```
# A valid snippet should starts with:
#
#            snippet trigger_word [ "description" [ options ] ]
#
# and end with:
#
#            endsnippet
#
# Snippet options:
#
#            b - Beginning of line.
#            i - In-word expansion.
#            w - Word boundary.
#            r - Regular expression
#            e - Custom context snippet
#            A - Snippet will be triggered automatically, when condition
matches.
#
# Basic example:
#
#            snippet emitter "emitter properties" b
#            private readonly ${1} = new Emitter<$2>()
#            public readonly ${1/^_(.*)/$1/}: Event<$2> = this.$1.event
#            endsnippet
#
# Online reference:
https://github.com/SirVer/ultisnips/blob/master/doc/UltiSnips.txt

snippet precode
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

#ifdef LOCAL
#include "debug.cpp"
#else
#define dbg(...)
#endif

using namespace std;
using namespace __gnu_pbds;
template <class T>
using ordered_set = tree<T, null_type, less_equal<T>, rb_tree_tag,
tree_order_statistics_node_update>;

#define pb push_back
#define sz(a) ((int)(a).size())
#define ff first
#define ss second
#define all(a) (a).begin(), (a).end()
```

```
#define allr(a) (a).rbegin(), (a).rend()
#define approx(a) fixed << setprecision(a)

template <class T> using pq = priority_queue<T>;
template <class T> using pqg = priority_queue<T, vector<T>, greater<T>>;
template <class T> void ckmin(T &a, const T &b) { a = min(a, b); }
template <class T> void ckmax(T &a, const T &b) { a = max(a, b); }

template <class T> void read(vector<T> &v);
template <class F, class S> void read(pair<F, S> &p);
template <class T, size_t Z> void read(array<T, Z> &a);
template <class T> void read(T &x) {cin >> x;}
template <class R, class... T> void read(R& r, T&... t){read(r); read(t...);};
template <class T> void read(vector<T> &v) {for(auto& x : v) read(x);}
template <class F, class S> void read(pair<F, S> &p) {read(p.ff, p.ss);}
template <class T, size_t Z> void read(array<T, Z> &a) { for(auto &x : a)
read(x); }

template <class F, class S> void pr(const pair<F, S> &x);
template <class T> void pr(const T &x) {cout << x;}
template <class R, class... T> void pr(const R& r, const T&... t) {pr(r);
pr(t...);}
template <class F, class S> void pr(const pair<F, S> &x) {pr("{", x.ff, ", ",
x.ss, "}\n");}
void ps() {pr("\n");}
template <class T> void ps(const T &x) {pr(x); ps();}
template <class T> void ps(vector<T> &v) {for(auto& x : v) pr(x, ' '); ps();}
template <class T, size_t Z> void ps(const array<T, Z> &a) { for(auto &x : a)
pr(x, ' '); ps(); }
template <class F, class S> void ps(const pair<F, S> &x) {pr(x.ff, ' ', x.ss);
ps();}
template <class R, class... T> void ps(const R& r,  const T &...t) {pr(r, ' ');
ps(t...);}

void solve(){
	${1}
}

int main(){
	ios::sync_with_stdio(0);
	cin.tie(0);
	int t = 1;
	cin >> t;
	while(t--){
		solve();
	}
}
endsnippet

snippet simple_precode
#include <bits/stdc++.h>
```

```
#ifdef LOCAL
#include "debug.cpp"
#else
#define dbg(...)
#endif

using namespace std;

void solve(){
        ${1}
}

int main(){
        ios::sync_with_stdio(0);
        cin.tie(0);
        int t = 1;
        cin >> t;
        while(t--){
                solve();
        }
}
endsnippet

snippet generator
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

#ifdef LOCAL
#include "debug.cpp"
#else
#define dbg(...)
#endif

using namespace std;
using namespace __gnu_pbds;
template <class T>
using ordered_set = tree<T, null_type, less_equal<T>, rb_tree_tag,
tree_order_statistics_node_update>;

#define pb push_back
#define sz(a) ((int)(a).size())
#define ff first
#define ss second
#define all(a) (a).begin(), (a).end()
#define allr(a) (a).rbegin(), (a).rend()
#define approx(a) fixed << setprecision(a)

template <class T> using pq = priority_queue<T>;
template <class T> using pqg = priority_queue<T, vector<T>, greater<T>>;
template <class T> void ckmin(T &a, const T &b) { a = min(a, b); }
template <class T> void ckmax(T &a, const T &b) { a = max(a, b); }

template <class T> void read(vector<T> &v);
template <class F, class S> void read(pair<F, S> &p);
template <class T, size_t Z> void read(array<T, Z> &a);
template <class T> void read(T &x) {cin >> x;}
template <class R, class... T> void read(R& r, T&... t){read(r); read(t...);};
template <class T> void read(vector<T> &v) {for(auto& x : v) read(x);}
template <class F, class S> void read(pair<F, S> &p) {read(p.ff, p.ss);}
template <class T, size_t Z> void read(array<T, Z> &a) { for(auto &x : a)
read(x); }

template <class F, class S> void pr(const pair<F, S> &x);
template <class T> void pr(const T &x) {cout << x;}
template <class R, class... T> void pr(const R& r, const T&... t) {pr(r);
pr(t...);}
template <class F, class S> void pr(const pair<F, S> &x) {pr("{", x.ff, ", ",
x.ss, "}\n");}
void ps() {pr("\n");}
template <class T> void ps(const T &x) {pr(x); ps();}
template <class T> void ps(vector<T> &v) {for(auto& x : v) pr(x, ' '); ps();}
template <class T, size_t Z> void ps(const array<T, Z> &a) { for(auto &x : a)
pr(x, ' '); ps(); }
template <class F, class S> void ps(const pair<F, S> &x) {pr(x.ff, ' ', x.ss);
ps();}
template <class R, class... T> void ps(const R& r,  const T &...t) {pr(r, ' ');
ps(t...);}

mt19937_64 rng((long long)
chrono::steady_clock::now().time_since_epoch().count());

int _myrandomint(int x) { return rng() % x; }
long long _myrandomlong(long long x) { return rng() % x; }

int gint(int mn, int mx){
	assert(mn <= mx);
	return rng() % (mx - mn + 1) + mn;
}

long long glong(long long mn, long long mx){
	assert(mn <= mx);
	return rng() % (mx - mn + 1) + mn;
}

char gchar(char mn = 'a', char mx = 'z'){
	assert(mn <= mx);
	int pmn = int(mn - 'a');
	int pmx = int(mx - 'a');
	assert(0 <= pmn && pmn < 26);
	assert(0 <= pmx && pmx < 26);
```

```
    int x = gint(pmn, pmx);
    char c = char('a' + x);
    return c;
}

string gstring(int max_size, char mn = 'a', char mx = 'z'){
    int size = gint(1, max_size);
    string ret = "";
    for(int i = 0; i < size; i++) ret += gchar(mn, mx);
    return ret;
}

vector<int> gvectorint(int size, int mn = 0, int mx = 1000){
    assert(size > 0);
    vector<int> a(size);
    for(int i = 0; i < size; i++) a[i] = gint(mn, mx);
    return a;
}

vector<long long> gvectorlong(int size, long long mn = 0, long long mx =
1000000000000LL){
    assert(size > 0);
    vector<long long> a(size);
    for(int i = 0; i < size; i++) a[i] = glong(mn, mx);
    return a;
}

vector<int> gpermutation(int size, int start = 1){
    vector<int> p(size);
    iota(p.begin(), p.end(), start);
    return p;
}

template <typename T>
void randomize(vector<T>& a){
    vector<int> order = gpermutation((int) a.size(), 0);
    random_shuffle(order.begin(), order.end(), _myrandomint);
    vector<T> b = a;
    for(int i = 0; i < (int) a.size(); i++) b[i] = a[order[i]];
    a.swap(b);
}

vector<vector<int>> ggraph();

vector<vector<int>> gtree(int n){
    struct DSU{
        vector<int> p, size;
        DSU(int n = 1e5){
            p.resize(n + 1), size.resize(n + 1,1);
            for(int i = 1; i <= n; i++) p[i] = i;
```

```
			}

			int find(int x){
				if(p[x] != x) p[x] = find(p[x]);
				return p[x];
			}

			void merge(int x, int y){
				x = find(x), y = find(y);
				if(x == y) return;
				if(size[x] < size[y]) swap(x, y);
				size[x] += size[y];
				p[y] = x;
			}

			int get_size(int x) {return size[find(x)];}
		};
		vector<vector<int>> adj(n + 1);

		vector<pair<int, int>> alledges;
		for(int u = 1; u <= n; u++){
			for(int v = u + 1; v <= n; v++){
				alledges.push_back({u, v});
			}
		}
		randomize(alledges);

		DSU dsu(n);
		for(auto [u, v] : alledges){
			if(dsu.find(u) == dsu.find(v)) continue;
			dsu.merge(u, v);
			adj[u].push_back(v);
			adj[v].push_back(u);
		}
		return adj;
}

int main(){
		ios::sync_with_stdio(0);
		cin.tie(0);

		${1}
}
endsnippet

snippet divisor
${1}template <class T> vector<T> divisor(T x) {
		vector<T> ans;
		for(T i = 1; i * i <= x; i++)
			if(x % i == 0) {
				ans.push_back(i);
```

```
            if(i * i != x) ans.push_back(x / i);
        }
    return ans;
}
endsnippet

snippet DSU
${1}struct DSU{
    vector<int> p, size;
    DSU(int n = 1e5){
        p.resize(n + 1), size.resize(n + 1,1);
        for(int i = 1; i <= n; i++) p[i] = i;
    }

    int find(int x){
        if(p[x] != x) p[x] = find(p[x]);
        return p[x];
    }

    void merge(int x, int y){
        x = find(x), y = find(y);
        if(x == y) return;
        if(size[x] < size[y]) swap(x, y);
        size[x] += size[y];
        p[y] = x;
    }

    int get_size(int x) {return size[find(x)];}
};
endsnippet

snippet Fenwick
// Fenwick Tree sirve para hacer consultas en O(log n) en el intervalo [0..k].
// Divide n en log(n) intervalos usando potencias de 2. Ejemplo:
// array        = {1, 3, 4, 8, 6, 1, 4, 2}
// Fenwick Tree = {1, 4, 4, 16, 6, 7, 4, 29}
// sum(1, 7) = sum(1, 4) + sum(5, 6) + sum(7, 7)

// highest_bit(x) : retorna la m├íxima potencia de 2 <= x
// query(x) : Retorna la suma (u otra consulta) del intervalo [0..x]
// get_kth(x) : Retorna la posici├│n k tal que sum[0..(k - 1)] < x <= sum[0..k]
// 1-indexed
template<class T>
${1}struct fenwick_tree{
    vector<T> bit;
    int n;
    fenwick_tree(int _n){
        n = _n + 1;
        bit.resize(n, 0);
    }
```

```
        fenwick_tree(vector<T>& values){
                n = values.size() + 1;
                bit.resize(n, 0);
                for(int i = 1; i < n; i++) upd(i, values[i - 1]);
        }

        int highest_bit(unsigned x){
                return x == 0 ? -1 : 31 - __builtin_clz(x);
        }

        T query(int k){
                T ans = 0;
                for(; k > 0; k -= k & -k) ans += bit[k];
                return ans;
        }

        T query(int left, int right){
                return query(right) - query(left - 1);
        }

        void upd(int k, T add){
                assert(0 < k && k < n);
                for(; k < n; k += k & -k) bit[k] += add;
        }

        int get_kth(T value){
                int index = 0;
                for(int i = 1 << highest_bit(n - 1); i > 0; i /= 2){
                        if(index + i < n && bit[index + i] < value)
                                value -= bit[index += i];
                }
                assert(index < n - 1);
                return index + 1; // one-based indexing
        }
};
endsnippet

snippet LCA
vector<vector<int>> adj;

${1}struct LCA{
        int n, l, timer;
        vector<vector<int>> up;
        vector<int> in, out, depth;

        LCA(int _n, int root = 1){
                timer = 0, l = ceil(log2(_n));
                n = _n + 1;
                in.resize(n), out.resize(n);
                up.resize(n, vector<int>(l + 1));
                depth.resize(n);
```

```
		depth[root] = 0;
		dfs(root, root);
	}

	void dfs(int v, int p){
		depth[v] = depth[p] + 1;
		in[v] = ++timer;
		up[v][0] = p;
		for(int i = 1; i <= l; i++) up[v][i] = up[up[v][i - 1]][i - 1];
		for(int u : adj[v]){
			if(u == p) continue;
			dfs(u, v);
		}
		out[v] = ++timer;
	}

	bool is_ancestor(int u, int v){
		return in[u] <= in[v] && out[u] >= out[v];
	}

	int query(int u, int v){
		if(is_ancestor(u, v)) return u;
		if(is_ancestor(v, u)) return v;
		for(int i = l; ~i; i--){
			if(!is_ancestor(up[u][i], v)) u = up[u][i];
		}
		return up[u][0];
	}

	int ancestor(int x, int k){
		assert(depth[x] >= k);
		for(int i = 0; i <= l; i++){
			if(k & (1 << i)) x = up[x][i];
		}
		return x;
	}

	int distance(int u, int v){
		return depth[u] + depth[v] - 2 * depth[query(u, v)];
	}
};
endsnippet

snippet RMQ
const int MAX_VALUE = 1E9 + 100;

template<typename T>
${1}struct RMQ{
	int n = 0;
	vector<vector<T>> st;
	int levels;
```

```
    RMQ() {}

    RMQ(vector<T>& values){
        build(values);
    }

    int highest_bit(unsigned x){
        return x == 0 ? -1 : 31 - __builtin_clz(x);
    }

    void resize(int lev, T value = 0){
        st.resize(n, vector<T>(lev + 1, value));
    }

    void modify(int pos, int level, T value){
        st[pos][level] = value;
    }

    void upd(int pos, int level){
        st[pos][level] = op(st[pos][level - 1], st[pos + (1 << (level -
1))][level - 1]);
    }

    void upd_level(int level){
        for(int pos = 0; pos <= n - (1 << level); pos++){
            upd(pos, level);
        }
    }

    T op(T& a, T& b){
        return min(a, b);
    }

    void build(vector<T>& values){
        n = values.size();
        levels = ceil(log2(n));
        resize(levels);

        for(int pos = 0; pos < n; pos++) modify(pos, 0, values[pos]);
        for(int i = 1; i <= levels; i++) upd_level(i);

    }

    // range [l..r], 0 <= l <= r < n
    // not overlapping O(log n)
    T query(int l, int r){
        if(l > r) swap(l, r);
        assert(0 <= l && r < n);
        int level = highest_bit(n);
        T ans = MAX_VALUE;
        for(int i = level; ~i; i--){
            if((1 << i) <= r - l + 1){
                ans = op(ans, st[l][i]);
                l += (1 << i);
            }
        }
        return ans;
    }

    //overlapping O(1)
    T query_over(int l, int r){
        if(l > r) swap(l , r);
        assert(0 <= l && r < n);
        int level = highest_bit(r - l + 1);
        return op(st[l][level], st[r - (1 <<level) + 1][level]);
    }
};
endsnippet

snippet fast_pow_mod
long long fast_pow(long long a, long long p, long long mod = 1e9 + 7){
    long long res = 1;
    while(p){
        if(p % 2 == 0){
            a = a * a % mod;
            p >>= 1;
        }else{
            res = res * a % mod;
            p--;
        }
    }
    return res;
}
endsnippet

snippet fast_pow
long long fast_pow(long long a, long long p){
    long long res = 1;
    while(p){
        if(p % 2 == 0){
            a = a * a;
            p >>= 1;
        }else{
            res = res * a;
            p--;
        }
    }
    return res;
}
endsnippet
```

```
snippet prefix "pi[i] = j donde j es el mayor tal que s[0...j] = s[i-j+1 ... i] (prefijo = sufijo de [0...i])" b
${1}vector<int> prefix(string s) {
	int n = s.length();
	vector<int> pi(n);
	for (int i = 1; i < n; i++) {
		int j = pi[i - 1];
		while (j > 0 && s[i] != s[j]) j = pi[j - 1];
		if (s[i] == s[j]) j++;
		pi[i] = j;
	}
	return pi;
}
endsnippet

snippet TopologicalSort
//adj : directed graph, vertices < adj.size()
${1}vector<int> top_sort(vector<vector<int>>& adj){
	int n = adj.size();
	bool cycle = false;
	vector<int> sorted, color(n, 0);
	function<void(int)> dfs = [&](int u){
		color[u] = 1;
		for(int v : adj[u]){
			if(color[v] == 0 && !cycle) dfs(v);
			else if(color[v] == 1) cycle = true;
		}
		color[u] = 2;
		sorted.push_back(u);
	};
	for(int i = 1; i < n; i++){
		if(color[i] == 0 && !cycle) dfs(i);
	}
	if(cycle){return {};}
	reverse(sorted.begin(), sorted.end());
	return sorted;
}
endsnippet

snippet Point
template<typename T>
${1}struct Point{
	T x, y;
	Point() : x(0), y(0) {}
	Point(const T& x_, const T& y_) : x(x_), y(y_) {}

	friend ostream& operator << (ostream& os, const Point& p){ return os << '(' << p.x << ',' << p.y << ')'; }
	friend istream& operator >> (istream& is, Point& p){ return is >> p.x >> p.y; }
```

```
        Point& operator += (const Point& other){ x += other.x, y += other.y;
return *this; }
        Point& operator -= (const Point& other){ x -= other.x, y -= other.y;
return *this; }
        Point& operator *= (const T& t) { x *= t, y *= t; return *this; }
        Point& operator /= (const T& t) { x /= t, y /= t; return *this; }

        friend Point operator + (const Point& p, const Point& q) { return
Point(p.x + q.x, p.y + q.y); }
        friend Point operator - (const Point& p, const Point& q) { return
Point(p.x - q.x, p.y - q.y); }
        friend Point operator * (const Point& p, const T& t) { return Point(p.x *
t, p.y * t); }
        friend Point operator * (const T& t ,const Point& p) { return Point(p.x *
t, p.y * t); }
        friend Point operator / (const Point& p, const T& t) { return Point(p.x /
t, p.y / t); }

        friend bool operator == (const Point& a, const Point& b) { return a.x ==
b.x && a.y == b.y; }
        friend bool operator != (const Point& a, const Point& b) { return !(a ==
b); }

        friend T dot(const Point& p, const Point& q){ return p.x * q.x + p.y *
q.y; }
        friend T cross(const Point& p, const Point& q){ return p.x * q.y - p.y *
q.x; }
        friend T cross3(const Point& p, const Point& q, const Point& r){ return
cross(q - p, r - p); }

        friend auto norm(const Point& p){ return p.x * p.x + p.y * p.y; };
        friend auto abs(const Point& p){ return sqrt(norm(p)); }
};
endsnippet

snippet Dinic
/*
* NOTAS
* edges[i] (u, v): Arista que conecta los nodos u y v
*             edges[i].capacity : capacidad total de la arista
*             edges[i].flow : cantidad de flow utilizado (flow <= capacity)
*
* pos = adj[u][0, 1, ...] : almacena para cada nodo u los ├¡ndices pos tal que
edges[pos].u = u
*             Cada v├(r)rtice v adyacente a u puede ser accedido como
edges[pos].v
*             Ejemplo arista (u, v):
*                     for(int pos : adj[u]) cout << u << ' ' << edges[pos].v <<
'\n';
*                     for(int i = 0; i < adj[u].size(); i++) cout << u << ' ' <<
edges[adj[u][i]].v << '\n';
```

```
*
* La estructura trabaja indexada en 1 pero al constructor debemos pasarle el N
original (sin aumentar en 1)
*
* ALGORITMO:
* 1.- Se realiza un BFS para etiquetar con niveles cada nodo.
