# [UFMG] Summergimurne?

Bruno Monteiro, Emanuel Silva e Bernardo Amorim

## Índice

# 1 Estruturas

## 1.1 DSU

```cpp
// Une dois conjuntos e acha a qual conjunto um elemento
   pertence por seu id
//
// dsu_build: O(n)
// find e unite: O(a(n)) ~= O(1) amortizado

int id[MAX], sz[MAX];

void dsu_build(int n) { for(int i=0; i<n; i++) sz[i] = 1,
   id[i] = i; }

int find(int a) { return id[a] = a == id[a] ? a :
   find(id[a]); }

void unite(int a, int b) {
    a = find(a), b = find(b);
    if(a == b) return;
    if(sz[a] < sz[b]) swap(a,b);

    sz[a] += sz[b];
    id[b] = a;
}
```

## 1.2 BIT

```cpp
// BIT de soma 1-based, v 0-based
// Para mudar o valor da posicao p para x,
// faca: poe(x - query(p, p), p)
// l_bound(x) retorna o menor p tal que
// query(1, p+1) > x    (0 based!)
//
// Complexidades:
// build - O(n)
// poe - O(log(n))
// query - O(log(n))
// l_bound - O(log(n))

int n;
int bit[MAX];
int v[MAX];

void build() {
    bit[0] = 0;
    for (int i = 1; i <= n; i++) bit[i] = v[i - 1];

    for (int i = 1; i <= n; i++) {
        int j = i + (i & -i);
        if (j <= n) bit[j] += bit[i];
    }
}

// soma x na posicao p
void poe(int x, int p) {
    for (; p <= n; p += p & -p) bit[p] += x;
}

// soma [1, p]
int pref(int p) {
    int ret = 0;
    for (; p; p -= p & -p) ret += bit[p];
    return ret;
}

// soma [a, b]
int query(int a, int b) {
    return pref(b) - pref(a - 1);
}

int l_bound(ll x) {
    int p = 0;
    for (int i = MAX2; i+1; i--) if (p + (1<<i) <= n
        and bit[p + (1<<i)] <= x) x -= bit[p += (1<<i)];
    return p;
}
```

## 1.3 SQRT Tree

```
// RMQ em O(log log n) com O(n log log n) pra buildar
// Funciona com qualquer operacao associativa
// Tao rapido quanto a sparse table, mas usa menos memoria
// (log log (1e9) < 5, entao a query eh praticamente O(1))
//
// build - O(n log log n)
// query - O(log log n)

namespace sqrtTree {
    int n, *v;
    int pref[4][MAX], sulf[4][MAX], getl[4][MAX],
        entre[4][MAX], sz[4];

    int op(int a, int b) { return min(a, b); }
    inline int getblk(int p, int i) { return
        (i-getl[p][i])/sz[p]; }
    void build(int p, int l, int r) {
        if (l+1 >= r) return;
        for (int i = l; i <= r; i++) getl[p][i] = l;
        for (int L = l; L <= r; L += sz[p]) {
            int R = min(L+sz[p]-1, r);
            pref[p][L] = v[L], sulf[p][R] = v[R];
            for (int i = L+1; i <= R; i++) pref[p][i] =
                op(pref[p][i-1], v[i]);
            for (int i = R-1; i >= L; i--) sulf[p][i] =
                op(v[i], sulf[p][i+1]);
            build(p+1, L, R);
        }
        for (int i = 0; i <= sz[p]; i++) {
            int at = entre[p][l+i*sz[p]+i] =
                sulf[p][l+i*sz[p]];
            for (int j = i+1; j <= sz[p]; j++)
                entre[p][l+i*sz[p]+j] = at =
                        op(at, sulf[p][l+j*sz[p]]);
        }
    }
    void build(int n2, int* v2) {
        n = n2, v = v2;
        for (int p = 0; p < 4; p++) sz[p] = n2 = sqrt(n2);
        build(0, 0, n-1);
    }
    int query(int l, int r) {
        if (l+1 >= r) return l == r ? v[l] : op(v[l], v[r]);
        int p = 0;
        while (getblk(p, l) == getblk(p, r)) p++;
        int ans = sulf[p][l], a = getblk(p, l)+1, b =
            getblk(p, r)-1;
        if (a <= b) ans = op(ans,
            entre[p][getl[p][l]+a*sz[p]+b]);
        return op(ans, pref[p][r]);
    }
}
```

## 1.4 Min queue - stack

```
// Tudo O(1) amortizado

template<class T> struct minstack {
    stack<pair<T, T>> s;

    void push(T x) {
        if (!s.size()) s.push({x, x});
        else s.push({x, std::min(s.top().second, x)});
    }
    T top() { return s.top().first; }
    T pop() {
        T ans = s.top().first;
        s.pop();
        return ans;
    }
    T size() { return s.size(); }
    T min() { return s.top().second; }
};

template<class T> struct minqueue {
    minstack<T> s1, s2;

    void push(T x) { s1.push(x); }
    void move() {
        if (s2.size()) return;
```

```
        while (s1.size()) {
            T x = s1.pop();
            s2.push(x);
        }
    }
    T front() { return move(), s2.top(); }
    T pop() { return move(), s2.pop(); }
    T size() { return s1.size()+s2.size(); }
    T min() {
        if (!s1.size()) return s2.min();
        else if (!s2.size()) return s1.min();
        return std::min(s1.min(), s2.min());
    }
};
```

## 1.5 Sparse Table

```
// Resolve RMQ
// MAX2 = log(MAX)
//
// Complexidades:
// build - O(n log(n))
// query - O(1)

namespace sparse {
    int m[MAX2][MAX], n;
    void build(int n2, int* v) {
        n = n2;
        for (int i = 0; i < n; i++) m[0][i] = v[i];
        for (int j = 1; (1<<j) <= n; j++) for (int i = 0;
            i+(1<<j) <= n; i++)
             m[j][i] = min(m[j-1][i], m[j-1][i+(1<<(j-1))]);
    }
    int query(int a, int b) {
        int j = __builtin_clz(1) - __builtin_clz(b-a+1);
        return min(m[j][a], m[j][b-(1<<j)+1]);
    }
}
```

## 1.6 BIT com update em range

```
// Operacoes 0-based
// query(l, r) retorna a soma de v[l..r]
// update(l, r, x) soma x em v[l..r]
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

namespace bit {
    ll bit[2][MAX+2];
    int n;

    void build(int n2, int* v) {
        n = n2;
        for (int i = 1; i <= n; i++)
            bit[1][min(n+1, i+(i&-i))] += bit[1][i] +=
               v[i-1];
    }
    ll get(int x, int i) {
        ll ret = 0;
        for (; i; i -= i&-i) ret += bit[x][i];
        return ret;
    }
    void add(int x, int i, ll val) {
        for (; i <= n; i += i&-i) bit[x][i] += val;
    }
    ll get2(int p) {
        return get(0, p) * p + get(1, p);
    }
    ll query(int l, int r) {
        return get2(r+1) - get2(l);
    }
    void update(int l, int r, ll x) {
        add(0, l+1, x), add(0, r+2, -x);
        add(1, l+1, -x*l), add(1, r+2, x*(r+1));
    }
};
```

## 1.7 Splay Tree

```
// SEMPRE QUE DESCER NA ARVORE, DAR SPLAY NO
// NODE MAIS PROFUNDO VISITADO
// Todas as operacoes sao O(log(n)) amortizado
// Se quiser colocar mais informacao no node,
// mudar em 'update'

template<typename T> struct splaytree {
	struct node {
		node *ch[2], *p;
		int sz;
		T val;
		node(T v) {
			ch[0] = ch[1] = p = NULL;
			sz = 1;
			val = v;
		}
		void update() {
			sz = 1;
			for (int i = 0; i < 2; i++) if (ch[i]) {
				sz += ch[i]->sz;
			}
		}
	};

	node* root;

	splaytree() { root = NULL; }
	splaytree(const splaytree& t) {
		throw logic_error("Nao copiar a splaytree!");
	}
	~splaytree() {
		vector<node*> q = {root};
		while (q.size()) {
			node* x = q.back(); q.pop_back();
			if (!x) continue;
			q.push_back(x->ch[0]), q.push_back(x->ch[1]);
			delete x;
		}
	}

	void rotate(node* x) { // x vai ficar em cima
		node *p = x->p, *pp = p->p;
		if (pp) pp->ch[pp->ch[1] == p] = x;
		bool d = p->ch[0] == x;
		p->ch[!d] = x->ch[d], x->ch[d] = p;
		if (p->ch[!d]) p->ch[!d]->p = p;
		x->p = pp, p->p = x;
		p->update(), x->update();
	}
	node* splay(node* x) {
		if (!x) return x;
		root = x;
		while (x->p) {
			node *p = x->p, *pp = p->p;
			if (!pp) return rotate(x), x; // zig
			if ((pp->ch[0] == p)^(p->ch[0] == x))
				rotate(x), rotate(x); // zigzag
			else rotate(p), rotate(x); // zigzig
		}
		return x;
	}
	node* insert(T v, bool lb=0) {
		if (!root) return lb ? NULL : root = new node(v);
		node *x = root, *last = NULL;;
		while (1) {
			bool d = x->val < v;
			if (!d) last = x;
			if (x->val == v) break;
			if (x->ch[d]) x = x->ch[d];
			else {
				if (lb) break;
				x->ch[d] = new node(v);
				x->ch[d]->p = x;
				x = x->ch[d];
				break;
			}
		}
		splay(x);
		return lb ? splay(last) : x;
	}
	int size() { return root ? root->sz : 0; }
	int count(T v) { return insert(v, 1) and root->val == v;
```

```
    }
    node* lower_bound(T v) { return insert(v, 1); }
    void erase(T v) {
        if (!count(v)) return;
        node *x = root, *l = x->ch[0];
        if (!l) {
            root = x->ch[1];
            if (root) root->p = NULL;
            return delete x;
        }
        root = l, l->p = NULL;
        while (l->ch[1]) l = l->ch[1];
        splay(l);
        l->ch[1] = x->ch[1];
        if (l->ch[1]) l->ch[1]->p = l;
        delete x;
        l->update();
    }
    int order_of_key(T v) {
        if (!lower_bound(v)) return root ? root->sz : 0;
        return root->ch[0] ? root->ch[0]->sz : 0;
    }
    node* find_by_order(int k) {
        if (k >= size()) return NULL;
        node* x = root;
        while (1) {
            if (x->ch[0] and x->ch[0]->sz >= k+1) x =
                x->ch[0];
            else {
                if (x->ch[0]) k -= x->ch[0]->sz;
                if (!k) return splay(x);
                k--, x = x->ch[1];
            }
        }
    }
    T min() {
        node* x = root;
        while (x->ch[0]) x = x->ch[0]; // max -> ch[1]
        return splay(x)->val;
    }
};
```

## 1.8 Treap Implicita

```
// Todas as operacoes custam
// O(log(n)) com alta probabilidade

mt19937 rng((int)
    chrono::steady_clock::now().time_since_epoch().count());

template<typename T> struct treap {
    struct node {
        node *l, *r;
        int p, sz;
        T val, sub, lazy;
        bool rev;
        node(T v) : l(NULL), r(NULL), p(rng()), sz(1),
            val(v), sub(v), lazy(0), rev(0) {}
        void prop() {
            if (lazy) {
                val += lazy, sub += lazy*sz;
                if (l) l->lazy += lazy;
                if (r) r->lazy += lazy;
            }
            if (rev) {
                swap(l, r);
                if (l) l->rev ^= 1;
                if (r) r->rev ^= 1;
            }
            lazy = 0, rev = 0;
        }
        void update() {
            sz = 1, sub = val;
            if (l) l->prop(), sz += l->sz, sub += l->sub;
            if (r) r->prop(), sz += r->sz, sub += r->sub;
        }
    };

    node* root;

    treap() { root = NULL; }
    treap(const treap& t) {
        throw logic_error("Nao copiar a treap!");
    }
```

```
~treap() {
    vector<node*> q = {root};
    while (q.size()) {
        node* x = q.back(); q.pop_back();
        if (!x) continue;
        q.push_back(x->l), q.push_back(x->r);
        delete x;
    }
}

int size(node* x) { return x ? x->sz : 0; }
int size() { return size(root); }
void join(node* l, node* r, node*& i) { // assume que l
    < r
    if (!l or !r) return void(i = l ? l : r);
    l->prop(), r->prop();
    if (l->p > r->p) join(l->r, r, l->r), i = l;
    else join(l, r->l, r->l), i = r;
    i->update();
}
void split(node* i, node*& l, node*& r, int v, int key =
    0) {
    if (!i) return void(r = l = NULL);
    i->prop();
    if (key + size(i->l) < v) split(i->r, i->r, r, v,
        key+size(i->l)+1), l = i;
    else split(i->l, l, i->l, v, key), r = i;
    i->update();
}
void push_back(T v) {
    node* i = new node(v);
    join(root, i, root);
}
T query(int l, int r) {
    node *L, *M, *R;
    split(root, M, R, r+1), split(M, L, M, l);
    T ans = M->sub;
    join(L, M, M), join(M, R, root);
    return ans;
}
void update(int l, int r, T s) {
    node *L, *M, *R;
        split(root, M, R, r+1), split(M, L, M, l);
        M->lazy += s;
        join(L, M, M), join(M, R, root);
    }
    void reverse(int l, int r) {
        node *L, *M, *R;
        split(root, M, R, r+1), split(M, L, M, l);
        M->rev ^= 1;
        join(L, M, M), join(M, R, root);
    }
};
```

## 1.9 Range color

```
// update(l, r, c) colore o range [l, r] com a cor c,
// e retorna os ranges que foram coloridos {l, r, cor}
// query(i) returna a cor da posicao i
//
// Complexidades (para q operacoes):
// update - O(log(q)) amortizado
// query - O(log(q))

template<typename T> struct color {
    set<tuple<int, int, T>> se;

    vector<tuple<int, int, T>> update(int l, int r, T val) {
        auto it = se.upper_bound({r, INF, val});
        if (it != se.begin() and get<1>(*prev(it)) > r) {
            auto [L, R, V] = *--it;
            se.erase(it);
            se.emplace(L, r, V), se.emplace(r+1, R, V);
        }
        it = se.lower_bound({l, -INF, val});
        if (it != se.begin() and get<1>(*prev(it)) >= l) {
            auto [L, R, V] = *--it;
            se.erase(it);
            se.emplace(L, l-1, V), it = se.emplace(l, R,
                V).first;
        }
        vector<tuple<int, int, T>> ret;
```

```
        for (; it != se.end() and get<0>(*it) <= r; it =
            se.erase(it))
             ret.push_back(*it);
        se.emplace(l, r, val);
        return ret;
    }
    T query(int i) {
        auto it = se.upper_bound({i, INF, T()});
        if (it == se.begin() or get<1>(*--it) < i) return
            -1; // nao tem
        return get<2>(*it);
    }
};
```

## 1.10 Li-Chao Tree

```
// Adiciona retas (ax+b), e computa o minimo entre as retas
// em um dado 'x'
// Cuidado com overflow!
// Se tiver overflow, tenta comprimir o 'x' ou usar
// convex hull trick
//
// O(log(MA-MI)), O(n) de memoria

template<ll MI = ll(-1e9), ll MA = ll(1e9)> struct lichao {
    struct line {
        ll a, b;
        array<int, 2> ch;
        line(ll a_ = 0, ll b_ = LINF) :
            a(a_), b(b_), ch({-1, -1}) {}
        ll operator ()(ll x) { return a*x + b; }
    };
    vector<line> ln;

    int ch(int p, int d) {
        if (ln[p].ch[d] == -1) {
            ln[p].ch[d] = ln.size();
            ln.emplace_back();
        }
        return ln[p].ch[d];
    }
    lichao() { ln.emplace_back(); }

    void add(line s, ll l=MI, ll r=MA, int p=0) {
        ll m = (l+r)/2;
        bool L = s(l) < ln[p](l);
        bool M = s(m) < ln[p](m);
        bool R = s(r) < ln[p](r);
        if (M) swap(ln[p], s), swap(ln[p].ch, s.ch);
        if (s.b == LINF) return;
        if (L != M) add(s, l, m-1, ch(p, 0));
        else if (R != M) add(s, m+1, r, ch(p, 1));
    }
    ll query(int x, ll l=MI, ll r=MA, int p=0) {
        ll m = (l+r)/2, ret = ln[p](x);
        if (ret == LINF) return ret;
        if (x < m) return min(ret, query(x, l, m-1, ch(p,
            0)));
        return min(ret, query(x, m+1, r, ch(p, 1)));
    }
};
```

## 1.11 BIT 2D

```
// BIT de soma 1-based
// Para mudar o valor da posicao (x, y) para k,
// faca: poe(x, y, k - sum(x, y, x, y))
//
// Complexidades:
// poe - O(log^2(n))
// query - O(log^2(n))

int n;
int bit[MAX][MAX];

void poe(int x, int y, int k) {
    for (int y2 = y; x <= n; x += x & -x)
        for (y = y2; y <= n; y += y & -y)
            bit[x][y] += k;
}
```

```cpp
int sum(int x, int y) {
    int ret = 0;
    for (int y2 = y; x; x -= x & -x)
        for (y = y2; y; y -= y & -y)
            ret += bit[x][y];

    return ret;
}

int query(int x, int y, int z, int w) {
    return sum(z, w) - sum(x-1, w)
        - sum(z, y-1) + sum(x-1, y-1);
}
```

## 1.12 Order Statistic Set

```cpp
// Funciona do C++11 pra cima

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <class T>
    using ord_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

// para declarar:
ord_set<int> s;
// coisas do set normal funcionam:
for (auto i : s) cout << i << endl;
cout << s.size() << endl;
// k-esimo maior elemento O(log|s|):
// k=0: menor elemento
cout << *s.find_by_order(k) << endl;
// quantos sao menores do que k O(log|s|):
cout << s.order_of_key(k) << endl;

// Para fazer um multiset, tem que
// usar ord_set<pair<int, int>> com o
// segundo parametro sendo algo para diferenciar
// os ementos iguais.
// s.order_of_key({k, -INF}) vai retornar o
// numero de elementos < k
```

## 1.13 Splay Tree Implicita

```cpp
// vector da NASA
// Um pouco mais rapido q a treap
// O construtor a partir do vector
// eh linear, todas as outras operacoes
// custam O(log(n)) amortizado

template<typename T> struct splay {
    struct node {
        node *ch[2], *p;
        int sz;
        T val, sub, lazy;
        bool rev;
        node(T v) {
            ch[0] = ch[1] = p = NULL;
            sz = 1;
            sub = val = v;
            lazy = 0;
            rev = false;
        }
        void prop() {
            if (lazy) {
                val += lazy, sub += lazy*sz;
                if (ch[0]) ch[0]->lazy += lazy;
                if (ch[1]) ch[1]->lazy += lazy;
            }
            if (rev) {
                swap(ch[0], ch[1]);
                if (ch[0]) ch[0]->rev ^= 1;
                if (ch[1]) ch[1]->rev ^= 1;
            }
            lazy = 0, rev = 0;
        }
        void update() {
            sz = 1, sub = val;
```

```
        for (int i = 0; i < 2; i++) if (ch[i]) {
            ch[i]->prop();
            sz += ch[i]->sz;
            sub += ch[i]->sub;
        }
    }
};

node* root;

splay() { root = NULL; }
splay(node* x) {
    root = x;
    if (root) root->p = NULL;
}
splay(vector<T> v) { // O(n)
    root = NULL;
    for (T i : v) {
        node* x = new node(i);
        x->ch[0] = root;
        if (root) root->p = x;
        root = x;
        root->update();
    }
}
splay(const splay& t) {
    throw logic_error("Nao copiar a splay!");
}
~splay() {
    vector<node*> q = {root};
    while (q.size()) {
        node* x = q.back(); q.pop_back();
        if (!x) continue;
        q.push_back(x->ch[0]), q.push_back(x->ch[1]);
        delete x;
    }
}

int size(node* x) { return x ? x->sz : 0; }
void rotate(node* x) { // x vai ficar em cima
    node *p = x->p, *pp = p->p;
    if (pp) pp->ch[pp->ch[1] == p] = x;
    bool d = p->ch[0] == x;
    p->ch[!d] = x->ch[d], x->ch[d] = p;
    if (p->ch[!d]) p->ch[!d]->p = p;
    x->p = pp, p->p = x;
    p->update(), x->update();
}
node* splaya(node* x) {
    if (!x) return x;
    root = x, x->update();
    while (x->p) {
        node *p = x->p, *pp = p->p;
        if (!pp) return rotate(x), x; // zig
        if ((pp->ch[0] == p)^(p->ch[0] == x))
            rotate(x), rotate(x); // zigzag
        else rotate(p), rotate(x); // zigzig
    }
    return x;
}
node* find(int v) {
    if (!root) return NULL;
    node *x = root;
    int key = 0;
    while (1) {
        x->prop();
        bool d = key + size(x->ch[0]) < v;
        if (key + size(x->ch[0]) != v and x->ch[d]) {
            if (d) key += size(x->ch[0])+1;
            x = x->ch[d];
        } else break;
    }
    return splaya(x);
}
int size() { return root ? root->sz : 0; }
void join(splay<T>& l) { // assume que l < *this
    if (!size()) swap(root, l.root);
    if (!size() or !l.size()) return;
    node* x = l.root;
    while (1) {
        x->prop();
        if (!x->ch[1]) break;
        x = x->ch[1];
    }
```

```cpp
        l.splaya(x), root->prop(), root->update();
        x->ch[1] = root, x->ch[1]->p = x;
        root = l.root, l.root = NULL;
        root->update();
    }
    node* split(int v) { // retorna os elementos < v
        if (v <= 0) return NULL;
        if (v >= size()) {
            node* ret = root;
            root = NULL;
            ret->update();
            return ret;
        }
        find(v);
        node* l = root->ch[0];
        root->ch[0] = NULL;
        if (l) l->p = NULL;
        root->update();
        return l;
    }
    T& operator [](int i) {
        find(i);
        return root->val;
    }
    void push_back(T v) { // O(1)
        node* r = new node(v);
        r->ch[0] = root;
        if (root) root->p = r;
        root = r, root->update();
    }
    T query(int l, int r) {
        splay<T> M(split(r+1));
        splay<T> L(M.split(l));
        T ans = M.root->sub;
        M.join(L), join(M);
        return ans;
    }
    void update(int l, int r, T s) {
        splay<T> M(split(r+1));
        splay<T> L(M.split(l));
        M.root->lazy += s;
        M.join(L), join(M);
    }
    void reverse(int l, int r) {
        splay<T> M(split(r+1));
        splay<T> L(M.split(l));
        M.root->rev ^= 1;
        M.join(L), join(M);
    }
    void erase(int l, int r) {
        splay<T> M(split(r+1));
        splay<T> L(M.split(l));
        join(L);
    }
};
```

## 1.14 Sparse Table Disjunta

```cpp
// Resolve qualquer operacao associativa
// MAX2 = log(MAX)
//
// Complexidades:
// build - O(n log(n))
// query - O(1)

namespace sparse {
    int m[MAX2][2*MAX], n, v[2*MAX];
    int op(int a, int b) { return min(a, b); }
    void build(int n2, int* v2) {
        n = n2;
        for (int i = 0; i < n; i++) v[i] = v2[i];
        while (n&(n-1)) n++;
        for (int j = 0; (1<<j) < n; j++) {
            int len = 1<<j;
            for (int c = len; c < n; c += 2*len) {
                m[j][c] = v[c], m[j][c-1] = v[c-1];
                for (int i = c+1; i <  c+len; i++) m[j][i] =
                    op(m[j][i-1], v[i]);
                for (int i = c-2; i >= c-len; i--) m[j][i] =
                    op(v[i], m[j][i+1]);
            }
        }
```

```
    }
    int query(int l, int r) {
        if (l == r) return v[l];
        int j = __builtin_clz(1) - __builtin_clz(l^r);
        return op(m[j][l], m[j][r]);
    }
}
```

## 1.15 Min queue - deque

```
// Tudo O(1) amortizado

template<class T> struct minqueue {
    deque<pair<T, int>> q;

    void push(T x) {
        int ct = 1;
        while (q.size() and x < q.front().first)
            ct += q.front().second, q.pop_front();
        q.push_front({x, ct});
    }
    void pop() {
        if (q.back().second > 1) q.back().second--;
        else q.pop_back();
    }
    T min() { return q.back().first; }
};
```

## 1.16 Split-Merge Set

```
// Representa um conjunto de inteiros nao negativos
// Todas as operacoes custam O(log(N)),
// em que N = maior elemento do set,
// exceto o merge, que custa O(log(N)) amortizado
// Usa O(min(N, n log(N))) de memoria, sendo 'n' o
// numero de elementos distintos no set

template<typename T, bool MULTI=false, typename SIZE_T=int>
   struct sms {
struct node {
    node *l, *r;
    SIZE_T cnt;
    node() : l(NULL), r(NULL), cnt(0) {}
    void update() {
        cnt = 0;
        if (l) cnt += l->cnt;
        if (r) cnt += r->cnt;
    }
};

node* root;
T N;

sms() : root(NULL), N(0) {}
sms(T v) : sms() { while (v >= N) N = 2*N+1; }
sms(const sms& t) : root(NULL), N(t.N) {
    for (SIZE_T i = 0; i < t.size(); i++) {
        T at = t[i];
        SIZE_T qt = t.count(at);
        insert(at, qt);
        i += qt-1;
    }
}
sms(initializer_list<T> v) : sms() { for (T i : v)
   insert(i); }
~sms() {
    vector<node*> q = {root};
    while (q.size()) {
        node* x = q.back(); q.pop_back();
        if (!x) continue;
        q.push_back(x->l), q.push_back(x->r);
        delete x;
    }
}

friend void swap(sms& a, sms& b) {
    swap(a.root, b.root), swap(a.N, b.N);
}
sms& operator =(const sms& v) {
    sms tmp = v;
    swap(tmp, *this);
```

```
	return *this;
}
SIZE_T size() const { return root ? root->cnt : 0; }
SIZE_T count(node* x) const { return x ? x->cnt : 0; }
void clear() {
	sms tmp;
	swap(*this, tmp);
}
void expand(T v) {
	for (; N < v; N = 2*N+1) if (root) {
		node* nroot = new node();
		nroot->l = root;
		root = nroot;
		root->update();
	}
}

node* insert(node* at, T idx, SIZE_T qt, T l, T r) {
	if (!at) at = new node();
	if (l == r) {
		at->cnt += qt;
		if (!MULTI) at->cnt = 1;
		return at;
	}
	T m = l + (r-l)/2;
	if (idx <= m) at->l = insert(at->l, idx, qt, l, m);
	else at->r = insert(at->r, idx, qt, m+1, r);
	return at->update(), at;
}
void insert(T v, SIZE_T qt=1) { // insere 'qt'
   ocorrencias de 'v'
	if (qt <= 0) return erase(v, -qt);
	assert(v >= 0);
	expand(v);
	root = insert(root, v, qt, 0, N);
}

node* erase(node* at, T idx, SIZE_T qt, T l, T r) {
	if (!at) return at;
	if (l == r) at->cnt = at->cnt < qt ? 0 : at->cnt -
	   qt;
	else {
```

```
		T m = l + (r-l)/2;
		if (idx <= m) at->l = erase(at->l, idx, qt, l,
		   m);
		else at->r = erase(at->r, idx, qt, m+1, r);
		at->update();
	}
	if (!at->cnt) delete at, at = NULL;
	return at;
}
void erase(T v, SIZE_T qt=1) { // remove 'qt'
   ocorrencias de 'v'
	if (v < 0 or v > N or !qt) return;
	if (qt < 0) insert(v, -qt);
	root = erase(root, v, qt, 0, N);
}
void erase_all(T v) { // remove todos os 'v'
	if (v < 0 or v > N) return;
	root = erase(root, v, numeric_limits<SIZE_T>::max(),
	   0, N);
}

SIZE_T count(node* at, T a, T b, T l, T r) const {
	if (!at or b < l or r < a) return 0;
	if (a <= l and r <= b) return at->cnt;
	T m = l + (r-l)/2;
	return count(at->l, a, b, l, m) + count(at->r, a, b,
	   m+1, r);
}
SIZE_T count(T v) const { return count(root, v, v, 0,
   N); }
SIZE_T order_of_key(T v) { return count(root, 0, v-1, 0,
   N); }
SIZE_T lower_bound(T v) { return order_of_key(v); }

const T operator [](SIZE_T i) const { // i-esimo menor
   elemento
	assert(i >= 0 and i < size());
	node* at = root;
	T l = 0, r = N;
	while (l < r) {
		T m = l + (r-l)/2;
		if (count(at->l) > i) at = at->l, r = m;
```

```
        else {
            i -= count(at->l);
            at = at->r; l = m+1;
        }
    }
    return l;
}

node* merge(node* l, node* r) {
    if (!l or !r) return l ? l : r;
    if (!l->l and !l->r) { // folha
        if (MULTI) l->cnt += r->cnt;
        delete r;
        return l;
    }
    l->l = merge(l->l, r->l), l->r = merge(l->r, r->r);
    l->update(), delete r;
    return l;
}
void merge(sms& s) { // mergeia dois sets
    if (N > s.N) swap(*this, s);
    expand(s.N);
    root = merge(root, s.root);
    s.root = NULL;
}

node* split(node*& x, SIZE_T k) {
    if (k <= 0 or !x) return NULL;
    node* ret = new node();
    if (!x->l and !x->r) x->cnt -= k, ret->cnt += k;
    else {
        if (k <= count(x->l)) ret->l = split(x->l, k);
        else {
            ret->r = split(x->r, k - count(x->l));
            swap(x->l, ret->l);
        }
        ret->update(), x->update();
    }
    if (!x->cnt) delete x, x = NULL;
    return ret;
}
void split(SIZE_T k, sms& s) { // pega os 'k' menores
        s.clear();
        s.root = split(root, min(k, size()));
        s.N = N;
    }
    // pega os menores que 'k'
    void split_val(T k, sms& s) { split(order_of_key(k), s);
        }
};
```

