edge : edges)
        ColorEdge(edge.first.first, edge.first.second);
    for (int v = 0; v < N; ++v)
        for (int c = 0; c <= K; ++c)
            if (color_to[v][c] > v) {
                assert(color_to[color_to[v][c]][c] == v);
                edge_colors[edges[make_pair(v, color_to[v][c])]] = c;
            }
    return edge_colors;
}
};
#undef int

```

## Math/BitowFFT.cpp

```

void xor_fft(ll* p, int n, bool inv) {
    for (int len = 1; 2 * len <= n; len <= 1) {
        for (int i = 0; i < n; i += 2 * len) {
            for (int j = 0; j < len; ++j) {
                ll u = p[i + j];
                ll v = p[i + len + j];
                p[i + j] = u + v;
                p[i + len + j] = u - v;
            }
        }
        if (inv) {
            for (int i = 0; i < n; ++i) {
                p[i] /= n; // uwaga gdy liczymy modulo!!
            }
        }
    }
}

void and_fft(ll* p, int n, bool inv) {
    for (int len = 1; 2 * len <= n; len <= 1) {
        for (int i = 0; i < n; i += 2 * len) {
            for (int j = 0; j < len; ++j) {
                ll u = p[i + j];
                ll v = p[i + len + j];
                if (!inv) {
                    p[i + j] = v;
                    p[i + len + j] = u + v;
                } else {
                    p[i + j] = -u + v;
                    p[i + len + j] = u;
                }
            }
        }
    }
}

void or_fft(ll* p, int n, bool inv) {
    for (int len = 1; 2 * len <= n; len <= 1) {
        for (int i = 0; i < n; i += 2 * len) {
            for (int j = 0; j < len; ++j) {
                ll u = p[i + j];
                ll v = p[i + len + j];
                if (!inv) {
                    p[i + j] = u + v;
                    p[i + len + j] = u;
                } else {
                    p[i + j] = v;
                    p[i + len + j] = u - v;
                }
            }
        }
    }
}

```

```

} } } } }

```

## Math/FFT.cpp

```

#define REP(i,n) for(int i = 0; i < int(n); ++i)
/*Precision error max_ans/1e15 (2.5e18) for (long) doubles.
So integer rounding works for doubles with answers 0.5*1e15,
e.g. for sizes 2^20 and RANDOM positive integers up to 45k.
Those values assume DBL_MANT_DIG=53 and LDBL_MANT_DIG=64.
For input in [0, M], you can decrease everything by M/2.
If there are many small vectors, uncomment "BRUTE FORCE".*/
typedef double ld; // 'long double' is 2.2 times slower
struct C {
    ld real, imag;
    C operator * (const C & he) const {
        return C{real * he.real - imag * he.imag, real * he.imag + imag * he.real};
    }
    void operator += (const C & he) {real += he.real; imag += he.imag; }
};

void dft(vector<C> & a, bool rev) {
    const int n = a.size();
    for (int i = 1, k = 0; i < n; ++i) {
        for (int bit = n / 2; (k ^ bit) < bit; bit /= 2);
        if (i < k) swap(a[i], a[k]);
    }
    for (int len = 1, who = 0; len < n; len *= 2, ++who) {
        static vector<C> t[30];
        vector<C> & om = t[who];
        if (om.empty()) {
            om.resize(len);
            const ld ang = 2 * acosl(0) / len;
            REP(i, len) om[i] = i%2 ? C{cos(i*ang), sin(i*ang)} : C{cos(i*ang), -sin(i*ang)};
        }
        for (int i = 0; i < n; i += 2 * len)
            REP(k, len) {
                const C x = a[i+k], y = a[i+k+len];
                * C{om[k].real, om[k].imag} * (rev ? -1 : 1);
                a[i+k] += y;
                a[i+k+len] = C{x.real - y.real, x.imag - y.imag};
            }
    }
    if (rev) REP(i, n) a[i].real /= n;
}

template<typename T> vector<T> multiply(const vector<T> &a, const vector<T> &b, bool split = false) {
    if (a.empty() || b.empty()) return {};
    int n = a.size() + b.size();
    vector<T> ans(n - 1);
    /* if(min(a.size(), b.size()) < 190) { // BRUTE FORCE
        REP(i, a.size()) REP(j, b.size()) ans[i+j] += a[i]*b[j];
        return ans; } */
    while(n & (n-1)) ++n;
    auto speed = [&](const vector<C> & w, int i, int k) {
        int j = i ? n - i : 0, r = k ? -1 : 1;
        return C{w[i].real + w[j].real * r, w[i].imag - w[j].imag * r} * (k ? C{0, -0.5} : C{0.5, 0});
    };
    if (!split) { // standard fast version
        vector<C> in(n), done(n);
        REP(i, a.size()) in[i].real = a[i];
        REP(i, b.size()) in[i].imag = b[i];
        dft(in, false);
        REP(i, n) done[i] = speed(in, i, 0) * speed(in, i, 1);
        dft(done, true);
        REP(i, ans.size()) ans[i] = is_integral<T>::value ?
            llround(done[i].real) : done[i].real;
    }
    else {

```

```

const int M = 1 << 15;
vector<C> t[2];
for (int x = 0; x < 2; ++x) {
    t[x].resize(n);
    const vector<T> &in = (x ? b : a);
    for (int i = 0; i < (int) in.size(); ++i)
        t[x][i] = C{ld(in[i] % M), ld(in[i] / M)};
    dft(t[x], false);
}
vector<C> d1(n), d2(n);
for (int i = 0; i < n; ++i) {
    d1[i] += speed(t[0], i, 0) * speed(t[1], i, 0);
    d1[i] += speed(t[0], i, 1) * speed(t[1], i, 1) * C{0, 1};
    d2[i] += speed(t[0], i, 0) * speed(t[1], i, 1);
    d2[i] += speed(t[0], i, 1) * speed(t[1], i, 0);
}
dft(d1, true);
dft(d2, true);
for (int i = 0; i < n; ++i)
    d1[i].imag /= n;
for (int i = 0; i < (int) ans.size(); ++i)
    ans[i] = (llround(d1[i].real) + llround(d2[i].real) % mod * M + llround(d1[i].imag)
% mod * (M * M)) % mod;
}
return ans;
}

```

## Math/Linear\_Function\_Convex\_Hull.cpp