*         Un nodo se puede alcanzar si su capacidad - flow usado > 0
*         Se debe llegar desde el nodo source hasta el nodo sink
* 2.- Se realizan m├║ltiples DFS hasta que ya no se pueda aumentar el flow
total
* 3.- Se repiten los pasos 1 y 2 hasta que ya no se pueda realizar el paso 1.
*         La suma de todos los flows es el max_flow
*
* OBSERVACIONES
* Para min cut, en el bucle for el aumento es i += 2 puesto que hacemos
add_edge(u, v) y add_edge(v, u) en grafo no dirigido.
* Para grafo dirifido el aumento deber├¡a ser i++
*
* Hay que considerar un flow_t (int o long long) de modo que mueda almacenar la
suma total de flow, no solo flows individuales
*/

template<class flow_t>
${1}struct Dinic{
	struct Edge{
		int u, v;
		flow_t capacity, flow = 0;
		Edge(int u, int v, flow_t capacity) : u(u), v(v),
capacity(capacity) {}
	};

	int index = 0, N = -1;
	vector<Edge> edges;
	vector<vector<int>> adj;
	vector<int> level, cnt;
	bool flow_called;

	Dinic(int N) : N(N) {
		adj.resize(N + 1);
		level.resize(N + 1);
		cnt.resize(N + 1);
		flow_called = false;
	}

	void add_edge(int u, int v, flow_t capacity){
		// 1-indexed
		assert(0 < u && u <= N && 0 < v && v <= N);
		edges.emplace_back(u, v, capacity);
		edges.emplace_back(v, u, 0);
		adj[u].push_back(index);
		adj[v].push_back(index + 1);
```

```
                index += 2;
        }

        bool bfs(int source, int sink){
                fill(level.begin(), level.end(), -1);
                level[source] = 0;
                queue<int> q;
                q.push(source);
                while(!q.empty()){
                        int cur = q.front(); q.pop();
                        for(int next : adj[cur]){
                                if(edges[next].capacity - edges[next].flow < 1)
continue;
                                if(level[edges[next].v] != -1) continue;
                                level[edges[next].v] = level[cur] + 1;
                                q.push(edges[next].v);
                        }
                }
                return level[sink] != -1;
        }

        flow_t dfs(int cur, int sink, flow_t min_flow =
numeric_limits<flow_t>::max()){
                if(min_flow == 0) return 0;
                if(cur == sink) return min_flow;
                for(int& i = cnt[cur]; i < adj[cur].size(); i++){
                        int idx = adj[cur][i];
                        int next = edges[idx].v;
                        if(level[cur] + 1 != level[next] || edges[idx].capacity -
edges[idx].flow < 1) continue;
                        flow_t mn = dfs(next, sink, min(min_flow,
edges[idx].capacity - edges[idx].flow));
                        if(mn == 0) continue;
                        edges[idx].flow += mn;
                        edges[idx ^ 1].flow -= mn;
                        return mn;
                }
                return 0;
        }

        flow_t flow(int source, int sink){
                flow_t total_flow = 0;
                while(bfs(source, sink)){
                        fill(cnt.begin(), cnt.end(), 0);
                        while(flow_t min_flow = dfs(source, sink)) total_flow +=
min_flow;
                }
                flow_called = true;
                return total_flow;
        }
```

```
    vector<pair<int, int>> min_cut(int source){
        assert(flow_called);
        vector<bool> reachable(N + 1, false);
        vector<pair<int, int>> cut;

        function<void(int)> dfs = [&](int cur){
            reachable[cur] = true;
            for(int next : adj[cur]){
                if(edges[next].capacity - edges[next].flow > 0 &&
!reachable[edges[next].v])
                    dfs(edges[next].v);
            }
        };
        dfs(source);
        for(int cur = 1; cur <= N; cur++){
            for(int i = 0 ; i < adj[cur].size(); i += 2){
                int next = adj[cur][i];
                if(reachable[cur] && !reachable[edges[next].v] &&
edges[next].capacity - edges[next].flow == 0)
                    cut.push_back({cur, edges[next].v});
            }
        }
        return cut;
    }
};
endsnippet

snippet binomialCoefficient
//Forma 1: Eficiente para calcular solo un coeficiente binomial en m├│dulo 10^9
+7

const int mod = 1e9 + 7;

long long fast_pow(long long a, long long p) {
    long long res = 1;
    while(p){
        if(p % 2 == 0) {
            a = a * a % mod;
            p >>= 1;
        }else{
            res = res * a % mod;
            p--;
        }
    }
    return res;
}

long long fact(int k){
    long long ans = 1;
    for(int i = 2; i <= k; i++)
        ans = (ans * i) % mod;
```

```
	return ans;
}

long long C(int n, int k){
	return ((fact(n) * fast_pow(fact(k), mod - 2)) % mod * (fast_pow(fact(n -
k), mod - 2))) % mod;
}

//Forma 2: Eficiente para calcular muchos coeficientes binomiales en m├│dulo
10^9 +7 (dp)

//const int mod = 1e9+7;
const int MXN = 1e6;
long long inv[MXN+1], fac[MXN+1], inv_fac[MXN+1];

void init(){
	inv[1] = 1;
	for(int i = 2; i <= MXN; i++){
		inv[i] = mod - (mod / i * inv[mod % i]) % mod;
	}
	fac[0] = inv_fac[0] = 1;
	for(int i = 1; i <= MXN; i++){
		fac[i] = (fac[i - 1] * i) % mod;
		inv_fac[i] = (inv_fac[i - 1] * inv[i]) % mod;
	}
}

long long BC(int n, int k){
	return fac[n] * inv_fac[k] % mod * inv_fac[n - k] %mod;
}
endsnippet

snippet Mo
int block_sz;

struct Query{
	int l, r, idx;
	bool operator< (const Query other){
		return make_pair(l / block_sz, r) < make_pair(other.l / block_sz,
other.r);
	}
};

/* Verificar el valor inicial y el tipo de dato apropiado
 * Modificar las funciones add y remove de acuerdo al problema
 * Modificar si es necesario la funcion get_answer
 * Si las consultas [l, r] estan indexadas en 1 entonces is_0index = false
 * Complejidad O((N + Q) * sqrt(N))
 */

template<class T>
```

```
${1}struct Mo{
        T answer = 0;
        vector<Query> queries;

        Mo(int N) { block_sz = (int)sqrt(N) + 1;}

        void add(int pos){

        }

        void remove(int pos){

        }

        T get_answer(){
                return answer;
        }

        void read_queries(int q){
                queries.resize(q);
                for(int i = 0; i < q; i++){
                        cin >> queries[i].l >> queries[i].r;
                }
        }

        vector<T> run(){
                bool is_0index = false;
                for(int i = 0; i < queries.size(); i++){
                        if(!is_0index) queries[i].l--, queries[i].r--;
                        queries[i].idx = i;
                }
                sort(queries.begin(), queries.end());
                vector<T> ans(queries.size());
                int left = 0, right = -1;
                for(auto q : queries){
                        while(left > q.l){ add(--left); }
                        while(right < q.r){ add(++right); }
                        while(left < q.l){ remove(left++); }
                        while(right > q.r){ remove(right--); }
                        ans[q.idx] = get_answer();
                }
                return ans;
        }
};
endsnippet

snippet MoHilbert
template<class T>
struct Mo{
        T answer = 0;
        int width;
```

```
vector<tuple<int64_t, int, int, int>> queries;

Mo(int N){
    width = 1 + floor(log2(N));
}

inline int64_t hilbertOrder(int x, int y, int pow, int rotate) {
    if (pow == 0) {
        return 0;
    }
    int hpow = 1 << (pow-1);
    int seg = (x < hpow) ? (
        (y < hpow) ? 0 : 3
    ) : (
        (y < hpow) ? 1 : 2
    );
    seg = (seg + rotate) & 3;
    const int rotateDelta[4] = {3, 0, 0, 1};
    int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
    int nrot = (rotate + rotateDelta[seg]) & 3;
    int64_t subSquareSize = int64_t(1) << (2*pow - 2);
    int64_t ans = seg * subSquareSize;
    int64_t add = hilbertOrder(nx, ny, pow-1, nrot);
    ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
    return ans;
}

void read_queries(int q){
    queries.resize(q);
    for(int i = 0; i < q; i++){
        int l, r; cin >> l >> r; l--, r--;
        queries[i] = {hilbertOrder(l, r, width, 0), l, r, i};
    }
}

void add(int pos){

}

void remove(int pos){

}

T get_answer(){
    return answer;
}


vector<T> run(){
    vector<T> ans(queries.size());
    sort(queries.begin(), queries.end(), [&](const auto& x, const auto&
y){
			return get<0>(x) < get<0>(y);
		});
		int left = 0, right = -1;
		for(auto [h, l, r, i] : queries){
			while(left > l){ add(--left); }
			while(right < r){ add(++right); }
			while(left < l){ remove(left++); }
			while(right > r){ remove(right--); }
			ans[i] = get_answer();
		}
		return ans;
	}
};
endsnippet

snippet IntegerSQRT
int64_t isqrt(int64_t N){
	assert(N >= 0);
	if(N == 0) return 0;
	int64_t M = sqrt(N) - 1;
	while(M + 1 <= N / (M + 1)) M++;
	return M;
}
endsnippet

snippet PI
const double PI = 3.141592653589793;
endsnippet

snippet MOD
${1}template <int MOD_> struct modnum {
	static constexpr int MOD = MOD_;
	int value;

	static int minv(int a, int m) {
		a %= m;
		assert(a);
		return a == 1 ? 1 : int(m - int64_t(minv(m, a)) * int64_t(m) / a);
	}

	modnum() : value(0) {}
	modnum(int64_t v) : value(int(v % MOD)) { if (value < 0) value += MOD; }

	explicit operator int() const { return value; }

	modnum inv() const { modnum res; res.value = minv(value, MOD); return
res; }
	friend modnum inv(const modnum& m) { return m.inv(); }

	friend ostream& operator << (ostream& out, const modnum& n) { return out
```

```
<< int(n); }
	friend istream& operator >> (istream& in, modnum& n) { int64_t v; in >>
v; n = modnum(v); return in; }
	friend bool operator == (const modnum& a, const modnum& b) { return
a.value == b.value; }
	friend bool operator != (const modnum& a, const modnum& b) { return
a.value != b.value; }
	friend bool operator > (const modnum& a, const modnum& b) { return
a.value > b.value; }
	friend bool operator < (const modnum& a, const modnum& b) { return
a.value < b.value; }

	modnum operator - () const { return neg(); }
	modnum operator + () const { return modnum(*this); }
	modnum& operator ++ () { value++; if (value == MOD) value = 0; return
*this; }
	modnum& operator -- () { if (value == 0) value = MOD; value--; return
*this; }
	modnum& operator += (const modnum& other) {
		value += other.value;
		if (value >= MOD) value -= MOD;
		return *this;
	}
	modnum& operator -= (const modnum& other) {
		value -= other.value;
		if (value < 0) value += MOD;
		return *this;
	}
	modnum& operator *= (const modnum& other) {
		value = int(int64_t(value) * int64_t(other.value) % MOD);
		return *this;
	}
	modnum& operator /= (const modnum& other) {
		return *this *= other.inv();
	}

	modnum neg() const { modnum res; res.value = value ? MOD - value : 0;
return res; }
	friend modnum neg(const modnum& m) { return m.neg(); }

	friend modnum operator + (const modnum& a, const modnum& b) { return
modnum(a) += b; }
	friend modnum operator - (const modnum& a, const modnum& b) { return
modnum(a) -= b; }
	friend modnum operator * (const modnum& a, const modnum& b) { return
modnum(a) *= b; }
	friend modnum operator / (const modnum& a, const modnum& b) { return
modnum(a) /= b; }
	friend modnum power(modnum a, long long p){
		modnum res = 1;
		while(p > 0){
```

```
                if(p % 2 == 1) res *= a;
                p >>= 1; a *= a;
            }
            return res;
        }
};

using mint = modnum<1000000007>; // modnum<998244353>;

endsnippet

snippet sieve

vector<int> lp, primes;

${1}void sieve(int n){
    lp.resize(n + 1);
    for(int i = 2; i <= n; i++){
        if(lp[i] == 0){ lp[i] = i, primes.push_back(i); }
        for(int j = 0; j < primes.size() && primes[j] <= lp[i] && i * primes[j] <= n; j++)
            lp[i * primes[j]] = primes[j];
    }
}
endsnippet

snippet mobius
/* Puedes usarlo con la convoluci├│n de Dirichlet
 * Puedes usarlo para inclusi├│n - exclusion de los factores primos
 * Ejemplo:
 * 60 = 2^2 * 3 * 5
 => mob(2) = mob(3) = mob(5) = -1
 => mob(2*3) = mob(2*5) = mob(3*5) = 1
 => mob(2*3*5) = -1
*/
const int N = 5e5 + 1;

/*
 * mob(n) = 1 : n = 1
 * mob(n) = 0 : si n no es libre de cuadrados
 * mob(n) = (-1)^k : n = p1 * p2 * ... * pk
*/
vector<int> mob(N);

/*
 * n = p1^q1 * p2^q2 * ... * pk^qk
 * d(n) = 1, p1, p2, ..., pk, p1p2, p1p3, ..., p1pk, p2p3
 * d(n) : todos los divisores libre de cuadrados de n
*/

//vector<vector<int>> d(N);
```

```
void mobius(){
    mob[1] = 1;
    for(int i = 2; i < N; i++){
        mob[i]--;
        for(int j = i + i; j < N; j += i)
            mob[j] -= mob[i];
    }
    /*for(int i = 1; i < N; i++){
        if(mob[i] == 0) continue;
        for(int j = i; j < N; j += i) d[j].push_back(i);
    }*/
}
endsnippet

snippet highest_bit
auto highest_bit = [&](unsigned x){
    return x == 0 ? -1 : 31 - __builtin_clz(x);
};
endsnippet

snippet SCC
${1}struct SCC{
    int N = 0, id;
    vector<vector<int>> adj;
    vector<int> ind, low;
    stack<int> s;
    vector<bool> in_stack;
    vector<vector<int>> components;
    vector<int> component_id;

    //1-indexed
    SCC(int n = 0){ N = n + 1, adj.assign(N, {}); }
    SCC(const vector<vector<int>> & _adj){ adj = _adj, N = adj.size(); }

    void add_edge(int from, int to){
        adj[from].push_back(to);
    }

    void dfs(int u){
        low[u] = ind[u] = id++;
        s.push(u);
        in_stack[u] = true;
        for(int v : adj[u]){
            if(ind[v] == -1){
                dfs(v);
                low[u] = min(low[u], low[v]);
            }else if(in_stack[v]){
                low[u] = min(low[u], ind[v]);
            }
        }
        if(low[u] == ind[u]){
            components.emplace_back();
            vector<int> & comp = components.back();
            while(true){
                assert(!s.empty());
                int x = s.top(); s.pop();
                in_stack[x] = false;
                component_id[x] = components.size() - 1;
                comp.push_back(x);
                if(x == u) break;
            }
        }
    }

    vector<vector<int>> get(){
        ind.assign(N, - 1); low.assign(N, -1); component_id.assign(N, -1);
        s = stack<int>();
        in_stack.assign(N, false);
        id = 0;
        components = {};
        for(int i = 1; i < N; i++)
            if(ind[i] == -1) dfs(i);

        // reverse(components.begin(), components.end()); return
components; // SCC in topological order
        return components; // SCC in reverse topological order
    }
};
endsnippet

snippet FFT
using float_type = long double;
using value_type = long long;

struct FFT{
    const float_type PI = acos((float_type) -1);
    vector<complex<float_type>> root;

    int round_up_power_two(int n){
        assert(n > 0);
        while(n & (n - 1)) n = (n | (n - 1)) + 1;
        return n;
    }