## 1.17 Treap

```
// Todas as operacoes custam
// O(log(n)) com alta probabilidade, exceto meld
// meld custa O(log^2 n) amortizado com alta prob.,
// e permite unir duas treaps sem restricao adicional
// Na pratica, esse meld tem constante muito boa e
// o pior caso eh meio estranho de acontecer

mt19937 rng((int)
    chrono::steady_clock::now().time_since_epoch().count());

template<typename T> struct treap {
    struct node {
        node *l, *r;
        int p, sz;
        T val, mi;
        node(T v) : l(NULL), r(NULL), p(rng()), sz(1),
            val(v), mi(v) {}
        void update() {
            sz = 1;
            mi = val;
            if (l) sz += l->sz, mi = min(mi, l->mi);
            if (r) sz += r->sz, mi = min(mi, r->mi);
        }
    };

    node* root;

    treap() { root = NULL; }
    treap(const treap& t) {
```

```
        throw logic_error("Nao copiar a treap!");
    }
    ~treap() {
        vector<node*> q = {root};
        while (q.size()) {
            node* x = q.back(); q.pop_back();
            if (!x) continue;
            q.push_back(x->l), q.push_back(x->r);
            delete x;
        }
    }

    int size(node* x) { return x ? x->sz : 0; }
    int size() { return size(root); }
    void join(node* l, node* r, node*& i) { // assume que l
       < r
        if (!l or !r) return void(i = l ? l : r);
        if (l->p > r->p) join(l->r, r, l->r), i = l;
        else join(l, r->l, r->l), i = r;
        i->update();
    }
    void split(node* i, node*& l, node*& r, T v) {
        if (!i) return void(r = l = NULL);
        if (i->val < v) split(i->r, i->r, r, v), l = i;
        else split(i->l, l, i->l, v), r = i;
        i->update();
    }
    int count(node* i, T v) {
        if (!i) return 0;
        if (i->val == v) return 1;
        if (v < i->val) return count(i->l, v);
        return count(i->r, v);
    }
    void index_split(node* i, node*& l, node*& r, int v, int
       key = 0) {
        if (!i) return void(r = l = NULL);
        if (key + size(i->l) < v) index_split(i->r, i->r, r,
           v, key+size(i->l)+1), l = i;
        else index_split(i->l, l, i->l, v, key), r = i;
        i->update();
    }
    int count(T v) {
        return count(root, v);
    }
    void insert(T v) {
        if (count(v)) return;
        node *L, *R;
        split(root, L, R, v);
        node* at = new node(v);
        join(L, at, L);
        join(L, R, root);
    }
    void erase(T v) {
        node *L, *M, *R;
        split(root, M, R, v+1), split(M, L, M, v);
        if (M) delete M;
        M = NULL;
        join(L, R, root);
    }
    void meld(treap& t) { // segmented merge
        node *L = root, *R = t.root;
        root = NULL;
        while (L or R) {
            if (!L or (L and R and L->mi > R->mi))
                std::swap(L, R);
            if (!R) join(root, L, root), L = NULL;
            else if (L->mi == R->mi) {
                node* LL;
                split(L, LL, L, R->mi+1);
                delete LL;
            } else {
                node* LL;
                split(L, LL, L, R->mi);
                join(root, LL, root);
            }
        }
        t.root = NULL;
    }
};
```

## 1.18 Treap Persistent Implicita

```
// Todas as operacoes custam
// O(log(n)) com alta probabilidade

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

struct node {
    node *l, *r;
    ll sz, val, sub;
    node(ll v) : l(NULL), r(NULL), sz(1), val(v), sub(v) {}
    node(node* x) : l(x->l), r(x->r), sz(x->sz),
        val(x->val), sub(x->sub) {}
    void update() {
        sz = 1, sub = val;
        if (l) sz += l->sz, sub += l->sub;
        if (r) sz += r->sz, sub += r->sub;
        sub %= MOD;
    }
};

ll size(node* x) { return x ? x->sz : 0; }
void update(node* x) { if (x) x->update(); }
node* copy(node* x) { return x ? new node(x) : NULL; }

node* join(node* l, node* r) {
    if (!l or !r) return l ? copy(l) : copy(r);
    node* ret;
    if (rng() % (size(l) + size(r)) < size(l)) {
        ret = copy(l);
        ret->r = join(ret->r, r);
    } else {
        ret = copy(r);
        ret->l = join(l, ret->l);
    }
    return update(ret), ret;
}

void split(node* x, node*& l, node*& r, ll v, ll key = 0) {
    if (!x) return void(l = r = NULL);
    if (key + size(x->l) < v) {
        l = copy(x);
        split(l->r, l->r, r, v, key+size(l->l)+1);
    } else {
        r = copy(x);
        split(r->l, l, r->l, v, key);
    }
    update(l), update(r);
}

vector<node*> treap;

void init(const vector<ll>& v) {
    treap = {NULL};
    for (auto i : v) treap[0] = join(treap[0], new node(i));
}
```

## 1.19 SQRT-decomposition

```
// Resolve RMQ
// 0-indexed
// MAX2 = sqrt(MAX)
//
// O bloco da posicao x eh
// sempre x/q
//
// Complexidades:
// build - O(n)
// query - O(sqrt(n))

int n, q;
int v[MAX];
int bl[MAX2];

void build() {
    q = (int) sqrt(n);

     // computa cada bloco
    for (int i = 0; i <= q; i++) {
        bl[i] = INF;
        for (int j = 0; j < q and q * i + j < n; j++)
            bl[i] = min(bl[i], v[q * i + j]);
    }
```

```
}

int query(int a, int b) {
    int ret = INF;

    // linear no bloco de a
    for (; a <= b and a % q; a++) ret = min(ret, v[a]);

    // bloco por bloco
    for (; a + q <= b; a += q) ret = min(ret, bl[a / q]);

    // linear no bloco de b
    for (; a <= b; a++) ret = min(ret, v[a]);

    return ret;
}
```

## 1.20 RMQ <O(n), O(1)> - cartesian tree

```
// O(n) pra buildar, query O(1)
// Para retornar o indice, basta
// trocar v[...] para ... na query

template<typename T> struct rmq {
    vector<T> v;
    int n, b;
    vector<int> id, st;
    vector<vector<int>> table;
    vector<vector<vector<int>>> entre;

    int op(int x, int y) { return v[x] < v[y] ? x : y; }
    rmq(vector<T>& v_) {
        v = v_, n = v.size();
        b = (__builtin_clz(1)-__builtin_clz(n)+1)/4 + 1;
        id.resize(n);
        table.assign(4*b, vector<int>((n+b-1)/b));
        entre.assign(1<<b<<b, vector<vector<int>>(b,
            vector<int>(b, -1)));
        for (int i = 0; i < n; i += b) {
            int at = 0, l = min(n, i+b);
            st.clear();
            for (int j = i; j < l; j++) {
                while (st.size() and op(st.back(), j) == j)
                    st.pop_back(), at *= 2;
                st.push_back(j), at = 2*at+1;
            }
            for (int j = i; j < l; j++) id[j] = at;
            if (entre[at][0][0] == -1) for (int x = 0; x <
                l-i; x++) {
                entre[at][x][x] = x;
                for (int y = x+1; y < l-i; y++)
                    entre[at][x][y] =
                        op(i+entre[at][x][y-1], i+y) - i;
            }
            table[0][i/b] = i+entre[at][0][l-i-1];
        }
        for (int j = 1; (1<<j) <= (n+b-1)/b; j++)
            for (int i = 0; i+(1<<j) <= (n+b-1)/b; i++)
                table[j][i] = op(table[j-1][i],
                    table[j-1][i+(1<<(j-1))]);
    }
    T query(int i, int j) {
        if (i/b == j/b) return
            v[i/b*b+entre[id[i]][i%b][j%b]];
        int x = i/b+1, y = j/b-1, ans = i;
        if (x <= y) {
            int t = __builtin_clz(1) - __builtin_clz(y-x+1);
            ans = op(ans, op(table[t][x],
                table[t][y-(1<<t)+1]));
        }
        ans = op(ans, op(i/b*b+entre[id[i]][i%b][b-1],
            j/b*b+entre[id[j]][0][j%b]));
        return v[ans];
    }
};
```

## 1.21 MergeSort Tree

```
// Se for construida sobre um array:
//      count(i, j, a, b) retorna quantos
```

```
//      elementos de v[i..j] pertencem a [a, b]
//      report(i, j, a, b) retorna os indices dos
//      elementos de v[i..j] que pertencem a [a, b]
//      retorna o vetor ordenado
// Se for construida sobre pontos (x, y):
//      count(x1, x2, y1, x2) retorna quantos pontos
//      pertencem ao retangulo (x1, y1), (x2, y2)
//      report(x1, x2, y1, y2) retorna os indices dos pontos
   que
//      pertencem ao retangulo (x1, y1), (x2, y2)
//      retorna os pontos ordenados lexicograficamente
//      (assume x1 <= x2, y1 <= y2)
//
// kth(y1, y2, k) retorna o indice do ponto com k-esimo menor
// x dentre os pontos que possuem y em [y1, y2] (0 based)
// Se quiser usar para achar k-esimo valor em range,
   construir
// com ms_tree t(v, true), e chamar kth(l, r, k)
//
// Usa O(n log(n)) de memoria
//
// Complexidades:
// construir - O(n log(n))
// count - O(log(n))
// report - O(log(n) + k) para k indices retornados
// kth - O(log(n))

template <typename T = int> struct ms_tree {
    vector<tuple<T, T, int>> v;
    int n;
    vector<vector<tuple<T, T, int>>> t; // {y, idx, left}
    vector<T> vy;

    ms_tree(vector<pair<T, T>>& vv) : n(vv.size()), t(4*n),
       vy(n) {
        for (int i = 0; i < n; i++)
            v.push_back({vv[i].first, vv[i].second, i});
        sort(v.begin(), v.end());
        build(1, 0, n-1);
        for (int i = 0; i < n; i++) vy[i] =
            get<0>(t[1][i+1]);
    }
    ms_tree(vector<T>& vv, bool inv = false) { // inv:
       inverte indice e valor
        vector<pair<T, T>> v2;
        for (int i = 0; i < vv.size(); i++)
            inv ? v2.push_back({vv[i], i}) :
                v2.push_back({i, vv[i]});
        *this = ms_tree(v2);
    }
    void build(int p, int l, int r) {
        t[p].push_back({get<0>(v[l]), get<0>(v[r]), 0}); //
           {min_x, max_x, 0}
        if (l == r) return t[p].push_back({get<1>(v[l]),
           get<2>(v[l]), 0});
        int m = (l+r)/2;
        build(2*p, l, m), build(2*p+1, m+1, r);

        int L = 0, R = 0;
        while (t[p].size() <= r-l+1) {
            int left = get<2>(t[p].back());
            if (L > m-l or (R+m+1 <= r and t[2*p+1][1+R] <
                t[2*p][1+L])) {
                 t[p].push_back(t[2*p+1][1 + R++]);
                 get<2>(t[p].back()) = left;
                 continue;
            }
            t[p].push_back(t[2*p][1 + L++]);
            get<2>(t[p].back()) = left+1;
        }
    }

    int get_l(T y) { return lower_bound(vy.begin(),
       vy.end(), y) - vy.begin(); }
    int get_r(T y) { return upper_bound(vy.begin(),
       vy.end(), y) - vy.begin(); }

    int count(T x1, T x2, T y1, T y2) {
        function<int(int, int, int)> dfs = [&](int p, int l,
           int r) {
             if (l == r or x2 < get<0>(t[p][0]) or
                get<1>(t[p][0]) < x1) return 0;
             if (x1 <= get<0>(t[p][0]) and get<1>(t[p][0]) <=
                x2) return r-l;
```

```
            int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
            return dfs(2*p, nl, nr) + dfs(2*p+1, l-nl, r-nr);
        };
        return dfs(1, get_l(y1), get_r(y2));
    }
    vector<int> report(T x1, T x2, T y1, T y2) {
        vector<int> ret;
        function<void(int, int, int)> dfs = [&](int p, int
            l, int r) {
            if (l == r or x2 < get<0>(t[p][0]) or
                get<1>(t[p][0]) < x1) return;
            if (x1 <= get<0>(t[p][0]) and get<1>(t[p][0]) <=
                x2) {
                for (int i = l; i < r; i++)
                    ret.push_back(get<1>(t[p][i+1]));
                return;
            }
            int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
            dfs(2*p, nl, nr), dfs(2*p+1, l-nl, r-nr);
        };
        dfs(1, get_l(y1), get_r(y2));
        return ret;
    }
    int kth(T y1, T y2, int k) {
        function<int(int, int, int)> dfs = [&](int p, int l,
            int r) {
            if (k >= r-l) {
                k -= r-l;
                return -1;
            }
            if (r-l == 1) return get<1>(t[p][l+1]);
            int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
            int left = dfs(2*p, nl, nr);
            if (left != -1) return left;
            return dfs(2*p+1, l-nl, r-nr);
        };
        return dfs(1, get_l(y1), get_r(y2));
    }
};
```

## 1.22 Split-Merge Set - Lazy

```
// Representa um conjunto de inteiros nao negativos
// Todas as operacoes custam O(log(N)),
// em que N = maior elemento do set,
// exceto o merge e o insert_range, que custa O(log(N))
    amortizado
// Usa O(min(N, n log(N))) de memoria, sendo 'n' o
// numero de elementos distintos no set

template<typename T> struct sms {
    struct node {
        node *l, *r;
        int cnt;
        bool flip;
        node() : l(NULL), r(NULL), cnt(0), flip(0) {}
        void update() {
            cnt = 0;
            if (l) cnt += l->cnt;
            if (r) cnt += r->cnt;
        }
    };

    void prop(node* x, int size) {
        if (!x or !x->flip) return;
        x->flip = 0;
        x->cnt = size - x->cnt;
        if (size > 1) {
            if (!x->l) x->l = new node();
            if (!x->r) x->r = new node();
            x->l->flip ^= 1;
            x->r->flip ^= 1;
        }
    }

    node* root;
    T N;

    sms() : root(NULL), N(0) {}
    sms(T v) : sms() { while (v >= N) N = 2*N+1; }
    sms(sms& t) : root(NULL), N(t.N) {
        for (int i = 0; i < t.size(); i++) insert(t[i]);
```

```
}
sms(initializer_list<T> v) : sms() { for (T i : v)
   insert(i); }
void destroy(node* r) {
    vector<node*> q = {r};
    while (q.size()) {
        node* x = q.back(); q.pop_back();
        if (!x) continue;
        q.push_back(x->l), q.push_back(x->r);
        delete x;
    }
}
~sms() { destroy(root); }

friend void swap(sms& a, sms& b) {
    swap(a.root, b.root), swap(a.N, b.N);
}
sms& operator =(const sms& v) {
    sms tmp = v;
    swap(tmp, *this);
    return *this;
}
int count(node* x, T size) {
    if (!x) return 0;
    prop(x, size);
    return x->cnt;
}
int size() { return count(root, N+1); }
void clear() {
    sms tmp;
    swap(*this, tmp);
}
void expand(T v) {
    for (; N < v; N = 2*N+1) if (root) {
        prop(root, N+1);
        node* nroot = new node();
        nroot->l = root;
        root = nroot;
        root->update();
    }
}
```

```
node* insert(node* at, T idx, T l, T r) {
    if (!at) at = new node();
    else prop(at, r-l+1);
    if (l == r) {
        at->cnt = 1;
        return at;
    }
    T m = l + (r-l)/2;
    if (idx <= m) at->l = insert(at->l, idx, l, m);
    else at->r = insert(at->r, idx, m+1, r);
    return at->update(), at;
}
void insert(T v) {
    assert(v >= 0);
    expand(v);
    root = insert(root, v, 0, N);
}

node* erase(node* at, T idx, T l, T r) {
    if (!at) return at;
    prop(at, r-l+1);
    if (l == r) at->cnt = 0;
    else {
        T m = l + (r-l)/2;
        if (idx <= m) at->l = erase(at->l, idx, l, m);
        else at->r = erase(at->r, idx, m+1, r);
        at->update();
    }
    return at;
}
void erase(T v) {
    if (v < 0 or v > N) return;
    root = erase(root, v, 0, N);
}

int count(node* at, T a, T b, T l, T r) {
    if (!at or b < l or r < a) return 0;
    prop(at, r-l+1);
    if (a <= l and r <= b) return at->cnt;
    T m = l + (r-l)/2;
    return count(at->l, a, b, l, m) + count(at->r, a, b,
       m+1, r);
```

```
}
int count(T v) { return count(root, v, v, 0, N); }
int order_of_key(T v) { return count(root, 0, v-1, 0,
   N); }
int lower_bound(T v) { return order_of_key(v); }

const T operator[](int i) { // i-esimo menor elemento
    assert(i >= 0 and i < size());
    node* at = root;
    T l = 0, r = N;
    while (l < r) {
        prop(at, r-l+1);
        T m = l + (r-l)/2;
        if (count(at->l, m-l+1) > i) at = at->l, r = m;
        else {
            i -= count(at->l, r-m);
            at = at->r; l = m+1;
        }
    }
    return l;
}

node* merge(node* a, node* b, T tam) {
    if (!a or !b) return a ? a : b;
    prop(a, tam), prop(b, tam);
    if (b->cnt == tam) swap(a, b);
    if (tam == 1 or a->cnt == tam) {
        destroy(b);
        return a;
    }
    a->l = merge(a->l, b->l, tam>>1), a->r = merge(a->r,
       b->r, tam>>1);
    a->update(), delete b;
    return a;
}
void merge(sms& s) { // mergeia dois sets
    if (N > s.N) swap(*this, s);
    expand(s.N);
    root = merge(root, s.root, N+1);
    s.root = NULL;
}
```

```
node* split(node*& x, int k, T tam) {
    if (k <= 0 or !x) return NULL;
    prop(x, tam);
    node* ret = new node();
    if (tam == 1) x->cnt = 0, ret->cnt = 1;
    else {
        if (k <= count(x->l, tam>>1)) ret->l =
           split(x->l, k, tam>>1);
        else {
            ret->r = split(x->r, k - count(x->l,
               tam>>1), tam>>1);
            swap(x->l, ret->l);
        }
        ret->update(), x->update();
    }
    return ret;
}
void split(int k, sms& s) { // pega os 'k' menores
    s.clear();
    s.root = split(root, min(k, size()), N+1);
    s.N = N;
}
// pega os menores que 'k'
void split_val(T k, sms& s) { split(order_of_key(k), s);
   }

void flip(node*& at, T a, T b, T l, T r) {
    if (!at) at = new node();
    else prop(at, r-l+1);
    if (a <= l and r <= b) {
        at->flip ^= 1;
        prop(at, r-l+1);
        return;
    }
    if (r < a or b < l) return;
    T m = l + (r-l)/2;
    flip(at->l, a, b, l, m), flip(at->r, a, b, m+1, r);
    at->update();
}
void flip(T l, T r) { // flipa os valores em [l, r]
    assert(l >= 0 and l <= r);
    expand(r);
```

```
        flip(root, l, r, 0, N);
    }
    // complemento considerando que o universo eh [0, lim]
    void complement(T lim) {
        assert(lim >= 0);
        if (lim > N) expand(lim);
        flip(root, 0, lim, 0, N);
        sms tmp;
        split_val(lim+1, tmp);
        swap(*this, tmp);
    }
    void insert_range(T l, T r) { // insere todo os valores
       em [l, r]
        sms tmp;
        tmp.flip(l, r);
        merge(tmp);
    }
};
```

## 1.23 SegTree 2D Iterativa

```
// Consultas 0-based
// Um valor inicial em (x, y) deve ser colocado em
   seg[x+n][y+n]
// Query: soma do retangulo ((x1, y1), (x2, y2))
// Update: muda o valor da posicao (x, y) para val
// Nao pergunte como que essa coisa funciona
//
// Para query com distancia de manhattan <= d, faca
// nx = x+y, ny = x-y
// Update em (nx, ny), query em ((nx-d, ny-d), (nx+d, ny+d))
//
// Se for de min/max, pode tirar os if's da 'query', e fazer
// sempre as 4 operacoes. Fica mais rapido
//
// Complexidades:
// build - O(n^2)
// query - O(log^2(n))
// update - O(log^2(n))

int seg[2*MAX][2*MAX], n;

void build() {
    for (int x = 2*n; x; x--) for (int y = 2*n; y; y--) {
        if (x < n) seg[x][y] = seg[2*x][y] + seg[2*x+1][y];
        if (y < n) seg[x][y] = seg[x][2*y] + seg[x][2*y+1];
    }
}

int query(int x1, int y1, int x2, int y2) {
    int ret = 0, y3 = y1 + n, y4 = y2 + n;
    for (x1 += n, x2 += n; x1 <= x2; ++x1 /= 2, --x2 /= 2)
        for (y1 = y3, y2 = y4; y1 <= y2; ++y1 /= 2, --y2 /=
           2) {
            if (x1%2 == 1 and y1%2 == 1) ret += seg[x1][y1];
            if (x1%2 == 1 and y2%2 == 0) ret += seg[x1][y2];
            if (x2%2 == 0 and y1%2 == 1) ret += seg[x2][y1];
            if (x2%2 == 0 and y2%2 == 0) ret += seg[x2][y2];
        }

    return ret;
}

void update(int x, int y, int val) {
    int y2 = y += n;
    for (x += n; x; x /= 2, y = y2) {
        if (x >= n) seg[x][y] = val;
        else seg[x][y] = seg[2*x][y] + seg[2*x+1][y];

        while (y /= 2) seg[x][y] = seg[x][2*y] +
           seg[x][2*y+1];
    }
}
```

## 1.24 SegTree PA

```
// Segtree de PA
// update_set(l, r, A, R) seta [l, r] para PA(A, R),
// update_add soma PA(A, R) em [l, r]
// query(l, r) retorna a soma de [l, r]
```

```
//
// PA(A, R) eh a PA: [A+R, A+2R, A+3R, ... ]
//
// Complexidades:
// construir - O(n)
// update_set, update_add, query - O(log(n))

struct seg_pa {
    struct Data {
        ll sum;
        ll set_a, set_r, add_a, add_r;
        Data() : sum(0), set_a(LINF), set_r(0), add_a(0),
            add_r(0) {}
    };
    vector<Data> seg;
    int n;

    seg_pa(int n_) {
        n = n_;
        seg = vector<Data>(4*n);
    }

    void prop(int p, int l, int r) {
        int tam = r-l+1;
        ll &sum = seg[p].sum, &set_a = seg[p].set_a, &set_r
            = seg[p].set_r,
             &add_a = seg[p].add_a, &add_r = seg[p].add_r;

        if (set_a != LINF) {
            set_a += add_a, set_r += add_r;
            sum = set_a*tam + set_r*tam*(tam+1)/2;
            if (l != r) {
                int m = (l+r)/2;

                seg[2*p].set_a = set_a;
                seg[2*p].set_r = set_r;
                seg[2*p].add_a = seg[2*p].add_r = 0;

                seg[2*p+1].set_a = set_a + set_r * (m-l+1);
                seg[2*p+1].set_r = set_r;
                seg[2*p+1].add_a = seg[2*p+1].add_r = 0;
            }
```

```
            set_a = LINF, set_r = 0;
            add_a = add_r = 0;
        } else if (add_a or add_r) {
            sum += add_a*tam + add_r*tam*(tam+1)/2;
            if (l != r) {
                int m = (l+r)/2;

                seg[2*p].add_a += add_a;
                seg[2*p].add_r += add_r;

                seg[2*p+1].add_a += add_a + add_r * (m-l+1);
                seg[2*p+1].add_r += add_r;
            }
            add_a = add_r = 0;
        }
    }

    int inter(pair<int, int> a, pair<int, int> b) {
        if (a.first > b.first) swap(a, b);
        return max(0, min(a.second, b.second) - b.first + 1);
    }
    ll set(int a, int b, ll aa, ll rr, int p, int l, int r) {
        prop(p, l, r);
        if (b < l or r < a) return seg[p].sum;
        if (a <= l and r <= b) {
            seg[p].set_a = aa;
            seg[p].set_r = rr;
            prop(p, l, r);
            return seg[p].sum;
        }
        int m = (l+r)/2;
        int tam_l = inter({l, m}, {a, b});
        return seg[p].sum = set(a, b, aa, rr, 2*p, l, m) +
            set(a, b, aa + rr * tam_l, rr, 2*p+1, m+1, r);
    }
    void update_set(int l, int r, ll aa, ll rr) {
        set(l, r, aa, rr, 1, 0, n-1);
    }
    ll add(int a, int b, ll aa, ll rr, int p, int l, int r) {
        prop(p, l, r);
        if (b < l or r < a) return seg[p].sum;
        if (a <= l and r <= b) {
```

```
            seg[p].add_a += aa;
            seg[p].add_r += rr;
            prop(p, l, r);
            return seg[p].sum;
        }
        int m = (l+r)/2;
        int tam_l = inter({l, m}, {a, b});
        return seg[p].sum = add(a, b, aa, rr, 2*p, l, m) +
            add(a, b, aa + rr * tam_l, rr, 2*p+1, m+1, r);
    }
    void update_add(int l, int r, ll aa, ll rr) {
        add(l, r, aa, rr, 1, 0, n-1);
    }
    ll query(int a, int b, int p, int l, int r) {
        prop(p, l, r);
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return seg[p].sum;
        int m = (l+r)/2;
        return query(a, b, 2*p, l, m) + query(a, b, 2*p+1,
           m+1, r);
    }
    ll query(int l, int r) { return query(l, r, 1, 0, n-1); }
};
```

## 1.25 SegTree Esparsa - Lazy

```
// Query: soma do range [a, b]
// Update: flipa os valores de [a, b]
// O MAX tem q ser Q log N para Q updates
//
// Complexidades:
// build - O(1)
// query - O(log(n))
// update - O(log(n))

namespace seg {
    int seg[MAX], lazy[MAX], R[MAX], L[MAX], ptr;
    int get_l(int i){
        if (L[i] == 0) L[i] = ptr++;
        return L[i];
    }
    int get_r(int i){
        if (R[i] == 0) R[i] = ptr++;
        return R[i];
    }

    void build() { ptr = 2; }

    void prop(int p, int l, int r) {
        if (!lazy[p]) return;
        seg[p] = r-l+1 - seg[p];
        if (l != r) lazy[get_l(p)]^=lazy[p],
           lazy[get_r(p)]^=lazy[p];
        lazy[p] = 0;
    }

    int query(int a, int b, int p=1, int l=0, int r=N-1) {
        prop(p, l, r);
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return seg[p];

        int m = (l+r)/2;
        return query(a, b, get_l(p), l, m)+query(a, b,
           get_r(p), m+1, r);
    }

    int update(int a, int b, int p=1, int l=0, int r=N-1) {
        prop(p, l, r);
        if (b < l or r < a) return seg[p];
        if (a <= l and r <= b) {
            lazy[p] ^= 1;
            prop(p, l, r);
            return seg[p];
        }
        int m = (l+r)/2;
        return seg[p] = update(a, b, get_l(p), l,
           m)+update(a, b, get_r(p), m+1, r);
    }
};
```