```

const ll is_query = -(1LL << 62);
struct Line {
    ll m, b;
    mutable function<const Line *(> succ;
    bool operator<(const Line &rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line *s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x; //ld?
    }
};
struct HullDynamic : public multiset<Line> {
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b) * (z->m - y->m) >= (y->b - z->b) * (y->m - x->m); //ld?
    }
    void insert_line(ll m, ll b) {
        auto y = insert({m, b});
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) {
            erase(y);
            return;
        }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) {x, is_query});
        return l.m * x + l.b;
    }
};

```

## Math/Massey.cpp

```

const int mod = 1e9 + 7; // const ll mod = (1ll)1e18 + 3;
template<class C> void add_self(C &a, C b) { a += b; if(a >= mod) a -= mod; }
template<class C> void sub_self(C &a, C b) { a -= b; if(a < 0) a += mod; }
int mul(int a, int b) { return (ll) a * b % mod; }
ll mul(ll a, ll b) { return (__int128) a * b % mod; }
template<class C> C my_pow(C a, C b) { /*...*/ }
template<class C> C my_inv(C a) { return my_pow<C>(a, mod - 2); }
template<class C> C negative(C a) { return (mod - a) % mod; }
template<class F> struct Massey {
    vector<F> start, coef; // 3 optional lines
    vector<vector<F>> powers;
    F memo_inv;
    // Start here and write the next ~25 lines until "STOP"
    int L; // L == coef.size() <= start.size()
    Massey(vector<F> in) { // 0(N^2)
        L = 0;
        const int N = in.size();
        vector<F> C{1}, B{1};
        for(int n = 0; n < N; ++n) {
            // assert(0 <= in[n] && in[n] < mod); // invalid input
            B.insert(B.begin(), 0);
            F d = 0;
            for(int i = 0; i <= L; ++i)
                add_self(d, mul(C[i], in[n-i]));
            if(d == 0) continue;
            vector<F> T = C;
            C.resize(max(B.size(), C.size()));
            for(int i = 0; i < (int) B.size(); ++i)
                sub_self(C[i], mul(d, B[i]));
            if(2 * L <= n) {
                L = n + 1 - L;
                B = T;
                d = my_inv(d);
                for(F &x : B) x = mul(x, d);
            }
        }
        cerr << "L = " << L << "\n";
        assert(2 * L <= N - 2); // NO RELATION FOUND :(
        // == STOP ==
        for(int i = 1; i < (int) C.size(); ++i)
            coef.push_back(negative(C[i]));
        assert((int) coef.size() == L);
        for(int i = 0; i < L; ++i)
            start.push_back(in[i]);
        while(!coef.empty() && coef.back() == 0) { coef.pop_back(); --L; }
        if(!coef.empty()) memo_inv = my_inv(coef.back());
        powers.push_back(coef);
    }
    vector<F> mul_cut(vector<F> a, vector<F> b) {
        vector<F> r(2 * L - 1);
        for(int i = 0; i < L; ++i)
            for(int j = 0; j < L; ++j)
                add_self(r[i+j], mul(a[i], b[j]));
        while((int) r.size() > L) {
            F value = mul(r.back(), memo_inv); // div(r.back(), coef.back());
            const int X = r.size();
            add_self(r[X-L-1], value);
            for(int i = 0; i < L; ++i)
                sub_self(r[X-L+i], mul(value, coef[i]));
            assert(r.back() == 0);
            r.pop_back();
        }
        return r;
    }
};

```

```
F get(ll k) { //  $O(L^2 * \log(k))$ 
  if(k < (int) start.size()) return start[k];
  if(L == 0) return 0;
  k -= start.size();
  vector<F> vec = coef;
  for(int i = 0; (1LL <= i) <= k; ++i) {
    if(i == (int) powers.size())
      powers.push_back(mul_cut(powers.back(), powers.back()));
    if(k & (1LL <= i))
      vec = mul_cut(vec, powers[i]);
  }
  F total = 0;
  for(int i = 0; i < L; ++i)
    add_self(total, mul(vec[i], start[(int)start.size()-1-i]));
  return total; };
```

## Math/Multipoint\_Evaluation.cpp

```
ll dzial(ll a, ll b) {...}
vll inv(vll p, int n) {
  assert(p[0]);
  ll odw=dzial(1, p[0]);
  for (ll &i : p) i=(i*odw)%mod;
  vll q{1};
  for (int i=1; i<n; i<=1) {
    vll r=multiply(vll(p.begin(), p.begin()+min(2*i, (int)p.size())), q, true);
    r.resize(2*i);
    r.erase(r.begin(), r.begin()+i);
    for (ll &j : r) j=(mod-j)%mod;
    r=multiply(r, q, true);
    q.resize(2*i);
    for (int j=0; j<min(i, (int)r.size()); j++) q[i+j]=r[j];
  }
  q.resize(n);
  while(!q.empty() && !q.back()) q.pop_back();
  for (ll &i : p) i=(i*odw)%mod;
  return q;
}
vll div(vll a, vll b) {
  int s=(int)a.size()-(int)b.size()+1;
  if (s<=0) return {};
  reverse(a.begin(), a.end()); reverse(b.begin(), b.end());
  b=inv(b, s);
  a=multiply(a, b, true);
  a.resize(s);
  reverse(a.begin(), a.end());
  return a;
}
vll rem(vll a, vll b) {
  vll d=div(a, b);
  d=multiply(d, b, true);
  for (int i=0; i<(int)d.size(); i++) a[i]=(a[i]-d[i]+mod)%mod;
  while(!a.empty() && !a.back()) a.pop_back();
  return a;
}
void precalc(int v, vector<vll> &help, vll query) {
  if ((int)help.size()<=v) help.resize(v+1);
  int s=query.size();
  if (s==1) {
    help[v]={mod-query[0]%mod, 1};
    return;
  }
  vll a, b;
  for (int i=0; i<s/2; i++) a.push_back(query[i]);
  for (int i=s/2; i<s; i++) b.push_back(query[i]);
  precalc(v*2, help, a); precalc(v*2+1, help, b);
  help[v]=multiply(help[v*2], help[v*2+1], true);
}
```