    void reorder(int n, vector<complex<float_type>>& a){
        assert((n & (n - 1)) == 0); // n = 2^k
        int zeros = __builtin_ctz(n); // return k
        for(int i = 0; i < n; i++){
            int bit_reverse = 0;
            for(int it = 0, cur_i = i; it < zeros; it++, cur_i >>= 1){
                bit_reverse <<= 1; bit_reverse += (cur_i & 1);
            }
            if(i < bit_reverse) swap(a[i], a[bit_reverse]);
        }
    }

    void fft(vector<complex<float_type>> &a){
        int n = a.size();
        assert((n & (n - 1)) == 0); // n = 2^k
        if(root.size() != n){
            root.resize(n);
            for(int i = 0; i < n; i++){
                root[i] = polar(float_type(1.0), float_type(2.0 * PI
* i / n));
            }
        }
        reorder(n, a);

        for(int depth = __builtin_ctz(n) - 1; ~depth; depth--){
            int m = (n >> (depth + 1));
            for(int k = 0; k < n; k += 2 * m){
                for(int i = 0; i < m; i++){
                    complex<float_type> even = a[k + i], odd = a[k
+ i + m];
                    a[k + i] = even + root[i << depth] * odd;
                    a[k + i + m] = even + root[(i + m) << depth] *
odd;
                }
            }
        }
    }

    vector<value_type> multiply(const vector<value_type> &a, const
vector<value_type> &b){
        int n = a.size(), m = b.size();
        int power_two = round_up_power_two(n + m - 1);
        vector<complex<float_type>> dfta(power_two, 0), dftb(power_two, 0);
        for(int i = 0; i < n; i++) dfta[i].real(a[i]);
        for(int i = 0; i < m; i++) dftb[i].real(b[i]);
        fft(dfta);
        fft(dftb);

        for(int i = 0; i < power_two; i++) dfta[i] = conj(dfta[i] *
dftb[i]);

        fft(dfta);
        vector<value_type> ans(n + m - 1);
        for(int i = 0; i < ans.size(); i++){
            ans[i] = (value_type) (dfta[i].real() / power_two + 0.5);
        }
        return ans;
    }
    template<int MOD> vector<value_type> multiplyMod(const
vector<value_type>& a, const vector<value_type>& b){
        if(a.empty() || b.empty()) return {};

        int n = a.size(), m = b.size();
        vector<value_type> ans(n + m - 1);
        int B = 32 - __builtin_clz(ans.size());
        int N = 1 << B;
        int cut = int(sqrt(MOD));
        vector<complex<float_type>> L(N), R(N), outs(N), outl(N);

        for(int i = 0; i < n; i++){
            L[i] = complex<float_type>((int) a[i] / cut, (int) a[i] %
cut);
        }

        for(int i = 0; i < m; i++){
            R[i] = complex<float_type>((int) b[i] / cut, (int) b[i] %
cut);
        }
        fft(L);
        fft(R);
        for(int i = 0; i < N; i++){
            int j = -i & (N - 1);
            outl[j] = (L[i] + conj(L[j])) * R[i] / float_type(2.0 * N);
            outs[j] = (L[i] - conj(L[j])) * R[i] / float_type(2.0 * N) /
complex<float_type>(0, 1);
        }
        fft(outl); fft(outs);

        for(int i = 0; i < n + m - 1; i++){
            value_type av = value_type(real(outl[i]) + .5), cv =
value_type(imag(outs[i]) + .5);
            value_type bv = value_type(imag(outl[i]) + .5) +
value_type(real(outs[i]) + .5);
            ans[i] = ((av % MOD * cut + bv) % MOD * cut + cv) % MOD;
        }
        return ans;
    }

}fft;
endsnippet

snippet ST1
// Usa esto cuando el orden de los intervalos no importan
// por ejemplo cuando se busca la suma en el intervalo [2..7]
// es lo mismo S[2..4] + S[5..7] que S[5..7] + S[2.4]
// 0-indexed

template<class T>
```

```
${1}struct segment_tree{
    int n;
    vector<T> tree;

    segment_tree(int n){
        this -> n = n;
        tree.resize(2 * n);
    }

    segment_tree(vector<T>& values){
        this -> n = values.size();
        tree.resize(2 * n);
        for(int i = 0; i < n; i++) upd(i, values[i]);
    }

    //CHANGE
    T compare(T a, T b){
        return max(a, b);
    }

    void modify(int index, T value){
        index += n;
        tree[index] = value;
        for(index >>= 1; index >= 1; index >>= 1) tree[index]=
compare(tree[2 * index], tree[2 * index + 1]);
    }

    void upd(int index, T value){
        index += n;
        tree[index] = compare(tree[index], value);
        for(index >>= 1; index >= 1; index >>= 1) tree[index]=
compare(tree[2 * index], tree[2 * index + 1]);
    }

    //BOTTOM - TOP
    T query(int first, int last){
        first += n, last += n;
        T ans = -1e9; //CHECK
        while(first <= last){
            if(first % 2 == 1) ans = compare(ans, tree[first++]);
            if(last % 2 == 0) ans = compare(ans, tree[last--]);
            first >>= 1, last >>= 1;
        }
        return ans;
    }

    //TOP - BOTTOM
    T query(int first, int last, int cur, int left_range, int right_range){
        T empty = 0; //CHECK
        if(last < left_range || first > right_range || cur >= tree.size())
return empty;
```

```
		if(first <= left_range && last >= right_range) return tree[cur];
		int mid_range = (left_range + right_range) / 2;
		return compare(query(first, last, 2 * cur, left_range, mid_range), 
query(first, last, 2 * cur + 1, mid_range + 1, right_range));
	}
};
endsnippet

snippet ST2
// Usa esto cuando S├ì importa el orden de los intervalos
// por ejemplo en la m├íxima secuencia correcta de '(' y ')'
// no es lo mismo A[3..4] + A[5..7] = () + (()
// que A[5..7] + A[3..4] = (() + ()

template<typename T>
${1}struct segment_tree{
	int n;
	vector<T> tree;

	segment_tree(int n){
		this -> n = n;
		tree.resize(4 * n);
	}

	segment_tree(vector<T>& values){
		this -> n = values.size();
		tree.resize(4 * n);
		build(values, 1, 0, n - 1);
	}

	//CHANGE
	T compare(T a, T b){
		return a + b;
	}

	void build(vector<T>& values, int cur = 1, int left_range = 0, int 
right_range = -1){
		if(right_range == -1) right_range = n - 1;

		if(left_range == right_range) tree[cur] = values[left_range];
		else{
			int mid_range = (left_range + right_range) / 2;
			build(values, 2 * cur, left_range, mid_range);
			build(values, 2 * cur + 1, mid_range + 1, right_range);
			tree[cur] = compare(tree[2 * cur], tree[2 * cur + 1]);
		}
	}
	// 0-indexed
	void modify(int index, T value, int cur = 1, int left_range = 0, int 
right_range = -1){
		if(right_range == -1) right_range = n - 1;
```

```
        if(left_range == right_range) tree[cur] = value;
        else{
            int mid_range = (left_range + right_range) / 2;
            if(index <= mid_range) modify(index, value, 2 * cur,
left_range, mid_range);
            else modify(index, value, 2 * cur + 1, mid_range + 1,
right_range);
            tree[cur] = compare(tree[2 * cur], tree[2 * cur + 1]);
        }
    }

    T query(int first, int last, int cur = 1, int left_range = 0, int
right_range = -1){
        if(right_range == -1) right_range = n - 1;

        T empty = 0; //CHECK
        if(first > last) return empty;
        if(first == left_range && last == right_range) return tree[cur];
        int mid_range = (left_range + right_range) / 2;
        T left = query(first, min(last, mid_range), 2 * cur, left_range,
mid_range);
        T right = query(max(first, mid_range + 1), last, 2 * cur + 1,
mid_range + 1, right_range);
        return compare(left, right);
    }
};
endsnippet

snippet factoring
template<typename T>
${1}vector<pair<T, int>> factor(T x) {
    vector<pair<T, int>> ans;
    for(T i = 2; i * i <= x; i++)
        if(x % i==0) {
            ans.pb({i, 1});
            while((x /= i) % i == 0) ans.back().second++;
        }
    if(x != 1) ans.pb({x, 1});
    return ans;
}
endsnippet

snippet prime_factoring
template<typename T>
${1}vector<T> prime_factor(T x) {
    vector<T> ans;
    for(T i = 2; i * i <= x; i++)
        if(x % i==0) {
            ans.pb(i);
            while(x % i == 0) x /= i;
```

```
		}
	if(x != 1) ans.pb(x);
	return ans;
}
endsnippet

snippet suffix_array
// O(N log N)
${1}vector<int> suffix_array(string s){
	s += '$';
	int n = s.size();
	const int alphabet = 256;
	vector<int> p(n), c(n), cnt(max(n, alphabet), 0);

	for(int i = 0; i < n; i++) cnt[s[i]]++;
	for(int i = 1; i < alphabet; i++) cnt[i] += cnt[i - 1];
	for(int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
	c[p[0]] = 0;
	int classes = 1;
	for(int i = 1; i < n; i++){
		if(s[p[i]] != s[p[i - 1]]) classes++;
		c[p[i]] = classes - 1;
	}
	vector<int> pn(n), cn(n);
	for(int h = 0; (1 << h)  < n; h++){
		for(int i = 0; i < n; i++){
			pn[i] = p[i] - (1 << h);
			if(pn[i] < 0) pn[i] += n;
		}
		fill(cnt.begin(), cnt.begin() + classes, 0);

		for(int i = 0; i < n; i++) cnt[c[pn[i]]]++;
		for(int i = 1; i < classes; i++) cnt[i] += cnt[i - 1];
		for(int i = n - 1; ~i; i--) p[--cnt[c[pn[i]]]] = pn[i];

		cn[p[0]] = 0;
		classes = 1;
		for(int i = 1; i < n; i++){
			pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n] };
			pair<int, int> prev = {c[p[i - 1]], c[(p[i - 1] + (1 << h))
% n]};
			if(cur != prev) classes++;
			cn[p[i]] = classes - 1;
		}
		c.swap(cn);
	}
	p.erase(p.begin());
	return p;
}
endsnippet
```

```
snippet random
// rng() : [0 - 2^32 - 1]
mt19937 rng((unsigned int)
chrono::steady_clock::now().time_since_epoch().count());
endsnippet

snippet BC
vector<mint> inverse, fact, inv_fact;

void generateBC(int N = 1e5){
	const int mod = mint().MOD;

	inverse.resize(N + 1); fact.resize(N + 1); inv_fact.resize(N + 1);
	inverse[1] = 1;

	for(int i = 2; i <= N; i++)
		inverse[i] = mod - (mod / i * inverse[mod % i]);

	fact[0] = inv_fact[0] = 1;
	for(int i = 1; i <= N; i++){
		fact[i] = fact[i - 1] * mint(i);
		inv_fact[i] = inv_fact[i - 1] * inverse[i];
	}
};

mint C(int n, int k){
	if(k > n) return mint(0);
	assert(n < fact.size() && k < fact.size());
	return fact[n] * inv_fact[k] * inv_fact[n - k];
}
endsnippet

snippet bridges
vector<vector<int>> adj;
vector<bool> used;
vector<int> in, low;
int timer = 0;

void dfs_bridges(int u, int p = -1){
	used[u] = true;
	low[u] = in[u] = ++timer;
	for(int v : adj[u]){
		if(v == p) continue;
		if(used[v]) low[u] = min(low[u], in[v]);
		else {
			dfs_bridges(v, u);
			low[u] = min(low[u], low[v]);
		}
	}
}
```

```
void find_bridges(){
    const int N = adj.size();

    used = vector<bool>(N);
    in = vector<int>(N);
    low = vector<int>(N);

    for(int u = 1; u < N; u++){
        if(used[u]) continue;
        dfs_bridges(u);
    }
}

bool is_bridge(int u, int v){
    if(in[u] > in[v]) swap(u, v);
    return (low[v] > in[u]);
}

endsnippet

snippet MergeSortTree
template<class T>
${1}struct merge_sort_tree{
    int n;
    vector<vector<T>> tree;

    merge_sort_tree(int n){
        this -> n = n;
        tree.resize(5 * n);
    }

    // build(1, 0, A.size() - 1)
    void build(int cur, int left, int right){
        if(left == right){
            tree[cur].push_back(A[left]);
            return;
        }
        int mid = (left + right) / 2;
        build(2 * cur, left, mid);
        build(2 * cur + 1, mid + 1, right);
        tree[cur] = merge(tree[2 * cur], tree[2 * cur + 1]);
    }

    vector<T> merge(vector<T>& A, vector<T>& B){
        int i = 0, j = 0;
        vector<T> ret;
        while(i < A.size() && j < B.size()){
            if(A[i] < B[j]) ret.push_back(A[i++]);
            else ret.push_back(B[j++]);
        }
        while(i < A.size()) ret.push_back(A[i++]);
        while(j < B.size()) ret.push_back(B[j++]);
        return ret;
    }

    // Modify
    void query(int cur, int left, int right, int ql, int qr){
        if(right < ql || left > qr) return;

        if(ql <= left && right <= qr){
            // Do something
            return;
        }

        int mid = (left + right) / 2;
        query(2 * cur, left, mid, ql, qr);
        query(2 * cur + 1, mid + 1, right, ql, qr);
    }
};
endsnippet

snippet lazyST "Segment Tree que permite hacer updates en rangos. Consultas y updates indexadas en 0" b
template<typename T>
${1}struct lazy_segment_tree{
    int n;
    const T EMPTY = 0; // CHECK
    vector<T> tree;
    vector<int> lazy;

    lazy_segment_tree(int n){
        this -> n = n;
        tree.resize(4 * n);
        lazy.resize(4 * n);
    }

    T compare(T& a, T& b){
        return a + b; // CHECK
    }

    void push_op(T& a, int& lazy_value){
        a ^= lazy_value; // CHECK
    }

    void lazy_op(int& lazy_child, int& lazy_parent){
        lazy_child ^= lazy_parent; // CHECK
    }

    void push(int cur){
        if(lazy[cur] == EMPTY) return;
        push_op(tree[2 * cur], lazy[cur]);
        push_op(tree[2 * cur + 1], lazy[cur]);
```

```
            lazy_op(lazy[2 * cur], lazy[cur]);
            lazy_op(lazy[2 * cur + 1], lazy[cur]);
            lazy[cur] = EMPTY;
    }

    void build(vector<T>& values, int cur = 1, int left_range = 0, int
right_range = -1){
            if(right_range == -1) right_range = n - 1;
            lazy[cur] = EMPTY;
            if(left_range == right_range){
                if(left_range >= 0 && left_range < n) {
                    tree[cur] = values[left_range];
                }
                return;
            }
            int mid_range = (left_range + right_range) / 2;
            build(values, 2 * cur, left_range, mid_range);
            build(values, 2 * cur + 1, mid_range + 1, right_range);
            tree[cur] = compare(tree[2 * cur], tree[2 * cur + 1]);
    }

    // 0-indexed
    void upd(int first, int last, T value, int cur = 1, int left_range = 0,
int right_range = -1){
            if(right_range == -1) right_range = n - 1;
            if(first > last) return;
            if(first == left_range && right_range == last){
                push_op(tree[cur], value);
                lazy_op(lazy[cur], value);
                return;
            }
            push(cur);

            int mid_range = (left_range + right_range) / 2;
            upd(first, min(last, mid_range), value, 2 * cur, left_range,
mid_range);
            upd(max(first, mid_range + 1), last, value, 2 * cur + 1, mid_range
+ 1, right_range);
            tree[cur] = compare(tree[2 * cur], tree[2 * cur + 1]);
    }

    T query(int first, int last, int cur = 1, int left_range = 0, int
right_range = -1){
            if(right_range == -1) right_range = n - 1;
            if(first > last) return EMPTY;
            if(left_range > last || right_range < first) return EMPTY;
            if(first == left_range && right_range == last) {
                return tree[cur];
            }

            push(cur);
```

```
        int mid_range = (left_range + right_range) / 2;
        T ql = query(first, min(last, mid_range), 2 * cur, left_range,
mid_range);
        T qr = query(max(first, mid_range + 1), last, 2 * cur + 1,
mid_range + 1, right_range);
        return compare(ql, qr);
    }
};
endsnippet

snippet MinCostMaxFlow "Calcula el m├¡nimo costo al enviar el m├íximo flujo.