## 1.26 SegTree Esparsa - O(q) memoria

```
// Query: min do range [a, b]
// Update: troca o valor de uma posicao
// Usa O(q) de memoria para q updates
//
// Complexidades:
// query - O(log(n))
// update - O(log(n))

template<typename T> struct seg {
	struct node {
		node* ch[2];
		char d;
		T v;

		T mi;

		node(int d_, T v_, T val) : d(d_), v(v_) {
			ch[0] = ch[1] = NULL;
			mi = val;
		}
		node(node* x) : d(x->d), v(x->v), mi(x->mi) {
			ch[0] = x->ch[0], ch[1] = x->ch[1];
		}
		void update() {
			mi = numeric_limits<T>::max();
			for (int i = 0; i < 2; i++) if (ch[i])
				mi = min(mi, ch[i]->mi);
		}
	};

	node* root;
	char n;

	seg() : root(NULL), n(0) {}
	~seg() {
		std::vector<node*> q = {root};
		while (q.size()) {
			node* x = q.back(); q.pop_back();
			if (!x) continue;
			q.push_back(x->ch[0]), q.push_back(x->ch[1]);
			delete x;
		}
	}

	char msb(T v, char l, char r) { // msb in range (l, r]
		for (char i = r; i > l; i--) if (v>>i&1) return i;
		return -1;
	}
	void cut(node* at, T v, char i) {
		char d = msb(v ^ at->v, at->d, i);
		if (d == -1) return; // no need to split
		node* nxt = new node(at);
		at->ch[v>>d&1] = NULL;
		at->ch[!(v>>d&1)] = nxt;
		at->d = d;
	}

	node* update(node* at, T idx, T val, char i) {
		if (!at) return new node(-1, idx, val);
		cut(at, idx, i);
		if (at->d == -1) { // leaf
			at->mi = val;
			return at;
		}
		bool dir = idx>>at->d&1;
		at->ch[dir] = update(at->ch[dir], idx, val, at->d-1);
		at->update();
		return at;
	}
	void update(T idx, T val) {
		while (idx>>n) n++;
		root = update(root, idx, val, n-1);
	}

	T query(node* at, T a, T b, T l, T r, char i) {
		if (!at or b < l or r < a) return
			numeric_limits<T>::max();
		if (a <= l and r <= b) return at->mi;
		T m = l + (r-l)/2;
		if (at->d < i) {
			if ((at->v>>i&1) == 0) return query(at, a, b, l,
				m, i-1);
```

```
            else return query(at, a, b, m+1, r, i-1);
        }
        return min(query(at->ch[0], a, b, l, m, i-1),
            query(at->ch[1], a, b, m+1, r, i-1));
    }
    T query(T l, T r) { return query(root, l, r, 0,
        (1<<n)-1, n-1); }
};
```

## 1.27 SegTree Iterativa com Lazy Propagation

```
// Query: soma do range [a, b]
// Update: soma x em cada elemento do range [a, b]
// Para mudar, mudar as funcoes junta, poe e query
// LOG = ceil(log2(MAX))
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

namespace seg {
    ll seg[2*MAX], lazy[2*MAX];
    int n;

    ll junta(ll a, ll b) {
        return a+b;
    }

    // soma x na posicao p de tamanho tam
    void poe(int p, ll x, int tam, bool prop=1) {
        seg[p] += x*tam;
        if (prop and p < n) lazy[p] += x;
    }

    // atualiza todos os pais da folha p
    void sobe(int p) {
        for (int tam = 2; p /= 2; tam *= 2) {
            seg[p] = junta(seg[2*p], seg[2*p+1]);
            poe(p, lazy[p], tam, 0);
        }
    }

    // propaga o caminho da raiz ate a folha p
    void prop(int p) {
        int tam = 1 << (LOG-1);
        for (int s = LOG; s; s--, tam /= 2) {
            int i = p >> s;
            if (lazy[i]) {
                poe(2*i, lazy[i], tam);
                poe(2*i+1, lazy[i], tam);
                lazy[i] = 0;
            }
        }
    }

    void build(int n2, int* v) {
        n = n2;
        for (int i = 0; i < n; i++) seg[n+i] = v[i];
        for (int i = n-1; i; i--) seg[i] = junta(seg[2*i],
            seg[2*i+1]);
        for (int i = 0; i < 2*n; i++) lazy[i] = 0;
    }

    ll query(int a, int b) {
        ll ret = 0;
        for (prop(a+=n), prop(b+=n); a <= b; ++a/=2, --b/=2)
            {
            if (a%2 == 1) ret = junta(ret, seg[a]);
            if (b%2 == 0) ret = junta(ret, seg[b]);
        }
        return ret;
    }

    void update(int a, int b, int x) {
        int a2 = a += n, b2 = b += n, tam = 1;
        for (; a <= b; ++a/=2, --b/=2, tam *= 2) {
            if (a%2 == 1) poe(a, x, tam);
            if (b%2 == 0) poe(b, x, tam);
        }
        sobe(a2), sobe(b2);
    }
```

```
};
```

## 1.28 SegTree Colorida

```
// Cada posicao tem um valor e uma cor
// O construtor receve um vector de {valor, cor}
// e o numero de cores (as cores devem estar em [0, c-1])
// query(c, a, b) retorna a soma dos valores
// de todo mundo em [a, b] que tem cor c
// update(c, a, b, x) soma x em todo mundo em
// [a, b] que tem cor c
// paint(c1, c2, a, b) faz com que todo mundo
// em [a, b] que tem cor c1 passe a ter cor c2
//
// Complexidades:
// construir - O(n log(n)) espaco e tempo
// query - O(log(n))
// update - O(log(n))
// paint - O(log(n)) amortizado

struct seg_color {
	struct node {
		node *l, *r;
		int cnt;
		ll val, lazy;
		node() : l(NULL), r(NULL), cnt(0), val(0), lazy(0) {}
		void update() {
			cnt = 0, val = 0;
			for (auto i : {l, r}) if (i) {
				i->prop();
				cnt += i->cnt, val += i->val;
			}
		}
		void prop() {
			if (!lazy) return;
			val += lazy*(ll)cnt;
			for (auto i : {l, r}) if (i) i->lazy += lazy;
			lazy = 0;
		}
	};

	int n;
	vector<node*> seg;

	seg_color(vector<pair<int, int>>& v, int c) :
		n(v.size()), seg(c, NULL) {
		for (int i = 0; i < n; i++)
			seg[v[i].second] = insert(seg[v[i].second], i,
				v[i].first, 0, n-1);
	}
	~seg_color() {
		queue<node*> q;
		for (auto i : seg) q.push(i);
		while (q.size()) {
			auto i = q.front(); q.pop();
			if (!i) continue;
			q.push(i->l), q.push(i->r);
			delete i;
		}
	}

	node* insert(node* at, int idx, int val, int l, int r) {
		if (!at) at = new node();
		if (l == r) return at->cnt = 1, at->val = val, at;
		int m = (l+r)/2;
		if (idx <= m) at->l = insert(at->l, idx, val, l, m);
		else at->r = insert(at->r, idx, val, m+1, r);
		return at->update(), at;
	}
	ll query(node* at, int a, int b, int l, int r) {
		if (!at or b < l or r < a) return 0;
		at->prop();
		if (a <= l and r <= b) return at->val;
		int m = (l+r)/2;
		return query(at->l, a, b, l, m) + query(at->r, a, b,
			m+1, r);
	}
	ll query(int c, int a, int b) { return query(seg[c], a,
		b, 0, n-1); }
	void update(node* at, int a, int b, int x, int l, int r)
		{
		if (!at or b < l or r < a) return;
```

```
        at->prop();
        if (a <= l and r <= b) {
            at->lazy += x;
            return void(at->prop());
        }
        int m = (l+r)/2;
        update(at->l, a, b, x, l, m), update(at->r, a, b, x,
            m+1, r);
        at->update();
    }
    void update(int c, int a, int b, int x) { update(seg[c],
        a, b, x, 0, n-1); }
    void paint(node*& from, node*& to, int a, int b, int l,
        int r) {
        if (to == from or !from or b < l or r < a) return;
        from->prop();
        if (to) to->prop();
        if (a <= l and r <= b) {
            if (!to) {
                to = from;
                from = NULL;
                return;
            }
            int m = (l+r)/2;
            paint(from->l, to->l, a, b, l, m),
                paint(from->r, to->r, a, b, m+1, r);
            to->update();
            delete from;
            from = NULL;
            return;
        }
        if (!to) to = new node();
        int m = (l+r)/2;
        paint(from->l, to->l, a, b, l, m), paint(from->r,
            to->r, a, b, m+1, r);
        from->update(), to->update();
    }
    void paint(int c1, int c2, int a, int b) {
        paint(seg[c1], seg[c2], a, b, 0, n-1); }
};
```

## 1.29 SegTree Iterativa

```
// Consultas 0-based
// Valores iniciais devem estar em (seg[n], ... , seg[2*n-1])
// Query: soma do range [a, b]
// Update: muda o valor da posicao p para x
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

int seg[2 * MAX];
int n;

void build() {
    for (int i = n - 1; i; i--) seg[i] = seg[2*i] +
        seg[2*i+1];
}

int query(int a, int b) {
    int ret = 0;
    for(a += n, b += n; a <= b; ++a /= 2, --b /= 2) {
        if (a % 2 == 1) ret += seg[a];
        if (b % 2 == 0) ret += seg[b];
    }
    return ret;
}

void update(int p, int x) {
    seg[p += n] = x;
    while (p /= 2) seg[p] = seg[2*p] + seg[2*p+1];
}
```

## 1.30 SegTree Beats

```
// query(a, b) - {{min(v[a..b]), max(v[a..b])}, sum(v[a..b])}
// updatemin(a, b, x) faz com que v[i] <- min(v[i], x),
// para i em [a, b]
// updatemax faz o mesmo com max, e updatesum soma x
```

```
// em todo mundo do intervalo [a, b]
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log^2 (n)) amortizado
// (se nao usar updatesum, fica log(n) amortizado)

#define f first
#define s second

namespace beats {
    struct node {
        int tam;
        ll sum, lazy; // lazy pra soma
        ll mi1, mi2, mi; // mi = #mi1
        ll ma1, ma2, ma; // ma = #ma1

        node(ll x = 0) {
            sum = mi1 = ma1 = x;
            mi2 = LINF, ma2 = -LINF;
            mi = ma = tam = 1;
            lazy = 0;
        }
        node(const node& l, const node& r) {
            sum = l.sum + r.sum, tam = l.tam + r.tam;
            lazy = 0;
            if (l.mi1 > r.mi1) {
                mi1 = r.mi1, mi = r.mi;
                mi2 = min(l.mi1, r.mi2);
            } else if (l.mi1 < r.mi1) {
                mi1 = l.mi1, mi = l.mi;
                mi2 = min(r.mi1, l.mi2);
            } else {
                mi1 = l.mi1, mi = l.mi+r.mi;
                mi2 = min(l.mi2, r.mi2);
            }
            if (l.ma1 < r.ma1) {
                ma1 = r.ma1, ma = r.ma;
                ma2 = max(l.ma1, r.ma2);
            } else if (l.ma1 > r.ma1) {
                ma1 = l.ma1, ma = l.ma;
```

```
                ma2 = max(r.ma1, l.ma2);
            } else {
                ma1 = l.ma1, ma = l.ma+r.ma;
                ma2 = max(l.ma2, r.ma2);
            }
        }
        void setmin(ll x) {
            if (x >= ma1) return;
            sum += (x - ma1)*ma;
            if (mi1 == ma1) mi1 = x;
            if (mi2 == ma1) mi2 = x;
            ma1 = x;
        }
        void setmax(ll x) {
            if (x <= mi1) return;
            sum += (x - mi1)*mi;
            if (ma1 == mi1) ma1 = x;
            if (ma2 == mi1) ma2 = x;
            mi1 = x;
        }
        void setsum(ll x) {
            mi1 += x, mi2 += x, ma1 += x, ma2 += x;
            sum += x*tam;
            lazy += x;
        }
    };

    node seg[4*MAX];
    int n, *v;

    node build(int p=1, int l=0, int r=n-1) {
        if (l == r) return seg[p] = {v[l]};
        int m = (l+r)/2;
        return seg[p] = {build(2*p, l, m), build(2*p+1, m+1,
            r)};
    }
    void build(int n2, int* v2) {
        n = n2, v = v2;
        build();
    }
    void prop(int p, int l, int r) {
        if (l == r) return;
```

```
    for (int k = 0; k < 2; k++) {
        if (seg[p].lazy) seg[2*p+k].setsum(seg[p].lazy);
        seg[2*p+k].setmin(seg[p].ma1);
        seg[2*p+k].setmax(seg[p].mi1);
    }
    seg[p].lazy = 0;
}
pair<pair<ll, ll>, ll> query(int a, int b, int p=1, int
    l=0, int r=n-1) {
    if (b < l or r < a) return {{LINF, -LINF}, 0};
    if (a <= l and r <= b) return {{seg[p].mi1,
        seg[p].ma1}, seg[p].sum};
    prop(p, l, r);
    int m = (l+r)/2;
    auto L = query(a, b, 2*p, l, m), R = query(a, b,
        2*p+1, m+1, r);
    return {{min(L.f.f, R.f.f), max(L.f.s, R.f.s)},
        L.s+R.s};
}
node updatemin(int a, int b, ll x, int p=1, int l=0, int
    r=n-1) {
    if (b < l or r < a or seg[p].ma1 <= x) return seg[p];
    if (a <= l and r <= b and seg[p].ma2 < x) {
        seg[p].setmin(x);
        return seg[p];
    }
    prop(p, l, r);
    int m = (l+r)/2;
    return seg[p] = {updatemin(a, b, x, 2*p, l, m),
                    updatemin(a, b, x, 2*p+1, m+1, r)};
}
node updatemax(int a, int b, ll x, int p=1, int l=0, int
    r=n-1) {
    if (b < l or r < a or seg[p].mi1 >= x) return seg[p];
    if (a <= l and r <= b and seg[p].mi2 > x) {
        seg[p].setmax(x);
        return seg[p];
    }
    prop(p, l, r);
    int m = (l+r)/2;
    return seg[p] = {updatemax(a, b, x, 2*p, l, m),
                    updatemax(a, b, x, 2*p+1, m+1, r)};
    }
    node updatesum(int a, int b, ll x, int p=1, int l=0, int
        r=n-1) {
        if (b < l or r < a) return seg[p];
        if (a <= l and r <= b) {
            seg[p].setsum(x);
            return seg[p];
        }
        prop(p, l, r);
        int m = (l+r)/2;
        return seg[p] = {updatesum(a, b, x, 2*p, l, m),
                        updatesum(a, b, x, 2*p+1, m+1, r)};
    }
};
```

## 1.31 SegTree Persistente

```
// SegTree de soma, update de somar numa posicao
//
// query(a, b, t) retorna a query de [a, b] na versao t
// update(a, x, t) faz um update v[a]+=x a partir da
// versao de t, criando uma nova versao e retornando seu id
// Por default, faz o update a partir da ultima versao
//
// build - O(n)
// query - O(log(n))
// update - O(log(n))

const int MAX = 3e4+10, UPD = 2e5+10, LOG = 20;
const int MAXS = 4*MAX+UPD*LOG;

namespace perseg {
    ll seg[MAXS];
    int rt[UPD], L[MAXS], R[MAXS], cnt, t;
    int n, *v;

    ll build(int p, int l, int r) {
        if (l == r) return seg[p] = v[l];
        L[p] = cnt++, R[p] = cnt++;
        int m = (l+r)/2;
```

```
        return seg[p] = build(L[p], l, m) + build(R[p], m+1,
            r);
    }
    void build(int n2, int* v2) {
        n = n2, v = v2;
        rt[0] = cnt++;
        build(0, 0, n-1);
    }
    ll query(int a, int b, int p, int l, int r) {
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return seg[p];
        int m = (l+r)/2;
        return query(a, b, L[p], l, m) + query(a, b, R[p],
            m+1, r);
    }
    ll query(int a, int b, int tt) {
        return query(a, b, rt[tt], 0, n-1);
    }
    ll update(int a, int x, int lp, int p, int l, int r) {
        if (l == r) return seg[p] = seg[lp]+x;
        int m = (l+r)/2;
        if (a <= m)
            return seg[p] = update(a, x, L[lp], L[p]=cnt++,
                l, m) + seg[R[p]=R[lp]];
        return seg[p] = seg[L[p]=L[lp]] + update(a, x,
            R[lp], R[p]=cnt++, m+1, r);
    }
    int update(int a, int x, int tt=t) {
        update(a, x, rt[tt], rt[++t]=cnt++, 0, n-1);
        return t;
    }
};
```

## 1.32 SegTree

```
// Recursiva com Lazy Propagation
// Query: soma do range [a, b]
// Update: soma x em cada elemento do range [a, b]
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

namespace seg {
    ll seg[4*MAX], lazy[4*MAX];
    int n, *v;

    ll build(int p=1, int l=0, int r=n-1) {
        lazy[p] = 0;
        if (l == r) return seg[p] = v[l];
        int m = (l+r)/2;
        return seg[p] = build(2*p, l, m) + build(2*p+1, m+1,
            r);
    }
    void build(int n2, int* v2) {
        n = n2, v = v2;
        build();
    }
    void prop(int p, int l, int r) {
        seg[p] += lazy[p]*(r-l+1);
        if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] +=
            lazy[p];
        lazy[p] = 0;
    }
    ll query(int a, int b, int p=1, int l=0, int r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) return seg[p];
        if (b < l or r < a) return 0;
        int m = (l+r)/2;
        return query(a, b, 2*p, l, m) + query(a, b, 2*p+1,
            m+1, r);
    }
    ll update(int a, int b, int x, int p=1, int l=0, int
        r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) {
            lazy[p] += x;
            prop(p, l, r);
            return seg[p];
        }
        if (b < l or r < a) return seg[p];
```

```
        int m = (l+r)/2;
        return seg[p] = update(a, b, x, 2*p, l, m) +
            update(a, b, x, 2*p+1, m+1, r);
    }
};


// Se tiver uma seg de max, da pra descobrir em O(log(n))
// o primeiro e ultimo elemento >= val numa range:

// primeira posicao >= val em [a, b] (ou -1 se nao tem)
int get_left(int a, int b, int val, int p=1, int l=0, int
    r=n-1) {
    if (b < l or r < a or seg[p] < val) return -1;
    if (r == l) return l;
    int m = (l+r)/2;
    int x = get_left(a, b, val, 2*p, l, m);
    if (x != -1) return x;
    return get_left(a, b, val, 2*p+1, m+1, r);
}
// ultima posicao >= val em [a, b] (ou -1 se nao tem)
int get_right(int a, int b, int val, int p=1, int l=0, int
    r=n-1) {
    if (b < l or r < a or seg[p] < val) return -1;
    if (r == l) return l;
    int m = (l+r)/2;
    int x = get_right(a, b, val, 2*p+1, m+1, r);
    if (x != -1) return x;
    return get_right(a, b, val, 2*p, l, m);
}

// Se tiver uma seg de soma sobre um array nao negativo v,
    da pra
// descobrir em O(log(n)) o maior j tal que
    v[i]+v[i+1]+...+v[j-1] < val

int lower_bound(int i, ll& val, int p, int l, int r) {
    if (r < i) return n;
    if (i <= l and seg[p] < val) {
        val -= seg[p];
        return n;
    }
    if (l == r) return l;
    int m = (l+r)/2;
    int x = lower_bound(i, val, 2*p, l, m);
    if (x != n) return x;
    return lower_bound(i, val, 2*p+1, m+1, r);
}
```

## 1.33 DSU Persistente

```
// Persistencia parcial, ou seja, tem que ir
// incrementando o 't' no une
//
// Complexidades:
// build - O(n)
// find - O(log(n))
// une - O(log(n))

int n, p[MAX], sz[MAX], ti[MAX];

void build() {
    for (int i = 0; i < n; i++) {
        p[i] = i;
        sz[i] = 1;
        ti[i] = -INF;
    }
}

int find(int k, int t) {
    if (p[k] == k or ti[k] > t) return k;
    return find(p[k], t);
}

void une(int a, int b, int t) {
    a = find(a, t); b = find(b, t);
    if (a == b) return;
    if (sz[a] > sz[b]) swap(a, b);

    sz[b] += sz[a];
    p[a] = b;
    ti[a] = t;
```

```
}
```

## 1.34 RMQ <O(n), O(1)> - min queue

```
// O(n) pra buildar, query O(1)
// Se tiver varios minimos, retorna
// o de menor indice

template<typename T> struct rmq {
    vector<T> v;
    int n; static const int b = 30;
    vector<int> mask, t;

    int op(int x, int y) { return v[x] <= v[y] ? x : y; }
    int msb(int x) { return
        __builtin_clz(1)-__builtin_clz(x); }
    int small(int r, int sz = b) { return
        r-msb(mask[r]&((1<<sz)-1)); }
    rmq() {}
    rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n),
        t(n) {
        for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
            at = (at<<1)&((1<<b)-1);
            while (at and op(i-msb(at&-at), i) == i) at ^=
                at&-at;
        }
        for (int i = 0; i < n/b; i++) t[i] = small(b*i+b-1);
        for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0;
            i+(1<<j) <= n/b; i++)
            t[n/b*j+i] = op(t[n/b*(j-1)+i],
                t[n/b*(j-1)+i+(1<<(j-1))]);
    }
    int index_query(int l, int r) {
        if (r-l+1 <= b) return small(r, r-l+1);
        int x = l/b+1, y = r/b-1;
        if (x > y) return op(small(l+b-1), small(r));
        int j = msb(y-x+1);
        int ans = op(small(l+b-1), op(t[n/b*j+x],
            t[n/b*j+y-(1<<j)+1]));
        return op(ans, small(r));
    }
    T query(int l, int r) { return v[index_query(l, r)]; }
};
```

## 1.35 Wavelet Tree

```
// Usa O(sigma + n log(sigma)) de memoria,
// onde sigma = MAXN - MINN
// Depois do build, o v fica ordenado
// count(i, j, x, y) retorna o numero de elementos de
// v[i, j) que pertencem a [x, y]
// kth(i, j, k) retorna o elemento que estaria
// na poscicao k-1 de v[i, j), se ele fosse ordenado
// sum(i, j, x, y) retorna a soma dos elementos de
// v[i, j) que pertencem a [x, y]
// sumk(i, j, k) retorna a soma dos k-esimos menores
// elementos de v[i, j) (sum(i, j, 1) retorna o menor)
//
// Complexidades:
// build - O(n log(sigma))
// count - O(log(sigma))
// kth   - O(log(sigma))
// sum   - O(log(sigma))
// sumk  - O(log(sigma))

int n, v[MAX];
vector<int> esq[4*(MAXN-MINN)], pref[4*(MAXN-MINN)];

void build(int b = 0, int e = n, int p = 1, int l = MINN,
    int r = MAXN) {
    int m = (l+r)/2; esq[p].push_back(0);
        pref[p].push_back(0);
    for (int i = b; i < e; i++) {
        esq[p].push_back(esq[p].back()+(v[i]<=m));
        pref[p].push_back(pref[p].back()+v[i]);
    }
    if (l == r) return;
    int m2 = stable_partition(v+b, v+e, [=](int i){return i
        <= m;}) - v;
    build(b, m2, 2*p, l, m), build(m2, e, 2*p+1, m+1, r);
```

```
}

int count(int i, int j, int x, int y, int p = 1, int l =
    MINN, int r = MAXN) {
    if (y < l or r < x) return 0;
    if (x <= l and r <= y) return j-i;
    int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
    return count(ei, ej, x, y, 2*p, l, m)+count(i-ei, j-ej,
        x, y, 2*p+1, m+1, r);
}

int kth(int i, int j, int k, int p=1, int l = MINN, int r =
    MAXN) {
    if (l == r) return l;
    int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
    if (k <= ej-ei) return kth(ei, ej, k, 2*p, l, m);
    return kth(i-ei, j-ej, k-(ej-ei), 2*p+1, m+1, r);
}

int sum(int i, int j, int x, int y, int p = 1, int l = MINN,
    int r = MAXN) {
    if (y < l or r < x) return 0;
    if (x <= l and r <= y) return pref[p][j]-pref[p][i];
    int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
    return sum(ei, ej, x, y, 2*p, l, m) + sum(i-ei, j-ej, x,
        y, 2*p+1, m+1, r);
}

int sumk(int i, int j, int k, int p = 1, int l = MINN, int r
    = MAXN) {
    if (l == r) return l*k;
    int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
    if (k <= ej-ei) return sumk(ei, ej, k, 2*p, l, m);
    return pref[2*p][ej]-pref[2*p][ei]+sumk(i-ei, j-ej,
        k-(ej-ei), 2*p+1, m+1, r);
}
```

# 2 Grafos

## 2.1 Dinic

```
// O(min(m * max_flow, n^2 m))
// Grafo com capacidades 1 -> O(sqrt(n)*m)

struct dinic {
    const bool scaling = false; // com scaling -> O(nm
        log(MAXCAP)),
    int lim;                    // com constante alta
    struct edge {
        int to, cap, rev, flow; // para, capacidade, id da
            reversa, fluxo
        bool res; // se a aresta eh residual
        edge(int to_, int cap_, int rev_, bool res_)
            : to(to_), cap(cap_), rev(rev_), flow(0),
                res(res_) {}
    };

    vector<vector<edge>> g;
    vector<int> lev, beg;
    ll F;
    dinic(int n) : g(n), F(0) {}

    void add(int a, int b, int c) { // de a pra b com cap. c
        g[a].push_back(edge(b, c, g[b].size(), false));
        g[b].push_back(edge(a, 0, g[a].size()-1, true));
    }
    bool bfs(int s, int t) {
        lev = vector<int>(g.size(), -1); lev[s] = 0;
        beg = vector<int>(g.size(), 0);
        queue<int> q; q.push(s);
        while (q.size()) {
            int u = q.front(); q.pop();
            for (auto& i : g[u]) {
                if (lev[i.to] != -1 or (i.flow == i.cap))
                    continue;
                if (scaling and i.cap - i.flow < lim)
                    continue;
                lev[i.to] = lev[u] + 1;
```

```cpp
                q.push(i.to);
            }
        }
        return lev[t] != -1;
    }
    int dfs(int v, int s, int f = INF){
        if (!f or v == s) return f;
        for (int& i = beg[v]; i < g[v].size(); i++) {
            auto& e = g[v][i];
            if (lev[e.to] != lev[v] + 1) continue;
            int foi = dfs(e.to, s, min(f, e.cap - e.flow));
            if (!foi) continue;
            e.flow += foi, g[e.to][e.rev].flow -= foi;
            return foi;
        }
        return 0;
    }
    ll max_flow(int s, int t) {
        for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
            while (bfs(s, t)) while (int ff = dfs(s, t)) F
                += ff;
        return F;
    }
    vector<pair<int, int> > get_cut(int s, int t) {
        max_flow(s, t);
        vector<pair<int, int> > cut;
        vector<int> vis(g.size(), 0), st = {s};
        vis[s] = 1;
        while (st.size()) {
            int u = st.back(); st.pop_back();
            for (auto e : g[u]) if (!vis[e.to] and e.flow <
                e.cap)
                vis[e.to] = 1, st.push_back(e.to);
        }
        for (int i = 0; i < g.size(); i++) for (auto e :
            g[i])
            if (vis[i] and !vis[e.to] and !e.res)
                cut.push_back({i, e.to});
        return cut;
    }
};
```

## 2.2 Kruskal

```cpp
// Gera e retorna uma AGM e seu custo total a partir do
   vetor de arestas (edg)
// do grafo
//
// O(m log(m) + m a(m))

vector<tuple<int, int, int>> edg; // {peso,[x,y]}

// DSU em O(a(n))
void dsu_build();
int find(int a);
void unite(int a, int b);

pair<ll,vector<tuple<int, int, int>>> kruskal(int n) {
    dsu_build(n);
    sort(edg.begin(), edg.end());

    ll cost = 0;
    vector<tuple<int, int, int>> mst;
    for (auto [w,x,y] : edg) if (find(x) != find(y)) {
        mst.push_back({w,x,y});
        cost += w;
        unite(x,y);
    }
    return {cost,mst};
}
```

## 2.3 Sack (DSU em arvores)

```cpp
// Responde queries de todas as sub-arvores
// offline
//
// O(n log(n))

int sz[MAX], cor[MAX], cnt[MAX];
vector<int> g[MAX];

void build(int k, int d=0) {
```

```
    sz[k] = 1;
    for (auto& i : g[k]) {
        build(i, d+1); sz[k] += sz[i];
        if (sz[i] > sz[g[k][0]]) swap(i, g[k][0]);
    }
}

void compute(int k, int x, bool dont=1) {
    cnt[cor[k]] += x;
    for (int i = dont; i < g[k].size(); i++)
        compute(g[k][i], x, 0);
}

void solve(int k, bool keep=0) {
    for (int i = int(g[k].size())-1; i >= 0; i--)
        solve(g[k][i], !i);
    compute(k, 1);

        // agora cnt[i] tem quantas vezes a cor
        // i aparece na sub-arvore do k

    if (!keep) compute(k, -1, 0);
}
```