```
void calc(int v, vll &res, vector<vll> &help, vll wek) {
  wek=rem(wek, help[v]);
  if ((int)help[v].size()==2) {
    res.push_back(wek.empty() ? 0 : wek[0]);
    return;
  }
  calc(v*2, res, help, wek);
  calc(v*2+1, res, help, wek);
}
vll multi_eva(vll wek, vll query) {
  vector<vll> help;
  precalc(1, help, query);
  vll res;
  calc(1, res, help, wek);
  return res;
}
```

## Math/Simplex.cpp

```
struct Simplex { // Maximize  $c \cdot x$  subject to  $Ax \leq b$ .
  using T = double; // Initialize the structure, set A, b, c and then run
  vector<vector<T>> A; // solve(). Max objective is stored in res. To recover
  vector<T> b, c; // the best result, use getVars().
  int V, E;
  vector<int> eqIds, varIds, cols;
  T res;
  static constexpr T kEps = 1e-9;
  Simplex(int vars, int eqs) : A(eqs, vector<T>(vars)), b(eqs), c(vars),
    V(vars), E(eqs), eqIds(eqs), varIds(vars), res(0) {
    iota(varIds.begin(), varIds.end(), 0);
    iota(eqIds.begin(), eqIds.end(), vars);
  }
  void pivot(int eq, int var) {
    T coef = 1 / A[eq][var];
    cols.clear();
    for (int i = 0; i < V; i++) {
      if (abs(A[eq][i]) > kEps) { cols.push_back(i); A[eq][i] *= coef; }
    }
    A[eq][var] *= coef; b[eq] *= coef;
    for (int row = 0; row < E; row++) {
      if (row == eq || abs(A[row][var]) < kEps) { continue; }
      T k = -A[row][var];
      A[row][var] = 0;
      for (int i : cols) { A[row][i] += k * A[eq][i]; }
      b[row] += k * b[eq];
    }
    T q = c[var]; c[var] = 0;
    for (int i : cols) { c[i] -= q * A[eq][i]; }
    res += q * b[eq];
    swap(varIds[var], eqIds[eq]);
  }
  bool solve() {
    while (true) {
      int eq = -1, var = -1;
      for (int i = 0; i < E; i++) { if (b[i] < -kEps) { eq = i; break; } }
      if (eq == -1) { break; }
      for (int i = 0; i < V; i++) { if (A[eq][i] < -kEps) { var = i; break; } }
      if (var == -1) { res = -1e9; return false; /* No solution */ }
      pivot(eq, var);
    }
    while (true) {
      int var = -1, eq = -1;
      for (int i = 0; i < V; i++) { if (c[i] > kEps) { var = i; break; } }
      if (var == -1) { break; }
      for (int i = 0; i < E; i++) {
        if (A[i][var] < kEps) { continue; }

```

```

    if (eq >= 0 && b[i] / A[i][var] >= b[eq] / A[eq][var]) { continue; }
    eq = i;
}
if (eq == -1) { res = 1e9; return false; /* Unbounded */ }
pivot(eq, var);
return true;
}
vector<T> getVars() { // Optimal assignment of variables.
vector<T> result(V);
for (int i = 0; i < E; i++) if (eqIds[i] < V) result[eqIds[i]] = b[i];
return result;
}
};

Math/Simpson.cpp
// Either run integral(A, B) once or split the interval [A, B] into up to ~1000
// smaller intervals -- if the function f behaves oddly or the interval is long.
ld simp(ld low, ld high, const ld * old, vector<ld> & nowe) {
const int n = 500; // n must be even!!! Try n = 2 and n = 10.
nowe.resize(n + 1);
ld total = 0, jump = (high - low) / n;
for (int i = 0; i <= n; ++i) {
int mul = i == 0 || i == n ? 1 : 2 + i % 2 * 2; // 1 4 2 4 2 ... 4 1
nowe[i] = !old || i % 2 ? f(low + i * jump) : old[i/2];
total += nowe[i] * mul; // uses a global function ld f(ld x)
}
return total * (high - low) / n / 3;
}
ld rec(ld low, ld high, ld prv, const vector<ld> & old) {
ld mid = (low + high) / 2;
vector<ld> left, right;
ld L = simp(low, mid, old.data(), left);
ld R = simp(mid, high, old.data() + old.size() / 2, right);
ld del = L + R - prv; // try without del/15
if (abs(del) < 15 * eps) return L + R + del / 15; // eps ~ required abs precision
return rec(low, mid, L, left) + rec(mid, high, R, right);
}
ld integral(ld low, ld high) {
vector<ld> old;
ld prv = simp(low, high, 0, old);
return rec(low, high, prv, old);
}
Math/Wzory.tex

```

## Math/Wzory.tex

$$\begin{aligned}
 \int \sqrt{x^2 + 1} dx &= \frac{1}{2} (x\sqrt{x^2 + 1} + \operatorname{arcsinh} x) + c & \int \sqrt{1 - x^2} dx &= \frac{1}{2} (x\sqrt{1 - x^2} + \arcsin x) + c \\
 \int \frac{1}{ax^2 + bx + c} dx &= \frac{2}{\sqrt{4ac - b^2}} \arctan \frac{2ax + b}{\sqrt{4ac - b^2}} \quad (\Delta < 0) \\
 \int \frac{x}{ax^2 + bx + c} dx &= \frac{1}{2a} \ln |ax^2 + bx + c| - \frac{b}{2a} \int \frac{dx}{ax^2 + bx + c} \\
 \int \tan x dx &= -\ln |\cos x| + c \\
 (\arcsin x)' &= \frac{1}{\sqrt{1 - x^2}}, & (\arccos x)' &= -\frac{1}{\sqrt{1 - x^2}} \\
 \operatorname{tgamma}(t) = \Gamma(t) &= \int_0^\infty e^{t-1} e^{-x} dx \\
 \frac{1}{\pi} &= 0.31831, & \pi^2 &= 9.86960, & \frac{1}{\pi^2} &= 0.10132, & \frac{1}{e} &= 0.36788, & \gamma &= 0.577215664901532 \\
 H_n &= \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + O(n^{-4}) \\
 \ln n! &= n \ln n - n + \frac{1}{2} \ln(2\pi n) + \frac{1}{12n} - \frac{1}{360n^3} + \frac{1}{1260n^5} - O(n^{-7})
 \end{aligned}$$

Jeśli  $n = 2^{a_0} \cdot p_1^{2 \cdot a_1} \cdot \dots \cdot p_r^{2 \cdot a_r} \cdot q_1^{b_1} \cdot \dots \cdot q_s^{b_s}$  i  $B = \prod (b_i + 1)$ , gdzie  $p_i$  to liczby postaci  $4 \cdot k + 3$ , a  $q_i$  to liczby postaci  $4 \cdot k + 1$ , to liczba sposobów na zapisanie  $n$  jako sumy dwóch kwadratów liczb naturalnych wynosi:

- 0, jeśli któraś z liczb  $a_i$  nie jest całkowita
- $\frac{B}{2}$ , jeśli  $B$  jest parzyste
- $\frac{B - (-1)^{a_0}}{2}$ , jeśli  $B$  jest nieparzyste

$$\begin{aligned}
 S_j &= \sum_{1 \leq i_1 < i_2 < \dots < i_j \leq n} |A_{i_1} \cap A_{i_2} \cap \dots \cap A_{i_j}| \\
 \sum_{j \geq k} \binom{j}{k} (-1)^{j+k} S_j &- \text{liczba elementów należących do dokładnie } k \text{ zbiorów} \\
 \sum_{j \geq k} \binom{j-1}{k-1} (-1)^{j+k} S_j &- \text{liczba elementów należących do co najmniej } k \text{ zbiorów.} \\
 \sum_{j \geq 1} (-2)^{j-1} S_j &- \text{liczba elementów należących do nieparzystej liczby zbiorów}
 \end{aligned}$$

Zaczynając w punkcie 0, idąc w każdym ruchu o  $-1$  z prawdopodobieństwem  $p$  i o  $+1$  z  $1 - p$ , kończąc w  $-A$  lub  $B$  prawdopodobieństwo dojścia do  $B$  wynosi  $\frac{1+r+r^2+\dots+r^{A-1}}{1+r+r^2+\dots+r^{A+B-1}} = \frac{1-r^A}{1-r^{A+B}} = 1 - \frac{1+r'+r'^2+\dots+r'^{B-1}}{1+r'+r'^2+\dots+r'^{A+B-1}} = 1 - \frac{1-r'^B}{1-r'^{A+B}}$ , (\*działa o ile  $p \neq \frac{1}{2}$ ) gdzie  $r = \frac{p}{1-p}$ ,  $r' = \frac{1}{r}$ , dla  $p = \frac{1}{2}$  równe  $\frac{A}{A+B}$ .

Dla  $x_1, x_2, \dots, x_n$  macierz  $a_{i,j} = x_i^{j-1}$  ma wyznacznik  $\prod_{i < j} (x_j - x_i)$  i odwrotność

$$b_{i,j} = (-1)^i \frac{\sum_{A \subset [n] \setminus \{j\}, |A|=n-i} \prod_{k \in A} x_k}{\prod_{k \neq j} (x_k - x_j)}$$

## Miscellaneous/De\_Bruijn.cpp