(Benq)" b
template<class type>
${1}struct MCMF{
    struct Edge{
        int u, v;
        type capacity, flow = 0, cost;

        Edge(int u, int v, type capacity, type cost) : u(u), v(v),
capacity(capacity), cost(cost) {}
    };

    int N, index = 0;
    vector<Edge> edges;
    vector<vector<int>> adj;
    vector<type> dist;
    vector<type> pot;
    vector<int> parent;

    MCMF(int N) : N(N) {
        adj.resize(N + 1);
        dist.resize(N + 1);
        pot.resize(N + 1);
        parent.resize(N + 1);
    }

    void add_edge(int u, int v, int capacity, int cost){
        assert(0 < u && u <= N && 0 < v && v <= N);
        adj[u].push_back(index);
        adj[v].push_back(index + 1);
        edges.push_back({u, v, capacity, cost});
        edges.push_back({v, u, 0, -cost});
        index += 2;
    }

    bool path(int source, int sink){
        const type inf = numeric_limits<type>::max();
        fill(dist.begin(), dist.end(), inf);
        using Node = pair<type, int>;
        priority_queue<Node, vector<Node>, greater<Node>> pq;
        dist[source] = 0;
        pq.push({0, source});
        while(!pq.empty()){
            auto [d, u] = pq.top(); pq.pop();
            if(dist[u] < d) continue;
            for(int pos : adj[u]){
                const Edge& e = edges[pos];
                type new_dist = d + e.cost + pot[u] - pot[e.v];
                if(e.flow < e.capacity && dist[e.v] > new_dist){
                    dist[e.v] = new_dist;
                    parent[e.v] = pos;
                    pq.push({dist[e.v], e.v});
                }
            }
        }
        return (dist[sink] != inf);
    }

    pair<type, type> get_cost(int source, int sink){
        for(int it = 1; it <= N; it++){
            for(int pos = 0; pos < edges.size(); pos++){
                const Edge& e = edges[pos];
                if(e.capacity > 0){
                    pot[e.v] = min(pot[e.v], pot[edges[pos ^ 1].v]
+ e.cost);
                }
            }
        }
        type total_flow = 0;
        type total_cost = 0;

        while(path(source, sink)){
            for(int i = 0; i <= N; i++) pot[i] += dist[i];
            type flow_add = numeric_limits<type>::max();
            for(int cur = sink; cur != source; cur = edges[parent[cur] ^
1].v){
                const Edge& e = edges[parent[cur]];
                flow_add = min(flow_add, e.capacity - e.flow);
            }
            total_flow += flow_add;
            total_cost += (pot[sink] - pot[source]) * flow_add;

            for(int cur = sink; cur != source; cur = edges[parent[cur] ^
1].v){
                edges[parent[cur]].flow += flow_add;
                edges[parent[cur] ^ 1].flow -= flow_add;
            }
        }
        return {total_flow, total_cost};
    }
};
endsnippet
```

```
snippet KMP-SEARCH "Cuenta ocurrencias de un string T en S"
${1}int kmp_search(string& s, string& t, vector<int>& good){
        const int N = s.size(), M = t.size();
        int count = 0;
        vector<int> lps = prefix(t);
        int si = 0, ti = 0;

        while((N - si) >= (M - ti)){
                if(s[si] == t[ti]){
                        si++, ti++;
                }

                if(ti == M){
                        count++;
                        good.push_back(si - ti);
                        ti = lps[ti - 1];
                }
                else if(si < N && s[si] != t[ti]){
                        if(ti != 0) ti = lps[ti - 1];
                        else si++;
                }
        }
        return count;
}
endsnippet

snippet Hashing
template<class container>
${1}struct Hashing{
        const long long M = (1LL << 61) - 1;
        long long B;

        vector<long long> powB; // powB[i] = B^i mod M

        vector<long long> prefix_hash; // prefix_hash[i] = hashing(s[0] : s[i -
1])

        __int128 mul(long long x, long long y) { return (__int128) x * y; }
        long long mod_mul(long long x, long long y) { return mul(x, y) % M; }

        Hashing(const container& s) : prefix_hash(s.size() + 1){
                mt19937
rng((uint32_t)chrono::steady_clock::now().time_since_epoch().count());
                this -> B = uniform_int_distribution<long long>(0, M - 1)(rng);

                powB = {1};
                while(powB.size() < s.size()){
                        powB.push_back(mod_mul(powB.back(), B));
                }
                prefix_hash[0] = 0;
        for(int i = 0; i < s.size(); i++){
            prefix_hash[i + 1] = (mod_mul(prefix_hash[i], B) + s[i]) %
M;
        }
    }

    long long get_hash(int start, int end){
        long long val = prefix_hash[end + 1] - mod_mul(prefix_hash[start],
powB[end - start + 1]);
        return (val % M + M) % M;
    }
};
endsnippet

snippet XorBasis
${1}struct Xor_basis{
    int bits, size;
    vector<int> basis;

    Xor_basis() { bits = 32, size = 0; }

    Xor_basis(int bits) : bits(bits) {
        size = 0;
        basis.resize(bits);
    }

    void insert(int mask){
        for(int i = 0; i < bits; i++){
            if((mask & (1 << i)) == 0) continue;
            if(!basis[i]){
                basis[i] = mask;
                size++;
                return;
            }
            mask ^= basis[i];
        }
    }
};
endsnippet

snippet NTT
const int MOD = 998244353;
const int ROOT = 3;

${1}struct NTT{
    vector<int> roots{0, 1};

    int round_up_power_two(int n){
        assert(n > 0);
        while(n & (n - 1)) n = (n | (n - 1)) + 1;
        return n;
        }

        int power(int a, int b){
                int ans = 1;
                while(b > 0){
                        if(b % 2 == 1) ans = 1LL * ans * a % MOD;
                        a = 1LL * a * a % MOD;
                        b >>= 1;
                }
                return ans;
        }

        void reorder(int n, vector<int>& a){
                assert((n & (n - 1)) == 0); // n = 2^k
                int zeros = __builtin_ctz(n); // return k
                for(int i = 0; i < n; i++){
                        int bit_reverse = 0;
                        for(int it = 0, cur_i = i; it < zeros; it++, cur_i >>= 1){
                                bit_reverse <<= 1; bit_reverse += (cur_i & 1);
                        }
                        if(i < bit_reverse) swap(a[i], a[bit_reverse]);
                }
        }

        void dft(vector<int> &a) {
                int n = a.size();
                assert((n & (n - 1)) == 0); // n = 2^k

                reorder(n, a);

                if (int(roots.size()) < n) {
                        int k = __builtin_ctz(roots.size());
                        roots.resize(n);
                        while ((1 << k) < n) {
                                int e = power(ROOT, (MOD - 1) >> (k + 1));
                                for (int i = 1 << (k - 1); i < (1 << k); ++i) {
                                        roots[2 * i] = roots[i];
                                        roots[2 * i + 1] = 1LL * roots[i] * e % MOD;
                                }
                                ++k;
                        }
                }
                for (int k = 1; k < n; k *= 2){
                        for (int i = 0; i < n; i += 2 * k){
                                for (int j = 0; j < k; ++j) {
                                        int u = a[i + j];
                                        int v = 1LL * a[i + j + k] * roots[k + j] %
MOD;
                                        a[i + j] = (u + v);
                                        if(a[i + j] >= MOD) a[i + j] -= MOD;
```

```
                    a[i + j + k] = (u - v);
                    if(a[i + j + k] < 0 ) a[i + j + k] += MOD;
                }
            }
        }
    }

    void invdft(vector<int> &a){
        int n = a.size();
        reverse(a.begin() + 1, a.end());
        dft(a);
        int inv = power(n, MOD - 2);
        for(int i = 0; i < n; i++) a[i] = 1LL * a[i] * inv % MOD;
    }

    vector<int> multiply(vector<int> a, vector<int> b){
        const int n = a.size(), m = b.size();
        int power_two = round_up_power_two(n + m - 1);
        a.resize(power_two);
        b.resize(power_two);

        dft(a);
        dft(b);

        for(int i = 0; i < power_two; i++) a[i] = 1LL * a[i] * b[i] % MOD;

        invdft(a);
        a.resize(n + m - 1);
        return a;
    }
}nnt;
endsnippet

snippet geometry "Geometry Namespace"
${1}namespace geometry{

    const double PI = 3.141592653589793;
    const double eps = 1e-9;
    using float_type = long double;
    using coord_type = long double;

    int sign(coord_type x){
        return x < -eps ? -1 : x > eps;
    }

    struct Point{
        coord_type x, y;
        Point() : x(0), y(0) {}
        Point(const coord_type& x_, const coord_type& y_) : x(x_), y(y_) {}

        friend ostream& operator << (ostream& os, const Point& p){ return
```

```
os << '(' << p.x << ',' << p.y << ')'; }
             //friend ostream& operator << (ostream& os, const Point& p){ return
os << p.x << ' ' << p.y; }
             friend istream& operator >> (istream& is, Point& p){ return is >>
p.x >> p.y; }

             Point& operator += (const Point& other){ x += other.x, y +=
other.y; return *this; }
             Point& operator -= (const Point& other){ x -= other.x, y -=
other.y; return *this; }
             Point& operator *= (const coord_type& t) { x *= t, y *= t; return
*this; }
             Point& operator /= (const coord_type& t) { x /= t, y /= t; return
*this; }

             friend Point operator + (const Point& p, const Point& q) { return
Point(p.x + q.x, p.y + q.y); }
             friend Point operator - (const Point& p, const Point& q) { return
Point(p.x - q.x, p.y - q.y); }
             friend Point operator * (const Point& p, const coord_type& t) {
return Point(p.x * t, p.y * t); }
             friend Point operator * (const coord_type& t ,const Point& p) {
return Point(p.x * t, p.y * t); }
             friend Point operator / (const Point& p, const coord_type& t) {
return Point(p.x / t, p.y / t); }

             friend auto norm(const Point& p){ return p.x * p.x + p.y * p.y; };
             friend auto abs(const Point& p){ return sqrt(norm(p)); }

             //friend bool operator == (const Point& a, const Point& b) { return
a.x == b.x && a.y == b.y; }
             friend bool operator == (const Point& a, const Point& b) { return
sign(abs(b - a)) == 0; }
             friend bool operator != (const Point& a, const Point& b) { return
!(a == b); }
             friend bool operator < (const Point& a, const Point& b) { return
(a.x < b.x || (a.x == b.x && a.y < b.y)); }

             friend coord_type dot(const Point& p, const Point& q){ return p.x *
q.x + p.y * q.y; }
             friend coord_type cross(const Point& p, const Point& q){ return p.x
* q.y - p.y * q.x; }
             friend coord_type cross3(const Point& p, const Point& q, const
Point& r){ return cross(q - p, r - p); }
      };

      // Angles:
      // dot(A, B)  = |A||B| cos(x)
      // cross(A, B)      = |A||B| sin(x)

      // clockwise angles : Return \angle AOB
```

```
    float_type get_angle(Point& OA, Point& OB){
        float_type dot_product = dot(OA, OB), cross_product = cross(OA, OB);
        float_type angle = atan2(cross_product, dot_product);
        angle *= float_type(180.0) / PI;
        return angle;
    }

    // clockwise angles : Return \angle AOB
    float_type get_angle(Point& A, Point& O, Point& B){
        Point OA = A - O, OB = B - O;
        return get_angle(OA, OB);
    }

    // location of A relative to BC
    // 1: left, 0: collinear, -1: right
    int location(Point& A, Point& B, Point& C){
        return sign(cross3(A, B, C));
    }

    // is A between the segment BC?