## 2.4 Block-Cut Tree

```
// Cria a block-cut tree, uma arvore com os blocos
// e os pontos de articulacao
// Blocos sao componentes 2-vertice-conexos maximais
// Uma 2-coloracao da arvore eh tal que uma cor sao
// os blocos, e a outra cor sao os pontos de art.
// art[i] responde se i eh ponto de articulacao
// Funciona pra grafo nao conexo, e ja limpa tudo
//
// O(n+m)

vector<int> g[MAX];
stack<int> s;
int id[MAX], art[MAX], pos[MAX];
vector<vector<int>> blocks, tree;

int dfs(int i, int &t, int p = -1) {
    int lo = id[i] = t++;
    s.push(i);
    for (int j : g[i]) if (j != p) {
        if (id[j] == -1) {
            int val = dfs(j, t, i);
            lo = min(lo, val);

            if (val >= id[i]) {
                art[i]++;
                blocks.emplace_back(1, i);
                while (blocks.back().back() != j)
                    blocks.back().push_back(s.top()),
                        s.pop();
            }
            // if (val > id[i]) aresta i-j eh ponte
        }
        else lo = min(lo, id[j]);
    }
    if (p == -1 and art[i]) art[i]--;
    return lo;
}

void build(int n) {
    for (int i = 0; i < n; i++) id[i] = -1, art[i] = 0;
    blocks.clear(), tree.clear();
    while (s.size()) s.pop();
    int t = 0;

    for (int i = 0; i < n; i++) if (id[i] == -1) dfs(i, t,
        -1);

    tree.resize(blocks.size()); // no maximo 2*n
    for (int i = 0; i < n; i++) if (art[i])
        pos[i] = tree.size(), tree.emplace_back();

    for (int i = 0; i < blocks.size(); i++) for (int j :
        blocks[i]) {
        if (!art[j]) pos[j] = i;
        else tree[i].push_back(pos[j]),
            tree[pos[j]].push_back(i);
```

```
    }
}
```

## 2.5 Topological Sort

```
// Retorna uma ordenacaoo topologica de g
// Se g nao for DAG retorna um vetor vazio
//
// O(n + m)

vector<int> g[MAX];

vector<int> topo_sort(int n) {
    vector<int> ret(n,-1), vis(n,0);

    int pos = n-1, dag = 1;
    function<void(int)> dfs = [&] (int v) {
        vis[v] = 1;
        for (auto u : g[v]) {
            if (vis[u] == 1) dag = 0;
            else if (!vis[u]) dfs(u);
        }
        ret[pos--] = v, vis[v] = 2;
    };

    for (int i=0; i<n; i++) if (!vis[i]) dfs(i);

    if (!dag) ret.clear();
    return ret;
}
```

## 2.6 Max flow com lower bound

```
// Manda passar pelo menos 'lb' de fluxo
// em cada aresta
//
// O(dinic)

struct lb_max_flow : dinic {
    vector<int> d;
    vector<int> e;
    lb_max_flow(int n):dinic(n + 2), d(n, 0){}
    void add(int a, int b, int c, int lb = 0){
        c -= lb;
        d[a] -= lb;
        d[b] += lb;
        dinic::add(a, b, c);
    }
    bool check_flow(int src, int snk, int F){
        int n = d.size();
        d[src] += F;
        d[snk] -= F;

        for (int i = 0; i < n; i++){
            if (d[i] > 0){
                dinic::add(n, i, d[i]);
            } else if (d[i] < 0){
                dinic::add(i, n+1, -d[i]);
            }
        }

        int f = max_flow(n, n+1);
        return (f == F);
    }
};
```

## 2.7 Prufer code

```
// Traduz de lista de arestas para prufer code
// e vice-versa
// Os vertices tem label de 0 a n-1
// Todo array com n-2 posicoes e valores de
// 0 a n-1 sao prufer codes validos
//
// O(n)

vector<int> to_prufer(vector<pair<int, int>> tree) {
    int n = tree.size()+1;
```

```
    vector<int> d(n, 0);
    vector<vector<int>> g(n);
    for (auto [a, b] : tree) d[a]++, d[b]++,
        g[a].push_back(b), g[b].push_back(a);
    vector<int> pai(n, -1);
    queue<int> q; q.push(n-1);
    while (q.size()) {
        int u = q.front(); q.pop();
        for (int v : g[u]) if (v != pai[u])
            pai[v] = u, q.push(v);
    }
    int idx, x;
    idx = x = find(d.begin(), d.end(), 1) - d.begin();
    vector<int> ret;
    for (int i = 0; i < n-2; i++) {
        int y = pai[x];
        ret.push_back(y);
        if (--d[y] == 1 and y < idx) x = y;
        else idx = x = find(d.begin()+idx+1, d.end(), 1) -
           d.begin();
    }
    return ret;
}

vector<pair<int, int>> from_prufer(vector<int> p) {
    int n = p.size()+2;
    vector<int> d(n, 1);
    for (int i : p) d[i]++;
    p.push_back(n-1);
    int idx, x;
    idx = x = find(d.begin(), d.end(), 1) - d.begin();
    vector<pair<int, int>> ret;
    for (int y : p) {
        ret.push_back({x, y});
        if (--d[y] == 1 and y < idx) x = y;
        else idx = x = find(d.begin()+idx+1, d.end(), 1) -
           d.begin();
    }
    return ret;
}
```

## 2.8 Tarjan para SCC

```
// O(n + m)

vector<int> g[MAX];
stack<int> s;
int vis[MAX], comp[MAX];
int id[MAX];

// se quiser comprimir ciclo ou achar ponte em grafo nao
   direcionado,
// colocar um if na dfs para nao voltar pro pai da DFS tree
int dfs(int i, int& t) {
    int lo = id[i] = t++;
    s.push(i);
    vis[i] = 2;

    for (int j : g[i]) {
        if (!vis[j]) lo = min(lo, dfs(j, t));
        else if (vis[j] == 2) lo = min(lo, id[j]);
    }

    // aresta de i pro pai eh uma ponte (no caso nao
       direcionado)
    if (lo == id[i]) while (1) {
        int u = s.top(); s.pop();
        vis[u] = 1, comp[u] = i;
        if (u == i) break;
    }

    return lo;
}

void tarjan(int n) {
    int t = 0;
    for (int i = 0; i < n; i++) vis[i] = 0;

    for (int i = 0; i < n; i++) if (!vis[i]) dfs(i, t);
}
```

## 2.9 Dijkstra

```
// encontra menor distancia de x
// para todos os vertices
// se ao final do algoritmo d[i] = LINF,
// entao x nao alcanca i
//
// O(m log(n))

ll d[MAX];
vector<pair<int,int>> g[MAX]; // {vizinho, peso}

int n;

void dijkstra(int x) {
    for (int i=0; i < n; i++) d[i] = LINF;
    d[x] = 0;
    priority_queue<pair<ll,int>> pq;
    pq.push({0,x});

    while (pq.size()) {
        auto [ndist,u] = pq.top(); pq.pop();
        if (-ndist > d[u]) continue;

        for (auto [idx,w] : g[u]) if (d[idx] > d[u] + w) {
            d[idx] = d[u] + w;
            pq.push({-d[idx], idx});
        }
    }
}
```

## 2.10 Floyd-Warshall

```
// encontra o menor caminho entre todo
// par de vertices e detecta ciclo negativo
// returna 1 sse ha ciclo negativo
// d[i][i] deve ser 0
// para i != j, d[i][j] deve ser w se ha uma aresta
// (i, j) de peso w, INF caso contrario
//
// O(n^3)

int n;
int d[MAX][MAX];

bool floyd_warshall() {
    for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

    for (int i = 0; i < n; i++)
        if (d[i][i] < 0) return 1;

    return 0;
}
```

## 2.11 Virtual Tree

```
// Comprime uma arvore dado um conjunto S de vertices, de
   forma que
// o conjunto de vertices da arvore comprimida contenha S e
   seja
// minimal e fechado sobre a operacao de LCA
// Se |S| = k, a arvore comprimida tem O(k) vertices
//
// O(k log(k))

template<typename T> struct rmq {
    vector<T> v;
    int n; static const int b = 30;
    vector<int> mask, t;

    int op(int x, int y) { return v[x] < v[y] ? x : y; }
    int msb(int x) { return
        __builtin_clz(1)-__builtin_clz(x); }
    rmq() {}
    rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n),
        t(n) {
        for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
```

```
            at = (at<<1)&((1<<b)-1);
            while (at and op(i, i-msb(at&-at)) == i) at ^=
                at&-at;
        }
        for (int i = 0; i < n/b; i++) t[i] =
            b*i+b-1-msb(mask[b*i+b-1]);
        for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0;
            i+(1<<j) <= n/b; i++)
            t[n/b*j+i] = op(t[n/b*(j-1)+i],
                t[n/b*(j-1)+i+(1<<(j-1))]);
    }
    int small(int r, int sz = b) { return
        r-msb(mask[r]&((1<<sz)-1)); }
    T query(int l, int r) {
        if (r-l+1 <= b) return small(r, r-l+1);
        int ans = op(small(l+b-1), small(r));
        int x = l/b+1, y = r/b-1;
        if (x <= y) {
            int j = msb(y-x+1);
            ans = op(ans, op(t[n/b*j+x],
                t[n/b*j+y-(1<<j)+1]));
        }
        return ans;
    }
};

namespace lca {
    vector<int> g[MAX];
    int v[2*MAX], pos[MAX], dep[2*MAX];
    int t;
    rmq<int> RMQ;

    void dfs(int i, int d = 0, int p = -1) {
        v[t] = i, pos[i] = t, dep[t++] = d;
        for (int j : g[i]) if (j != p) {
            dfs(j, d+1, i);
            v[t] = i, dep[t++] = d;
        }
    }
    void build(int n, int root) {
        t = 0;
        dfs(root);
        RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
    }
    int lca(int a, int b) {
        a = pos[a], b = pos[b];
        return v[RMQ.query(min(a, b), max(a, b))];
    }
    int dist(int a, int b) {
        return dep[pos[a]] + dep[pos[b]] - 2*dep[pos[lca(a,
            b)]];
    }
}

vector<int> virt[MAX];

#warning lembrar de buildar o LCA antes
int build_virt(vector<int> v) {
    auto cmp = [&](int i, int j) { return lca::pos[i] <
        lca::pos[j]; };
    sort(v.begin(), v.end(), cmp);
    for (int i = v.size()-1; i; i--)
        v.push_back(lca::lca(v[i], v[i-1]));
    sort(v.begin(), v.end(), cmp);
    v.erase(unique(v.begin(), v.end()), v.end());
    for (int i : v) virt[i].clear();
    for (int i = 1; i < v.size(); i++) {
#warning soh to colocando aresta descendo
        virt[lca::lca(v[i-1], v[i])].push_back(v[i]);
    }
    return v[0];
}
```

## 2.12 Bellman-Ford

```
// Calcula a menor distancia
// entre a e todos os vertices e
// detecta ciclo negativo
// Retorna 1 se ha ciclo negativo
// Nao precisa representar o grafo,
// soh armazenar as arestas
//
```

```
// O(nm)

int n, m;
int d[MAX];
vector<pair<int, int>> ar; // vetor de arestas
vector<int> w;              // peso das arestas

bool bellman_ford(int a) {
    for (int i = 0; i < n; i++) d[i] = INF;
    d[a] = 0;

    for (int i = 0; i <= n; i++)
        for (int j = 0; j < m; j++) {
            if (d[ar[j].second] > d[ar[j].first] + w[j]) {
                if (i == n) return 1;

                d[ar[j].second] = d[ar[j].first] + w[j];
            }
        }

    return 0;
}
```

## 2.13 AGM Direcionada

```
// Fala o menor custo para selecionar arestas tal que
// o vertice 'r' alcance todos
// Se nao tem como, retorna LINF
//
// O(m log(n))

struct node {
    pair<ll, int> val;
    ll lazy;
    node *l, *r;
    node() {}
    node(pair<int, int> v) : val(v), lazy(0), l(NULL),
        r(NULL) {}

    void prop() {
        val.first += lazy;
        if (l) l->lazy += lazy;
        if (r) r->lazy += lazy;
        lazy = 0;
    }
};
void merge(node*& a, node* b) {
    if (!a) swap(a, b);
    if (!b) return;
    a->prop(), b->prop();
    if (a->val > b->val) swap(a, b);
    merge(rand()%2 ? a->l : a->r, b);
}
pair<ll, int> pop(node*& R) {
    R->prop();
    auto ret = R->val;
    node* tmp = R;
    merge(R->l, R->r);
    R = R->l;
    if (R) R->lazy -= ret.first;
    delete tmp;
    return ret;
}
void apaga(node* R) { if (R) apaga(R->l), apaga(R->r),
    delete R; }

ll dmst(int n, int r, vector<pair<pair<int, int>, int>>& ar)
    {
    vector<int> p(n); iota(p.begin(), p.end(), 0);
    function<int(int)> find = [&](int k) { return
        p[k]==k?k:p[k]=find(p[k]); };
    vector<node*> h(n);
    for (auto e : ar) merge(h[e.first.second], new
        node({e.second, e.first.first}));
    vector<int> pai(n, -1), path(n);
    pai[r] = r;
    ll ans = 0;

    for (int i = 0; i < n; i++) { // vai conectando todo
        mundo
        int u = i, at = 0;
        while (pai[u] == -1) {
```

```
            if (!h[u]) { // nao tem
                for (auto i : h) apaga(i);
                return LINF;
            }
            path[at++] = u, pai[u] = i;
            auto [mi, v] = pop(h[u]);
            ans += mi;

            if (pai[u = find(v)] == i) { // ciclo
                while (find(v = path[--at]) != u)
                    merge(h[u], h[v]), h[v] = NULL,
                       p[find(v)] = u;
                pai[u] = -1;
            }
        }
    }
    for (auto i : h) apaga(i);
    return ans;
}
```

## 2.14 Blossom - matching maximo em grafo geral

```
// O(n^3)
// Se for bipartido, nao precisa da funcao
// 'contract', e roda em O(nm)

vector<int> g[MAX];
int match[MAX]; // match[i] = com quem i esta matchzado ou -1
int n, pai[MAX], base[MAX], vis[MAX];
queue<int> q;

void contract(int u, int v, bool first = 1) {
    static vector<bool> bloss;
    static int l;
    if (first) {
        bloss = vector<bool>(n, 0);
        vector<bool> teve(n, 0);
        int k = u; l = v;
        while (1) {
            teve[k = base[k]] = 1;
            if (match[k] == -1) break;
            k = pai[match[k]];
        }
        while (!teve[l = base[l]]) l = pai[match[l]];
    }
    while (base[u] != l) {
        bloss[base[u]] = bloss[base[match[u]]] = 1;
        pai[u] = v;
        v = match[u];
        u = pai[match[u]];
    }
    if (!first) return;
    contract(v, u, 0);
    for (int i = 0; i < n; i++) if (bloss[base[i]]) {
        base[i] = l;
        if (!vis[i]) q.push(i);
        vis[i] = 1;
    }
}

int getpath(int s) {
    for (int i = 0; i < n; i++) base[i] = i, pai[i] = -1,
       vis[i] = 0;
    vis[s] = 1; q = queue<int>(); q.push(s);
    while (q.size()) {
        int u = q.front(); q.pop();
        for (int i : g[u]) {
            if (base[i] == base[u] or match[u] == i)
                continue;
            if (i == s or (match[i] != -1 and pai[match[i]]
                != -1))
                 contract(u, i);
            else if (pai[i] == -1) {
                 pai[i] = u;
                 if (match[i] == -1) return i;
                 i = match[i];
                 vis[i] = 1; q.push(i);
            }
        }
    }
    return -1;
}
```

```
int blossom() {
	int ans = 0;
	memset(match, -1, sizeof(match));
	for (int i = 0; i < n; i++) if (match[i] == -1)
		for (int j : g[i]) if (match[j] == -1) {
			match[i] = j;
			match[j] = i;
			ans++;
			break;
		}
	for (int i = 0; i < n; i++) if (match[i] == -1) {
		int j = getpath(i);
		if (j == -1) continue;
		ans++;
		while (j != -1) {
			int p = pai[j], pp = match[p];
			match[p] = j;
			match[j] = p;
			j = pp;
		}
	}
	return ans;
}
```

## 2.15 LCA com RMQ

```
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// dist(a, b) retorna a distancia entre a e b
//
// Complexidades:
// build - O(n)
// lca - O(1)
// dist - O(1)

template<typename T> struct rmq {
	vector<T> v;
	int n; static const int b = 30;
	vector<int> mask, t;

	int op(int x, int y) { return v[x] < v[y] ? x : y; }
	int msb(int x) { return
		__builtin_clz(1)-__builtin_clz(x); }
	rmq() {}
	rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n),
		t(n) {
		for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
			at = (at<<1)&((1<<b)-1);
			while (at and op(i, i-msb(at&-at)) == i) at ^=
				at&-at;
		}
		for (int i = 0; i < n/b; i++) t[i] =
			b*i+b-1-msb(mask[b*i+b-1]);
		for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0;
			i+(1<<j) <= n/b; i++)
			t[n/b*j+i] = op(t[n/b*(j-1)+i],
				t[n/b*(j-1)+i+(1<<(j-1))]);
	}
	int small(int r, int sz = b) { return
		r-msb(mask[r]&((1<<sz)-1)); }
	T query(int l, int r) {
		if (r-l+1 <= b) return small(r, r-l+1);
		int ans = op(small(l+b-1), small(r));
		int x = l/b+1, y = r/b-1;
		if (x <= y) {
			int j = msb(y-x+1);
			ans = op(ans, op(t[n/b*j+x],
				t[n/b*j+y-(1<<j)+1]));
		}
		return ans;
	}
};

namespace lca {
	vector<int> g[MAX];
	int v[2*MAX], pos[MAX], dep[2*MAX];
	int t;
	rmq<int> RMQ;

	void dfs(int i, int d = 0, int p = -1) {
		v[t] = i, pos[i] = t, dep[t++] = d;
```

```
        for (int j : g[i]) if (j != p) {
            dfs(j, d+1, i);
            v[t] = i, dep[t++] = d;
        }
    }
    void build(int n, int root) {
        t = 0;
        dfs(root);
        RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
    }
    int lca(int a, int b) {
        a = pos[a], b = pos[b];
        return v[RMQ.query(min(a, b), max(a, b))];
    }
    int dist(int a, int b) {
        return dep[pos[a]] + dep[pos[b]] - 2*dep[pos[lca(a,
           b)]];
    }
}
```

## 2.16 Heavy-Light Decomposition sem Update

```
// query de min do caminho
//
// Complexidades:
// build - O(n)
// query_path - O(log(n))

#define f first
#define s second

namespace hld {
    vector<pair<int, int> > g[MAX];
    int pos[MAX], sz[MAX];
    int sobe[MAX], pai[MAX];
    int h[MAX], v[MAX], t;
    int men[MAX], seg[2*MAX];

    void build_hld(int k, int p = -1, int f = 1) {
        v[pos[k] = t++] = sobe[k]; sz[k] = 1;
        for (auto& i : g[k]) if (i.first != p) {
            sobe[i.first] = i.second; pai[i.first] = k;
            h[i.first] = (i == g[k][0] ? h[k] : i.first);
            men[i.first] = (i == g[k][0] ? min(men[k],
               i.second) : i.second);
            build_hld(i.first, k, f); sz[k] += sz[i.first];

            if (sz[i.first] > sz[g[k][0].first] or
               g[k][0].first == p)
                swap(i, g[k][0]);
        }
        if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
    }
    void build(int root = 0) {
        t = 0;
        build_hld(root);
        for (int i = 0; i < t; i++) seg[i+t] = v[i];
        for (int i = t-1; i; i--) seg[i] = min(seg[2*i],
           seg[2*i+1]);
    }
    int query_path(int a, int b) {
        if (a == b) return INF;
        if (pos[a] < pos[b]) swap(a, b);

        if (h[a] != h[b]) return min(men[a],
           query_path(pai[h[a]], b));
        int ans = INF, x = pos[b]+1+t, y = pos[a]+t;
        for (; x <= y; ++x/=2, --y/=2) ans = min({ans,
           seg[x], seg[y]});
        return ans;
    }
};
```

## 2.17 Heavy-Light Decomposition - aresta

```
// SegTree de soma
// query / update de soma das arestas
//
// Complexidades:
// build - O(n)
```

```cpp
// query_path - O(log^2 (n))
// update_path - O(log^2 (n))
// query_subtree - O(log(n))
// update_subtree - O(log(n))

namespace seg { ... }

namespace hld {
    vector<pair<int, int> > g[MAX];
    int pos[MAX], sz[MAX];
    int sobe[MAX], pai[MAX];
    int h[MAX], v[MAX], t;

    void build_hld(int k, int p = -1, int f = 1) {
        v[pos[k] = t++] = sobe[k]; sz[k] = 1;
        for (auto& i : g[k]) if (i.first != p) {
            auto [u, w] = i;
            sobe[u] = w; pai[u] = k;
            h[u] = (i == g[k][0] ? h[k] : u);
            build_hld(u, k, f); sz[k] += sz[u];

            if (sz[u] > sz[g[k][0].first] or g[k][0].first
                == p)
                swap(i, g[k][0]);
        }
        if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
    }
    void build(int root = 0) {
        t = 0;
        build_hld(root);
        seg::build(t, v);
    }
    ll query_path(int a, int b) {
        if (a == b) return 0;
        if (pos[a] < pos[b]) swap(a, b);

        if (h[a] == h[b]) return seg::query(pos[b]+1,
            pos[a]);
        return seg::query(pos[h[a]], pos[a]) +
            query_path(pai[h[a]], b);
    }
    void update_path(int a, int b, int x) {
        if (a == b) return;
        if (pos[a] < pos[b]) swap(a, b);

        if (h[a] == h[b]) return (void)seg::update(pos[b]+1,
            pos[a], x);
        seg::update(pos[h[a]], pos[a], x);
            update_path(pai[h[a]], b, x);
    }
    ll query_subtree(int a) {
        if (sz[a] == 1) return 0;
        return seg::query(pos[a]+1, pos[a]+sz[a]-1);
    }
    void update_subtree(int a, int x) {
        if (sz[a] == 1) return;
        seg::update(pos[a]+1, pos[a]+sz[a]-1, x);
    }
    int lca(int a, int b) {
        if (pos[a] < pos[b]) swap(a, b);
        return h[a] == h[b] ? b : lca(pai[h[a]], b);
    }
}
```

## 2.18 LCA com binary lifting

```cpp
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// MAX2 = ceil(log(MAX))
//
// Complexidades:
// build - O(n log(n))
// lca - O(log(n))

vector<vector<int> > g(MAX);
int n, p;
int pai[MAX2][MAX];
int in[MAX], out[MAX];

void dfs(int k) {
    in[k] = p++;
    for (int i = 0; i < (int) g[k].size(); i++)
```

```
        if (in[g[k][i]] == -1) {
            pai[0][g[k][i]] = k;
            dfs(g[k][i]);
        }
    out[k] = p++;
}

void build(int raiz) {
    for (int i = 0; i < n; i++) pai[0][i] = i;
    p = 0, memset(in, -1, sizeof in);
    dfs(raiz);

    // pd dos pais
    for (int k = 1; k < MAX2; k++) for (int i = 0; i < n;
       i++)
        pai[k][i] = pai[k - 1][pai[k - 1][i]];
}

bool anc(int a, int b) { // se a eh ancestral de b
    return in[a] <= in[b] and out[a] >= out[b];
}

int lca(int a, int b) {
    if (anc(a, b)) return a;
    if (anc(b, a)) return b;

    // sobe a
    for (int k = MAX2 - 1; k >= 0; k--)
        if (!anc(pai[k][a], b)) a = pai[k][a];

    return pai[0][a];
}

// Alternativamente:
// 'binary lifting' gastando O(n) de memoria
// Da pra add folhas e fazer queries online
// 3 vezes o tempo do binary lifting normal
//
// build - O(n)
// kth, lca, dist - O(log(n))

int d[MAX], p[MAX], pp[MAX];

void set_root(int i) { p[i] = pp[i] = i, d[i] = 0; }

void add_leaf(int i, int u) {
    p[i] = u, d[i] = d[u]+1;
    pp[i] = 2*d[pp[u]] == d[pp[pp[u]]]+d[u] ? pp[pp[u]] : u;
}

int kth(int i, int k) {
    int dd = max(0, d[i]-k);
    while (d[i] > dd) i = d[pp[i]] >= dd ? pp[i] : p[i];
    return i;
}

int lca(int a, int b) {
    if (d[a] < d[b]) swap(a, b);
    while (d[a] > d[b]) a = d[pp[a]] >= d[b] ? pp[a] : p[a];
    while (a != b) {
        if (pp[a] != pp[b]) a = pp[a], b = pp[b];
        else a = p[a], b = p[b];
    }
    return a;
}

int dist(int a, int b) { return d[a]+d[b]-2*d[lca(a,b)]; }

vector<int> g[MAX];

void build(int i, int pai=-1) {
    if (pai == -1) set_root(i);
    for (int j : g[i]) if (j != pai) {
        add_leaf(j, i);
        build(j, i);
    }
}
```

## 2.19 Heavy-Light Decomposition - vertice

```
// SegTree de soma
// query / update de soma dos vertices
```

```
//
// Complexidades:
// build - O(n)
// query_path - O(log^2 (n))
// update_path - O(log^2 (n))
// query_subtree - O(log(n))
// update_subtree - O(log(n))

namespace seg { ... }

namespace hld {
    vector<int> g[MAX];
    int pos[MAX], sz[MAX];
    int peso[MAX], pai[MAX];
    int h[MAX], v[MAX], t;

    void build_hld(int k, int p = -1, int f = 1) {
        v[pos[k] = t++] = peso[k]; sz[k] = 1;
        for (auto& i : g[k]) if (i != p) {
            pai[i] = k;
            h[i] = (i == g[k][0] ? h[k] : i);
            build_hld(i, k, f); sz[k] += sz[i];

            if (sz[i] > sz[g[k][0]] or g[k][0] == p) swap(i,
                g[k][0]);
        }
        if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
    }
    void build(int root = 0) {
        t = 0;
        build_hld(root);
        seg::build(t, v);
    }
    ll query_path(int a, int b) {
        if (pos[a] < pos[b]) swap(a, b);

        if (h[a] == h[b]) return seg::query(pos[b], pos[a]);
        return seg::query(pos[h[a]], pos[a]) +
            query_path(pai[h[a]], b);
    }
    void update_path(int a, int b, int x) {
        if (pos[a] < pos[b]) swap(a, b);

        if (h[a] == h[b]) return (void)seg::update(pos[b],
            pos[a], x);
        seg::update(pos[h[a]], pos[a], x);
            update_path(pai[h[a]], b, x);
    }
    ll query_subtree(int a) {
        return seg::query(pos[a], pos[a]+sz[a]-1);
    }
    void update_subtree(int a, int x) {
        seg::update(pos[a], pos[a]+sz[a]-1, x);
    }
    int lca(int a, int b) {
        if (pos[a] < pos[b]) swap(a, b);
        return h[a] == h[b] ? b : lca(pai[h[a]], b);
    }
}
```

## 2.20 LCA com HLD

```
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// Para buildar pasta chamar build(root)
// anc(a, b) responde se 'a' eh ancestral de 'b'
//
// Complexidades:
// build - O(n)
// lca - O(log(n))
// anc - O(1)

vector<int> g[MAX];
int pos[MAX], h[MAX], sz[MAX];
int pai[MAX], t;

void build(int k, int p = -1, int f = 1) {
    pos[k] = t++; sz[k] = 1;
    for (int& i : g[k]) if (i != p) {
        pai[i] = k;
        h[i] = (i == g[k][0] ? h[k] : i);
        build(i, k, f); sz[k] += sz[i];
```

```
        if (sz[i] > sz[g[k][0]] or g[k][0] == p) swap(i,
            g[k][0]);
    }
    if (p*f == -1) t = 0, h[k] = k, build(k, -1, 0);
}

int lca(int a, int b) {
    if (pos[a] < pos[b]) swap(a, b);
    return h[a] == h[b] ? b : lca(pai[h[a]], b);
}

bool anc(int a, int b) {
    return pos[a] <= pos[b] and pos[b] <= pos[a]+sz[a]-1;
}
```