```

//Generuje słowo cykliczne, Działa dla alph >= 2
vector<int> de_bruijn(int len, int alph){
vector<int> res, lyn{0};

```

```

while (lyn[0] != alph - 1){
    int r = lyn.size();
    if (len % r == 0)
        for (int c : lyn)
            res.push_back(c);
    for (int i = r; i <= len - 1; ++i)
        lyn.push_back(lyn[i - r]);
    while (lyn.back() == alph - 1)
        lyn.pop_back();
    lyn.back()++;
}
res.push_back(alph - 1);
return res;
}

```

#### Number\_Theory/Cornacchia.cpp

```

using T = int; // Typ, w którym mieści się modulo.
using T2 = long long; // Typ, w którym mieści się kwadrat modulo.
struct Pierwiastek { // Dla pierwszego p > 2, szuka pierwiastków modulo p.
    Pierwiastek(T p) : s(0), p(p), m(p - 1) { //template <int p> jeśli stałe p
        assert(p > 2);
        while (m % 2 == 0) { m /= 2; s++; }
        c = 2;
        while (Pot(c, p / 2) == 1) c = rand() % (p - 1) + 1; //rand() musi być duży
    }
    Pot(T a, T pot) const {
        T r = 1;
        while (pot) {
            if (pot & 1) r = r * (T2) a % p;
            a = a * (T2) a % p;
            pot >>= 1;
        }
        return r;
    }
    // Zwraca a**pot % p.
    T Licz(T a) const { // Znajduje pierwiastek z a modulo p.
        if (a == 0) return 0; // Sprawdza, czy a jest resztą
        if (Pot(a, p / 2) != 1) return -1; // kwadratową. Jeśli nie, zwraca -1.
        T z = Pot(c, m);
        T v = Pot(a, m / 2);
        T u = (T2) v * v % p;
        v = (T2) v * a % p;
        u = (T2) u * a % p;
        for (int i = s - 1; i >= 1; i--) {
            if (Pot(u, T(1) << (i - 1)) != 1) {
                u = (T2) u * z % p;
                u = (T2) u * z % p;
                v = (T2) v * z % p;
            }
            z = (T2) z * z % p;
        }
        return v; // Pierwiastkami liczby a są: {v, (p - v) % p}.
    }
private: int s;
        T p, m, c;
};
template<class num_t>
inline num_t isqrt(num_t k) {
    num_t r = sqrt(k) + 1;
    while (r * r > k) r--;
    return r;
}
pii cornacchia(int p, int d) { //dla p pierwszego, znajduje a, b: p = a^2 + db^2
    int x0 = Pierwiastek(p).Licz(p - d);
    if (x0 == -1) return {-1, -1};
    x0 = max(x0, p - x0);
    int a = p, b = x0;
}

```

```

int l = isqrt(p);
while (b > l) {
    int r = a % b;
    a = b;
    b = r;
}
int c = p - b * b;
if (c % d) return {-1, -1};
c /= d;
int cc = isqrt(c);
if (cc * cc != c) return {-1, -1};
return {b, cc};
}

```

#### Number\_Theory/CRT.cpp

```

void eukl(ll &x, ll &y, ll a, ll b) {
    if (!a) { x = 0; y = 1; return; }
    eukl(y, x, b % a, a);
    x -= y * (b / a);
}
ll mno(ll a, ll b, ll mod) { /*...*/ }
pll crt2(ll p, ll a, ll q, ll b) { //x==p(mod a), x==q(mod b)
    if (a == -1) return {-1, -1}; // Zarówno wynik, jak i argumenty są
    ll x, y; // postaci x = first (mod second).
    eukl(x, y, a, b); // Jeśli kongruencja jest niespełnialna
    ll nwd = x * a + y * b; // to zwraca (-1, -1).
    if ((p % nwd) != (q % nwd))
        return {-1, -1};
    a /= nwd; b /= nwd;
    ll nww = a * b;
    ll ret = mno(x * a, q / nwd, nww) + mno(y * b, p / nwd, nww);
    if ((ret % nww) < 0) ret += nww;
    return {ret * nwd + (p % nwd), nww * nwd};
}

```

#### Number\_Theory/Euklides.cpp