    bool between(Point& A, Point& B, Point& C){
        coord_type mnx = A.x - min(B.x, C.x);
        coord_type mny = A.y - min(B.y, C.y);
        coord_type mxx = max(B.x, C.x) - A.x;
        coord_type mxy = max(B.y, C.y) - A.y;
        return (sign(mnx) >= 0 && sign(mny) >= 0 && sign(mxx) >= 0 && sign(mxy) >= 0);
    }

    vector<Point> segment_intersection(Point A, Point B, Point P, Point Q){
        vector<Point> intersection = {};
        for(int i = 0; i < 2; i++){
            if(location(A, P, Q) == 0 && between(A, P, Q)){
                if(location(B, P, Q) != 0) intersection = {A};
                else if(between(B, P, Q)) intersection = {A, B};
                else if(between(P, A, B)) intersection = {A, P};
                else { assert(between(Q, A, B)); intersection = {A, Q}; }
                if(intersection.size() == 2 && intersection[0] == intersection[1]) intersection.pop_back();
                sort(intersection.begin(), intersection.end());
                return intersection;
            }
            if(location(B, P, Q) == 0 && between(B, P, Q)){
                if(location(A, P, Q) != 0) intersection = {B};
                else if(between(P, A, B)) intersection = {B, P};
                else { assert(between(Q, A, B)); intersection = {B, Q}; }
                if(intersection.size() == 2 && intersection[0] == intersection[1]) intersection.pop_back();
                sort(intersection.begin(), intersection.end());
                return intersection;
            }
            swap(A, P); swap(B, Q);
        }
        if(location(A, P, Q) != location(B, P, Q) && location(P, A, B) != location(Q, A, B)){
            Point C = B - A, R = Q - P; // AB = A + C * lambda, PQ = P + R * lambda
            float_type lambda = (float_type) cross((P - A), R) / cross(C, R);
            Point intersec = A + lambda * C;
            intersection = {intersec};
            return intersection;
        }
        return intersection;
    }

    struct Line{
        Point a, b;
        bool is_line = false;
        coord_type A, B, C; // Line: Ax + By = C

        Line() {}
        Line(Point& a, Point& b, bool line = false) : a(a), b(b) {
            if(line){
                is_line = true;
                A = a.y - b.y;
                B = b.x - a.x;
                coord_type Z = sqrt(A * A + B * B); //
                A /= Z, B /= Z;
                C = A * a.x + B * a.y;
            }
        }

        friend ostream& operator << (ostream& os, const Line& l){ return os << l.a << " --- " << l.b; }
        //friend ostream& operator << (ostream& os, const Line& l){ return os << l.a << ' ' << l.b; }
        friend istream& operator >> (istream& is, Line& l){ return is >> l.a >> l.b; }

        bool parallel(Line& L1, Line& L2){
            return sign(cross(L1.b - L1.a, L2.b - L2.a)) == 0;
        }

        bool orthogonal(Line& L1, Line& L2){
            return sign(dot(L1.b - L1.a, L2.b - L2.a)) == 0;
        }

        // Projection of P onto segment L
```

```
             friend Point projection(Point& P, Line& L){
                   return L.a + (L.b - L.a) * dot(P - L.a, L.b - L.a) /
norm(L.b - L.a);
             }

             // Reflection of P on segment L
             friend Point reflection(Point& P, Line& L){
                   Point Q = projection(P, L);
                   return Q + Q - P;
             }

             friend bool check_segment_intersection(Line L1, Line L2){
                   assert(!L1.is_line && !L2.is_line);
                   return !segment_intersection(L1.a, L1.b, L2.a,
L2.b).empty();
             }

             // Point of intersection or nullopt if L1 || L2
             friend optional<Point> line_intersection(Line& L1, Line& L2){
                   assert(L1.is_line && L2.is_line);
                   float_type det = L1.A * L2.B - L2.A * L1.B;
                   if(det == 0) return nullopt;
                   float_type x = (L2.B * L1.C - L1.B * L2.C) / det;
                   float_type y = (L1.A * L2.C - L2.A * L1.C) / det;
                   Point ans(x, y);
                   return ans;
             }

             friend optional<Point> line_segment_intersection(Line L1, Line L2){
                   if(!L1.is_line) swap(L1, L2);
                   assert(L1.is_line && !L2.is_line);
                   Line L2ext = Line(L2.a, L2.b, true);
                   auto P = line_intersection(L1, L2ext);
                   if(!P) return nullopt;
                   if(between(*P, L2.a, L2.b)) return *P;
                   return nullopt;
             }

             friend float_type distance_point_line(Point& P, Line& L){
                   float_type dist = cross3(P, L.a, L.b) / abs(L.b - L.a);
                   if(!L.is_line){
                          if(sign(dot(L.b - L.a, P - L.a)) <= 0) return abs(P -
L.a);

                          if(sign(dot(L.a - L.b, P - L.b)) <= 0) return abs(P -
L.b);
                   }
                   return abs(dist);
             }

             friend float_type distance_segment_segment(Line& L1, Line& L2){
                   assert(!L1.is_line && !L2.is_line);
```

```
            if(check_segment_intersection(L1, L2)) return float_type(0);
            float_type ans = distance_point_line(L1.a, L2);
            ans = min(ans, distance_point_line(L1.b, L2));
            ans = min(ans, distance_point_line(L2.a, L1));
            ans = min(ans, distance_point_line(L2.b, L1));
            return ans;
        }
    };

    struct Polygon{
        int n;
        vector<Point> p;
        Polygon() {};
        Polygon(int n) : n(n) { p.resize(n); }
        Polygon(vector<Point> points) {
            n = points.size();
            p = points;
        }

        friend ostream& operator << (ostream& os, const Polygon& Pol){
            for(int i = 0; i < Pol.n; i++) os << Pol.p[i] << '\n';
            return os;
        }
        friend istream& operator >> (istream& is, Polygon& Pol){
            for(int i = 0; i < Pol.n; i++) is >> Pol.p[i];
            return is;
        }

        auto& operator[](size_t i) { return p[i]; }
        auto const& operator[] (size_t i) const {return p[i];}

        float_type area(){
            float_type ans = 0;
            for(int i = 0; i < n; i++)
                ans += cross(p[i], p[(i + 1) % n]);
            return abs(ans) / 2;
        }

        bool is_convex(){
            bool ans = true;
            for(int i = 0; i < n; i++){
                ans &= (location(p[(i + 2) % n], p[i], p[(i + 1) % n]) != -1);
            }
            return ans;
        }

        // If the polygon is not convex, call convex_hull
        float_type diameter(){
            float_type ans = 0;
            for(int i = 0, j = n < 2 ? 0 : 1; i < j; i++){
                while(true){
                    ans = max(ans, norm(p[i] - p[j]));
                    int ii = (i + 1) % n;
                    int jj = (j + 1) % n;
                    if(sign(cross(p[ii] - p[i], p[jj] - p[j])) < 0) break;
                    j = jj;
                }
            }
            return sqrt(ans);
        }

        // 1: P inside Pol,  0: P in segment,  -1: P outside Pol
        friend int location_polygon(Polygon& Pol, Point P){
            Point inf_point(1e9 + 1, P.y + 1);
            int ans = 0;
            for(int i = 0; i < Pol.n; i++){
                Point& a = Pol.p[i], b = Pol.p[(i + 1) % Pol.n];
                if(location(P, a, b) == 0 && between(P, a, b)) return 0;
                ans += check_segment_intersection(Line(a, b), Line(P, inf_point));
            }
            return (ans & 1 ? 1 : -1);
        }

        // clockwise order
        friend Polygon convex_hull(vector<Point> p){
            int n = (int) p.size();
            sort(p.begin(), p.end());
            Polygon hull;
            for(int it = 0; it < 2; it++){
                int size = hull.p.size();
                for(int i = 0; i < n; i++){
                    while(hull.p.size() - size >= 2){
                        int s = hull.p.size();
                        if(cross(hull[s-1] - hull[s-2], p[i] - hull[s-2]) <= 0) break;
                        hull.p.pop_back();
                    }
                    hull.p.push_back(p[i]);
                }
                hull.p.pop_back();
                reverse(p.begin(), p.end());
            }
            return hull;
        }
    };

    // Point on segment PQ at a distance {dist} from {start}
    Point moveBy(Point& P, Point& Q, Point& start, float_type dist){
        assert(location(start, P, Q) == 0 && between(start, P, Q));
        assert(sign(abs(Q - start) - dist) >= 0);
        Point right = start + Point(1, 0);
        float_type angle = get_angle(right, start, Q);
        Point delta(cos(angle * PI / float_type(180.0)), sin(angle * PI /
float_type(180.0)));
        Point ans = start + dist * delta;
        return ans;
    }

    // Sorting points in counterclockwise order that starts from the half
line x <= 0, y = 0
    void sort_by_argument(vector<Point>& points){
        sort(points.begin(), points.end(), [&](const Point& A, const Point&
B){
            return atan2(A.y, A.x) < atan2(B.y, B.x);
        });
    }

    // Closest pair (Euclidean distance) O(N log N)
    pair<Point, Point> closest_pair(vector<Point> p){
        int n = p.size();
        sort(p.begin(), p.end());
        set<pair<coord_type, coord_type>> s;
        coord_type best = 9e18;
        int j = 0;
        pair<Point, Point> ans;
        for(int i = 0; i < n; i++){
            coord_type d = sqrt(best) + eps;
            while(j < i && p[i].x - p[j].x >= d){
                s.erase({p[j].y, p[j].x});
                j++;
            }
            auto range_left = s.lower_bound({p[i].y - d, p[i].x});
            auto range_right = s.upper_bound({p[i].y + d, p[i].x});
            for(auto it = range_left; it != range_right; it++){
                coord_type dx = p[i].x - it -> second;
                coord_type dy = p[i].y - it -> first;
                coord_type dist = dx * dx + dy * dy;
                if(dist < best){
                    best = dist;
                    ans.first = p[i];
                    ans.second = Point(it -> second, it -> first);
                }
            }
            s.insert({p[i].y, p[i].x});
        }
        return ans;
    }

}
```

```
using namespace geometry;
endsnippet

snippet waveletTree
int MXN = 1e6;
vector<int> sorted;
int N = -1;
bool compress = false;

${1}struct wavelet_tree{
    int low, high;
    wavelet_tree *left, *right;
    vector<int> b;

    wavelet_tree(vector<int>& A, bool large = true, int from = 0, int to = -
1, int x = 0, int y = MXN){
        if(to == -1) to = N = A.size();

        if(large && !compress){
            compress = true;
            sorted = A;
            sort(sorted.begin(), sorted.end());
            sorted.erase(unique(sorted.begin(), sorted.end()),
sorted.end());

            for(int i = 0; i < (int) A.size(); i++){
                A[i] = lower_bound(sorted.begin(), sorted.end(),
A[i]) - sorted.begin();
            }
            y = (int) sorted.size() - 1;
        }
        left = right = NULL;

        low = x, high = y;
        if(low == high or from >= to) return;

        int mid = (low + high) / 2;

        auto f = [mid](int x){ return x <= mid; };

        b.reserve(to - from + 1);
        b.push_back(0);

        for(int pos = from; pos < to; pos++) b.push_back(b.back() +
f(A[pos]));

        auto pivot = stable_partition(A.begin() + from, A.begin() + to, f)
- A.begin();
        left = new wavelet_tree(A, large, from, pivot, low, mid);
        right = new wavelet_tree(A, large, pivot, to, mid + 1, high);
        }

        int find_kth(int l, int r, int k){
                if(l > r) return 0;
                if(low == high) return low;
                int inLeft = b[r + 1] - b[l];
                int lb = b[l];
                int rb = b[r + 1];

                if(k <= inLeft) return this -> left -> find_kth(lb, rb - 1, k);
                return this -> right -> find_kth(l - lb, r - rb, k - inLeft);
        }

        int LTE(int l, int r, int k) {
                if(l > r or k < low) return 0;
                if(high <= k) return r - l + 1;

                int lb = b[l], rb = b[r + 1];
                return this -> left -> LTE(lb, rb - 1, k) + this -> right -> LTE(l
- lb, r - rb, k);
        }

        int count(int l, int r, int k) {
                if(l > r or k < low or k > high) return 0;
                if(low == high) return r - l + 1;

                int lb = b[l], rb = b[r + 1], mid = (low + high) / 2;

                if(k <= mid) return this -> left -> count(lb, rb - 1, k);
                return this -> right -> count(l - lb, r - rb, k);
        }

        int get_kth(int l, int r, int k){
                assert(l >= 0 && r < N && k >= 1 && k <= r - l + 1);
                int ans = find_kth(l, r, k);
                return ((int) sorted.size() == 0 ? ans : sorted[ans]);
        }

        int get_LTE(int l, int r, int k){
                assert(l >= 0 && r < N);
                if(sorted.size() != 0){
                        int pos = upper_bound(sorted.begin(), sorted.end(), k) -
sorted.begin() - 1;
                        if(pos == -1) return 0;
                        k = pos;
                }
                return LTE(l, r, k);
        }

        int get_count(int l, int r, int k){
                assert(l >= 0 && r < N);
        if(sorted.size() != 0){
            int pos = lower_bound(sorted.begin(), sorted.end(), k) -
sorted.begin();
            if(pos == sorted.size() || sorted[pos] != k) return 0;
            k = pos;
        }
        return count(l, r, k);
    }

    ~wavelet_tree(){
        delete left;
        delete right;
    }
};
endsnippet

snippet CountingPrimes "N <= 10^11 : 0.2ms. @@ No olvidar colocar init() en
main() @@ Verificado en https://judge.yosupo.jp/problem/counting_primes" b
const int MAXN = 100;
const int MAXM = 100010;
const int MAXP = 10000010;

int prime_cnt[MAXP];
int64_t dp[MAXN][MAXM];
bitset<MAXP> is_prime;
vector<int> primes;

void sieve(){
    is_prime[2] = true;
    for(int i = 3; i < MAXP; i += 2) is_prime[i] = true;

    for (int i = 3; i * i < MAXP; i += 2)
        for (int j = i * i; is_prime[i] && j < MAXP; j += (i << 1))
            is_prime[j] = false;

    for (int i = 1; i < MAXP; i++){
        prime_cnt[i] = prime_cnt[i - 1] + is_prime[i];
        if (is_prime[i]) primes.push_back(i);
    }
}

void init(){
    sieve();
    for (int m = 0; m < MAXM; m++) dp[0][m] = m;
    for (int n = 1; n < MAXN; n++)
        for (int m = 0; m < MAXM; m++)
            dp[n][m] = dp[n - 1][m] - dp[n - 1][m / primes[n - 1]];