## 2.21 MinCostMaxFlow

```
// min_cost_flow(s, t, f) computa o par (fluxo, custo)
// com max(fluxo) <= f que tenha min(custo)
// min_cost_flow(s, t) -> Fluxo maximo de custo minimo de s
   pra t
// Se for um dag, da pra substituir o SPFA por uma DP pra nao
// para O(nm) no comeco
// Se nao tiver aresta com custo negativo, nao precisa do
   SPFA
//
// O(nm + f * m log n)

template<typename T> struct mcmf {
    struct edge {
        int to, rev, flow, cap; // para, id da reversa,
            fluxo, capacidade
        bool res; // se eh reversa
        T cost; // custo da unidade de fluxo
        edge() : to(0), rev(0), flow(0), cap(0), cost(0),
            res(false) {}
        edge(int to_, int rev_, int flow_, int cap_, T
            cost_, bool res_)
            : to(to_), rev(rev_), flow(flow_), cap(cap_),
                res(res_), cost(cost_) {}
    };

    vector<vector<edge>> g;
    vector<int> par_idx, par;
    T inf;
    vector<T> dist;

    mcmf(int n) : g(n), par_idx(n), par(n),
        inf(numeric_limits<T>::max()/3) {}

    void add(int u, int v, int w, T cost) { // de u pra v
        com cap w e custo cost
        edge a = edge(v, g[v].size(), 0, w, cost, false);
        edge b = edge(u, g[u].size(), 0, 0, -cost, true);

        g[u].push_back(a);
        g[v].push_back(b);
    }

    vector<T> spfa(int s) { // nao precisa se nao tiver
        custo negativo
        deque<int> q;
        vector<bool> is_inside(g.size(), 0);
        dist = vector<T>(g.size(), inf);

        dist[s] = 0;
        q.push_back(s);
        is_inside[s] = true;

        while (!q.empty()) {
            int v = q.front();
            q.pop_front();
            is_inside[v] = false;

            for (int i = 0; i < g[v].size(); i++) {
                auto [to, rev, flow, cap, res, cost] =
                    g[v][i];
                if (flow < cap and dist[v] + cost <
                    dist[to]) {
                    dist[to] = dist[v] + cost;
```

```
                if (is_inside[to]) continue;
                if (!q.empty() and dist[to] >
                    dist[q.front()]) q.push_back(to);
                else q.push_front(to);
                is_inside[to] = true;
            }
        }
    }
    return dist;
}
bool dijkstra(int s, int t, vector<T>& pot) {
    priority_queue<pair<T, int>, vector<pair<T, int>>,
        greater<>> q;
    dist = vector<T>(g.size(), inf);
    dist[s] = 0;
    q.emplace(0, s);
    while (q.size()) {
        auto [d, v] = q.top();
        q.pop();
        if (dist[v] < d) continue;
        for (int i = 0; i < g[v].size(); i++) {
            auto [to, rev, flow, cap, res, cost] =
                g[v][i];
            cost += pot[v] - pot[to];
            if (flow < cap and dist[v] + cost <
                dist[to]) {
                 dist[to] = dist[v] + cost;
                 q.emplace(dist[to], to);
                 par_idx[to] = i, par[to] = v;
            }
        }
    }
    return dist[t] < inf;
}

pair<int, T> min_cost_flow(int s, int t, int flow = INF)
   {
    vector<T> pot(g.size(), 0);
    pot = spfa(s); // mudar algoritmo de caminho minimo
       aqui

    int f = 0;
    T ret = 0;
    while (f < flow and dijkstra(s, t, pot)) {
        for (int i = 0; i < g.size(); i++)
            if (dist[i] < inf) pot[i] += dist[i];

        int mn_flow = flow - f, u = t;
        while (u != s){
            mn_flow = min(mn_flow,
                g[par[u]][par_idx[u]].cap -
                   g[par[u]][par_idx[u]].flow);
            u = par[u];
        }

        ret += pot[t] * mn_flow;

        u = t;
        while (u != s) {
            g[par[u]][par_idx[u]].flow += mn_flow;
            g[u][g[par[u]][par_idx[u]].rev].flow -=
               mn_flow;
            u = par[u];
        }

        f += mn_flow;
    }

    return make_pair(f, ret);
}

// Opcional: retorna as arestas originais por onde passa
   flow = cap
vector<pair<int,int>> recover() {
    vector<pair<int,int>> used;
    for (int i = 0; i < g.size(); i++) for (edge e :
       g[i])
        if(e.flow == e.cap && !e.res) used.push_back({i,
           e.to});
    return used;
}
};
```

## 2.22 Algoritmo de Kuhn

```
// Computa matching maximo em grafo bipartido
// 'n' e 'm' sao quantos vertices tem em cada particao
// chamar add(i, j) para add aresta entre o cara i
// da particao A, e o cara j da particao B
// (entao i < n, j < m)
// Para recuperar o matching, basta olhar 'ma' e 'mb'
// recover() recupera o min vertex cover como um par de
// {caras da particao A, caras da particao B}
//
// O(|V| * |E|)
// Na pratica, parece rodar tao rapido quanto o Dinic

mt19937 rng((int)
   chrono::steady_clock::now().time_since_epoch().count());

struct kuhn {
    int n, m;
    vector<vector<int>> g;
    vector<int> vis, ma, mb;

    kuhn(int n_, int m_) : n(n_), m(m_), g(n),
        vis(n+m), ma(n, -1), mb(m, -1) {}

    void add(int a, int b) { g[a].push_back(b); }

    bool dfs(int i) {
        vis[i] = 1;
        for (int j : g[i]) if (!vis[n+j]) {
            vis[n+j] = 1;
            if (mb[j] == -1 or dfs(mb[j])) {
                ma[i] = j, mb[j] = i;
                return true;
            }
        }
        return false;
    }
    int matching() {
        int ret = 0, aum = 1;
        for (auto& i : g) shuffle(i.begin(), i.end(), rng);
        while (aum) {
            for (int j = 0; j < m; j++) vis[n+j] = 0;
            aum = 0;
            for (int i = 0; i < n; i++)
                if (ma[i] == -1 and dfs(i)) ret++, aum = 1;
        }
        return ret;
    }
    pair<vector<int>, vector<int>> recover() {
        matching();
        for (int i = 0; i < n+m; i++) vis[i] = 0;
        for (int i = 0; i < n; i++) if (ma[i] == -1) dfs(i);
        vector<int> ca, cb;
        for (int i = 0; i < n; i++) if (!vis[i])
            ca.push_back(i);
        for (int i = 0; i < m; i++) if (vis[n+i])
            cb.push_back(i);
        return {ca, cb};
    }
};
```

## 2.23 Functional Graph

```
// rt[i] fala o ID da raiz associada ao vertice i
// d[i] fala a profundidade (0 sse ta no ciclo)
// pos[i] fala a posicao de i no array que eh a concat. dos
   ciclos
// build(f, val) recebe a funcao f e o custo de ir de
// i para f[i] (por default, val = f)
// f_k(i, k) fala onde i vai parar se seguir k arestas
// path(i, k) fala o custo (soma) seguir k arestas a partir
   de i
// Se quiser outra operacao, da pra alterar facil o codigo
// Codigo um pouco louco, tenho que admitir
//
// build - O(n)
// f_k   - O(log(min(n, k)))
// path  - O(log(min(n, k)))

namespace func_graph {
    int n;
```

```cpp
int f[MAX], vis[MAX], d[MAX];
int p[MAX], pp[MAX], rt[MAX], pos[MAX];
int sz[MAX], comp;
vector<vector<int>> ciclo;
ll val[MAX], jmp[MAX], seg[2*MAX];

ll op(ll a, ll b) { return a+b; }; // mudar a operacao
    aqui
void dfs(int i, int t = 2) {
    vis[i] = t;
    if (vis[f[i]] >= 2) { // comeca ciclo - f[i] eh o
        rep.
        d[i] = 0, rt[i] = comp;
        sz[comp] = t - vis[f[i]] + 1;
        p[i] = pp[i] = i, jmp[i] = val[i];
        ciclo.emplace_back();
        ciclo.back().push_back(i);
    } else {
        if (!vis[f[i]]) dfs(f[i], t+1);
        rt[i] = rt[f[i]];
        if (sz[comp]+1) { // to no ciclo
            d[i] = 0;
            p[i] = pp[i] = i, jmp[i] = val[i];
            ciclo.back().push_back(i);
        } else { // nao to no ciclo
            d[i] = d[f[i]]+1, p[i] = f[i];
            pp[i] = 2*d[pp[f[i]]] ==
                d[pp[pp[f[i]]]]+d[f[i]] ? pp[pp[f[i]]] :
                f[i];
            jmp[i] = pp[i] == f[i] ? val[i] : op(val[i],
                op(jmp[f[i]], jmp[pp[f[i]]]));
        }
    }
    if (f[ciclo[rt[i]][0]] == i) comp++; // fim do ciclo
    vis[i] = 1;
}
void build(vector<int> f_, vector<int> val_ = {}) {
    n = f_.size(), comp = 0;
    if (!val_.size()) val_ = f_;
    for (int i = 0; i < n; i++)
        f[i] = f_[i], val[i] = val_[i], vis[i] = 0,
            sz[i] = -1;
    ciclo.clear();
    for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);
    int t = 0;
    for (auto& c : ciclo) {
        reverse(c.begin(), c.end());
        for (int j : c) {
            pos[j] = t;
            seg[n+t] = val[j];
            t++;
        }
    }
    for (int i = n-1; i; i--) seg[i] = op(seg[2*i],
        seg[2*i+1]);
}

int f_k(int i, ll k) {
    while (d[i] and k) {
        int big = d[i] - d[pp[i]];
        if (big <= k) k -= big, i = pp[i];
        else k--, i = p[i];
    }
    if (!k) return i;
    return ciclo[rt[i]][(pos[i] - pos[ciclo[rt[i]][0]] +
        k) % sz[rt[i]]];
}
ll path(int i, ll k) {
    auto query = [&](int l, int r) {
        ll q = 0;
        for (l += n, r += n; l <= r; ++l/=2, --r/=2) {
            if (l%2 == 1) q = op(q, seg[l]);
            if (r%2 == 0) q = op(q, seg[r]);
        }
        return q;
    };
    ll ret = 0;
    while (d[i] and k) {
        int big = d[i] - d[pp[i]];
        if (big <= k) k -= big, ret = op(ret, jmp[i]), i
            = pp[i];
        else k--, ret = op(ret, val[i]), i = p[i];
    }
```

```
        if (!k) return ret;
        int first = pos[ciclo[rt[i]][0]], last =
            pos[ciclo[rt[i]].back()];

        // k/sz[rt[i]] voltas completas
        if (k/sz[rt[i]]) ret = op(ret, k/sz[rt[i]] *
            query(first, last));

        k %= sz[rt[i]];
        if (!k) return ret;
        int l = pos[i], r = first + (pos[i] - first + k - 1)
            % sz[rt[i]];
        if (l <= r) return op(ret, query(l, r));
        return op(ret, op(query(l, last), query(first, r)));
    }
}
```

## 2.24 Euler Path / Euler Cycle

```
// Para declarar: 'euler<true> E(n);' se quiser
// direcionado e com 'n' vertices
// As funcoes retornam um par com um booleano
// indicando se possui o cycle/path que voce pediu,
// e um vector de {vertice, id da aresta para chegar no
    vertice}
// Se for get_path, na primeira posicao o id vai ser -1
// get_path(src) tenta achar um caminho ou ciclo euleriano
// comecando no vertice 'src'.
// Se achar um ciclo, o primeiro e ultimo vertice serao
    'src'.
// Se for um P3, um possiveo retorno seria [0, 1, 2, 0]
// get_cycle() acha um ciclo euleriano se o grafo for
    euleriano.
// Se for um P3, um possivel retorno seria [0, 1, 2]
// (vertie inicial nao repete)
//
// O(n+m)

template<bool directed=false> struct euler {
    int n;
    vector<vector<pair<int, int>>> g;
    vector<int> used;

    euler(int n_) : n(n_), g(n) {}
    void add(int a, int b) {
        int at = used.size();
        used.push_back(0);
        g[a].push_back({b, at});
        if (!directed) g[b].push_back({a, at});
    }
#warning chamar para o src certo!
    pair<bool, vector<pair<int, int>>> get_path(int src) {
        if (!used.size()) return {true, {}};
        vector<int> beg(n, 0);
        for (int& i : used) i = 0;
        // {{vertice, anterior}, label}
        vector<pair<pair<int, int>, int>> ret, st = {{{src,
            -1}, -1}};
        while (st.size()) {
            int at = st.back().first.first;
            int& it = beg[at];
            while (it < g[at].size() and
                used[g[at][it].second]) it++;
            if (it == g[at].size()) {
                if (ret.size() and ret.back().first.second
                    != at)
                    return {false, {}};
                ret.push_back(st.back()), st.pop_back();
            } else {
                st.push_back({{g[at][it].first, at},
                    g[at][it].second});
                used[g[at][it].second] = 1;
            }
        }
        if (ret.size() != used.size()+1) return {false, {}};
        vector<pair<int, int>> ans;
        for (auto i : ret) ans.push_back({i.first.first,
            i.second});
        reverse(ans.begin(), ans.end());
        return {true, ans};
    }
    pair<bool, vector<pair<int, int>>> get_cycle() {
```

```
        if (!used.size()) return {true, {}};
        int src = 0;
        while (!g[src].size()) src++;
        auto ans = get_path(src);
        if (!ans.first or ans.second[0].first !=
            ans.second.back().first)
            return {false, {}};
        ans.second[0].second = ans.second.back().second;
        ans.second.pop_back();
        return ans;
    }
};
```

## 2.25 Dominator Tree - Kawakami

```
// Se vira pra usar ai
//
// build - O(n)
// dominates - O(1)

int n;

namespace DTree {
    vector<int> g[MAX];

    // The dominator tree
    vector<int> tree[MAX];
    int dfs_l[MAX], dfs_r[MAX];

    // Auxiliary data
    vector<int> rg[MAX], bucket[MAX];
    int idom[MAX], sdom[MAX], prv[MAX], pre[MAX];
    int ancestor[MAX], label[MAX];
    vector<int> preorder;

    void dfs(int v) {
        static int t = 0;
        pre[v] = ++t;
        sdom[v] = label[v] = v;
        preorder.push_back(v);
        for (int nxt: g[v]) {
            if (sdom[nxt] == -1) {
                prv[nxt] = v;
                dfs(nxt);
            }
            rg[nxt].push_back(v);
        }
    }
    int eval(int v) {
        if (ancestor[v] == -1) return v;
        if (ancestor[ancestor[v]] == -1) return label[v];
        int u = eval(ancestor[v]);
        if (pre[sdom[u]] < pre[sdom[label[v]]]) label[v] = u;
        ancestor[v] = ancestor[u];
        return label[v];
    }
    void dfs2(int v) {
        static int t = 0;
        dfs_l[v] = t++;
        for (int nxt: tree[v]) dfs2(nxt);
        dfs_r[v] = t++;
    }
    void build(int s) {
        for (int i = 0; i < n; i++) {
            sdom[i] = pre[i] = ancestor[i] = -1;
            rg[i].clear();
            tree[i].clear();
            bucket[i].clear();
        }
        preorder.clear();
        dfs(s);
        if (preorder.size() == 1) return;
        for (int i = int(preorder.size()) - 1; i >= 1; i--) {
            int w = preorder[i];
            for (int v: rg[w]) {
                int u = eval(v);
                if (pre[sdom[u]] < pre[sdom[w]]) sdom[w] =
                    sdom[u];
            }
            bucket[sdom[w]].push_back(w);
            ancestor[w] = prv[w];
            for (int v: bucket[prv[w]]) {
```

```
            int u = eval(v);
            idom[v] = (u == v) ? sdom[v] : u;
        }
        bucket[prv[w]].clear();
    }
    for (int i = 1; i < preorder.size(); i++) {
        int w = preorder[i];
        if (idom[w] != sdom[w]) idom[w] = idom[idom[w]];
        tree[idom[w]].push_back(w);
    }
    idom[s] = sdom[s] = -1;
    dfs2(s);
}

// Whether every path from s to v passes through u
bool dominates(int u, int v) {
    if (pre[v] == -1) return 1; // vacuously true
    return dfs_l[u] <= dfs_l[v] && dfs_r[v] <= dfs_r[u];
}
};
```

## 2.26 Line Tree

```
// Reduz min-query em arvore para RMQ
// Se o grafo nao for uma arvore, as queries
// sao sobre a arvore geradora maxima
// Queries de minimo
//
// build - O(n log(n))
// query - O(log(n))

int n;

namespace linetree {
    int id[MAX], seg[2*MAX], pos[MAX];
    vector<int> v[MAX], val[MAX];
    vector<pair<int, pair<int, int> > > ar;

    void add(int a, int b, int p) { ar.push_back({p, {a,
        b}}); }
    void build() {
        sort(ar.rbegin(), ar.rend());
        for (int i = 0; i < n; i++) id[i] = i, v[i] = {i},
            val[i].clear();
        for (auto i : ar) {
            int a = id[i.second.first], b =
                id[i.second.second];
            if (a == b) continue;
            if (v[a].size() < v[b].size()) swap(a, b);
            for (auto j : v[b]) id[j] = a, v[a].push_back(j);
            val[a].push_back(i.first);
            for (auto j : val[b]) val[a].push_back(j);
            v[b].clear(), val[b].clear();
        }
        vector<int> vv;
        for (int i = 0; i < n; i++) for (int j = 0; j <
            v[i].size(); j++) {
            pos[v[i][j]] = vv.size();
            if (j + 1 < v[i].size()) vv.push_back(val[i][j]);
            else vv.push_back(0);
        }
        for (int i = n; i < 2*n; i++) seg[i] = vv[i-n];
        for (int i = n-1; i; i--) seg[i] = min(seg[2*i],
            seg[2*i+1]);
    }
    int query(int a, int b) {
        if (id[a] != id[b]) return 0; // nao estao conectados
        a = pos[a], b = pos[b];
        if (a > b) swap(a, b);
        b--;
        int ans = INF;
        for (a += n, b += n; a <= b; ++a/=2, --b/=2) ans =
            min({ans, seg[a], seg[b]});
        return ans;
    }
};
```

## 2.27 Isomorfismo de arvores

```
// thash() retorna o hash da arvore (usando centroids como
```

```
    vertices especiais).
// Duas arvores sao isomorfas sse seu hash eh o mesmo
//
// O(|V|.log(|V|))

map<vector<int>, int> mphash;

struct tree {
    int n;
    vector<vector<int>> g;
    vector<int> sz, cs;

    tree(int n_) : n(n_), g(n_), sz(n_) {}

    void dfs_centroid(int v, int p) {
        sz[v] = 1;
        bool cent = true;
        for (int u : g[v]) if (u != p) {
            dfs_centroid(u, v), sz[v] += sz[u];
            if(sz[u] > n/2) cent = false;
        }
        if (cent and n - sz[v] <= n/2) cs.push_back(v);
    }
    int fhash(int v, int p) {
        vector<int> h;
        for (int u : g[v]) if (u != p) h.push_back(fhash(u,
            v));
        sort(h.begin(), h.end());
        if (!mphash.count(h)) mphash[h] = mphash.size();
        return mphash[h];
    }
    ll thash() {
        cs.clear();
        dfs_centroid(0, -1);
        if (cs.size() == 1) return fhash(cs[0], -1);
        ll h1 = fhash(cs[0], cs[1]), h2 = fhash(cs[1],
            cs[0]);
        return (min(h1, h2) << 30) + max(h1, h2);
    }
};
```

## 2.28 Link-cut Tree - vertice

```
// Valores nos vertices
// make_tree(v, w) cria uma nova arvore com um
// vertice soh com valor 'w'
// rootify(v) torna v a raiz de sua arvore
// query(v, w) retorna a soma do caminho v--w
// update(v, w, x) soma x nos vertices do caminho v--w
//
// Todas as operacoes sao O(log(n)) amortizado

namespace lct {
    struct node {
        int p, ch[2];
        ll val, sub;
        bool rev;
        int sz;
        ll lazy;
        node() {}
        node(int v) : p(-1), val(v), sub(v), rev(0), sz(1),
            lazy(0) {
            ch[0] = ch[1] = -1;
        }
    };

    node t[MAX];

    void prop(int x) {
        if (t[x].lazy) {
            t[x].val += t[x].lazy, t[x].sub +=
                t[x].lazy*t[x].sz;
            if (t[x].ch[0]+1) t[t[x].ch[0]].lazy +=
                t[x].lazy;
            if (t[x].ch[1]+1) t[t[x].ch[1]].lazy +=
                t[x].lazy;
        }
        if (t[x].rev) {
            swap(t[x].ch[0], t[x].ch[1]);
            if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
            if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
        }
        t[x].lazy = 0, t[x].rev = 0;
```

```
}
void update(int x) {
	t[x].sz = 1, t[x].sub = t[x].val;
	for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
		prop(t[x].ch[i]);
		t[x].sz += t[t[x].ch[i]].sz;
		t[x].sub += t[t[x].ch[i]].sub;
	}
}
bool is_root(int x) {
	return t[x].p == -1 or (t[t[x].p].ch[0] != x and
	   t[t[x].p].ch[1] != x);
}
void rotate(int x) {
	int p = t[x].p, pp = t[p].p;
	if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
	bool d = t[p].ch[0] == x;
	t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
	if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
	t[x].p = pp, t[p].p = x;
	update(p), update(x);
}
int splay(int x) {
	while (!is_root(x)) {
		int p = t[x].p, pp = t[p].p;
		if (!is_root(p)) prop(pp);
		prop(p), prop(x);
		if (!is_root(p)) rotate((t[pp].ch[0] ==
		   p)^(t[p].ch[0] == x) ? x : p);
		rotate(x);
	}
	return prop(x), x;
}
int access(int v) {
	int last = -1;
	for (int w = v; w+1; update(last = w), splay(v), w =
	   t[v].p)
		splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
	return last;
}
void make_tree(int v, int w) { t[v] = node(w); }
int find_root(int v) {
		access(v), prop(v);
		while (t[v].ch[0]+1) v = t[v].ch[0], prop(v);
		return splay(v);
	}
	bool connected(int v, int w) {
		access(v), access(w);
		return v == w ? true : t[v].p != -1;
	}
	void rootify(int v) {
		access(v);
		t[v].rev ^= 1;
	}
	ll query(int v, int w) {
		rootify(w), access(v);
		return t[v].sub;
	}
	void update(int v, int w, int x) {
		rootify(w), access(v);
		t[v].lazy += x;
	}
	void link(int v, int w) {
		rootify(w);
		t[w].p = v;
	}
	void cut(int v, int w) {
		rootify(w), access(v);
		t[v].ch[0] = t[t[v].ch[0]].p = -1;
	}
	int lca(int v, int w) {
		access(v);
		return access(w);
	}
}
```

## 2.29 Link-cut Tree - aresta

```
// Valores nas arestas
// rootify(v) torna v a raiz de sua arvore
// query(v, w) retorna a soma do caminho v--w
// update(v, w, x) soma x nas arestas do caminho v--w
```

```
//
// Todas as operacoes sao O(log(n)) amortizado

namespace lct {
	struct node {
		int p, ch[2];
		ll val, sub;
		bool rev;
		int sz, ar;
		ll lazy;
		node() {}
		node(int v, int ar_) :
		p(-1), val(v), sub(v), rev(0), sz(ar_), ar(ar_),
			lazy(0) {
			ch[0] = ch[1] = -1;
		}
	};

	node t[2*MAX]; // MAXN + MAXQ
	map<pair<int, int>, int> aresta;
	int sz;

	void prop(int x) {
		if (t[x].lazy) {
			if (t[x].ar) t[x].val += t[x].lazy;
			t[x].sub += t[x].lazy*t[x].sz;
			if (t[x].ch[0]+1) t[t[x].ch[0]].lazy +=
				t[x].lazy;
			if (t[x].ch[1]+1) t[t[x].ch[1]].lazy +=
				t[x].lazy;
		}
		if (t[x].rev) {
			swap(t[x].ch[0], t[x].ch[1]);
			if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
			if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
		}
		t[x].lazy = 0, t[x].rev = 0;
	}
	void update(int x) {
		t[x].sz = t[x].ar, t[x].sub = t[x].val;
		for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
			prop(t[x].ch[i]);
			t[x].sz += t[t[x].ch[i]].sz;
			t[x].sub += t[t[x].ch[i]].sub;
		}
	}
	bool is_root(int x) {
		return t[x].p == -1 or (t[t[x].p].ch[0] != x and
			t[t[x].p].ch[1] != x);
	}
	void rotate(int x) {
		int p = t[x].p, pp = t[p].p;
		if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
		bool d = t[p].ch[0] == x;
		t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
		if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
		t[x].p = pp, t[p].p = x;
		update(p), update(x);
	}
	int splay(int x) {
		while (!is_root(x)) {
			int p = t[x].p, pp = t[p].p;
			if (!is_root(p)) prop(pp);
			prop(p), prop(x);
			if (!is_root(p)) rotate((t[pp].ch[0] ==
				p)^(t[p].ch[0] == x) ? x : p);
			rotate(x);
		}
		return prop(x), x;
	}
	int access(int v) {
		int last = -1;
		for (int w = v; w+1; update(last = w), splay(v), w =
			t[v].p)
			splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
		return last;
	}
	void make_tree(int v, int w=0, int ar=0) { t[v] =
		node(w, ar); }
	int find_root(int v) {
		access(v), prop(v);
		while (t[v].ch[0]+1) v = t[v].ch[0], prop(v);
		return splay(v);
	}
```

```
	bool conn(int v, int w) {
		access(v), access(w);
		return v == w ? true : t[v].p != -1;
	}
	void rootify(int v) {
		access(v);
		t[v].rev ^= 1;
	}
	ll query(int v, int w) {
		rootify(w), access(v);
		return t[v].sub;
	}
	void update(int v, int w, int x) {
		rootify(w), access(v);
		t[v].lazy += x;
	}
	void link_(int v, int w) {
		rootify(w);
		t[w].p = v;
	}
	void link(int v, int w, int x) { // v--w com peso x
		int id = MAX + sz++;
		aresta[make_pair(v, w)] = id;
		make_tree(id, x, 1);
		link_(v, id), link_(id, w);
	}
	void cut_(int v, int w) {
		rootify(w), access(v);
		t[v].ch[0] = t[t[v].ch[0]].p = -1;
	}
	void cut(int v, int w) {
		int id = aresta[make_pair(v, w)];
		cut_(v, id), cut_(id, w);
	}
	int lca(int v, int w) {
		access(v);
		return access(w);
	}
}
```

## 2.30 Link-cut Tree

```
// Link-cut tree padrao
//
// Todas as operacoes sao O(log(n)) amortizado

namespace lct {
	struct node {
		int p, ch[2];
		node() { p = ch[0] = ch[1] = -1; }
	};

	node t[MAX];

	bool is_root(int x) {
		return t[x].p == -1 or (t[t[x].p].ch[0] != x and
			t[t[x].p].ch[1] != x);
	}
	void rotate(int x) {
		int p = t[x].p, pp = t[p].p;
		if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
		bool d = t[p].ch[0] == x;
		t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
		if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
		t[x].p = pp, t[p].p = x;
	}
	void splay(int x) {
		while (!is_root(x)) {
			int p = t[x].p, pp = t[p].p;
			if (!is_root(p)) rotate((t[pp].ch[0] ==
				p)^(t[p].ch[0] == x) ? x : p);
			rotate(x);
		}
	}
	int access(int v) {
		int last = -1;
		for (int w = v; w+1; last = w, splay(v), w = t[v].p)
			splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
		return last;
	}
	int find_root(int v) {
		access(v);
```

```
        while (t[v].ch[0]+1) v = t[v].ch[0];
        return splay(v), v;
    }
    void link(int v, int w) { // v deve ser raiz
        access(v);
        t[v].p = w;
    }
    void cut(int v) { // remove aresta de v pro pai
        access(v);
        t[v].ch[0] = t[t[v].ch[0]].p = -1;
    }
    int lca(int v, int w) {
        return access(v), access(w);
    }
}
```

## 2.31 Centro de arvore

```
// Retorna o diametro e o(s) centro(s) da arvore
// Uma arvore tem sempre um ou dois centros e estes estao no
   meio do diametro
//
// O(n)

vector<int> g[MAX];
int d[MAX], par[MAX];

pair<int, vector<int>> center() {
    int f, df;
    function<void(int)> dfs = [&] (int v) {
        if(d[v] > df) f = v, df = d[v];
        for(int u : g[v]) if(u != par[v])
            d[u] = d[v] + 1, par[u] = v, dfs(u);
    };

    f = df = par[0] = -1, d[0] = 0;
    dfs(0);
    int root = f;
    f = df = par[root] = -1, d[root] = 0;
    dfs(root);
```

```
    vector<int> c;
    while (f != -1) {
        if (d[f] == df/2 or d[f] == (df+1)/2) c.push_back(f);
        f = par[f];
    }

    return {df, c};
}
```

## 2.32 Kosaraju

```
// O(n + m)

int n;
vector<int> g[MAX];
vector<int> gi[MAX]; // grafo invertido
int vis[MAX];
stack<int> S;
int comp[MAX]; // componente conexo de cada vertice

void dfs(int k) {
    vis[k] = 1;
    for (int i = 0; i < (int) g[k].size(); i++)
        if (!vis[g[k][i]]) dfs(g[k][i]);

    S.push(k);
}

void scc(int k, int c) {
    vis[k] = 1;
    comp[k] = c;
    for (int i = 0; i < (int) gi[k].size(); i++)
        if (!vis[gi[k][i]]) scc(gi[k][i], c);
}

void kosaraju() {
    for (int i = 0; i < n; i++) vis[i] = 0;
    for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);
```

```
    for (int i = 0; i < n; i++) vis[i] = 0;
    while (S.size()) {
        int u = S.top();
        S.pop();
        if (!vis[u]) scc(u, u);
    }
}
```