```

ll trzesienie(ll a, ll b, ll c) {
    if (c < 0) return 0;
    if (a > b) swap(a, b);
    ll p = c / b;
    ll k = b / a;
    ll d = (c - p * b) / a;
    return trzesienie(b - k * a, a, c - a * (k * p + d + 1)) + (p + 1) * (d + 1) + k * p * (p + 1) / 2;
} //counts pairs of nonnegative integers (x, y) such that ax + by <= c

```

#### Number\_Theory/Fibonacci Cycle.txt

Znaleziony cykl niekoniecznie jest najmniejszym cyklem, wiadomo jednak, że cykl(m) <= 6m  
 Dla m, które nie są postaci 2 \* 5^r, zachodzi ograniczenie: cykl(m) <= 4m.  
 Jeśli nie zrobi się Nww, to wiadomo, że cykl(m) <= m \* 2\*\*(1 + #dz.pierw.m).  
 cycle(2) = 3, cycle(5) = 20, cycle(p == 1 or 9 mod 10) = p - 1  
 cycle(p == 3 or 7 mod 10) = 2(p+1) cycle(p^k) = p^{k-1} \* cycle(k)  
 cycle(product p\_i^a\_i) = lcm(cycle(p\_i^a\_i))

#### Number\_Theory/Phi.cpp

```

struct Coprimes {
    vector<ll> w, dp;
    int gdz(ll v) {
        if (v <= w.back() / v)
            return v - 1;
        return w.size() - w.back() / v;
    }
    ll phi(ll n) {
        for (ll i = 1; i * i <= n; i++) {
            w.push_back(i);
            if ((n / i) != i)
                w.push_back(n / i);
        }
        sort(w.begin(), w.end());
        for (ll i : w) {

```

```

    dp.push_back(i*(i+1)/2);
    for (ll j=1; j*j<=i; j++) {
        if (i>1) dp.back()-=dp[gdz(j)]*(i/j-i/(j+1));
        if (j>1 && j<=i/(j+1)) dp.back()-=dp[gdz(i/j)];
    }
    return dp.back();
}
ll ask(ll v) { return dp[gdz(v)]; } //v==n/u for some u
};

```

#### Number\_Theory/Pi.cpp

```

struct Primes {
    vector<ll> w, dp;
    int gdz(ll v) {
        if (v<=w.back()/v)
            return v-1;
        return w.size()-w.back()/v;
    }
    ll pi(ll n) {
        for (ll i=1; i*i<=n; i++) {
            w.push_back(i);
            if ((n/i)!=i)
                w.push_back(n/i);
        }
        sort(w.begin(), w.end());
        for (ll i : w)
            dp.push_back(i-1);
        for (ll i=1; (i+1)*(i+1)<=n; i++) {
            if (dp[i]==dp[i-1]) continue;
            for (int j=(int)w.size()-1; w[j]>=(i+1)*(i+1); j--)
                dp[j]-=dp[gdz(w[j]/(i+1))]-dp[i-1];
        }
        return dp.back();
    }
    ll ask(ll v) { return dp[gdz(v)]; } //v==n/u for some u
};

```

#### Number\_Theory/Rho.cpp

```

// szybkie, faktoryzuje liczby z przedziału [1e18, 1e18 + 1e5] w 9s na cfie
namespace rho {
    const int maxv = 50;
    const int maxp = 1e6 + 7; // preprocesujemy rozkład do maxp - 1
    int ptot;
    int d[maxp];
    int pr[maxp];
    vector<ll> ans; // rozkład, czyszcimy zazwyczaj po kazdym wywołaniu decompose()
    inline ll mod(ll a, ll n) {
        if (a >= n) a -= n;
        return a;
    }
    inline ll add(ll a, ll b, ll n) {
        a += b;
        mod(a, n);
        return a;
    }
    inline ll mul(ll x, ll y, ll p) { // uwaga, ta funkcja rzuca overflow, tak ma byc
        ll ret = x * y - (ll)((ld)x * y / p + 0.5) * p;
        return ret < 0 ? ret + p : ret;
    }
    ll fast(ll x, ll k, ll p) {
        ll ret = 1 % p;
        for (; k > 0; k >>= 1, x = mul(x, x, p)) (k & 1) && (ret = mul(ret, x, p));
        return ret;
    }
    bool rabin(ll n) {
        if (n == 2) return 1;

```

```

    if (n < 2 || !(n & 1)) return 0;
    ll s = 0, r = n - 1;
    for (; !(r & 1); r >>= 1, ++s)
        ;
    for (int i = 0; pr[i] < n && pr[i] < maxv; ++i) {
        ll cur = fast(pr[i], r, n), nxt;
        for (int j = 0; j < s; ++j) {
            nxt = mul(cur, cur, n);
            if (nxt == 1 && cur != 1 && cur != n - 1) return false;
            cur = nxt;
        }
        if (cur != 1) return false;
    }
    return true;
}
ll factor(ll n) {
    static ll seq[maxp];
    while (true) {
        ll x = rand() % n, y = x, c = rand() % n;
        ll *px = seq, *py = seq, tim = 0, prd = 1;
        while (true) {
            *py++ = y = add(mul(y, y, n), c, n);
            *py++ = y = add(mul(y, y, n), c, n);
            if ((x = *px++) == y) break;
            ll tmp = prd;
            prd = mul(prd, abs(y - x), n);
            if (!prd) return gcd(tmp, n);
            if ((++tim) == maxv) {
                if ((prd = gcd(prd, n)) > 1 && prd < n) return prd;
                tim = 0;
            }
        }
        if (tim && (prd = gcd(prd, n)) > 1 && prd < n) return prd;
    }
}

```

```

void decompose(ll n) { // glowna funkcja

```

```

    if (n < maxp) {
        while (n > 1) ans.pb(d[n]), n /= d[n];
    } else if (rabin(n))
        ans.pb(n);
    else {
        ll fact = factor(n);
        decompose(fact), decompose(n / fact);
    }
}

```