}

int64_t phi(int64_t m, int n){
    if (n == 0) return m;
```

```
	if (m < MAXM && n < MAXN) return dp[n][m];
	if (1LL * primes[n - 1] * primes[n - 1] >= m && m < MAXP) return
prime_cnt[m] - n + 1;
	return phi(m, n - 1) - phi(m / primes[n - 1], n - 1);
}

// init() in main()
// m <= 10^11   -->  < 200ms
// counting primes from [1 : m]

int64_t lehmer(int64_t m){
	if (m < MAXP) return prime_cnt[m];

	int s = sqrt(0.5 + m), y = cbrt(0.5 + m);
	int a = prime_cnt[y];
	int64_t ans = phi(m, a) + a - 1;

	for (int i = a; primes[i] <= s; i++)
		ans = ans - lehmer(m / primes[i]) + lehmer(primes[i]) - 1;

	return ans;
}
endsnippet

snippet sieve1e9 "Sieve: N <= 10^9 ~ 0.5s" b
// credit: min_25
// takes 0.5s for n = 1e9
${1}vector<int> sieve(const int N, const int Q = 17, const int L = 1 << 15) {
	static const int rs[] = {1, 7, 11, 13, 17, 19, 23, 29};
	struct P {
		P(int p) : p(p) {}
		int p; int pos[8];
	};

	auto approx_prime_count = [] (const int N) -> int {
		return N > 60184 ? N / (log(N) - 1.1) : max(1., N / (log(N) -
1.11)) + 1;
	};

	const int v = sqrt(N), vv = sqrt(v);
	vector<bool> isp(v + 1, true);
	for (int i = 2; i <= vv; ++i) if (isp[i]) {
		for (int j = i * i; j <= v; j += i) isp[j] = false;
	}

	const int rsize = approx_prime_count(N + 30);
	vector<int> primes = {2, 3, 5}; int psize = 3;
	primes.resize(rsize);

	vector<P> sprimes; size_t pbeg = 0;
	int prod = 1;
```

```
    for (int p = 7; p <= v; ++p) {
        if (!isp[p]) continue;
        if (p <= Q) prod *= p, ++pbeg, primes[psize++] = p;
        auto pp = P(p);
        for (int t = 0; t < 8; ++t) {
            int j = (p <= Q) ? p : p * p;
            while (j % 30 != rs[t]) j += p << 1;
            pp.pos[t] = j / 30;
        }
        sprimes.push_back(pp);
    }

    vector<unsigned char> pre(prod, 0xFF);

    for (size_t pi = 0; pi < pbeg; ++pi) {
        auto pp = sprimes[pi]; const int p = pp.p;
        for (int t = 0; t < 8; ++t) {
            const unsigned char m = ~(1 << t);
            for (int i = pp.pos[t]; i < prod; i += p) pre[i] &= m;
        }
    }

    const int block_size = (L + prod - 1) / prod * prod;
    vector<unsigned char> block(block_size); unsigned char* pblock =
block.data();
    const int M = (N + 29) / 30;

    for (int beg = 0; beg < M; beg += block_size, pblock -= block_size) {
        int end = min(M, beg + block_size);
        for (int i = beg; i < end; i += prod)
            copy(pre.begin(), pre.end(), pblock + i);

        if (beg == 0) pblock[0] &= 0xFE;

        for (size_t pi = pbeg; pi < sprimes.size(); ++pi) {
            auto& pp = sprimes[pi];
            const int p = pp.p;
            for (int t = 0; t < 8; ++t) {
                int i = pp.pos[t]; const unsigned char m = ~(1 << t);
                for (; i < end; i += p) pblock[i] &= m;
                pp.pos[t] = i;
            }
        }

        for(int i = beg; i < end; ++i) {
            for (int m = pblock[i]; m > 0; m &= m - 1) {
                primes[psize++] = i * 30 + rs[__builtin_ctz(m)];
            }
        }
    }
    assert(psize <= rsize);
```

```
		while (psize > 0 && primes[psize - 1] > N) --psize;
		primes.resize(psize);
		return primes;
}
endsnippet

snippet LCA+RMQ "LCA + Sparse Table" b
typedef int rmq_type;
const int MAX_VALUE = 1E9 + 100;
vector<vector<int>> adj;
vector<int> rmq_init;

template<typename T>
${1}struct RMQ{
	int n = 0;
	vector<vector<T>> st;
	int levels;

	RMQ() {}

	RMQ(vector<T>& values){
		build(values);
	}

	int highest_bit(unsigned x){
		return x == 0 ? -1 : 31 - __builtin_clz(x);
	}

	void resize(int _n, int lev){
		st.resize(_n, vector<T>(lev + 1));
	}

	void modify(int pos, int level, int value){ // CHECK
		st[pos][level] = value;
	}

	void upd(int pos, int level){
		st[pos][level] = op(st[pos][level - 1], st[pos + (1 << (level - 1))][level - 1]);
	}

	void upd(int node, int ancestor, int level){
		st[node][level] = op(st[node][level - 1], st[ancestor][level - 1]);
	}

	void upd_level(int level){
		for(int pos = 0; pos <= n - (1 << level); pos++){
			upd(pos, level);
		}
	}
```

```
        T op(T& a, T& b){ // CHECK
                return min(a, b);
        }

        void build(vector<T>& values){
                n = values.size();
                levels = ceil(log2(n));
                resize(n, levels);

                for(int pos = 0; pos < n; pos++) modify(pos, 0, values[pos]);
                for(int i = 1; i <= levels; i++) upd_level(i);

        }

        // range [l..r], 0 <= l <= r < n
        // not overlapping O(log n)
        T query(int l, int r){
                if(l > r) swap(l, r);
                assert(0 <= l && r < n);
                int level = highest_bit(n);
                T ans = MAX_VALUE;
                for(int i = level; ~i; i--){
                        if((1 << i) <= r - l + 1){
                                ans = op(ans, st[l][i]);
                                l += (1 << i);
                        }
                }
                return ans;
        }

        //overlapping O(1)
        T query_over(int l, int r){
                if(l > r) swap(l , r);
                assert(0 <= l && r < n);
                int level = highest_bit(r - l + 1);
                return op(st[l][level], st[r - (1 <<level) + 1][level]);
        }
};

struct LCA{
        int n, l, timer;
        vector<vector<int>> up;
        vector<int> in, out, depth;
        RMQ<rmq_type> rmq;
        bool build_rmq = false;

        LCA(int _n, int root = 1, bool with_rmq = false){
                timer = 0, l = ceil(log2(_n));
                n = _n + 1;
                in.resize(n), out.resize(n);
                up.resize(n, vector<int>(l + 1));
```

```
	depth.resize(n);
	depth[root] = 0;
	build_rmq = with_rmq;
	if(build_rmq) rmq.resize(n, l);
	dfs(root, root);
}

void dfs(int v, int p){
	depth[v] = depth[p] + 1;
	in[v] = ++timer;
	up[v][0] = p; // [0] : parent

	if(build_rmq){ // rmq_init : 0-indexed, [0] : node
		rmq.modify(v, 0, rmq_init[v - 1]);
	}

	for(int i = 1; i <= l; i++){
		up[v][i] = up[up[v][i - 1]][i - 1];
		if(build_rmq) rmq.upd(v, up[v][i - 1], i);

	}
	for(int u : adj[v]){
		if(u == p) continue;
		dfs(u, v);
	}
	out[v] = ++timer;
}

bool is_ancestor(int u, int v){
	return in[u] <= in[v] && out[u] >= out[v];
}

int query(int u, int v){
	if(is_ancestor(u, v)) return u;
	if(is_ancestor(v, u)) return v;
	for(int i = l; ~i; i--){
		if(!is_ancestor(up[u][i], v)) u = up[u][i];
	}
	return up[u][0];
}

rmq_type query_rmq(int u, int v){ // CHECK
	assert(build_rmq);
	int anc = query(u, v);
	int dist_from_u = depth[u] - depth[anc];
	int dist_from_v = depth[v] - depth[anc];

	rmq_type ans;
	for(int i = 0; i <= l; i++){
		if(dist_from_u & (1 << i)){
			//ans = merge_operation(ans, rmq.st[u][i]);
```

```
                u = up[u][i];
            }
        }

        for(int i = 0; i <= l; i++){
            if(dist_from_v & (1 << i)){
                //ans = merge_operation(ans, rmq.st[v][i]);
                v = up[v][i];
            }
        }
        // ans = merge_operation(ans, rmq.st[anc][0]);
        return ans;
    }

    int ancestor(int x, int k){
        assert(depth[x] >= k);
        for(int i = 0; i <= l; i++){
            if(k & (1 << i)) x = up[x][i];
        }
        return x;
    }

    int distance(int u, int v){
        return depth[u] + depth[v] - 2 * depth[query(u, v)];
    }
};
endsnippet

snippet PHI "Precalcular la funcion phi de Euler" b
const int MXN = 1e5 + 1;
vector<int> phi(MXN);

void run_phi(){
    iota(phi.begin(), phi.end(), 0);
    for(int i = 2; i < MXN; i++){
        if(phi[i] != i) continue;
        for(int j = i; j < MXN; j += i){
            phi[j] -= phi[j] / i;
        }
    }
}
endsnippet

snippet Zfunction "Funcion Z: z[i] mayor prefijo empezando en i que es prefijo de S" b
${1}vector<int> Z(string& s){
    int N = (int) s.size();
    vector<int> z(N);
    z[0] = N;

    int l = 0, r = 0;
```

```
	for(int i = 1; i < N; i++){
		if(i < r) z[i] = min(r - i, z[i - l]);
		while(i + z[i] < N && s[z[i]] == s[i + z[i]]) z[i]++;
		if(i + z[i] > r){
			l = i, r = i + z[i];
		}
	}
	return z;
}
endsnippet

snippet Trie "Modificar dependiendo de la inserci├│n (vector o cadena)" b

const int MXN = 1e6 + 1;
const int bits = 30;

int trie[MXN][bits];
int next_node;

//int cnt[MXN];
//bool end[MXN];

void insert(string& s, int add){
	int node = 0;
	for(auto c : s){
		if(::trie[node][c - '0'] == 0){
			::trie[node][c - '0'] = ++next_node;
		}
		node = ::trie[node][c - '0'];
		// cnt[node] += add;
	}
	// end[node] = true;
}
endsnippet

snippet Manacher "Centro en i: 2 * len[2*i] + 1, Centro entre i e i+1: 2 * len[2*i + 1]. Ver: https://judge.yosupo.jp/submission/183016" b
vector<int> manacher(const string &s) {
	int n = (int) s.size();
	vector<int> len(2 * n - 1, 0);

	int l = -1, r = -1;
	for (int center = 0; center < 2 * n - 1; center++) {
		int i = (center + 1) / 2;
		int j = center / 2;

		if(i < r) len[center] = min(r - i, len[2 * (l + r) - center]);

		while (j + len[center] + 1 < n && i - len[center] - 1 >= 0) {
			if (s[j + len[center] + 1] != s[i - len[center] - 1]) break;
			len[center]++;
```

```
            if (j + len[center] > r) {
                l = i - len[center];
                r = j + len[center];
            }
        }
    }
    return len;
}
endsnippet

snippet SuffixArray-InducedSort "SuffixArray usando InducedSort" b
// Source: https://github.com/ShahjalalShohag/code-library/blob/main/Strings/Suffix%20Array.cpp

${1}void induced_sort(const vector<int> &vec, int val_range, vector<int> &SA, const vector<bool> &sl, const vector<int> &lms_idx) {
    vector<int> l(val_range, 0), r(val_range, 0);
    for (int c : vec) {
        if (c + 1 < val_range) ++l[c + 1];
        ++r[c];
    }
    partial_sum(l.begin(), l.end(), l.begin());
    partial_sum(r.begin(), r.end(), r.begin());
    fill(SA.begin(), SA.end(), -1);

    for (int i = lms_idx.size() - 1; i >= 0; --i) SA[--r[vec[lms_idx[i]]]] = lms_idx[i];
    for (int i : SA){
        if (i >= 1 && sl[i - 1]) SA[l[vec[i - 1]]++] = i - 1;
    }

    fill(r.begin(), r.end(), 0);
    for (int c : vec) ++r[c];

    partial_sum(r.begin(), r.end(), r.begin());

    for (int k = SA.size() - 1, i = SA[k]; k >= 1; --k, i = SA[k]){
        if (i >= 1 && !sl[i - 1]) SA[--r[vec[i - 1]]] = i - 1;
    }
}

${2}vector<int> SA_IS(const vector<int> &vec, int val_range) {
    const int n = vec.size();
    vector<int> SA(n), lms_idx;
    vector<bool> sl(n);

    sl[n - 1] = false;
    for (int i = n - 2; i >= 0; --i) {
        sl[i] = (vec[i] > vec[i + 1] || (vec[i] == vec[i + 1] && sl[i +
1]));
        if (sl[i] && !sl[i + 1]) lms_idx.push_back(i + 1);
    }

    reverse(lms_idx.begin(), lms_idx.end());
    induced_sort(vec, val_range, SA, sl, lms_idx);
    vector<int> new_lms_idx(lms_idx.size()), lms_vec(lms_idx.size());

    for (int i = 0, k = 0; i < n; ++i){
        if (!sl[SA[i]] && SA[i] >= 1 && sl[SA[i] - 1])
            new_lms_idx[k++] = SA[i];
    }

    int cur = 0;
    SA[n - 1] = cur;

    for (size_t k = 1; k < new_lms_idx.size(); ++k) {
        int i = new_lms_idx[k - 1], j = new_lms_idx[k];
        if (vec[i] != vec[j]) {
            SA[j] = ++cur;
            continue;
        }
        bool flag = false;
        for (int a = i + 1, b = j + 1;; ++a, ++b) {
            if (vec[a] != vec[b]) {
                flag = true;
                break;
            }
            if ((!sl[a] && sl[a - 1]) || (!sl[b] && sl[b - 1])) {
                flag = !((!sl[a] && sl[a - 1]) && (!sl[b] && sl[b -
1]));

                break;
            }
        }
        SA[j] = (flag ? ++cur : cur);
    }

    for (size_t i = 0; i < lms_idx.size(); ++i) lms_vec[i] = SA[lms_idx[i]];

    if (cur + 1 < (int)lms_idx.size()) {
        auto lms_SA = SA_IS(lms_vec, cur + 1);
        for (size_t i = 0; i < lms_idx.size(); ++i) {
            new_lms_idx[i] = lms_idx[lms_SA[i]];
        }
    }
    induced_sort(vec, val_range, SA, sl, new_lms_idx);
    return SA;
}

${3}vector<int> suffix_array(const string &s, const int LIM = 128) {
    vector<int> vec(s.size() + 1);
```

```
        copy(begin(s), end(s), begin(vec));
        vec.back() = '$';
        auto ret = SA_IS(vec, LIM);
        ret.erase(ret.begin());
        return ret;
    }

    ${4}struct SuffixArray {
        int n;
        string s;
        vector<int> sa, rank, lcp;
        static const int LG = 18;
        vector<vector<int>> t;
        vector<int> lg;

        SuffixArray() {}
        SuffixArray(string _s) {
            n = _s.size();
            s = _s;
            sa = suffix_array(s); // O(N)
            rank.resize(n);
            for (int i = 0; i < n; i++) rank[sa[i]] = i;
            costruct_lcp(); // O(N)

            /*
            * O(N log N)
            * prec();
            * build();
            */
        }

        void costruct_lcp() {
            int k = 0;
            lcp.resize(n - 1, 0);
            for (int i = 0; i < n; i++) {
                if (rank[i] == n - 1) {
                    k = 0;
                    continue;
                }
                int j = sa[rank[i] + 1];
                while (i + k < n && j + k < n && s[i + k] == s[j + k])  k++;
                lcp[rank[i]] = k;
                if (k)  k--;
            }
        }

        void prec() {
            lg.resize(n, 0);
            for (int i = 2; i < n; i++) lg[i] = lg[i / 2] + 1;
        }
        void build() {
            int sz = n - 1;
            t.resize(sz);
            for (int i = 0; i < sz; i++) {
                t[i].resize(LG);
                t[i][0] = lcp[i];
            }
            for (int k = 1; k < LG; ++k) {
                for (int i = 0; i + (1 << k) - 1 < sz; ++i) {
                    t[i][k] = min(t[i][k - 1], t[i + (1 << (k - 1))][k - 1]);
                }
            }
        }

        int query(int l, int r) { // minimum of lcp[l], ..., lcp[r]
            int k = lg[r - l + 1];
            return min(t[l][k], t[r - (1 << k) + 1][k]);
        }

        int get_lcp(int i, int j) { // lcp of suffix starting from i and j
            if (i == j) return n - i;
            int l = rank[i], r = rank[j];
            if (l > r) swap(l, r);
            return query(l, r - 1);
        }

        int lower_bound(string &t) {
            int l = 0, r = n - 1, k = t.size(), ans = n;
            while (l <= r) {
                int mid = l + r >> 1;
                if (s.substr(sa[mid], min(n - sa[mid], k)) >= t) ans = mid, r = mid - 1;
                else l = mid + 1;
            }
            return ans;
        }

        int upper_bound(string &t) {
            int l = 0, r = n - 1, k = t.size(), ans = n;
            while (l <= r) {
                int mid = l + r >> 1;