## 2.33 Euler Tour Tree

```
// Mantem uma floresta enraizada dinamicamente
// e permite queries/updates em sub-arvore
//
// Chamar ETT E(n, v), passando n = numero de vertices
// e v = vector com os valores de cada vertice (se for vazio,
// constroi tudo com 0
//
// link(v, u) cria uma aresta de v pra u, de forma que u se
   torna
// o pai de v (eh preciso que v seja raiz anteriormente)
// cut(v) corta a resta de v para o pai
// query(v) retorna a soma dos valores da sub-arvore de v
// update(v, val) soma val em todos os vertices da
   sub-arvore de v
// update_v(v, val) muda o valor do vertice v para val
// is_in_subtree(v, u) responde se o vertice u esta na
   sub-arvore de v
//
// Tudo O(log(n)) com alta probabilidade

mt19937 rng((int)
   chrono::steady_clock::now().time_since_epoch().count());

template<typename T> struct ETT {
    // treap
    struct node {
        node *l, *r, *p;
        int pr, sz;
        T val, sub, lazy;
        int id;
        bool f; // se eh o 'first'
        int qt_f; // numero de firsts na subarvore
        node(int id_, T v, bool f_ = 0) : l(NULL), r(NULL),
           p(NULL), pr(rng()),
            sz(1), val(v), sub(v), lazy(), id(id_), f(f_),
               qt_f(f_) {}
        void prop() {
            if (lazy != T()) {
                if (f) val += lazy;
                sub += lazy*sz;
                if (l) l->lazy += lazy;
                if (r) r->lazy += lazy;
            }
            lazy = T();
        }
        void update() {
            sz = 1, sub = val, qt_f = f;
            if (l) l->prop(), sz += l->sz, sub += l->sub,
               qt_f += l->qt_f;
            if (r) r->prop(), sz += r->sz, sub += r->sub,
               qt_f += r->qt_f;
        }
    };

    node* root;

    int size(node* x) { return x ? x->sz : 0; }
    void join(node* l, node* r, node*& i) { // assume que l
       < r
        if (!l or !r) return void(i = l ? l : r);
        l->prop(), r->prop();
        if (l->pr > r->pr) join(l->r, r, l->r), l->r->p = i
           = l;
        else join(l, r->l, r->l), r->l->p = i = r;
        i->update();
    }
    void split(node* i, node*& l, node*& r, int v, int key =
       0) {
        if (!i) return void(r = l = NULL);
        i->prop();
        if (key + size(i->l) < v) {
            split(i->r, i->r, r, v, key+size(i->l)+1), l = i;
```

```
            if (r) r->p = NULL;
            if (i->r) i->r->p = i;
        } else {
            split(i->l, l, i->l, v, key), r = i;
            if (l) l->p = NULL;
            if (i->l) i->l->p = i;
        }
        i->update();
    }
    int get_idx(node* i) {
        int ret = size(i->l);
        for (; i->p; i = i->p) {
            node* pai = i->p;
            if (i != pai->l) ret += size(pai->l) + 1;
        }
        return ret;
    }
    node* get_min(node* i) {
        if (!i) return NULL;
        return i->l ? get_min(i->l) : i;
    }
    node* get_max(node* i) {
        if (!i) return NULL;
        return i->r ? get_max(i->r) : i;
    }
    // fim da treap

    vector<node*> first, last;

    ETT(int n, vector<T> v = {}) : root(NULL), first(n),
       last(n) {
        if (!v.size()) v = vector<T>(n);
        for (int i = 0; i < n; i++) {
            first[i] = last[i] = new node(i, v[i], 1);
            join(root, first[i], root);
        }
    }
    ETT(const ETT& t) { throw logic_error("Nao copiar a
       ETT!"); }
    ~ETT() {
        vector<node*> q = {root};
        while (q.size()) {
            node* x = q.back(); q.pop_back();
            if (!x) continue;
            q.push_back(x->l), q.push_back(x->r);
            delete x;
        }
    }

    pair<int, int> get_range(int i) {
        return {get_idx(first[i]), get_idx(last[i])};
    }
    void link(int v, int u) { // 'v' tem que ser raiz
        auto [lv, rv] = get_range(v);
        int ru = get_idx(last[u]);

        node* V;
        node *L, *M, *R;
        split(root, M, R, rv+1), split(M, L, M, lv);
        V = M;
        join(L, R, root);

        split(root, L, R, ru+1);
        join(L, V, L);
        join(L, last[u] = new node(u, T() /* elemento neutro
           */), L);
        join(L, R, root);
    }
    void cut(int v) {
        auto [l, r] = get_range(v);

        node *L, *M, *R;
        split(root, M, R, r+1), split(M, L, M, l);
        node *LL = get_max(L), *RR = get_min(R);
        if (LL and RR and LL->id == RR->id) { // remove
           duplicata
            if (last[RR->id] == RR) last[RR->id] = LL;
            node *A, *B;
            split(R, A, B, 1);
            delete A;
            R = B;
        }
        join(L, R, root);
        join(root, M, root);
```

```
    }
    T query(int v) {
        auto [l, r] = get_range(v);
        node *L, *M, *R;
        split(root, M, R, r+1), split(M, L, M, l);
        T ans = M->sub;
        join(L, M, M), join(M, R, root);
        return ans;
    }
    void update(int v, T val) { // soma val em todo mundo da
        subarvore
        auto [l, r] = get_range(v);
        node *L, *M, *R;
        split(root, M, R, r+1), split(M, L, M, l);
        M->lazy += val;
        join(L, M, M), join(M, R, root);
    }
    void update_v(int v, T val) { // muda o valor de v pra
        val
        int l = get_idx(first[v]);
        node *L, *M, *R;
        split(root, M, R, l+1), split(M, L, M, l);
        M->val = M->sub = val;
        join(L, M, M), join(M, R, root);
    }
    bool is_in_subtree(int v, int u) { // se u ta na subtree
        de v
        auto [lv, rv] = get_range(v);
        auto [lu, ru] = get_range(u);
        return lv <= lu and ru <= rv;
    }

    void print(node* i) {
        if (!i) return;
        print(i->l);
        cout << i->id+1 << " ";
        print(i->r);
    }
    void print() { print(root); cout << endl; }
};
```

## 2.34 Centroid

```
// Computa os 2 centroids da arvore
//
// O(n)

int n, subsize[MAX];
vector<int> g[MAX];

void dfs(int k, int p=-1) {
    subsize[k] = 1;
    for (int i : g[k]) if (i != p) {
        dfs(i, k);
        subsize[k] += subsize[i];
    }
}

int centroid(int k, int p=-1, int size=-1) {
    if (size == -1) size = subsize[k];
    for (int i : g[k]) if (i != p) if (subsize[i] > size/2)
        return centroid(i, k, size);
    return k;
}

pair<int, int> centroids(int k=0) {
    dfs(k);
    int i = centroid(k), i2 = i;
    for (int j : g[i]) if (2*subsize[j] == subsize[k]) i2 =
       j;
    return {i, i2};
}
```

## 2.35 Vertex cover

```
// Encontra o tamanho do vertex cover minimo
// Da pra alterar facil pra achar os vertices
// Parece rodar com < 2 s pra N = 90
//
// O(n * 1.38^n)
```

```cpp
namespace cover {
    const int MAX = 96;
    vector<int> g[MAX];
    bitset<MAX> bs[MAX];
    int n;

    void add(int i, int j) {
        if (i == j) return;
        n = max({n, i+1, j+1});
        bs[i][j] = bs[j][i] = 1;
    }

    int rec(bitset<MAX> m) {
        int ans = 0;
        for (int x = 0; x < n; x++) if (m[x]) {
            bitset<MAX> comp;
            function<void(int)> dfs = [&](int i) {
                comp[i] = 1, m[i] = 0;
                for (int j : g[i]) if (m[j]) dfs(j);
            };
            dfs(x);

            int ma, deg = -1, cyc = 1;
            for (int i = 0; i < n; i++) if (comp[i]) {
                int d = (bs[i]&comp).count();
                if (d <= 1) cyc = 0;
                if (d > deg) deg = d, ma = i;
            }
            if (deg <= 2) { // caminho ou ciclo
                ans += (comp.count() + cyc) / 2;
                continue;
            }
            comp[ma] = 0;

            // ou ta no cover, ou nao ta no cover
            ans += min(1 + rec(comp), deg + rec(comp & ~
              bs[ma]));
        }
        return ans;
    }
    int solve() {
        bitset<MAX> m;
        for (int i = 0; i < n; i++) {
            m[i] = 1;
            for (int j = 0; j < n; j++)
                if (bs[i][j]) g[i].push_back(j);
        }
        return rec(m);
    }
}
```

## 2.36 Centroid decomposition

```cpp
// Computa pai[i] = pai de i na arv. da centroid
// Descomentar o codigo comentado para computar
// dist[i][x] = distancia na arv. original entre o i e
// o x-esimo ancestral na arv. da centroid
//
// O(n log(n))

int n;
vector<int> g[MAX];
int subsize[MAX];
int rem[MAX];
int pai[MAX];

void dfs(int k, int last) {
    subsize[k] = 1;
    for (int i : g[k])
        if (i != last and !rem[i]) {
            dfs(i, k);
            subsize[k] += subsize[i];
        }
}

int centroid(int k, int last, int size) {
    for (int i : g[k]) {
        if (rem[i] or i == last) continue;
        if (subsize[i] > size / 2)
            return centroid(i, k, size);
    }
```

```
    // k eh o centroid
    return k;
}

//vector<int> dist[MAX];
//void dfs_dist(int k, int last, int d=0) {
//    dist[k].push_back(d);
//    for (int j : g[k]) if (j != last and !rem[j])
//        dfs_dist(j, k, d+1);
//}

void decomp(int k, int last = -1) {
    dfs(k, k);

    // acha e tira o centroid
    int c = centroid(k, k, subsize[k]);
    rem[c] = 1;
    pai[c] = last;
    //dfs_dist(c, c);

    // decompoe as sub-arvores
    for (int i : g[c]) if (!rem[i]) decomp(i, c);
}

void build() {
    memset(rem, 0, sizeof rem);
    decomp(0);
    //for (int i = 0; i < n; i++) reverse(dist[i].begin(),
       dist[i].end());
}
```

# 3 Problemas

## 3.1 Inversion Count

```
// Computa o numero de inversoes para transformar
// l em r (se nao tem como, retorna -1)
//
// O(n log(n))

template<typename T> ll inv_count(vector<T> l, vector<T> r =
    {}) {
    if (!r.size()) {
        r = l;
        sort(r.begin(), r.end());
    }
    int n = l.size();
    vector<int> v(n), bit(n);
    vector<pair<T, int>> w;
    for (int i = 0; i < n; i++) w.push_back({r[i], i+1});
    sort(w.begin(), w.end());
    for (int i = 0; i < n; i++) {
        auto it = lower_bound(w.begin(), w.end(),
            make_pair(l[i], 0));
        if (it == w.end() or it->first != l[i]) return -1;
            // nao da
        v[i] = it->second;
        it->second = -1;
    }

    ll ans = 0;
    for (int i = n-1; i >= 0; i--) {
        for (int j = v[i]-1; j; j -= j&-j) ans += bit[j];
        for (int j = v[i]; j < n; j += j&-j) bit[j]++;
    }
    return ans;
}
```

## 3.2 Gray Code

```
// Gera uma permutacao de 0 a 2^n-1, de forma que
// duas posicoes adjacentes diferem em exatamente 1 bit
//
// O(2^n)

vector<int> gray_code(int n) {
    vector<int> ret(1<<n);
    for (int i = 0; i < (1<<n); i++) ret[i] = i^(i>>1);
    return ret;
```

```
}
```

## 3.3 Points Inside Polygon

```
// Encontra quais pontos estao
// dentro de um poligono simples nao convexo
// o poligono tem lados paralelos aos eixos
// Pontos na borda estao dentro
// Pontos podem estar em ordem horaria ou anti-horaria
//
// O(n log(n))

typedef long long ll;

const ll N = 1e9+10;
const int MAX = 1e5+10;
int ta[MAX];

namespace seg {
    unordered_map<ll, int> seg;
    int query(int a, int b, ll p, ll l, ll r) {
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return seg[p];
        ll m = (l+r)/2;
        return query(a, b, 2*p, l, m)+query(a, b, 2*p+1,
           m+1, r);
    }
    int query(ll p) {
        return query(0, p+N, 1, 0, 2*N);
    }
    int update(ll i, int x, ll p, ll l, ll r) {
        if (i < l or r < i) return seg[p];
        if (l == r) return seg[p] += x;
        ll m = (l+r)/2;
        return seg[p] = update(i, x, 2*p, l, m)+update(i, x,
           2*p+1, m+1, r);
    }
    void update(ll a, ll b, int x) {
        if (a > b) return;
        update(a+N, x, 1, 0, 2*N);
        update(b+N+1, -x, 1, 0, 2*N);
    }
};

void pointsInsidePol(vector<pair<int, int>>& pol,
    vector<pair<int, int>>& v) {
    vector<pair<int, pair<int, pair<int, int>> > > ev; //
       {x, {tipo, {a, b}}}
    // -1: poe ; id: query ; 1e9: tira
    for (int i = 0; i < v.size(); i++)
        ev.pb({v[i].first, {i, {v[i].second, v[i].second}}});
    for (int i = 0; i < pol.size(); i++) {
        pair<int, int> u = pol[i], v = pol[(i+1)%pol.size()];
        if (u.second == v.second) {
            ev.pb({min(u.first, v.first), {-1, {u.second,
               u.second}}});
            ev.pb({max(u.first, v.first), {N, {u.second,
               u.second}}});
            continue;
        }
        int t = N;
        if (u.second > v.second) t = -1;
        ev.pb({u.first, {t, {min(u.second, v.second)+1,
           max(u.second, v.second)}}});
    }

    sort(ev.begin(), ev.end());
    for (int i = 0; i < v.size(); i++) ta[i] = 0;
    for (auto i : ev) {
        pair<int, pair<int, int>> j = i.second;
        if (j.first == -1) seg::update(j.second.first,
           j.second.second, 1);
        else if (j.first == N) seg::update(j.second.first,
           j.second.second, -1);
        else if (seg::query(j.second.first)) ta[j.first] =
           1; // ta dentro
    }
}
```

## 3.4 Sweep Direction

```
// Passa por todas as ordenacoes dos pontos definitas por
    "direcoes"
// Assume que nao existem pontos coincidentes
//
// O(n^2 log n)

void sweep_direction(vector<pt> v) {
    int n = v.size();
    sort(v.begin(), v.end(), [](pt a, pt b) {
        if (a.x != b.x) return a.x < b.x;
        return a.y > b.y;
    });
    vector<int> at(n);
    iota(at.begin(), at.end(), 0);
    vector<pair<int, int>> swapp;
    for (int i = 0; i < n; i++) for (int j = i+1; j < n; j++)
        swapp.push_back({i, j}), swapp.push_back({j, i});

    sort(swapp.begin(), swapp.end(), [&](auto a, auto b) {
        pt A = rotate90(v[a.first] - v[a.second]);
        pt B = rotate90(v[b.first] - v[b.second]);
        if (quad(A) == quad(B) and !sarea2(pt(0, 0), A, B))
            return a < b;
        return compare_angle(A, B);
    });
    for (auto par : swapp) {
        assert(abs(at[par.first] - at[par.second]) == 1);
        int l = min(at[par.first], at[par.second]),
            r = n-1 - max(at[par.first], at[par.second]);
        // l e r sao quantos caras tem de cada lado do par
            de pontos
        // (cada par eh visitado duas vezes)
        swap(v[at[par.first]], v[at[par.second]]);
        swap(at[par.first], at[par.second]);
    }
}
```

## 3.5 Area da Uniao de Retangulos

```
// O(n log(n))

const int MAX = 1e5+10;
namespace seg {
    pair<int, ll> seg[4*MAX];
    ll lazy[4*MAX], *v;
    int n;

    pair<int, ll> merge(pair<int, ll> l, pair<int, ll> r){
        if (l.second == r.second) return {l.first+r.first,
            l.second};
        else if (l.second < r.second) return l;
        else return r;
    }

    pair<int, ll> build(int p=1, int l=0, int r=n-1) {
        lazy[p] = 0;
        if (l == r) return seg[p] = {1, v[l]};
        int m = (l+r)/2;
        return seg[p] = merge(build(2*p, l, m), build(2*p+1,
            m+1, r));
    }
    void build(int n2, ll* v2) {
        n = n2, v = v2;
        build();
    }
    void prop(int p, int l, int r) {
        seg[p].second += lazy[p];
        if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] +=
            lazy[p];
        lazy[p] = 0;
    }
    pair<int, ll> query(int a, int b, int p=1, int l=0, int
        r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) return seg[p];
        if (b < l or r < a) return {0, LINF};
        int m = (l+r)/2;
        return merge(query(a, b, 2*p, l, m), query(a, b,
            2*p+1, m+1, r));
    }
    pair<int, ll> update(int a, int b, int x, int p=1, int
        l=0, int r=n-1) {
```

```
        prop(p, l, r);
        if (a <= l and r <= b) {
            lazy[p] += x;
            prop(p, l, r);
            return seg[p];
        }
        if (b < l or r < a) return seg[p];
        int m = (l+r)/2;
        return seg[p] = merge(update(a, b, x, 2*p, l, m),
                update(a, b, x, 2*p+1, m+1, r));
    }
};

ll seg_vec[MAX];

ll area_sq(vector<pair<pair<int, int>, pair<int, int>>> &sq){
    vector<pair<pair<int, int>, pair<int, int>>> up;
    for (auto it : sq){
        int x1, y1, x2, y2;
        tie(x1, y1) = it.first;
        tie(x2, y2) = it.second;
        up.push_back({{x1+1,  1}, {y1, y2}});
        up.push_back({{x2+1, -1}, {y1, y2}});
    }
    sort(up.begin(), up.end());
    memset(seg_vec, 0, sizeof seg_vec);
    ll H_MAX = MAX;
    seg::build(H_MAX-1, seg_vec);
    auto it = up.begin();
    ll ans = 0;
    while (it != up.end()){
        ll L = (*it).first.first;
        while (it != up.end() && (*it).first.first == L){
            int x, inc, y1, y2;
            tie(x, inc) = it->first;
            tie(y1, y2) = it->second;
            seg::update(y1+1, y2, inc);
            it++;
        }
        if (it == up.end()) break;
        ll R = (*it).first.first;
        ll W = R-L;
        auto jt = seg::query(0, H_MAX-1);
        ll H = H_MAX - 1;
        if (jt.second == 0) H -= jt.first;
        ans += W*H;
    }
    return ans;
}
```

## 3.6 LIS - Longest Increasing Subsequence

```
// Calcula e retorna uma LIS
//
// O(n.log(n))

template<typename T> vector<T> lis(vector<T>& v) {
    int n = v.size(), m = -1;
    vector<T> d(n+1, INF);
    vector<int> l(n);
    d[0] = -INF;

    for (int i = 0; i < n; i++) {
        // Para non-decreasing use upper_bound()
        int t = lower_bound(d.begin(), d.end(), v[i]) -
            d.begin();
        d[t] = v[i], l[i] = t, m = max(m, t);
    }

    int p = n;
    vector<T> ret;
    while (p--) if (l[p] == m) {
        ret.push_back(v[p]);
        m--;
    }
    reverse(ret.begin(),ret.end());

    return ret;
}
```

## 3.7 Distinct Range Query - Persistent Segtree

```
// build - O(n (log n + log(sigma)))
// query - O(log(sigma))

const int MAX = 3e4+10, LOG = 20;
const int MAXS = 4*MAX+MAX*LOG;

namespace perseg {
    ll seg[MAXS];
    int rt[MAX], L[MAXS], R[MAXS], cnt, t;
    int n, *v;

    ll build(int p, int l, int r) {
        if (l == r) return seg[p] = 0;
        L[p] = cnt++, R[p] = cnt++;
        int m = (l+r)/2;
        return seg[p] = build(L[p], l, m) + build(R[p], m+1,
           r);
    }
    void build(int n2) {
        n = n2;
        rt[0] = cnt++;
        build(0, 0, n-1);
    }
    ll query(int a, int b, int p, int l, int r) {
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return seg[p];
        int m = (l+r)/2;
        return query(a, b, L[p], l, m) + query(a, b, R[p],
           m+1, r);
    }
    ll query(int a, int b, int tt) {
        return query(a, b, rt[tt], 0, n-1);
    }
    ll update(int a, int x, int lp, int p, int l, int r) {
        if (l == r) return seg[p] = seg[lp]+x;

        int m = (l+r)/2;
        if (a <= m)
            return seg[p] = update(a, x, L[lp], L[p]=cnt++,
               l, m) + seg[R[p]=R[lp]];
        return seg[p] = seg[L[p]=L[lp]] + update(a, x,
           R[lp], R[p]=cnt++, m+1, r);
    }
    void update(int a, int x, int tt=t) {
        update(a, x, rt[tt], rt[++t]=cnt++, 0, n-1);
    }
};

int qt[MAX];

void build(vector<int>& v) {
    int n = v.size();
    perseg::build(n);
    map<int, int> last;
    int at = 0;
    for (int i = 0; i < n; i++) {
        if (last.count(v[i])) {
            perseg::update(last[v[i]], -1);
            at++;
        }
        perseg::update(i, 1);
        qt[i] = ++at;
        last[v[i]] = i;
    }
}

int query(int l, int r) {
    return perseg::query(l, r, qt[r]);
}
```

## 3.8 Coloracao de Grafo de Intervalo

```
// Colore os intervalos com o numero minimo
// de cores de tal forma que dois intervalos
// que se interceptam tem cores diferentes
// As cores vao de 1 ate n
//
// O(n log(n))

vector<int> coloring(vector<pair<int, int>>& v) {
```

```
    int n = v.size();
    vector<pair<int, pair<int, int>>> ev;
    for (int i = 0; i < n; i++) {
        ev.push_back({v[i].first, {1, i}});
        ev.push_back({v[i].second, {0, i}});
    }
    sort(ev.begin(), ev.end());
    vector<int> ans(n), avl(n);
    for (int i = 0; i < n; i++) avl.push_back(n-i);
    for (auto i : ev) {
        if (i.second.first == 1) {
            ans[i.second.second] = avl.back();
            avl.pop_back();
        } else avl.push_back(ans[i.second.second]);
    }
    return ans;
}
```

## 3.9 MO - DSU

```
// Dado uma lista de arestas de um grafo, responde
// para cada query(l, r), quantos componentes conexos
// o grafo tem se soh considerar as arestas l, l+1, ..., r
// Da pra adaptar pra usar MO com qualquer estrutura
   rollbackavel
//
// O(m sqrt(q) log(n))

struct dsu {
    int n, ans;
    vector<int> p, sz;
    stack<int> S;

    dsu(int n_) : n(n_), ans(n), p(n), sz(n) {
        for (int i = 0; i < n; i++) p[i] = i, sz[i] = 1;
    }
    int find(int k) {
        while (p[k] != k) k = p[k];
        return k;
    }
    void add(pair<int, int> x) {
        int a = x.first, b = x.second;
        a = find(a), b = find(b);
        if (a == b) return S.push(-1);
        ans--;
        if (sz[a] > sz[b]) swap(a, b);
        S.push(a);
        sz[b] += sz[a];
        p[a] = b;
    }
    int query() { return ans; }
    void rollback() {
        int u = S.top(); S.pop();
        if (u == -1) return;
        sz[p[u]] -= sz[u];
        p[u] = u;
        ans++;
    }
};

int n;
vector<pair<int, int>> ar; // vetor com as arestas

vector<int> MO(vector<pair<int, int>> &q) {
    int SQ = ar.size() / sqrt(q.size()) + 1;
    int m = q.size();
    vector<int> ord(m);
    iota(ord.begin(), ord.end(), 0);
    sort(ord.begin(), ord.end(), [&](int l, int r) {
        if (q[l].first / SQ != q[r].first / SQ) return
           q[l].first < q[r].first;
        return q[l].second < q[r].second;
    });
    vector<int> ret(m);

    for (int i = 0; i < m; i++) {
        dsu D(n);
        int fim = q[ord[i]].first/SQ*SQ + SQ - 1;
        int last_r = fim;
        int j = i-1;
        while (j+1 < m and q[ord[j+1]].first / SQ ==
           q[ord[i]].first / SQ) {
```

```
            auto [l, r] = q[ord[++j]];

            if (l / SQ == r / SQ) {
                dsu D2(n);
                for (int k = l; k <= r; k++) D2.add(ar[k]);
                ret[ord[j]] = D2.query();
                continue;
            }

            while (last_r < r) D.add(ar[++last_r]);
            for (int k = l; k <= fim; k++) D.add(ar[k]);

            ret[ord[j]] = D.query();

            for (int k = l; k <= fim; k++) D.rollback();
        }
        i = j;
    }
    return ret;
}
```

## 3.10 Distancia maxima entre dois pontos

```
// max_dist2(v) - O(n log(n))
// max_dist_manhattan - O(n)

// Quadrado da Distancia Euclidiana (precisa copiar
   convex_hull, ccw e pt)
ll max_dist2(vector<pt> v) {
    v = convex_hull(v);
    if (v.size() <= 2) return dist2(v[0], v[1%v.size()]);
    ll ans = 0;
    int n = v.size(), j = 0;
    for (int i = 0; i < n; i++) {
        while (!ccw(v[(i+1)%n]-v[i], pt(0, 0),
            v[(j+1)%n]-v[j])) j = (j+1)%n;
        ans = max({ans, dist2(v[i], v[j]), dist2(v[(i+1)%n],
            v[j])});
    }
    return ans;
}

// Distancia de Manhattan
template<typename T> T max_dist_manhattan(vector<pair<T, T>>
   v) {
    T min_sum, max_sum, min_dif, max_dif;
    min_sum = max_sum = v[0].first + v[0].second;
    min_dif = max_dif = v[0].first - v[0].second;
    for (auto [x, y] : v) {
        min_sum = min(min_sum, x+y);
        max_sum = max(max_sum, x+y);
        min_dif = min(min_dif, x-y);
        max_dif = max(max_dif, x-y);
    }
    return max(max_sum - min_sum, max_dif - min_dif);
}
```

## 3.11 Conectividade Dinamica

```
// Offline com Divide and Conquer e
// DSU com rollback
// O(n log^2(n))

typedef pair<int, int> T;

namespace data {
    int n, ans;
    int p[MAX], sz[MAX];
    stack<int> S;

    void build(int n2) {
        n = n2;
        for (int i = 0; i < n; i++) p[i] = i, sz[i] = 1;
        ans = n;
    }
    int find(int k) {
        while (p[k] != k) k = p[k];
        return k;
    }
    void add(T x) {
```

```
        int a = x.first, b = x.second;
        a = find(a), b = find(b);
        if (a == b) return S.push(-1);
        ans--;
        if (sz[a] > sz[b]) swap(a, b);
        S.push(a);
        sz[b] += sz[a];
        p[a] = b;
    }
    int query() {
        return ans;
    }
    void rollback() {
        int u = S.top(); S.pop();
        if (u == -1) return;
        sz[p[u]] -= sz[u];
        p[u] = u;
        ans++;
    }
};

int ponta[MAX]; // outra ponta do intervalo ou -1 se for
   query
int ans[MAX], n, q;
T qu[MAX];

void solve(int l = 0, int r = q-1) {
    if (l >= r) {
        ans[l] = data::query(); // agora a estrutura ta certa
        return;
    }
    int m = (l+r)/2, qnt = 1;
    for (int i = m+1; i <= r; i++) if (ponta[i]+1 and
       ponta[i] < l)
        data::add(qu[i]), qnt++;
    solve(l, m);
    while (--qnt) data::rollback();
    for (int i = l; i <= m; i++) if (ponta[i]+1 and ponta[i]
       > r)
        data::add(qu[i]), qnt++;
    solve(m+1, r);
    while (qnt--) data::rollback();
}
```

## 3.12 Arpa's Trick

```
// Responde RMQ em O((n+q)log(n)) offline
// Adicionar as queries usando arpa::add(a, b)
// A resposta vai ta em ans[], na ordem que foram colocadas

int n, v[MAX], ans[MAX];

namespace arpa {
    int p[MAX], cnt;
    stack<int> s;
    vector<pair<int, int>> l[MAX];

    int find(int k) { return p[k] == k ? k : p[k] =
       find(p[k]); }
    void add(int a, int b) { l[b].push_back({a, cnt++}); }
    void solve() {
        for (int i = 0; (p[i]=i) < n; s.push(i++)) {
            while (s.size() and v[s.top()] >= v[i])
               p[s.top()] = i, s.pop();
            for (auto q : l[i]) ans[q.second] =
               v[find(q.first)];
        }
    }
}
```