```

void init() { // wywołujemy przed pierwsza faktoryzacja, raz na caly program

```

```

    d[1] = 1;
    for (int i = 2; i * i < maxp; ++i)
        if (!d[i])
            for (int j = i * i; j < maxp; j += i) d[j] = i;
    for (int i = 2; i < maxp; ++i)
        if (!d[i]) {
            d[i] = i;
            pr[ptot++] = i;
        }
}

```

```

} // namespace rho

```

#### Strings/Aho.cpp

```

const int MAXN = 404, sigma = 26;
int term[MAXN], len[MAXN], to[MAXN][sigma], link[MAXN], sz = 1;
void add_str(string s) {
    int cur = 0;
    for(auto c: s) {
        if(!to[cur][c - 'a']) {
            to[cur][c - 'a'] = sz++;

```



```

    len[to[cur][c - 'a']] = len[cur] + 1;
}
cur = to[cur][c - 'a'];
}
term[cur] = cur;
}
void push_links() {
    int que[sz];
    int st = 0, fi = 1;
    que[0] = 0;
    while(st < fi) {
        int V = que[st++];
        int U = link[V];
        if(!term[V]) term[V] = term[U];
        for(int c = 0; c < sigma; c++)
            if(to[V][c]) {
                link[to[V][c]] = V ? to[U][c] : 0;
                que[fi++] = to[V][c];
            }
        else to[V][c] = to[U][c];
    }
}

Strings/Eertree.cpp

const int maxn=1000*1000+7, alfa=26;
int len[maxn], link[maxn], to[maxn][alfa], slink[maxn], diff[maxn];
int ans[maxn], z[maxn], sz, last, n;
pair <int,int> series_ans[maxn];
char s[maxn];
void init() {
    s[n++] = len[1] = -1;
    link[0] = 1;
    sz = 2;
}
int get_link(int v) {
    while(s[n-len[v]-2] != s[n-1]) v = link[v];
    return v;
}
void add_letter(char c) {
    s[n++] = c - 'a';
    last = get_link(last);
    if(!to[last][c]) {
        len[sz] = len[last] + 2;
        link[sz] = to[get_link(link[last])][c];
        diff[sz] = len[sz] - len[link[sz]];
        slink[sz] = (diff[sz] == diff[link[sz]] ? slink[link[sz]] : link[sz]);
        to[last][c] = sz++;
    }
    last = to[last][c];
}
int main() {
    init();
    for(int i=1; i<=nn; i++) {
        add_letter(tekst[i]);
        for(int v=last; len[v]>0; v=slink[v]) {
            series_ans[v] = {ans[i - (len[slink[v]] + diff[v])], i - (len[slink[v]] + diff[v])};
            if(diff[v] == diff[link[v]])
                series_ans[v] = min(series_ans[v], series_ans[link[v]]);
            if (!(i&1)) {
                if (series_ans[v].first+1<ans[i]) {
                    ans[i] = series_ans[v].first + 1;
                    z[i] = series_ans[v].second;
                }
            }
        }
    }
    if (!(i&1) && tekst[i]==tekst[i-1] && ans[i-2]<ans[i]) {

```

```

        ans[i] = min(ans[i], ans[i-2]);
        z[i] = i-2;
    }
}

Strings/Graf_Podsłow.cpp

//TODO: some comments on what those functions are
struct suffix_automaton {
    vector<map<char,int>> edges;
    vector<int> link, length;
    int last;
    suffix_automaton(string s) {
        edges.push_back(map<char,int>());
        link.push_back(-1);
        length.push_back(0);
        last = 0;
        for (int i=0; i<(int)s.size(); i++) {
            edges.push_back(map<char,int>());
            length.push_back(i+1);
            link.push_back(0);
            int r = edges.size() - 1;
            int p = last;
            while (p >= 0 && !edges[p].count(s[i])) {
                edges[p][s[i]] = r;
                p = link[p];
            }
            if (p != -1) {
                int q = edges[p][s[i]];
                if (length[p] + 1 == length[q]) {
                    link[r] = q;
                }
                else {
                    edges.push_back(edges[q]);
                    length.push_back(length[p] + 1);
                    link.push_back(link[q]);
                    int qq = edges.size()-1;
                    link[q] = link[r] = qq;
                    while (p >= 0 && edges[p][s[i]] == q) {
                        edges[p][s[i]] = qq;
                        p = link[p];
                    }
                }
            }
            last = r;
        }
    }
};

Strings/Lyndon.cpp

// 1) Przyjmuje słowo s (wypełnione na pozycjach 0, 1, ..., n-1).
// Dzieli słowo s na pewną liczbę słów Lyndona p_1, ... p_k tak, że:
// p_1 >= p_2 >= ... >= p_k (leksykograficznie)
// Podział jest zapisywany w tablicy b - na i-tej pozycji jest true,
// jeśli nastąpiło cięcie przed i-tą literką.
// 2) Znajduje minimalne leksykograficznie przesunięcie cykliczne słowa.
// 3) Znajduje minimalny leksykograficznie sufiks słowa.
void lyndon(char * s, // Słowo zaczynające się na pozycji 0:
             // 2) s powinno być sklejone: xx.
             int n, // Długość słowa s (licząc ew. podwojenie).
             int& suf, // 3) pozycja minimalnego leksykograficznie sufiksu.
             int& cyk, // 2) pozycja minimalnego leksykograficznie przes. cykl.
             bool* b) { // Tablica cięcia b.
    for (int i = 0; i < n; i++) b[i] = false; // wykomentuj, jeśli nie 1)
    int p = 0, k = 0, m = 1;
    while (p < n) {
        if (m == n or s[m] < s[k]) {

```



```

if (p < n / 2) cyk = p; // wykomentuj, jeśli nie 2)
while (p <= k) {
    p += m - k;
    if (p < n) {
        suf = p; // wykomentuj, jeśli nie 3)
        b[p] = true; // wykomentuj, jeśli nie 1)
    }
}
m = (k = p) + 1;
} else if (s[m++] != s[k++]) k = p;
}
}

```

#### Strings/Manacher.cpp