                if (s.substr(sa[mid], min(n - sa[mid], k)) > t) ans = mid, r = mid - 1;
                else l = mid + 1;
            }
            return ans;
        }

        // occurrences of s[p, ..., p + len - 1]
        pair<int, int> find_occurrence(int p, int len) {
            p = rank[p];
            pair<int, int> ans = {p, p};
            int l = 0, r = p - 1;
            while (l <= r) {
                int mid = l + r >> 1;
                if (query(mid, p - 1) >= len) ans.first = mid, r = mid - 1;
                else l = mid + 1;
            }
            l = p + 1, r = n - 1;
            while (l <= r) {
                int mid = l + r >> 1;
                if (query(p, mid - 1) >= len) ans.second = mid, l = mid + 1;
                else r = mid - 1;
            }
            return ans;
        }
};
endsnippet

snippet highest_base "En la posici├│n x est├í el n├║mero A tal que A^x <= inf
(inf = 1e18)" b
const vector<long long> highest_base = {0, 1000000000000000000, 1000000000,
1000000, 31622, 3981, 1000, 372, 177, 100, 63, 43, 31, 24, 19, 15, 13, 11, 10,
8, 7, 7, 6, 6, 5, 5, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1};
endsnippet

snippet sequence "Referencia: https://atcoder.jp/contests/abc324/tasks/abc324_g"
b
struct Element{
    int index, value;

    Element() { index = -1, value = -1; }
    Element(int index, int value) : index(index), value(value) {}
};

struct cmp_index{
    bool operator() (const Element& a, const Element& b) const {
        return a.index < b.index;
    }
};

struct cmp_value{
    bool operator() (const Element& a, const Element& b) const {
        if(a.value != b.value) return a.value < b.value;
        return a.index < b.index;
    }
};

${1}struct sequence{
    set<Element, cmp_index> groups_by_idx;
    set<Element, cmp_value> groups_by_value;

    int size() const { return (int) groups_by_value.size(); }

    void swap(sequence& other){
        groups_by_idx.swap(other.groups_by_idx);
        groups_by_value.swap(other.groups_by_value);
    }

    void erase(const Element& x){
        groups_by_idx.erase(x);
        groups_by_value.erase(x);
    }

    void insert(const Element& x){
        groups_by_idx.insert(x);
        groups_by_value.insert(x);
    }

    int count_grater(int x){
        auto left = groups_by_value.begin();
        auto right = groups_by_value.end();
        int ans = 0;
        while(true){
            if(left -> value > x) {
                return size() - ans;
            }
            left++;
            right--;

            if(right -> value <= x){
                return ans;
            }
            ans++;
        }
        assert(false);
    }
};

void split_index(sequence& A, sequence& B, int left){
    if(left == 0){
        A.swap(B);
        return;
    }
    if(left >= A.size()) return;

    if(left * 2 <= A.size()){
        while(B.size() < left){
            auto element = *A.groups_by_idx.begin();
            A.erase(element);
            B.insert(element);
```

```
		}
		A.swap(B);
	}else{
		while(A.size() > left){
			auto element = *A.groups_by_idx.rbegin();
			A.erase(element);
			B.insert(element);
		}
	}
}

void split_value(sequence& A, sequence& B, int x){
	if(A.size() == 0) return;

	int cnt = A.count_grater(x);

	if(cnt == 0) return;
	if(cnt == A.size()){ A.swap(B); return; }

	cnt = A.size() - cnt;

	if(cnt * 2 <= A.size()){
		while(B.size() < cnt){
			auto element = *A.groups_by_value.begin();
			A.erase(element);
			B.insert(element);
		}
		A.swap(B);
	}else{
		while(A.size() > cnt){
			auto element = *A.groups_by_value.rbegin();
			A.erase(element);
			B.insert(element);
		}
	}
}
endsnippet

snippet extgcd "Solving ax + by = gcd(a, b)" b
long long extgcd(long long a, long long b, long long& x, long long& y){
	if(a == 0){
		x = 0;
		y = 1;
		return b;
	}
	long long x1, y1;
	long long d = extgcd(b % a, a, x1, y1);
	x = y1 - (b / a) * x1;
	y = x1;
	return d;
}
```

```
endsnippet

snippet diophantine "Resolviendo ecuaciones diofanticas" b
// Find minimum x such that Ax = B mod M
long long solve_cong(long long A, long long B, long long M){
	A = (A % M + M) % M;
	B = (B % M + M) % M;

	long long g = __gcd(A, M);

	if(B % g != 0) return -1;

	A /= g;
	M /= g;
	B /= g;

	long long x, y;
	g = extgcd(A, M, x, y);

	x = (x % M + M) % M;
	x = (x * B) % M;
	x %= M / g;

	return x;
};
endsnippet

snippet Matrix "Matrix class" b
template <class T>
${1}struct Matrix{
	int row = -1, col = -1;
	vector<vector<T>> A;

	Matrix() {
		row = col = 2;
		A = {{1, 0}, {0, 1}};
	}

	Matrix(int n, int m = -1){
		if(m == -1) m = n;
		A.resize(n, vector<T>(m));
		row = n, col = m;
	}

	Matrix(vector<T>& X){
		assert((int) X.size() > 0);
		row = 1;
		col = (int) X.size();
		A.resize(row, vector<T>(col));

		for(int i = 0; i < col; i++) A[0][i] = X[i];
}

Matrix(vector<vector<T>>& X){
        assert((int) X.size() > 0);
        row = (int) X.size();
        col = (int) X[0].size();
        assert((int) X[0].size() > 0);
        A.resize(row, vector<T>(col));

        for(int i = 0; i < row; i++)
                for(int j = 0; j < col; j++)
                        A[i][j] = X[i][j];
}

auto& operator[](size_t i) { return A[i]; }
auto const& operator[] (size_t i) const {return A[i];}

Matrix& operator += (const Matrix& other){
        assert(row == other.row && col == other.col);
        for(int i = 0; i < row; i++){
                for(int j = 0; j < col; j++){
                        A[i][j] += other[i][j];
                }
        }
        return *this;
}

Matrix& operator -= (const Matrix& other){
        assert(row == other.row && col == other.col);
        for(int i = 0; i < row; i++){
                for(int j = 0; j < col; j++){
                        A[i][j] -= other[i][j];
                }
        }
        return *this;
}

Matrix& operator*=(const Matrix& other){
        assert(col == other.row);
        vector<vector<T>> B(row, vector<T>(other.col));
        for(int i = 0; i < row; i++){
                for(int j = 0; j < col; j++){
                        for(int k = 0; k < other.col; k++){
                                B[i][k] += A[i][j] * other[j][k];
                        }
                }
        }
        A = B;
        return *this;
}
```

```
void setUnit(){
    assert(row != -1 && col != -1 && row == col);
    for(int i = 0; i < row; i++){
        this -> A[i][i] = 1;
    }
}
T det(){
    assert(row == col);
    Matrix<T> y = A;
    T ans = 1;
    for(int i = 0; i < y.row; i++){
        if(y[i][i] == 0){
            bool found = false;
            for(int j = i + 1; j < y.row; j++){
                if(y[j][i] != 0){
                    swap(y[i], y[j]);
                    ans = -ans;
                    found = true; break;
                }
            }
            if(!found) return 0;
        }
        for(int j = i + 1; j < y.row; j++){
            T ratio = y[j][i] / y[i][i];
            for(int k = i; k < y.row; k++){
                y[j][k] -= ratio * y[i][k];
            }
        }
        ans *= y[i][i];
    }
    return ans;
}

friend Matrix operator + (Matrix<T> X, Matrix<T> Y){ return X += Y; }
friend Matrix operator - (Matrix<T> X, Matrix<T> Y){ return X -= Y; }
friend Matrix operator * (Matrix<T> X, Matrix<T> Y){ return X *= Y; }
friend bool operator == (Matrix<T> X, Matrix<T> Y){
    if(X.row != Y.row || X.col != Y.col) return false;
    for(int i = 0; i < X.row; i++)
        for(int j = 0; j < X.col; j++)
            if(X[i][j] != Y[i][j])
                return false;
    return true;
}
friend bool operator != (Matrix<T> X, Matrix<T> Y){ return !(X == Y); }
friend istream& operator >> (istream& in, Matrix<T>& c){
    for(int i = 0; i < c.row; i++)
        for(int j = 0; j < c.col; j++)
            in >> c[i][j];
    return in;
}
```

```
    friend ostream& operator << (ostream& out, Matrix<T>& c){
        for(int i = 0; i < c.row; i++)
            for(int j = 0; j < c.col; j++)
                out << c[i][j] << " \n"[j == c.col - 1];
        return out;

    }

    friend Matrix<T> join(Matrix<T> X, Matrix<T> Y){
        assert(X.row == Y.row);
        Matrix<T> ret(X.row, X.col + Y.col);
        for(int i = 0; i < X.row; i++){
            for(int j = 0; j < X.col + Y.col; j++){
                ret[i][j] = (j < X.col ? X[i][j] : Y[i][j - X.col]);
            }
        }
        return ret;
    }

    friend Matrix<T> power(Matrix<T> x, long long k){
        assert(x.row == x.col);
        Matrix<T> ans(x.row, x.col); ans.setUnit();
        while(k > 0){
            if(k & 1) ans *= x;
            k >>= 1LL; x *= x;
        }
        return ans;
    }

    friend pair<int, Matrix<T>> gauss(Matrix<T> x, Matrix<T>& ans){
        int n = (int) x.row; assert(n > 0);
        int m = (int) x.col - 1;

        vector<int> where(m, -1);

        auto abs1 = [&](T x){ return (x < 0 ? -x : x); };
        auto is_zero = [&](T x){
            if(std::is_same<T, double>::value){ double eps = 1e9; return
x < eps; }
            if(std::is_same<T, long double>::value){ long double eps =
1e9; return x < eps; }
            return x == 0;
        };

        int free = 0;
        int col = 0, row = 0;
        for(; col < m && row < n; col++){
            int max_row = row;
            for(int i = row + 1; i < n; i++){
                if(abs1(x[i][col]) > abs1(x[max_row][col]))
                    max_row = i;
            }
            if(is_zero(x[max_row][col])) {
                free++; continue;
            }
            if(max_row != row){
                for(int i = col; i <= m; i++)
                    swap(x[max_row][i], x[row][i]);
            }
            where[col] = row;
            for(int i = 0; i < n; i++){
                if(i == row) continue;
                T ratio = x[i][col] / x[row][col];
                for(int j = col; j <= m; j++)
                    x[i][j] -= x[row][j] * ratio;
            }
            row++;
        }
        while(col < m){ col++; free++; }

        ans = Matrix<T>(m, 1);

        for(int i = 0; i < m; i++){
            if(where[i] != -1)
                ans[i][0] = x[where[i]][m] / x[where[i]][i];
        }

        Matrix<T> base(0);
        for(int i = 0; i < n; i++){
            T sum = 0;
            for(int j = 0; j < m; j++)
                sum += ans[j][0] * x[i][j];

            if(!is_zero(sum - x[i][m])) return {0, base};
        }
        if(free == 0) return {1, base};

        base = Matrix<T>(free, m);
        int base_row = 0;
        for(int i = 0; i < m; i++){
            if(where[i] == -1) {
                for(int j = 0; j < m; j++){
                    if(i == j) base[base_row][j] = 1;
                    else if(where[j] == -1) base[base_row][j] = 0;
                    else{
                        base[base_row][j] -= x[where[j]][i] /
x[where[j]][j];
                    }
                }
                base_row++;
            }
        }
        return {2, base};
    }

    // Verification: https://judge.yosupo.jp/problem/system_of_linear_equations
    friend tuple<int, Matrix<T>, Matrix<T>> solve(Matrix<T> equations, Matrix<T> solutions){
        assert(equations.row == solutions.row && solutions.col == 1);
        Matrix<T> ans;
        auto [cnt, base] = gauss(join(equations, solutions), ans);
        return {cnt, base, ans};
    }
};
endsnippet

snippet erase_ordered_set "Funcion para eliminar en un ordered set" b
template <class T>
void oserase(ordered_set<T>& os, T val){
    int pos = os.order_of_key(val);
    auto it = os.find_by_order(pos);
    os.erase(it);
}
endsnippet

snippet area_of_union_of_rectangles "Suma de areas de rectangulos (0 <= l, r <= 1e9)" b
vector<int> y;

template<typename T>
${1}struct lazy_segment_tree{
    int n;
    const int EMPTY = 0; // CHECK
    vector<T> tree;
    vector<int> lazy;

    lazy_segment_tree(int n){
        this -> n = n;
        tree.resize(4 * n);
        lazy.resize(4 * n);
    }

    T compare(T& a, T& b){
        T ret;
        if(a.first < b.first) ret = a;
        else if(a.first > b.first) ret = b;
        else {
            ret = a;
            ret.second += b.second;
        }
        return ret;
    }
    void push_op(T& a, int& lazy_value){
        a.first += lazy_value;
    }

    void lazy_op(int& lazy_child, int& lazy_parent){
        lazy_child += lazy_parent; // CHECK
    }

    void push(int cur){
        if(lazy[cur] == EMPTY) return;
        push_op(tree[2 * cur], lazy[cur]);
        push_op(tree[2 * cur + 1], lazy[cur]);
        lazy_op(lazy[2 * cur], lazy[cur]);
        lazy_op(lazy[2 * cur + 1], lazy[cur]);
        lazy[cur] = EMPTY;
    }

    void build(int cur = 1, int left_range = 0, int right_range = -1){
        if(right_range == -1) right_range = n - 1;
        lazy[cur] = EMPTY;
        if(left_range == right_range){
            if(left_range >= 0 && left_range < n) {
                tree[cur] = {0, y[left_range + 1] - y[left_range]};
            }
            return;
        }
        int mid_range = (left_range + right_range) / 2;
        build(2 * cur, left_range, mid_range);
        build(2 * cur + 1, mid_range + 1, right_range);
        tree[cur] = compare(tree[2 * cur], tree[2 * cur + 1]);
    }

    // 0-indexed
    void upd(int first, int last, int value, int cur = 1, int left_range = 0,
int right_range = -1){
        if(right_range == -1) right_range = n - 1;
        if(first > last) return;
        if(first == left_range && right_range == last){
            push_op(tree[cur], value);
            lazy_op(lazy[cur], value);
            return;
        }
        push(cur);

        int mid_range = (left_range + right_range) / 2;
        upd(first, min(last, mid_range), value, 2 * cur, left_range,
mid_range);
        upd(max(first, mid_range + 1), last, value, 2 * cur + 1, mid_range
+ 1, right_range);
        tree[cur] = compare(tree[2 * cur], tree[2 * cur + 1]);
    }

    T query(int first, int last, int cur = 1, int left_range = 0, int
right_range = -1){
        if(right_range == -1) right_range = n - 1;
        if(first > last) return {-1e9, 0};
        if(left_range > last || right_range < first) return {-1e9, 0};
        if(first == left_range && right_range == last) {
            return tree[cur];
        }

        push(cur);
        int mid_range = (left_range + right_range) / 2;
        T ql = query(first, min(last, mid_range), 2 * cur, left_range,
mid_range);
        T qr = query(max(first, mid_range + 1), last, 2 * cur + 1,
mid_range + 1, right_range);
        return compare(ql, qr);
    }
};

// rect[i] = {l, r, d, u}
int64_t area_of_union_of_rectangles(vector<array<int, 4>>& rect){
    int n = (int) rect.size();
    if(!n) return 0;

    struct Event{
        int x, ly, ry;