## 3.13 Simple Polygon

```
// Verifica se um poligono com n pontos eh simples
//
// O(n log n)

bool operator < (const line& a, const line& b) { //
   comparador pro sweepline
    if (a.p == b.p) return ccw(a.p, a.q, b.q);
    if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or a.p.x+eps
       < b.p.x))
```

```
        return ccw(a.p, a.q, b.p);
    return ccw(a.p, b.q, b.p);
}

bool simple(vector<pt> v) {
    auto intersects = [&](pair<line, int> a, pair<line, int>
       b) {
        if ((a.second+1)%v.size() == b.second or
            (b.second+1)%v.size() == a.second) return false;
        return interseg(a.first, b.first);
    };
    vector<line> seg;
    vector<pair<pt, pair<int, int>>> w;
    for (int i = 0; i < v.size(); i++) {
        pt at = v[i], nxt = v[(i+1)%v.size()];
        if (nxt < at) swap(at, nxt);
        seg.push_back(line(at, nxt));
        w.push_back({at, {0, i}});
        w.push_back({nxt, {1, i}});
        // casos degenerados estranhos
        if (isinseg(v[(i+2)%v.size()], line(at, nxt)))
            return 0;
        if (isinseg(v[(i+v.size()-1)%v.size()], line(at,
            nxt))) return 0;
    }
    sort(w.begin(), w.end());
    set<pair<line, int>> se;
    for (auto i : w) {
        line at = seg[i.second.second];
        if (i.second.first == 0) {
            auto nxt = se.lower_bound({at, i.second.second});
            if (nxt != se.end() and intersects(*nxt, {at,
                i.second.second})) return 0;
            if (nxt != se.begin() and intersects(*(--nxt),
                {at, i.second.second})) return 0;
            se.insert({at, i.second.second});
        } else {
            auto nxt = se.upper_bound({at,
                i.second.second}), cur = nxt, prev = --cur;
            if (nxt != se.end() and prev != se.begin()
                and intersects(*nxt, *(--prev))) return 0;
            se.erase(cur);
        }
    }
    return 1;
}
```

## 3.14 Algoritmo MO - queries em caminhos de arvore

```
// Problema que resolve: https://www.spoj.com/problems/COT2/
//
// Complexidade sendo c = O(update) e SQ = sqrt(n):
// O((n + q) * sqrt(n) * c)

const int MAX = 40010, SQ = 400;

vector<int> g[MAX];

namespace LCA { ... }

int in[MAX], out[MAX], vtx[2 * MAX];
bool on[MAX];

int dif, freq[MAX];
vector<int> w;

void dfs(int v, int p, int &t) {
    vtx[t] = v, in[v] = t++;
    for (int u : g[v]) if (u != p) {
        dfs(u, v, t);
    }
    vtx[t] = v, out[v] = t++;
}

void update(int p) { // faca alteracoes aqui
    int v = vtx[p];
    if (not on[v]) { // insere vtx v
        dif += (freq[w[v]] == 0);
        freq[w[v]]++;
    }
    else { // retira o vertice v
        dif -= (freq[w[v]] == 1);
```

```
        freq[w[v]]--;
    }
    on[v] = not on[v];
}

vector<tuple<int, int, int>> build_queries(const
    vector<pair<int, int>>& q) {
    LCA::build(0);
    vector<tuple<int, int, int>> ret;
    for (auto [l, r] : q){
        if (in[r] < in[l]) swap(l, r);
        int p = LCA::lca(l, r);
        int init = (p == l) ? in[l] : out[l];
        ret.emplace_back(init, in[r], in[p]);
    }
    return ret;
}

vector<int> mo_tree(const vector<pair<int, int>>& vq){
    int t = 0;
    dfs(0, -1, t);

    auto q = build_queries(vq);

    vector<int> ord(q.size());
    iota(ord.begin(), ord.end(), 0);
    sort(ord.begin(), ord.end(), [&] (int l, int r) {
        int bl = get<0>(q[l]) / SQ, br = get<0>(q[r]) / SQ;
        if (bl != br) return bl < br;
        else if (bl % 2 == 1) return get<1>(q[l]) <
            get<1>(q[r]);
        else return get<1>(q[l]) > get<1>(q[r]);
    });

    memset(freq, 0, sizeof freq);
    dif = 0;

    vector<int> ret(q.size());
    int l = 0, r = -1;
    for (int i : ord) {
        auto [ql, qr, qp] = q[i];
        while (r < qr) update(++r);
        while (l > ql) update(--l);
        while (l < ql) update(l++);
        while (r > qr) update(r--);

        if (qp < l or qp > r) { // se LCA estah entre as
            pontas
            update(qp);
            ret[i] = dif;
            update(qp);
        }
        else ret[i] = dif;
    }
    return ret;
}
```

## 3.15 Area Maxima de Histograma

```
// Assume que todas as barras tem largura 1,
// e altura dada no vetor v
//
// O(n)

ll area(vector<int> v) {
    ll ret = 0;
    stack<int> s;
    // valores iniciais pra dar tudo certo
    v.insert(v.begin(), -1);
    v.insert(v.end(), -1);
    s.push(0);

    for(int i = 0; i < (int) v.size(); i++) {
        while (v[s.top()] > v[i]) {
            ll h = v[s.top()]; s.pop();
            ret = max(ret, h * (i - s.top() - 1));
        }
        s.push(i);
    }

    return ret;
}
```

## 3.16 LIS2 - Longest Increasing Subsequence

```
// Calcula o tamanho da LIS
//
// O(n.log(n))

template<typename T> int lis(vector<T> &v){
    vector<T> ans;
    for (T t : v){
        // Para non-decreasing use upper_bound()
        auto it = lower_bound(ans.begin(), ans.end(), t);
        if (it == ans.end()) ans.push_back(t);
        else *it = t;
    }
    return ans.size();
}
```

## 3.17 Mininum Enclosing Circle

```
// O(n) com alta probabilidade

const double EPS = 1e-12;
mt19937 rng((int)
   chrono::steady_clock::now().time_since_epoch().count());

struct pt {
    double x, y;
    pt(double x_ = 0, double y_ = 0) : x(x_), y(y_) {}
    pt operator + (const pt& p) const { return pt(x+p.x,
        y+p.y); }
    pt operator - (const pt& p) const { return pt(x-p.x,
        y-p.y); }
    pt operator * (double c) const { return pt(x*c, y*c); }
    pt operator / (double c) const { return pt(x/c, y/c); }
};

double dot(pt p, pt q) { return p.x*q.x+p.y*q.y; }
double cross(pt p, pt q) { return p.x*q.y-p.y*q.x; }
double dist(pt p, pt q) { return sqrt(dot(p-q, p-q)); }

pt center(pt p, pt q, pt r) {
    pt a = p-r, b = q-r;
    pt c = pt(dot(a, p+r)/2, dot(b, q+r)/2);
    return pt(cross(c, pt(a.y, b.y)), cross(pt(a.x, b.x),
        c)) / cross(a, b);
}

struct circle {
    pt cen;
    double r;
    circle(pt cen_, double r_) : cen(cen_), r(r_) {}
    circle(pt a, pt b, pt c) {
        cen = center(a, b, c);
        r = dist(cen, a);
    }
    bool inside(pt p) { return dist(p, cen) < r+EPS; }
};

circle minCirc(vector<pt> v) {
    shuffle(v.begin(), v.end(), rng);
    circle ret = circle(pt(0, 0), 0);
    for (int i = 0; i < v.size(); i++) if
        (!ret.inside(v[i])) {
        ret = circle(v[i], 0);
        for (int j = 0; j < i; j++) if (!ret.inside(v[j])) {
            ret = circle((v[i]+v[j])/2, dist(v[i], v[j])/2);
            for (int k = 0; k < j; k++) if
                (!ret.inside(v[k]))
                ret = circle(v[i], v[j], v[k]);
        }
    }
    return ret;
}
```

## 3.18 Conj. Indep. Maximo com Peso em Grafo de Intervalo

```
// Retorna os indices ordenados dos
// intervalos selecionados
```

```
// Se tiver empate, retorna o que minimiza o comprimento
   total
//
// O(n log(n))

vector<int> ind_set(vector<tuple<int, int, int>>& v) {
	vector<tuple<int, int, int>> w;
	for (int i = 0; i < v.size(); i++) {
		w.push_back(tuple(get<0>(v[i]), 0, i));
		w.push_back(tuple(get<1>(v[i]), 1, i));
	}
	sort(w.begin(), w.end());

	vector<int> nxt(v.size());
	vector<pair<ll, int>> dp(v.size());
	int last = -1;
	for (auto [fim, t, i] : w) {
		if (t == 0) {
			nxt[i] = last;
			continue;
		}
		dp[i] = {0, 0};
		if (last != -1) dp[i] = max(dp[i], dp[last]);
		pair<ll, int> pega = {get<2>(v[i]), -(get<1>(v[i]) -
		   get<0>(v[i]) + 1)};
		if (nxt[i] != -1) pega.first += dp[nxt[i]].first,
		   pega.second += dp[nxt[i]].second;
		if (pega > dp[i]) dp[i] = pega;
		else nxt[i] = last;
		last = i;
	}
	pair<ll, int> ans = {0, 0};
	int idx = -1;
	for (int i = 0; i < v.size(); i++) if (dp[i] > ans) ans
	   = dp[i], idx = i;
	vector<int> ret;
	while (idx != -1) {
		if (get<2>(v[idx]) > 0 and
			(nxt[idx] == -1 or get<1>(v[nxt[idx]]) <
			   get<0>(v[idx]))) ret.push_back(idx);
		idx = nxt[idx];
	}
	sort(ret.begin(), ret.end());
	return ret;
}
```

## 3.19 Conectividade Dinamica 2

```
// Offline com link-cut trees
// O(n log(n))

namespace lct {
	struct node {
		int p, ch[2];
		int val, sub;
		bool rev;
		node() {}
		node(int v) : p(-1), val(v), sub(v), rev(0) { ch[0]
		   = ch[1] = -1; }
	};

	node t[2*MAX]; // MAXN + MAXQ
	map<pair<int, int>, int> aresta;
	int sz;

	void prop(int x) {
		if (t[x].rev) {
			swap(t[x].ch[0], t[x].ch[1]);
			if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
			if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
		}
		t[x].rev = 0;
	}
	void update(int x) {
		t[x].sub = t[x].val;
		for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
			prop(t[x].ch[i]);
			t[x].sub = min(t[x].sub, t[t[x].ch[i]].sub);
		}
	}
	bool is_root(int x) {
		return t[x].p == -1 or (t[t[x].p].ch[0] != x and
```

```
        t[t[x].p].ch[1] != x);
}
void rotate(int x) {
    int p = t[x].p, pp = t[p].p;
    if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
    bool d = t[p].ch[0] == x;
    t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
    if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
    t[x].p = pp, t[p].p = x;
    update(p), update(x);
}
int splay(int x) {
    while (!is_root(x)) {
        int p = t[x].p, pp = t[p].p;
        if (!is_root(p)) prop(pp);
        prop(p), prop(x);
        if (!is_root(p)) rotate((t[pp].ch[0] ==
           p)^(t[p].ch[0] == x) ? x : p);
        rotate(x);
    }
    return prop(x), x;
}
int access(int v) {
    int last = -1;
    for (int w = v; w+1; update(last = w), splay(v), w =
        t[v].p)
         splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
    return last;
}
void make_tree(int v, int w=INF) { t[v] = node(w); }
bool conn(int v, int w) {
    access(v), access(w);
    return v == w ? true : t[v].p != -1;
}
void rootify(int v) {
    access(v);
    t[v].rev ^= 1;
}
int query(int v, int w) {
    rootify(w), access(v);
    return t[v].sub;
}
```

```
    void link_(int v, int w) {
        rootify(w);
        t[w].p = v;
    }
    void link(int v, int w, int x) { // v--w com peso x
        int id = MAX + sz++;
        aresta[make_pair(v, w)] = id;
        make_tree(id, x);
        link_(v, id), link_(id, w);
    }
    void cut_(int v, int w) {
        rootify(w), access(v);
        t[v].ch[0] = t[t[v].ch[0]].p = -1;
    }
    void cut(int v, int w) {
        int id = aresta[make_pair(v, w)];
        cut_(v, id), cut_(id, w);
    }
}

void dyn_conn() {
    int n, q; cin >> n >> q;
    vector<int> p(2*q, -1); // outra ponta do intervalo
    for (int i = 0; i < n; i++) lct::make_tree(i);
    vector<pair<int, int>> qu(q);
    map<pair<int, int>, int> m;
    for (int i = 0; i < q; i++) {
        char c; cin >> c;
        if (c == '?') continue;
        int a, b; cin >> a >> b; a--, b--;
        if (a > b) swap(a, b);
        qu[i] = {a, b};
        if (c == '+') {
            p[i] = i+q, p[i+q] = i;
            m[make_pair(a, b)] = i;
        } else {
            int j = m[make_pair(a, b)];
            p[i] = j, p[j] = i;
        }
    }
    int ans = n;
    for (int i = 0; i < q; i++) {
```

```
        if (p[i] == -1) {
            cout << ans << endl; // numero de comp conexos
            continue;
        }
        int a = qu[i].first, b = qu[i].second;
        if (p[i] > i) { // +
            if (lct::conn(a, b)) {
                int mi = lct::query(a, b);
                if (p[i] < mi) {
                    p[p[i]] = p[i];
                    continue;
                }
                lct::cut(qu[p[mi]].first, qu[p[mi]].second),
                    ans++;
                p[mi] = mi;
            }
            lct::link(a, b, p[i]), ans--;
        } else if (p[i] != i) lct::cut(a, b), ans++; // -
    }
}
```

## 3.20 Mo - numero de distintos em range

```
// Para ter o bound abaixo, escolher
// SQ = n / sqrt(q)
//
// O(n * sqrt(q))

const int MAX = 3e4+10;
const int SQ = sqrt(MAX);
int v[MAX];

int ans, freq[MAX];

inline void insert(int p) {
    int o = v[p];
    freq[o]++;
    ans += (freq[o] == 1);
}
inline void erase(int p) {
    int o = v[p];
    ans -= (freq[o] == 1);
    freq[o]--;
}

inline ll hilbert(int x, int y) {
    static int N = (1 << 20);
    int rx, ry, s;
    ll d = 0;
    for (s = N/2; s>0; s /= 2) {
        rx = (x & s) > 0;
        ry = (y & s) > 0;
        d += s * ll(s) * ((3 * rx) ^ ry);
        if (ry == 0) {
            if (rx == 1) {
                x = N-1 - x;
                y = N-1 - y;
            }
            swap(x, y);
        }
    }
    return d;
}

#define HILBERT true
vector<int> MO(vector<pair<int, int>> &q) {
    ans = 0;
    int m = q.size();
    vector<int> ord(m);
    iota(ord.begin(), ord.end(), 0);
#if HILBERT
    vector<ll> h(m);
    for (int i = 0; i < m; i++) h[i] = hilbert(q[i].first,
        q[i].second);
    sort(ord.begin(), ord.end(), [&](int l, int r) { return
        h[l] < h[r]; });
#else
    sort(ord.begin(), ord.end(), [&](int l, int r) {
        if (q[l].first / SQ != q[r].first / SQ) return
            q[l].first < q[r].first;
        if ((q[l].first / SQ) % 2) return q[l].second >
```

```
            q[r].second;
        return q[l].second < q[r].second;
    });
#endif
    vector<int> ret(m);
    int l = 0, r = -1;

    for (int i : ord) {
        int ql, qr;
        tie(ql, qr) = q[i];
        while (r < qr) insert(++r);
        while (l > ql) insert(--l);
        while (l < ql) erase(l++);
        while (r > qr) erase(r--);
        ret[i] = ans;
    }
    return ret;
}
```

## 3.21 Algoritmo Hungaro

```
// Resolve o problema de assignment (matriz n x n)
// Colocar os valores da matriz em 'a' (pode < 0)
// assignment() retorna um par com o valor do
// assignment minimo, e a coluna escolhida por cada linha
//
// O(n^3)

template<typename T> struct hungarian {
    int n;
    vector<vector<T>> a;
    vector<T> u, v;
    vector<int> p, way;
    T inf;

    hungarian(int n_) : n(n_), u(n+1), v(n+1), p(n+1),
        way(n+1) {
        a = vector<vector<T>>(n, vector<T>(n));
        inf = numeric_limits<T>::max();
    }
    pair<T, vector<int>> assignment() {
        for (int i = 1; i <= n; i++) {
            p[0] = i;
            int j0 = 0;
            vector<T> minv(n+1, inf);
            vector<int> used(n+1, 0);
            do {
                used[j0] = true;
                int i0 = p[j0], j1 = -1;
                T delta = inf;
                for (int j = 1; j <= n; j++) if (!used[j]) {
                    T cur = a[i0-1][j-1] - u[i0] - v[j];
                    if (cur < minv[j]) minv[j] = cur, way[j]
                        = j0;
                    if (minv[j] < delta) delta = minv[j], j1
                        = j;
                }
                for (int j = 0; j <= n; j++)
                    if (used[j]) u[p[j]] += delta, v[j] -=
                        delta;
                    else minv[j] -= delta;
                j0 = j1;
            } while (p[j0] != 0);
            do {
                int j1 = way[j0];
                p[j0] = p[j1];
                j0 = j1;
            } while (j0);
        }
        vector<int> ans(n);
        for (int j = 1; j <= n; j++) ans[p[j]-1] = j-1;
        return make_pair(-v[0], ans);
    }
};
```

## 3.22 Dominator Points

```
// Se um ponto A tem ambas as coordenadas >= B, dizemos
// que A domina B
// is_dominated(p) fala se existe algum ponto no conjunto
```

```
// que domina p
// insert(p) insere p no conjunto
// (se p for dominado por alguem, nao vai inserir)
// o multiset 'quina' guarda informacao sobre os pontos
// nao dominados por um elemento do conjunto que nao dominam
// outro ponto nao dominado por um elemento do conjunto
// No caso, armazena os valores de x+y esses pontos
//
// Complexidades:
// is_dominated - O(log(n))
// insert - O(log(n)) amortizado
// query - O(1)

struct dominator_points {
    set<pair<int, int>> se;
    multiset<int> quina;

    bool is_dominated(pair<int, int> p) {
        auto it = se.lower_bound(p);
        if (it == se.end()) return 0;
        return it->second >= p.second;
    }
    void mid(pair<int, int> a, pair<int, int> b, bool rem) {
        pair<int, int> m = {a.first+1, b.second+1};
        int val = m.first + m.second;
        if (!rem) quina.insert(val);
        else quina.erase(quina.find(val));
    }
    bool insert(pair<int, int> p) {
        if (is_dominated(p)) return 0;
        auto it = se.lower_bound(p);
        if (it != se.begin() and it != se.end())
            mid(*prev(it), *it, 1);
        while (it != se.begin()) {
            it--;
            if (it->second > p.second) break;
            if (it != se.begin()) mid(*prev(it), *it, 1);
            it = se.erase(it);
        }
        it = se.insert(p).first;
        if (it != se.begin()) mid(*prev(it), *it, 0);
        if (next(it) != se.end()) mid(*it, *next(it), 0);
            return 1;
    }
    int query() {
        if (!quina.size()) return INF;
        return *quina.begin();
    }
};
```

## 3.23 Min fixed range

```
// https://codeforces.com/contest/1195/problem/E
//
// O(n)
// ans[i] = min_{0 <= j < k} v[i+j]

vector<int> min_k(vector<int> &v, int k){
    int n = v.size();
    deque<int> d;
    auto put = [&](int i){
        while (!d.empty() && v[d.back()] > v[i])
            d.pop_back();
        d.push_back(i);
    };
    for (int i = 0; i < k-1; i++)
        put(i);
    vector<int> ans(n-k+1);
    for (int i = 0; i < n-k+1; i++){
        put(i+k-1);
        while (i > d.front()) d.pop_front();
        ans[i] = v[d.front()];
    }
    return ans;
}
```

## 3.24 Binomial modular

```
// Computa C(n, k) mod m em O(m + log(m) log(n))
// = O(rapido)
```

```cpp
ll divi[MAX];

ll expo(ll a, ll b, ll m) {
    if (!b) return 1;
    ll ans = expo(a*a%m, b/2, m);
    if (b%2) ans *= a;
    return ans%m;
}

ll inv(ll a, ll b){
    return 1<a ? b - inv(b%a,a)*b/a : 1;
}

ll gcde(ll a, ll b, ll& x, ll& y) {
    if (!a) {
        x = 0;
        y = 1;
        return b;
    }

    ll X, Y;
    ll g = gcde(b % a, a, X, Y);
    x = Y - (b / a) * X;
    y = X;

    return g;
}

struct crt {
    ll a, m;

    crt(ll a_, ll m_) : a(a_), m(m_) {}
    crt operator * (crt C) {
        ll x, y;
        ll g = gcde(m, C.m, x, y);
        if ((a - C.a) % g) a = -1;
        if (a == -1 or C.a == -1) return crt(-1, 0);
        ll lcm = m/g*C.m;
        ll ans = a + (x*(C.a-a)/g % (C.m/g))*m;
        return crt((ans % lcm + lcm) % lcm, lcm);
    }
};

pair<ll, ll> divide_show(ll n, int p, int k, int pak) {
    if (n == 0) return {0, 1};
    ll blocos = n/pak, falta = n%pak;
    ll periodo = divi[pak], resto = divi[falta];
    ll r = expo(periodo, blocos, pak)*resto%pak;

    auto rec = divide_show(n/p, p, k, pak);
    ll y = n/p + rec.first;
    r = r*rec.second % pak;

    return {y, r};
}

ll solve_pak(ll n, ll x, int p, int k, int pak) {
    divi[0] = 1;
    for (int i = 1; i <= pak; i++) {
        divi[i] = divi[i-1];
        if (i%p) divi[i] = divi[i] * i % pak;
    }

    auto dn = divide_show(n, p, k, pak), dx = divide_show(x,
       p, k, pak),
        dnx = divide_show(n-x, p, k, pak);
    ll y = dn.first-dx.first-dnx.first, r =
      (dn.second*inv(dx.second, pak)%pak)*inv(dnx.second,
         pak)%pak;
    return expo(p, y, pak) * r % pak;
}

ll solve(ll n, ll x, int mod) {
    vector<pair<int, int>> f;
    int mod2 = mod;
    for (int i = 2; i*i <= mod2; i++) if (mod2%i==0) {
        int c = 0;
        while (mod2%i==0) mod2 /= i, c++;
        f.push_back({i, c});
    }
    if (mod2 > 1) f.push_back({mod2, 1});
    crt ans(0, 1);
    for (int i = 0; i < f.size(); i++) {
```

```
        int pak = 1;
        for (int j = 0; j < f[i].second; j++) pak *=
            f[i].first;
        ans = ans * crt(solve_pak(n, x, f[i].first,
            f[i].second, pak), pak);
    }
    return ans.a;
}
```

## 3.25 Triangulos em Grafos

```
// get_triangles(i) encontra todos os triangulos ijk no grafo
// Custo nas arestas
// retorna {custo do triangulo, {j, k}}
//
// O(m sqrt(m) log(n)) se chamar para todos os vertices

vector<pair<int, int>> g[MAX]; // {para, peso}

#warning o 'g' deve estar ordenado
vector<pair<int, pair<int, int>>> get_triangles(int i) {
    vector<pair<int, pair<int, int>>> tri;
    for (pair<int, int> j : g[i]) {
        int a = i, b = j.first;
        if (g[a].size() > g[b].size()) swap(a, b);
        for (pair<int, int> c : g[a]) if (c.first != b and
            c.first > j.first) {
            auto it = lower_bound(g[b].begin(), g[b].end(),
                make_pair(c.first, -INF));
            if (it == g[b].end() or it->first != c.first)
                continue;
            tri.push_back({j.second+c.second+it->second, {a
                == i ? b : a, c.first}});
        }
    }
    return tri;
}
```

## 3.26 Closest pair of points

```
// O(nlogn)

pair<pt, pt> closest_pair_of_points(vector<pt> &v){
    #warning changes v order
    int n = v.size();
    sort(v.begin(), v.end());
    for (int i = 1; i < n; i++){
        if (v[i] == v[i-1]){
            return make_pair(v[i-1], v[i]);
        }
    }
    auto cmp_y = [&](const pt &l, const pt &r){
        if (l.y != r.y) return l.y < r.y;
        return l.x < r.x;
    };
    set<pt, decltype(cmp_y)> s(cmp_y);
    int l = 0, r = -1;
    ll d2_min = numeric_limits<ll>::max();
    pt pl, pr;
    const int magic = 5;
    while (r+1 < n){
        auto it = s.insert(v[++r]).first;
        int cnt = magic/2;
        while (cnt-- && it != s.begin())
            it--;
        cnt = 0;
        while (cnt++ < magic && it != s.end()){
            if (!((*it) == v[r])){
                ll d2 = dist2(*it, v[r]);
                if (d2_min > d2){
                    d2_min = d2;
                    pl = *it;
                    pr = v[r];
                }
            }
            it++;
        }
        while (l < r && sq(v[l].x-v[r].x) > d2_min)
            s.erase(v[l++]);
    }
```

```
    return make_pair(pl, pr);
}
```

## 3.27 Segment Intersection

```
// Verifica, dado n segmentos, se existe algum par de
    segmentos
// que se intersecta
//
// O(n log n)

bool operator < (const line& a, const line& b) { //
    comparador pro sweepline
    if (a.p == b.p) return ccw(a.p, a.q, b.q);
    if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or a.p.x+eps
        < b.p.x))
        return ccw(a.p, a.q, b.p);
    return ccw(a.p, b.q, b.p);
}

bool has_intersection(vector<line> v) {
    auto intersects = [&](pair<line, int> a, pair<line, int>
        b) {
        return interseg(a.first, b.first);
    };
    vector<pair<pt, pair<int, int>>> w;
    for (int i = 0; i < v.size(); i++) {
        if (v[i].q < v[i].p) swap(v[i].p, v[i].q);
        w.push_back({v[i].p, {0, i}});
        w.push_back({v[i].q, {1, i}});
    }
    sort(w.begin(), w.end());
    set<pair<line, int>> se;
    for (auto i : w) {
        line at = v[i.second.second];
        if (i.second.first == 0) {
            auto nxt = se.lower_bound({at, i.second.second});
            if (nxt != se.end() and intersects(*nxt, {at,
                i.second.second})) return 1;
            if (nxt != se.begin() and intersects(*(--nxt),
                {at, i.second.second})) return 1;
            se.insert({at, i.second.second});
        } else {
            auto nxt = se.upper_bound({at,
                i.second.second}), cur = nxt, prev = --cur;
            if (nxt != se.end() and prev != se.begin()
                and intersects(*nxt, *(--prev))) return 1;
            se.erase(cur);
        }
    }
    return 0;
}
```

## 3.28 Distinct Range Query - Wavelet

```
// build - O(n (log n + log(sigma)))
// query - O(log(sigma))

int v[MAX], n, nxt[MAX];

namespace wav {
    vector<int> esq[4*(1+MAXN-MINN)];

    void build(int b = 0, int e = n, int p = 1, int l =
        MINN, int r = MAXN) {
        if (l == r) return;
        int m = (l+r)/2; esq[p].push_back(0);
        for (int i = b; i < e; i++)
            esq[p].push_back(esq[p].back()+(nxt[i]<=m));
        int m2 = stable_partition(nxt+b, nxt+e, [=](int
            i){return i <= m;}) - nxt;
        build(b, m2, 2*p, l, m), build(m2, e, 2*p+1, m+1, r);
    }

    int count(int i, int j, int x, int y, int p = 1, int l =
        MINN, int r = MAXN) {
        if (y < l or r < x) return 0;
        if (x <= l and r <= y) return j-i;
        int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
        return count(ei, ej, x, y, 2*p, l, m)+count(i-ei,
```

```
            j-ej, x, y, 2*p+1, m+1, r);
    }
}

void build() {
    for (int i = 0; i < n; i++) nxt[i] = MAXN+1;
    vector<pair<int, int>> t;
    for (int i = 0; i < n; i++) t.push_back({v[i], i});
    sort(t.begin(), t.end());
    for (int i = 0; i < n-1; i++) if (t[i].first ==
       t[i+1].first)
        nxt[t[i].second] = t[i+1].second;

    wav::build();
}

int query(int a, int b) {
    return wav::count(a, b+1, b+1, MAXN+1);
}
```

## 3.29 RMQ com Divide and Conquer

```
// Responde todas as queries em
// O(n log(n))

typedef pair<pair<int, int>, int> iii;
#define f first
#define s second

int n, q, v[MAX];
iii qu[MAX];
int ans[MAX], pref[MAX], sulf[MAX];

void solve(int l=0, int r=n-1, int ql=0, int qr=q-1) {
    if (l > r or ql > qr) return;
    int m = (l+r)/2;
    int qL = partition(qu+ql, qu+qr+1, [=](iii x){return
       x.f.s < m;}) - qu;
    int qR = partition(qu+qL, qu+qr+1, [=](iii x){return
       x.f.f <=m;}) - qu;

    pref[m] = sulf[m] = v[m];
    for (int i = m-1; i >= l; i--) pref[i] = min(v[i],
       pref[i+1]);
    for (int i = m+1; i <= r; i++) sulf[i] = min(v[i],
       sulf[i-1]);

    for (int i = qL; i < qR; i++)
        ans[qu[i].s] = min(pref[qu[i].f.f], sulf[qu[i].f.s]);

    solve(l, m-1, ql, qL-1), solve(m+1, r, qR, qr);
}
```