```

// @s[0..n-1] - napis długości @n.
// @r[0..2n-2] - tablica promieni palindromów.
// s: a b a a b a a c a a b b b b a a c a c
// r: 0 0 1 0 0 3 0 0 2 0 0 1 0 0 3 0 0 1 0 0 0 1 1 6 1 1 0 0 0 1 0 0 1 0 1 0 0
void Manacher(const char* s, int n, int* r) {
    for (int i = 0, m = 0, k = 0, p = 0; i < 2 * n - 1; m = i++ - 1) {
        while (p < k and i / 2 + r[m] != k)
            r[i++] = min(r[m--], (k + 1 - p++) / 2);
        while (k + 1 < n and p > 0 and s[k + 1] == s[p - 1])
            k++, p--;
        r[i] = (k + 1 - p++) / 2;
    }
}
bool is_pal(int p, int k) { //Przedział [p, k] obustronnie domknięty
    return r[p + k] >= (k - p + 1) / 2;
}

```

#### Strings/Pref.cpp

```

void Pref(const char* s, int n, int* p) {
    p[0] = n;
    int i = 1, m = 0;
    while (i < n) {
        while (m + i < n and s[m + i] == s[m]) m++;
        p[i++] = m;
        m = max(m - 1, 0);
        for (int k = 1; p[k] < m; m--) p[i++] = p[k++];
    }
}

```

#### Strings/Suf.cpp

```

void Sufar(const int* s, int n, int alpha, int* sa, int* lcp = nullptr) {
    if (n > 0) sa[0] = 0;
    if (n <= 1) return;
    VI roz(alpha + 1), wsk(alpha), typ(n + 1), ids(n, -1), news, pos;
    auto star = [&](int i) -> bool { return typ[i] == 3; };
    auto Indukuj = [&]() -> void {
        copy(roz.begin(), roz.end() - 1, wsk.begin());
        sa[wsk[s[n - 1]]++] = n - 1;
        REP(i, n)
            if (sa[i] > 0 && !typ[sa[i] - 1])
                sa[wsk[s[sa[i] - 1]]++] = sa[i] - 1;
        copy(roz.begin() + 1, roz.end(), wsk.begin());
        FORD(i, n - 1, 0)
            if (sa[i] > 0 && typ[sa[i] - 1])
                sa[--wsk[s[sa[i] - 1]]] = sa[i] - 1;
    };
    typ[n] = 3;
    FORD(i, n - 1, 0) {
        sa[i] = -1;
        roz[s[i] + 1]++;
        if (i != n - 1 && s[i] < s[i + 1] + !typ[i + 1]) {
            typ[i] = 1;
        }
        else if (typ[i + 1]) {

```

```

        typ[i + 1] = 3;
    } }
    partial_sum(ALL(roz), roz.begin());
    copy(roz.begin() + 1, roz.end(), wsk.begin());
    REP(i, n) if (star(i)) sa[--wsk[s[i]]] = i;
    Indukuj();
    int nast_id = 0, b = -1;
    REP(i, n) {
        int a = sa[i];
        if (!star(a)) continue;
        if (b >= 0) while (a == sa[i] || !star(a) || !star(b)) {
            if (star(a) != star(b) || s[a++] != s[b++]) {
                nast_id++;
                break;
            }
        }
        ids[b = sa[i]] = nast_id;
    }
    for (int i = 0; i < n; i++) {
        if (ids[i] == -1) continue;
        news.PB(ids[i]);
        pos.PB(i);
    }
    VI new_sa(SIZ(news));
    Sufar(news.data(), SIZ(news), nast_id + 1, new_sa.data());
    fill(sa, sa + n, -1);
    copy(roz.begin() + 1, roz.end(), wsk.begin());
    reverse(ALL(new_sa));
    for (int j : new_sa) sa[--wsk[s[pos[j]]]] = pos[j];
    Indukuj();
    if (lcp) {
        REP(i, n) ids[sa[i]] = i;
        for (int i = 0, k = 0; i < n; i++, k = max(0, k - 1)) {
            if (ids[i] == n - 1) { k = 0; continue; }
            const int j = sa[ids[i] + 1];
            while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
            lcp[ids[i]] = k;
        }
    }
}

```

#### Strings/Tablica\_Sufiksowa\_Log.cpp

```

/* TABLICA SUFIKSOWA O(n*log(n)) i O(n*log^2(n)). */
/* Sortuje leksykograficznie wszystkie sufiksy słowa. */
/* Dodatkowo liczy pomocnicze tablice rank i lcp. */
/* Wszystko indeksujemy od 1, ale podajemy wskaźnik na 0 */

```

```

struct suffix_array {
    vector<pair<pair<int,int>, int>> wek; // do log^2
    vector<int> ran; // do log
    vector<int> ile; // do log
    vector<int> kol;

    //n to limit na długość słowa
    suffix_array(int n) {
        wek.resize(n + 1, make_pair(make_pair(-1, -1), -1));
        ran.resize(2 * n + 1);
        ile.resize(n + 2);
        kol.resize(n + 2);
    }

    void sa_log_2(char *tab, int n, int *sa, int *ran, int *lcp) {
        int l = 0;
        for (int i = 1; i <= n; i++)
            ran[i] = tab[i];
        for (int h = 1; h <= n; h *= 2) {
            for (int i = 1; i <= n; i++)
                wek[i].first.first = ran[i];
            for (int i = 1; i + h <= n; i++)

```

```

    wek[i].first.second = ran[i + h];
    for (int i = n - h + 1; i <= n; i++)
        wek[i].first.second = 0;
    for (int i = 1; i <= n; i++)
        wek[i].second = i;
    sort(wek.begin() + 1, wek.begin() + 1 + n);
    le = 0;
    for (int i = 1; i <= n; i++) {
        if (wek[i].first != wek[i - 1].first)
            le++;
        ran[wek[i].second] = le;
    }
}

for (int i = 1; i <= n; i++)
    sa[ran[i]] = i;
le = 0;
for (int i = 1; i <= n; i++) {
    le = max(0, le - 1);
    if (ran[i] == n) {
        lcp[n] = 0;
        continue;
    }
    while (tab[i + le] == tab[sa[ran[i] + 1] + le])
        le++;
    lcp[ran[i]] = le;
}