        bool is_add;
        bool operator<(const Event& e) { return x < e.x; }
    };

    for(int i = 0; i < n; i++){
        y.push_back(rect[i][2]);
        y.push_back(rect[i][3]);
    }
    sort(y.begin(), y.end());
    y.erase(unique(y.begin(), y.end()), y.end());

    for(int i = 0; i < n; i++){
        rect[i][2] = lower_bound(y.begin(), y.end(), rect[i][2]) -
y.begin();
        rect[i][3] = lower_bound(y.begin(), y.end(), rect[i][3]) -
y.begin();
    }

    vector<Event> e;
    for(auto [l, r, d, u] : rect){
        e.push_back({l, d, u, true});
        e.push_back({r, d, u, false});
    }
    sort(e.begin(), e.end());

    int Y = (int) y.size();

    lazy_segment_tree<pair<int, int>> lst(Y - 1);
    lst.build();

    int64_t ans = 0;
    int last_x = 0;
    int total = y.back() - y[0];

    for(int i = 0; i < 2 * n; i++){
        auto [x, d, u, is_add] = e[i];

        auto [mn, mn_cnt] = lst.query(0, Y - 2);

        int dx = x - last_x;

        if(mn == 0) ans += 1LL * dx * (total - mn_cnt);
        else ans += 1LL * dx * total;

        last_x = x;
        if(is_add) lst.upd(d, u - 1, 1);
        else lst.upd(d, u - 1, -1);
    }

    return ans;
}
endsnippet

snippet ArticulationPoints "Find all articulation Points or Cutpoints" b
vector<vector<int>> adj;
vector<bool> used;
vector<int> in, low;
vector<bool> is_cutpoint;
int timer = 0;

void dfs_cutpoints(int u, int p = -1){
    used[u] = true;
    low[u] = in[u] = ++timer;
    int children = 0;
    for(int v : adj[u]){
        if(v == p) continue;
        if(used[v]) low[u] = min(low[u], in[v]);
        else {
            dfs_cutpoints(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] >= in[u] && p != -1){
                is_cutpoint[u] = true;
            }
            children++;
```

```
		}
	}
	if(p == -1 && children > 1){
		is_cutpoint[u] = true;
	}
}

void find_cutpoints(){
	const int N = adj.size();

	used = vector<bool>(N);
	in = vector<int>(N);
	low = vector<int>(N);
	is_cutpoint = vector<bool>(N);


	for(int u = 1; u < N; u++){
		if(used[u]) continue;
		dfs_cutpoints(u);
	}
}
endsnippet

snippet Montgomery_multiplication "https://judge.yosupo.jp/submission/201217" b
// https://github.com/maspypy/library/blob/main/mod/mongomery_modint.hpp
template<typename uintx_t, typename uint2x_t>
${1}class montgomery_multiplication {
	public:
	constexpr montgomery_multiplication() {}
	constexpr montgomery_multiplication(uintx_t _md) { set_mod(_md); }

	constexpr void set_mod(uintx_t _md) {
		assert(_md & 1 && _md <= uintx_t(1) << (w - 2));
		md = _md; n2 = -static_cast<uint2x_t>(md) % md; r = md;
		for (int i = 0; i < 5; i++) r *= 2 - md * r;
		r = -r;
	}

	// Assumptions: '0 <= x, y < 2 * mod'
	constexpr uintx_t mul(uintx_t x, uintx_t y) const { return
reduce(static_cast<uint2x_t>(x) * y); }

	// Assumptions: '0 <= a < 2 * mod'
	template<typename T>
	constexpr uintx_t pow(uintx_t a, T b) const {
		uintx_t ans = convert(1);
		for (; b; b >>= 1, a = mul(a, a)) if (b & 1) ans = mul(ans, a);
		return ans;
	}

	constexpr bool equiv(uintx_t x, uintx_t y) const { return (x >= md ? x -
md : x) == (y >= md ? y - md : y); }

        constexpr uintx_t mod() const { return md; }

        constexpr uintx_t convert(uintx_t x) const { return
reduce(static_cast<uint2x_t>(x) * n2); }

        constexpr uintx_t reduce(uint2x_t x) const { return (x +
static_cast<uint2x_t>(static_cast<uintx_t>(x) * r) * md) >> w; }

        constexpr uintx_t val(uintx_t x) const {
                uintx_t y = reduce(x);
                return y >= md ? y - md : y;
        }

        private:
        static constexpr int w = std::numeric_limits<uintx_t>::digits;

        uintx_t md, r, n2;
};

using montgomery_multiplication_32 = montgomery_multiplication<uint32_t,
uint64_t>;
using montgomery_multiplication_64 = montgomery_multiplication<uint64_t,
__uint128_t>;

template<typename uintx_t, typename uint2x_t, int id>
montgomery_multiplication<uintx_t, uint2x_t> global_montgomery_multiplication;
template<int id> montgomery_multiplication_32
&global_montgomery_multiplication_32 =
global_montgomery_multiplication<uint32_t, uint64_t, id>;
template<int id> montgomery_multiplication_64
&global_montgomery_multiplication_64 =
global_montgomery_multiplication<uint64_t, __uint128_t, id>;
endsnippet

snippet Brent'sAlgoritm "Mejora de Pollard's Rho creo :p" b
cuidado con n = 1
template<typename T>
${1}constexpr T nontrivial_divisor(T n, const T &x0 = 2, T c = 1) {
        if (!(n & 1)) return 2;

        using uintx_t  = std::conditional_t<(sizeof(T) <= 4), uint32_t,
uint64_t>;
        using uint2x_t = std::conditional_t<(sizeof(T) <= 4), uint64_t,
__uint128_t>;

        const montgomery_multiplication<uintx_t, uint2x_t> mm(n);

        #define abs_diff(x, y) ((x) > (y) ? (x) - (y) : (y) - (x))
```

```
	for (; ; c++) {
		uintx_t x = mm.convert(x0), g = 1, q = mm.convert(1), xs, y;

		auto f_eq = [&mm, &n, &c](uintx_t &a) -> void {
			a = mm.reduce(static_cast<uint2x_t>(a) * a + c);
		};

		int m = 1 << 7, l = 1;
		while (g == 1) {
			y = x;
			for (int i = 0; i < l; i++) f_eq(x);
			int k = 0;
			while (k < l && g == 1) {
				xs = x;
				for (int i = 0; i < m && i < l - k; i++) {
					f_eq(x);
					q = mm.mul(q, abs_diff(mm.val(x), mm.val(y)));
					}
					g = __gcd((uintx_t) mm.val(q), (uintx_t)n);
					k += m;
			}
			l <<= 1;
		}
		if (g == static_cast<uintx_t>(n)) {
			do {
				f_eq(xs);
				g = __gcd((uintx_t) abs_diff(mm.val(xs), mm.val(y)),
(uintx_t) n);
			} while (g == 1);
		}
		if (g != 1 && g != static_cast<uintx_t>(n)) return g;
	}

	#undef abs_diff

	__builtin_unreachable();
}
endsnippet
```

```
snippet Millerrabin "Test de primalidad, N <= 1e18" b
cuidado con n = 1
template<typename T>
```

```
${1}constexpr T millerrabin(T n){
	if(n < 2) return false;

	if(n == 2) return true;
	if(n % 2 == 0) return false;

	using uintx_t  = std::conditional_t<(sizeof(T) <= 4), uint32_t,
uint64_t>;
	using uint2x_t = std::conditional_t<(sizeof(T) <= 4), uint64_t,
__uint128_t>;

	const montgomery_multiplication<uintx_t, uint2x_t> mm(n);

	int r = 0;
	uintx_t d = n - 1;
	while ((d & 1) == 0) {
		d >>= 1;
		r++;
	}

	auto fast_pow = [&mm](uintx_t a, uintx_t p){
		uintx_t res = mm.convert(1);
		while(p > 0){
			if(p & 1) res = mm.mul(res, a);
			a = mm.mul(a, a);
			p >>= 1LL;
		}
		return mm.val(res);
	};

	auto check_composite = [&](uintx_t a, int s) {
		uintx_t x = fast_pow(a, d);

		if (x == 1 || x == n - 1) return false;

		x = mm.convert(x);
		for (int r = 1; r < s; r++) {
			x = mm.mul(x, x);
			if (mm.val(x) == n - 1)
				return false;
		}
		return true;
	};

	for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
		if (n == a)
			return true;
		if (check_composite(a, r))
			return false;
	}
	return true;
```

```
}
endsnippet

snippet factorize-N1e18 "Factorizar hasta 1e18.
https://judge.yosupo.jp/submission/201217" b
template<typename T>
${1}vector<T> factorize(T n){
	vector<T> ans;
	if(n == 1) return ans;

	ans = {n};
	for(int i = 0; i < ans.size(); i++){
		while(true){
			if(ans[i] < (long long) 1e6){
				if(lp[ans[i]] == ans[i]) break;
			}else if(millerrabin(ans[i])) break;

			// ans[i] is not prime

			long long d = (n > (long long) MXN ?
nontrivial_divisor(ans[i]) : (long long) lp[ans[i]]);
			ans[i] /= d;
			ans.push_back(d);
		}
	}
	sort(ans.begin(), ans.end());
	return ans;

}
endsnippet

snippet EulerianCycle "Retorna ciclo euleriano empezando y terminando en 1" b
${1}vector<int> get_eulerian_cycle(vector<vector<int>>& adj){
	int n = adj.size();
	vector<int> ans;

	vector<set<int>> other_adj(n);
	int start = 1, odd = 0;

	for(int i = 1; i < n; i++){
		if((int) adj[i].size() % 2 == 1){
			odd++;
		}
		other_adj[i] = set<int>(adj[i].begin(), adj[i].end());
	}
	if(odd > 0){
		return {};
	}

	stack<int> st;
	st.push(start);
	while(!st.empty()){
		int u = st.top();
		if(other_adj[u].empty()){
			ans.push_back(u);
			st.pop();
		}else{
			int v = *other_adj[u].begin();
			other_adj[u].erase(other_adj[u].begin());
			other_adj[v].erase(u);
			st.push(v);
		}
	}
	for(int i = 1; i < n; i++){
		if(other_adj[i].size() > 0){
			return {};
		}
	}
	return ans;
}
endsnippet

snippet LinkCutTree "Basically Link cut tree xd" b
template<typename T>
${1}struct link_cut_tree{
	struct Node{
		Node *parent;
		array<Node*, 2> child;
		bool rev;
		T value;
		T siz, vir;

		Node() {}

		Node(T value) : value(value), rev(false), siz(1), vir(0) {
			parent = child[0] = child[1] = nullptr;
		}

		bool is_root() const {
			return (!parent || (parent -> child[0] != this && parent ->
child[1] != this));
		}
	};

	vector<Node> nodes;

	link_cut_tree(int n){
		nodes.resize(n + 1);
		for(int u = 1; u <= n; u++) nodes[u] = Node(u);
	}
```

```
int side(Node *x) {
       Node* p = x -> parent;
       if(!p) return -1;
       return p -> child[0] == x ? 0 : (p -> child[1] == x ? 1 : -1);
}

void set(Node *par, int sid, Node *ch){
       par -> child[sid] = ch;
       if(ch) ch -> parent = par;
       pull(par);
}

void rotate(Node *x){
       Node *p = x -> parent; assert(p);
       Node *pp = p -> parent;

       int dx = side(x), dy = side(p);

       set(p, dx, x -> child[!dx]);
       set(x, !dx, p);
       if(~dy) set(pp, dy, x);
       x -> parent = pp;
}

void splay(Node* x) {
       push(x);
       while(!x -> is_root()){
              Node* p = x -> parent;
              if(p -> parent) push(p -> parent);
              push(p); push(x);

              if(!p -> is_root()){
                     int dx = side(x), dy = side(p);
                     rotate(dx != dy ? x : p);
              }
              rotate(x);
       }
}

void push(Node *x){
       if(!x || !x -> rev) return;
       Node *left = x -> child[0], *right = x -> child[1];

       if(left) left -> rev ^= 1;
       if(right) right -> rev ^= 1;
       swap(x -> child[0], x -> child[1]);
       x -> rev = false;
}

void pull(Node *x){
       x -> siz = x -> vir + 1;
```

```
        if(x -> child[0]){
                x -> siz += x -> child[0] -> siz;
        }
        if(x -> child[1]){
                x -> siz += x -> child[1] -> siz;
        }
}

Node* access(Node *x) {
        Node *last = nullptr;
        for(Node *y = x; y; y = y -> parent){
                splay(y);
                if(last) y -> vir -= last -> siz;
                if(y -> child[1]) y -> vir += y -> child[1] -> siz;
                y -> child[1] = last;
                last = y;
                pull(y);
        }
        splay(x);
        return last;
}

void makeroot(Node *x){
        access(x);
        x -> rev ^= 1;
        push(x);
}

Node* find_root(Node *x) {
        access(x);
        while(x -> child[0]) x = x -> child[0];
        access(x);
        return x;
}

void link(int u, int v){
        if(is_connected(u, v)) return;
        Node *par = &nodes[u], *ch = &nodes[v];
        makeroot(ch);
        access(par);
        set(ch, 0, par);
}

void cut(int u, int v){
        Node* par = &nodes[u];
        Node* ch = &nodes[v];
        makeroot(par);
        access(ch);
        if(ch -> child[0]){
                ch -> child[0] -> parent = nullptr;
                ch -> child[0] = nullptr;
        }
    }

    bool is_connected(int u, int v){
        return (find_root(&nodes[u]) == find_root(&nodes[v]));
    }

    int lca(int u, int v){
        if(!is_connected(u, v)) return -1;
        access(&nodes[u]);
        Node *uv = access(&nodes[v]);
        return uv -> value;
    }
};
endsnippet

snippet AhoCorasick "Automata que ayuda a buscar patrones"
const int SIZE = 5e5 + 10;
const int ALPH = 26;

int tr[SIZE][ALPH];
int fail[SIZE]; // fail[u] = the failure link for node
int seen[SIZE]; // // number of times a node has been visited
int nodes = 1;
vector<int> g[SIZE], leaf[SIZE];

${1}struct AhoCorasick{

    AhoCorasick(){
        memset(tr, 0, sizeof(tr));
        memset(fail, 0, sizeof(fail));
        memset(seen, 0, sizeof(seen));

    }

    void add(const string& word, const int& idx){
        int node = 1;
        for(char c : word){
            if(tr[node][c - 'a'] == 0) {
                tr[node][c - 'a'] = ++nodes;
            }
            node = tr[node][c - 'a'];
        }
        leaf[node].push_back(idx);
    }

    // BFS to building the failure and suffix links
    void build(){
        queue<int> q;
        int node = 1;
        fail[node] = 1;
        for (int i = 0; i < ALPH; i++) {
            int& next = tr[node][i];
            if (tr[node][i]) {
                fail[tr[node][i]] = node;
                q.push(tr[node][i]);
            } else {
                tr[node][i] = 1;
            }
        }

        while (!q.empty()) {
            node = q.front(); q.pop();
            for (int i = 0; i < ALPH; i++) {
                if (tr[node][i]) {
                    fail[tr[node][i]] = tr[fail[node]][i];
                    q.push(tr[node][i]);
                } else {
                    tr[node][i] = tr[fail[node]][i];
                }
            }
        }
        for (int i = 2; i <= nodes; i++) { g[fail[i]].push_back(i); }
    }

    void traverse(string& s, bool counting = true){
        int node = 1;
        for(int i = 0; i < (int) s.size(); i++){
            node = tr[node][s[i] - 'a'];
            seen[node] = (counting ? seen[node] + 1 : (seen[node] == -1
? i : seen[node]));
        }
    }

    vector<int> count(string& text, vector<string>& patterns){
        for(int i = 0; i < (int) patterns.size(); i++)
            add(patterns[i], i);

        build();
        traverse(text);
        vector<int> ans((int) patterns.size());

        function<int(int)> dfs = [&](int node){
            int cnt = seen[node];
            for(int son : g[node])
                cnt += dfs(son);

            for(int idx : leaf[node])
                ans[idx] = cnt;

            return cnt;
        };
        dfs(1);
        return ans;
    }

    // 0-indexed
    vector<int> find_first_position(string& text, vector<string>& patterns){
        memset(seen, -1, sizeof(seen));
        for(int i = 0; i < (int) patterns.size(); i++)
            add(patterns[i], i);

        build();
        traverse(text, false);
        vector<int> ans((int) patterns.size());

        function<int(int)> dfs = [&](int node){
            int mn = seen[node];
            for(int son : g[node]){
                int new_mn = dfs(son);
                if(mn == -1) mn = new_mn;
                else if(new_mn != -1 && new_mn < mn)  mn = new_mn;
            }

            for(int idx : leaf[node]){
                ans[idx] = mn;
                if(mn != -1)
                    ans[idx] -= patterns[idx].size() - 1;
            }

            return mn;
        };
        dfs(1);
        return ans;
    }
};
endsnippet
```