}

inline void bucketSort(int *val, int *tab, int n) {
    for (int i = 1; i <= n; ++i) {
        ile[1 + val[i]]++;
        kol[i] = tab[i];
    }
    for (int i = 1; i <= n; ++i) ile[i] += ile[i - 1];
    for (int i = 1; i <= n; ++i) {
        tab[++ile[val[kol[i]]]] = kol[i];
    }
    fill(ile.begin(), ile.end(), 0);
}

void sa_log(char *tab, int n, int *sa, int *rank, int *lcp) {
    vector<int> num(256);
    for (int i = 1; i <= n; ++i) num[(int) tab[i]] = 1;
    for (int i = 1; i < 256; ++i) num[i] += num[i - 1];
    for (int i = 1; i <= n; ++i) ran[i] = num[(int) tab[i]], sa[i] = i;

    for (int len = 1; len <= n; len *= 2) {
        bucketSort(ran.data() + len, sa, n);
        bucketSort(ran.data(), sa, n);

        int nval = 0;
        for (int i = 1; i <= n; ++i) {
            rank[sa[i]] = nval += (i == 1 ||
                ran[sa[i]] != ran[sa[i - 1]] ||
                ran[sa[i] + len] != ran[sa[i - 1] + len]);
        }
        for (int i = 1; i <= n; ++i) ran[i] = rank[i];
        if (nval == n) break;
    }

    int le = 0;
    for (int i = 1; i <= n; i++) {
        le = max(0, le - 1);
        if (rank[i] == n) {
            lcp[n] = 0;

```

```

        continue;
    }
    while (i + le <= n && tab[i + le] == tab[sa[rank[i] + 1] + le]) le++;
    lcp[rank[i]] = le;
}
};

Strings/Ukkonen.cpp
template <typename Char> struct Ukkonen {
    // Musi być ściśle większe niż jakakolwiek długość słowa.
    static constexpr int kInfinity = numeric_limits<int>::max();
    struct Node {
        map<Char, pair<Node*, pair<int, int>>> transition;
        Node* suflink;
    };
    // Ta metoda jest wywoływana zawsze gdy tworzona jest krawędź {node}[a, +oo).
    void CreateLeafCallback(Node* node, int a) {
        // printf("CreateLeafCallback({%p}{%d, +oo})\n", node, a);
    }
    // Ta metoda jest wywoływana zawsze gdy krawędź {node}[a, b] zamienia się
    // w dwie krawędzie: {node}[a, c-1], {middle}[c, b].
    void SplitEdgeCallback(Node* node, int a, int b, Node* middle, int c) {
        // printf("SplitEdgeCallback({%p}{%d, %d} -> {%p}{%d, %d} + {%p}{%d, %d})\n",
        //         node, a, b, node, a, c - 1, middle, c, b);
    }
    // vector<unique_ptr<Node>> nodes_to_delete; // Odkomentować w celu usuwania.
    Node* NewNode() {
        Node* node = new Node();
        // nodes_to_delete.insert(node); // Odkomentować w celu usuwania.
        return node;
    }
    vector<Char> text; // Słowo powinno zajmować indeksy [0..n-1].
    Node* root;
    Node* pin;
    Node* last_explicit_node; // Ostatni wierzchołek „explicit”.
    int last_length; // Liczba literek do ostatniego wierzchołka „implicit”.
    // "reserve" warto ustawić na maksymalną długość słowa, ale wcale nie trzeba.
    Ukkonen(const int reserve = 0) : root(nullptr), pin(nullptr) {
        text.reserve(reserve);
        root = NewNode();
        pin = NewNode();
        root->suflink = pin;
        last_explicit_node = root;
        last_length = 0;
    }
    void Canonize(Node** s, int* a, int b) {
        if (b < *a) return;
        pair<Node*, pair<int, int>> t = (*s)->transition[text[*a]];
        Node* sp = t.first;
        int ap = t.second.first;
        int bp = t.second.second;
        while (bp - ap <= b - *a) {
            *a = *a + bp - ap + 1;
            *s = sp;
            if (*a <= b) {
                t = (*s)->transition[text[*a]];
                sp = t.first;
                ap = t.second.first;
                bp = t.second.second;
            }
        }
    }
    bool TestAndSplit(Node* s, int a, int b, Char c, Node** ret) {
        if (a <= b) {
            pair<Node*, pair<int, int>>& t = s->transition[text[a]];

```

```

Node* sp = t.first;
int ap = t.second.first;
int bp = t.second.second;
if (c == text[ap + b - a + 1]) {
    *ret = s;
    return true;
}
*ret = NewNode();
t.second.second = ap + b - a;
t.first = *ret;
(*ret)->transition[text[ap + b - a + 1]] =
    make_pair(sp, make_pair(ap + b - a + 1, bp));
SplitEdgeCallback(s, ap, bp, *ret, ap + b - a + 1);
return false;
}
*ret = s;
return s->transition.find(c) != s->transition.end();
}
void Update(Node** s, int* a, int i) {
    Node* oldr = root;
    Node* r;
    bool end = TestAndSplit(*s, *a, i - 1, text[i], &r);
    while (!end) {
        CreateLeafCallback(r, i);
        r->transition[text[i]] = make_pair(nullptr, make_pair(i, kInfinity));
        if (oldr != root) oldr->suflink = r;
        oldr = r;
        *s = (*s)->suflink;
        Canonize(s, a, i - 1);
        end = TestAndSplit(*s, *a, i - 1, text[i], &r);
    }
    if (oldr != root) oldr->suflink = *s;
}
// Dodaje kolejną literę do drzewa.
void AddLetter(Char z) {
    const int i = static_cast<int>(text.size());
    text.push_back(z);
    auto it = pin->transition.find(z);
    if (it == pin->transition.end())
        pin->transition[z] = make_pair(root, make_pair(i, i));
    Update(&last_explicit_node, &last_length, i);
    Canonize(&last_explicit_node, &last_length, i);
}
// Zamienia wszystkie krawędzie: [x, +oo) -> [x, text.size()-1].
void ClearInfinities(Node* node = nullptr) {
    if (node == nullptr) node = root;
    for (auto& it : node->transition) {
        if (it.second.second == kInfinity)
            it.second.second = static_cast<int>(text.size()) - 1;
        else ClearInfinities(it.second.first);
    }
}
};
template <typename Char> constexpr int Ukkonen<Char>::kInfinity;
int main() { // Przykład użycia.
    string s = "abcdefgh#";
    Ukkonen<char> u(s.size() /* reserve */);
    for (char c : s) u.AddLetter(c);
    u.ClearInfinities();
}

```