## 3.30 Distinct Range Query com Update

```
// build - O(n log(n))
// query - O(log^2(n))
// update - O(log^2(n))

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <class T>
    using ord_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

int v[MAX], n, nxt[MAX], prv[MAX];
map<int, set<int> > ocor;

namespace bit {
    ord_set<pair<int, int>> bit[MAX];

    void build() {
        for (int i = 1; i <= n; i++)
           bit[i].insert({nxt[i-1], i-1});
        for (int i = 1; i <= n; i++) {
            int j = i + (i&-i);
            if (j <= n) for (auto x : bit[i])
                bit[j].insert(x);
        }
```

```
    }
    int pref(int p, int x) {
        int ret = 0;
        for (; p; p -= p&-p) ret += bit[p].order_of_key({x,
           -INF});
        return ret;
    }
    int query(int l, int r, int x) {
        return pref(r+1, x) - pref(l, x);
    }
    void update(int p, int x) {
        int p2 = p;
        for (p++; p <= n; p += p&-p) {
            bit[p].erase({nxt[p2], p2});
            bit[p].insert({x, p2});
        }
    }
}

void build() {
    for (int i = 0; i < n; i++) nxt[i] = INF;
    for (int i = 0; i < n; i++) prv[i] = -INF;
    vector<pair<int, int>> t;
    for (int i = 0; i < n; i++) t.push_back({v[i], i});
    sort(t.begin(), t.end());
    for (int i = 0; i < n; i++) {
        if (i and t[i].first == t[i-1].first)
            prv[t[i].second] = t[i-1].second;
        if (i+1 < n and t[i].first == t[i+1].first)
            nxt[t[i].second] = t[i+1].second;
    }

    for (int i = 0; i < n; i++) ocor[v[i]].insert(i);

    bit::build();
}

void muda(int p, int x) {
    bit::update(p, x);
    nxt[p] = x;
}

int query(int a, int b) {
    return b-a+1 - bit::query(a, b, b+1);
}

void update(int p, int x) { // mudar valor na pos. p para x
    if (prv[p] > -INF) muda(prv[p], nxt[p]);
    if (nxt[p] < INF) prv[nxt[p]] = prv[p];

    ocor[v[p]].erase(p);
    if (!ocor[x].size()) {
        muda(p, INF);
        prv[p] = -INF;
    } else if (*ocor[x].rbegin() < p) {
        int i = *ocor[x].rbegin();
        prv[p] = i;
        muda(p, INF);
        muda(i, p);
    } else {
        int i = *ocor[x].lower_bound(p);
        if (prv[i] > -INF) {
            muda(prv[i], p);
            prv[p] = prv[i];
        } else prv[p] = -INF;
        prv[i] = p;
        muda(p, i);
    }
    v[p] = x; ocor[x].insert(p);
}
```

# 4 Matematica

## 4.1 Division Trick

```
// Gera o conjunto n/i, pra todo i, em O(sqrt(n))
// copiei do github do tfg50

for(int l = 1, r; l <= n; l = r + 1) {
    r = n / (n / l);
    // n / i has the same value for l <= i <= r
```

```
}
```

## 4.2 Produto de dois long long mod m

```
// O(1)

typedef long long int ll;

ll mul(ll a, ll b, ll m) { // a*b % m
    ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
    return ret < 0 ? ret+m : ret;
}
```

## 4.3 Avaliacao de Interpolacao

```
// Dado 'n' pontos (i, y[i]), i \in [0, n),
// avalia o polinomio de grau n-1 que passa
// por esses pontos em 'x'
// Tudo modular, precisa do mint
//
// O(n log n)

mint evaluate_interpolation(int x, vector<mint> y) {
    int n = y.size();

    vector<mint> sulf(n+1, 1), fat(n, 1);
    for (int i = n-1; i >= 0; i--) sulf[i] = sulf[i+1] * (x
        - i);
    for (int i = 1; i < n; i++) fat[i] = fat[i-1] * i;

    mint pref = 1, ans = 0;
    for (int i = 0; i < n; pref *= (x - i++)) {
        mint num = pref * sulf[i+1];

        mint den = fat[i] * fat[n-1 - i];
        if ((n-1 - i)%2) den *= -1;

        ans += y[i] * num / den;
    }
    return ans;
}
```

## 4.4 Equacao Diofantina Linear

```
// Encontra o numero de solucoes de a*x + b*y = c,
// em que x \in [lx, rx] e y \in [ly, ry]
// Usar o comentario para recuperar as solucoes
// (note que o b ao final eh b/gcd(a, b))
// Cuidado com overflow! Tem que caber o quadrado dos valores
//
// O(log(min(a, b)))

template<typename T> tuple<ll, T, T> ext_gcd(ll a, ll b) {
    if (!a) return {b, 0, 1};
    auto [g, x, y] = ext_gcd<T>(b%a, a);
    return {g, y - b/a*x, x};
}

// numero de solucoes de a*[lx, rx] + b*[ly, ry] = c
template<typename T = ll> // usar __int128 se for ate 1e18
ll diophantine(ll a, ll b, ll c, ll lx, ll rx, ll ly, ll ry)
   {
    if (lx > rx or ly > ry) return 0;
    if (a == 0 and b == 0) return c ? 0 :
       (rx-lx+1)*(ry-ly+1);
    auto [g, x, y] = ext_gcd<T>(abs(a), abs(b));
    if (c % g != 0) return 0;
    if (a == 0) return (rx-lx+1)*(ly <= c/b and c/b <= ry);
    if (b == 0) return (ry-ly+1)*(lx <= c/a and c/a <= rx);
    x *= a/abs(a) * c/g, y *= b/abs(b) * c/g, a /= g, b /= g;

    auto shift = [&](T qt) { x += qt*b, y -= qt*a; };
    auto test = [&](T& k, ll mi, ll ma, ll coef, int t) {
        shift((mi - k)*t / coef);
        if (k < mi) shift(coef > 0 ? t : -t);
        if (k > ma) return pair<T, T>(rx+2, rx+1);
        T x1 = x;
        shift((ma - k)*t / coef);
```

```
        if (k > ma) shift(coef > 0 ? -t : t);
        return pair<T, T>(x1, x);
    };

    auto [l1, r1] = test(x, lx, rx, b, 1);
    auto [l2, r2] = test(y, ly, ry, a, -1);
    if (l2 > r2) swap(l2, r2);
    T l = max(l1, l2), r = min(r1, r2);
    if (l > r) return 0;
    ll k = (r-l) / abs(b) + 1;
    return k; // solucoes: x = l + [0, k)*|b|
}
```

## 4.5 Divisao de Polinomios

```
// Divide p1 por p2
// Retorna um par com o quociente e o resto
// Os coeficientes devem estar em ordem
// decrescente pelo grau. Ex:
// 3x^2 + 2x - 1 -> [3, 2, -1]
//
// O(nm), onde n e m sao os tamanhos dos
// polinomios

typedef vector<int> vi;

pair<vi, vi> div(vi p1, vi p2) {
    vi quoc, resto;
    int a = p1.size(), b = p2.size();
    for (int i = 0; i <= a - b; i++) {
        int k = p1[i] / p2[0];
        quoc.push_back(k);
        for (int j = i; j < i + b; j++)
            p1[j] -= k * p2[j - i];
    }

    for (int i = a - b + 1; i < a; i++)
        resto.push_back(p1[i]);

    return mp(quoc, resto);
}
```

## 4.6 Fast Walsh Hadamard Transform

```
// FWHT<'|'>(f) eh SOS DP
// FWHT<'&'>(f) eh soma de superset DP
// Se chamar com ^, usar tamanho potencia de 2!!
//
// O(n log(n))

template<char op, bool inv = false, class T> vector<T>
    FWHT(vector<T> f) {
    int n = f.size();
    for (int k = 0; (n-1)>>k; k++) for (int i = 0; i < n;
        i++) if (i>>k&1) {
        int j = i^(1<<k);
        if (op == '^') f[j] += f[i], f[i] = f[j] - 2*f[i];
        if (op == '|') f[i] += (inv ? -1 : 1) * f[j];
        if (op == '&') f[j] += (inv ? -1 : 1) * f[i];
    }
    if (op == '^' and inv) for (auto& i : f) i /= n;
    return f;
}
```

## 4.7 Logaritmo Discreto

```
// Resolve logaritmo discreto com o algoritmo baby step
    giant step
// Encontra o menor x tal que a^x = b (mod m)
// Se nao tem, retorna -1
//
// O(sqrt(m) * log(sqrt(m))

int dlog(int b, int a, int m) {
    if (a == 0) return b ? -1 : 1; // caso nao definido

    a %= m, b %= m;
    int k = 1, shift = 0;
```

```
    while (1) {
        int g = gcd(a, m);
        if (g == 1) break;

        if (b == k) return shift;
        if (b % g) return -1;
        b /= g, m /= g, shift++;
        k = (ll) k * a / g % m;
    }

    int sq = sqrt(m)+1, giant = 1;
    for (int i = 0; i < sq; i++) giant = (ll) giant * a % m;

    vector<pair<int, int>> baby;
    for (int i = 0, cur = b; i <= sq; i++) {
        baby.emplace_back(cur, i);
        cur = (ll) cur * a % m;
    }
    sort(baby.begin(), baby.end());

    for (int j = 1, cur = k; j <= sq; j++) {
        cur = (ll) cur * giant % m;
        auto it = lower_bound(baby.begin(), baby.end(),
            pair(cur, INF));
        if (it != baby.begin() and (--it)->first == cur)
             return sq * j - it->second + shift;
    }

    return -1;
}
```

## 4.8 2-SAT

```
// solve(n) Retorna se eh possivel atribuir valores
// pras 'n' variaveis
// ans[i] fala se a variavel 'i' eh verdadeira
// Pra chamar o negado da variavel 'i', usar ~i
//
// O(|V|+|E|)

namespace doisSAT {
#warning limpar o grafo
    vector<int> g[2*MAX];
    int vis[2*MAX], comp[2*MAX], id[2*MAX];
    stack<int> s;
    int ans[MAX];

    int dfs(int i, int& t) {
        int lo = id[i] = t++;
        s.push(i), vis[i] = 2;
        for (int j : g[i]) {
            if (!vis[j]) lo = min(lo, dfs(j, t));
            else if (vis[j] == 2) lo = min(lo, id[j]);
        }
        if (lo == id[i]) while (1) {
            int u = s.top(); s.pop();
            vis[u] = 1, comp[u] = i;
            if (ans[u>>1] == -1) ans[u>>1] = ~u&1;
            if (u == i) break;
        }
        return lo;
    }

    void tarjan(int n) {
        int t = 0;
        for (int i = 0; i < 2*n; i++) vis[i] = 0;
        for (int i = 0; i < 2*n; i++) if (!vis[i]) dfs(i, t);
    }

    void add_impl(int x, int y) { // x -> y = !x ou y
        x = x >= 0 ? 2*x : -2*x-1;
        y = y >= 0 ? 2*y : -2*y-1;
        g[x].push_back(y);
        g[y^1].push_back(x^1);
    }
    void add_cl(int x, int y) { // x ou y
        add_impl(~x, y);
    }
    void add_xor(int x, int y) { // x xor y
        add_cl(x, y), add_cl(~x, ~y);
    }
    void add_eq(int x, int y) { // x = y
```

```
        add_xor(~x, y);
    }
    void add_true(int x) { // x = T
        add_impl(~x, x);
    }

    bool solve(int n) {
        for (int i = 0; i < n; i++) ans[i] = -1;
        tarjan(n);
        for (int i = 0; i < n; i++)
            if (comp[2*i] == comp[2*i+1]) return 0;
        return 1;
    }
}
```

## 4.9 Variacoes do crivo de Eratosthenes

```
// "O" crivo
//
// Encontra maior divisor primo
// Um numero eh primo sse divi[x] == x
// fact fatora um numero <= lim
// A fatoracao sai ordenada
//
// crivo - O(n log(log(n)))
// fact - O(log(n))

int divi[MAX];

void crivo(int lim) {
    for (int i = 1; i <= lim; i++) divi[i] = 1;

    for (int i = 2; i <= lim; i++) if (divi[i] == 1)
        for (int j = i; j <= lim; j += i) divi[j] = i;
}

void fact(vector<int>& v, int n) {
    if (n != divi[n]) fact(v, n/divi[n]);
    v.push_back(divi[n]);
}
```

```
// Crivo linear
//
// Mesma coisa que o de cima, mas tambem
// calcula a lista de primos
//
// O(n)

int divi[MAX];
vector<int> primes;

void crivo(int lim) {
    divi[1] = 1;
    for (int i = 2; i <= lim; i++) {
        if (divi[i] == 0) divi[i] = i, primes.push_back(i);
        for (int j : primes) {
            if (j > divi[i] or i*j > lim) break;
            divi[i*j] = j;
        }
    }
}

// Crivo de divisores
//
// Encontra numero de divisores
// ou soma dos divisores
//
// O(n log(n))

int divi[MAX];

void crivo(int lim) {
    for (int i = 1; i <= lim; i++) divi[i] = 1;

    for (int i = 2; i <= lim; i++)
        for (int j = i; j <= lim; j += i) {
            // para numero de divisores
            divi[j]++;
            // para soma dos divisores
            divi[j] += i;
        }
}
```

```
// Crivo de totiente
//
// Encontra o valor da funcao
// totiente de Euler
//
// O(n log(log(n)))

int tot[MAX];

void crivo(int lim) {
    for (int i = 1; i <= lim; i++) tot[i] = i;

    for (int i = 2; i <= lim; i++) if (tot[i] == i)
        for (int j = i; j <= lim; j += i)
            tot[j] -= tot[j] / i;
}

// Crivo de funcao de mobius
//
// O(n log(log(n)))

char meb[MAX];

void crivo(int lim) {
    for (int i = 2; i <= lim; i++) meb[i] = 2;
    meb[1] = 1;
    for (int i = 2; i <= lim; i++) if (meb[i] == 2)
        for (int j = i; j <= lim; j += i) if (meb[j]) {
            if (meb[j] == 2) meb[j] = 1;
            meb[j] *= j/i%i ? -1 : 0;
        }
}

// Crivo linear de funcao multiplicativa
//
// Computa f(i) para todo 1 <= i <= n, sendo f
// uma funcao multiplicativa (se gcd(a,b) = 1,
// entao f(a*b) = f(a)*f(b))
// f_prime tem que computar f de um primo, e
// add_prime tem que computar f(p^(k+1)) dado f(p^k) e p
// Se quiser computar f(p^k) dado p e k, usar os comentarios
//
// O(n)

vector<int> primes;
int f[MAX], pot[MAX];
//int expo[MAX];

void sieve(int lim) {
    // Funcoes para soma dos divisores:
    auto f_prime = [](int p) { return p+1; };
    auto add_prime = [](int fpak, int p) { return fpak*p+1;
        };
    //auto f_pak = [](int p, int k) {};

    f[1] = 1;
    for (int i = 2; i <= lim; i++) {
        if (!pot[i]) {
            primes.push_back(i);
            f[i] = f_prime(i), pot[i] = i;
            //expo[i] = 1;
        }
        for (int p : primes) {
            if (i*p > lim) break;
            if (i%p == 0) {
                f[i*p] = f[i / pot[i]] *
                    add_prime(f[pot[i]], p);
                // se for descomentar, tirar a linha de cima
                    tambem
                //f[i*p] = f[i / pot[i]] * f_pak(p,
                    expo[i]+1);
                //expo[i*p] = expo[i]+1;
                pot[i*p] = pot[i] * p;
                break;
            } else {
                f[i*p] = f[i] * f[p];
                pot[i*p] = p;
                //expo[i*p] = 1;
            }
        }
    }
}
```

## 4.10 Algoritmo de Euclides estendido

```
// Acha x e y tal que ax + by = mdc(a, b) (nao eh unico)
// Assume a, b >= 0
//
// O(log(min(a, b)))

tuple<ll, ll, ll> ext_gcd(ll a, ll b) {
    if (!a) return {b, 0, 1};
    auto [g, x, y] = ext_gcd(b%a, a);
    return {g, y - b/a*x, x};
}
```

## 4.11 Karatsuba

```
// Os pragmas podem ajudar
// Para n ~ 2e5, roda em < 1 s
//
// O(n^1.58)

//#pragma GCC optimize("Ofast")
//#pragma GCC target ("avx,avx2")
template<typename T> void kar(T* a, T* b, int n, T* r, T*
    tmp) {
    if (n <= 64) {
        for (int i = 0; i < n; i++) for (int j = 0; j < n;
            j++)
            r[i+j] += a[i] * b[j];
        return;
    }
    int mid = n/2;
    T *atmp = tmp, *btmp = tmp+mid, *E = tmp+n;
    memset(E, 0, sizeof(E[0])*n);
    for (int i = 0; i < mid; i++) {
        atmp[i] = a[i] + a[i+mid];
        btmp[i] = b[i] + b[i+mid];
    }
    kar(atmp, btmp, mid, E, tmp+2*n);
    kar(a, b, mid, r, tmp+2*n);
    kar(a+mid, b+mid, mid, r+n, tmp+2*n);
    for (int i = 0; i < mid; i++) {
        T temp = r[i+mid];
        r[i+mid] += E[i] - r[i] - r[i+2*mid];
        r[i+2*mid] += E[i+mid] - temp - r[i+3*mid];
    }
}

template<typename T> vector<T> karatsuba(vector<T> a,
    vector<T> b) {
    int n = max(a.size(), b.size());
    while (n&(n-1)) n++;
    a.resize(n), b.resize(n);
    vector<T> ret(2*n), tmp(4*n);
    kar(&a[0], &b[0], n, &ret[0], &tmp[0]);
    return ret;
}
```

## 4.12 Exponenciacao rapida

```
// (x^y mod m) em O(log(y))

typedef long long int ll;

ll pow(ll x, ll y, ll m) { // iterativo
    ll ret = 1;
    while (y) {
        if (y & 1) ret = (ret * x) % m;
        y >>= 1;
        x = (x * x) % m;
    }
    return ret;
}

ll pow(ll x, ll y, ll m) { // recursivo
    if (!y) return 1;
    ll ans = pow(x*x%m, y/2, m);
    return y%2 ? x*ans%m : ans;
}
```

## 4.13 Inverso Modular

```
// Computa o inverso de a modulo b
// Se b eh primo, basta fazer
// a^(b-2)

ll inv(ll a, ll b) {
     return a > 1 ? b - inv(b%a, a)*b/a : 1;
}

// computa o inverso modular de 1..MAX-1 modulo um primo
ll inv[MAX]:
inv[1] = 1;
for (int i = 2; i < MAX; i++) inv[i] = MOD -
    MOD/i*inv[MOD%i]%MOD;
```

## 4.14 FFT

```
// chamar com vector<cplx> para FFT, ou vector<mint> para NTT
//
// O(n log(n))

template<typename T> void fft(vector<T> &a, bool f, int N,
    vector<int> &rev){
     for (int i = 0; i < N; i++)
         if (i < rev[i])
             swap(a[i], a[rev[i]]);
     int l, r, m;
     vector<T> roots(N);
     for (int n = 2; n <= N; n *= 2){
         T root = T::rt(f, n, N);
         roots[0] = 1;
         for (int i = 1; i < n/2; i++)
             roots[i] = roots[i-1]*root;
         for (int pos = 0; pos < N; pos += n){
             l = pos+0, r = pos+n/2, m = 0;
             while (m < n/2){
                 auto t = roots[m]*a[r];
                 a[r] = a[l] - t;
                 a[l] = a[l] + t;
                 l++; r++; m++;
             }
         }
     }
     if (f) {
         auto invN = T(1)/N;
         for(int i = 0; i < N; i++) a[i] = a[i]*invN;
     }
}

template<typename T> vector<T> convolution(vector<T> &a,
    vector<T> &b) {
     vector<T> l(a.begin(), a.end());
     vector<T> r(b.begin(), b.end());
     int ln = l.size(), rn = r.size();
     int N = ln+rn+1;
     int n = 1, log_n = 0;
     while (n <= N) { n <<= 1; log_n++; }
     vector<int> rev(n);
     for (int i = 0; i < n; ++i){
         rev[i] = 0;
         for (int j = 0; j < log_n; ++j)
             if (i & (1<<j))
                 rev[i] |= 1 << (log_n-1-j);
     }
     assert(N <= n);
     l.resize(n);
     r.resize(n);
     fft(l, false, n, rev);
     fft(r, false, n, rev);
     for (int i = 0; i < n; i++)
         l[i] *= r[i];
     fft(l, true, n, rev);
     return l;
}
```

## 4.15 Simplex

```
// Maximiza c^T x s.t. Ax <= b, x >= 0
//
```

```
// O(2^n), porem executa em O(n^3) no caso medio

const double eps = 1e-7;

namespace Simplex {
    vector<int> B;
    vector<vector<double>> T;
    int n, m;

    void pivot(int x, int y) {
        B[x] = y;
        for (int i = 0; i <= m; i++) if (i != y) T[x][i] /=
            T[x][y];
        T[x][y] = 1/T[x][y];
        for (int i = 0; i <= n; i++) if (i != x and
            abs(T[i][y]) > eps) {
            for (int j = 0; j <= m; j++) if (j != y) T[i][j]
                -= T[i][y] * T[x][j];
            T[i][y] = -T[i][y] * T[x][y];
        }
    }

    // Retorna o par (valor maximo, vetor solucao)
    pair<double, vector<double>> simplex(
            vector<vector<double>> A, vector<double> b,
                vector<double> c) {
        n = b.size(), m = c.size();
        B = vector<int> (n + 1, -1);
        T = vector(n + 1, vector<double>(m + 1));
        for (int i = 0; i < m; i++) T[0][i] = -c[i];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) T[i+1][j] = A[i][j];
            T[i+1][m] = b[i];
        }
        while (true) {
            int x = -1, y = -1;
            double mn = -eps;
            for (int i = 1; i <= n; i++) if(T[i][m] < mn) mn
                = T[i][m], x = i;
            if(x < 0) break;
            for (int i = 0; i < m; i++) if(T[x][i] < -eps) {
                y = i; break; }
            if (y < 0) return {-1e18, {}}; // sem solucao
                para Ax <= b
            pivot(x, y);
        }
        while (true) {
            int x = -1, y = -1;
            double mn = -eps;
            for (int i = 0; i < m; i++) if (T[0][i] < mn) mn
                = T[0][i], y = i;
            if (y < 0) break;
            mn = 1e200;
            for (int i = 1; i <= n; i++) if (T[i][y] > eps
                and T[i][m] / T[i][y] < mn)
                 mn = T[i][m] / T[i][y], x = i;

            if (x < 0) return {1e18, {}}; // c^T x eh
                ilimitado
            pivot(x, y);
        }
        vector<double> r(m);
        for (int i = 1; i <= n; i++) if (B[i] != -1) r[B[i]]
            = T[i][m];
        return {T[0][m], r};
    }
}
```

## 4.16 Binomial Distribution

```
// binom(n, k, p) retorna a probabilidade de k sucessos
// numa binomial(n, p)

mt19937 rng((int)
    chrono::steady_clock::now().time_since_epoch().count());

double logfact[MAX];
void calc(){
    logfact[0] = 0;
    for (int i = 1; i < MAX; i++)
        logfact[i] = logfact[i-1] + log(i);
```

```
}

double binom(int n, int k, double p){
    return exp(logfact[n] - logfact[k] - logfact[n-k] + k *
        log(p) + (n-k) * log(1 - p));
}

int main(){//if you want to sample from a bin(n, p)
    calc();
    int n; double p;
    cin >> n >> p;
    binomial_distribution<int> distribution(n, p);
    int IT = 1e5;
    vector<int> freq(n+1, 0);
    for (int i = 0; i < IT; i++){
        int v = distribution(rng);
        //P(v == k) = (n choose k)p^k (1-p)^(n-k) = binom(n,
            k, p)
        freq[v]++;
    }
    cout << fixed << setprecision(5);
    for (int i = 0; i <= n; i++)
        cout << double(freq[i])/IT << " ~= " << binom(n, i,
            p) << endl;
}
```

## 4.17 Miller-Rabin

```
// Testa se n eh primo, n <= 3 * 10^18
//
// O(log(n)), considerando multiplicacao
// e exponenciacao constantes

// multiplicacao modular

ll mul(ll a, ll b, ll m) {
    ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
    return ret < 0 ? ret+m : ret;
}

ll pow(ll x, ll y, ll m) {
    if (!y) return 1;
    ll ans = pow(mul(x, x, m), y/2, m);
    return y%2 ? mul(x, ans, m) : ans;
}

bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;
    ll r = __builtin_ctzll(n - 1), d = n >> r;

    // com esses primos, o teste funciona garantido para n
        <= 2^64
    // funciona para n <= 3*10^24 com os primos ate 41
    for (int a : {2, 325, 9375, 28178, 450775, 9780504,
        795265022}) {
        ll x = pow(a, d, n);
        if (x == 1 or x == n - 1 or a % n == 0) continue;

        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) break;
        }
        if (x != n - 1) return 0;
    }
    return 1;
}
```

## 4.18 Deteccao de ciclo - Tortoise and Hare

```
// Linear no tanto que tem que andar pra ciclar,
// O(1) de memoria
// Retorna um par com o tanto que tem que andar
// do f0 ate o inicio do ciclo e o tam do ciclo

pair<ll, ll> find_cycle() {
    ll tort = f(f0);
    ll hare = f(f(f0));
    ll t = 0;
```

```
    while (tort != hare) {
        tort = f(tort);
        hare = f(f(hare));
        t++;
    }
    ll st = 0;
    tort = f0;
    while (tort != hare) {
        tort = f(tort);
        hare = f(hare);
        st++;
    }

    ll len = 1;
    hare = f(tort);
    while (tort != hare) {
        hare = f(hare);
        len ++;
    }
    return {st, len};
}
```

## 4.19 Totiente

```
// O(sqrt(n))

int tot(int n){
    int ret = n;

    for (int i = 2; i*i <= n; i++) if (n % i == 0) {
        while (n % i == 0) n /= i;
        ret -= ret / i;
    }
    if (n > 1) ret -= ret / n;

    return ret;
}
```

## 4.20 Eliminacao Gaussiana

```
// Resolve sistema linear
// Retornar um par com o numero de solucoes
// e alguma solucao, caso exista
//
// O(n^2 * m)

template<typename T>
pair<int, vector<T>> gauss(vector<vector<T>> a, vector<T> b)
   {
    const double eps = 1e-6;
    int n = a.size(), m = a[0].size();
    for (int i = 0; i < n; i++) a[i].push_back(b[i]);

    vector<int> where(m, -1);
    for (int col = 0, row = 0; col < m and row < n; col++) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs(a[i][col]) > abs(a[sel][col])) sel = i;
        if (abs(a[sel][col]) < eps) continue;
        for (int i = col; i <= m; i++)
            swap(a[sel][i], a[row][i]);
        where[col] = row;

        for (int i = 0; i < n; i++) if (i != row) {
            T c = a[i][col] / a[row][col];
            for (int j = col; j <= m; j++)
                a[i][j] -= a[row][j] * c;
        }
        row++;
    }

    vector<T> ans(m, 0);
    for (int i = 0; i < m; i++) if (where[i] != -1)
        ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i = 0; i < n; i++) {
        T sum = 0;
        for (int j = 0; j < m; j++)
            sum += ans[j] * a[i][j];
        if (abs(sum - a[i][m]) > eps)
            return pair(0, vector<T>());
```

```
    }

    for (int i = 0; i < m; i++) if (where[i] == -1)
        return pair(INF, ans);
    return pair(1, ans);
}
```

## 4.21 Teorema Chines do Resto

```
// Combina equacoes modulares lineares: x = a (mod m)
// O m final eh o lcm dos m's, e a resposta eh unica mod o
   lcm
// Os m nao precisam ser coprimos
// Se nao tiver solucao, o 'a' vai ser -1

tuple<ll, ll, ll> ext_gcd(ll a, ll b) {
    if (!a) return {b, 0, 1};
    auto [g, x, y] = ext_gcd(b%a, a);
    return {g, y - b/a*x, x};
}

struct crt {
    ll a, m;

    crt() : a(0), m(1) {}
    crt(ll a_, ll m_) : a(a_), m(m_) {}
    crt operator * (crt C) {
        auto [g, x, y] = ext_gcd(m, C.m);
        if ((a - C.a) % g) a = -1;
        if (a == -1 or C.a == -1) return crt(-1, 0);
        ll lcm = m/g*C.m;
        ll ans = a + (x*(C.a-a)/g % (C.m/g))*m;
        return crt((ans % lcm + lcm) % lcm, lcm);
    }
};
```

## 4.22 Ordem de elemento do grupo

```
// Calcula a ordem de a em Z_n
// O grupo Zn eh ciclico sse n =
// 1, 2, 4, p^k ou 2 p^k, p primo impar
// Retorna -1 se nao achar
//
// O(sqrt(n) log(n))

int tot(int n); // totiente em O(sqrt(n))
int expo(int a, int b, int m); // (a^b)%m em O(log(b))

// acha todos os divisores ordenados em O(sqrt(n))
vector<int> div(int n) {
    vector<int> ret1, ret2;
    for (int i = 1; i*i <= n; i++) if (n % i == 0) {
        ret1.push_back(i);
        if (i*i != n) ret2.push_back(n/i);
    }

    for (int i = ret2.size()-1; i+1; i--)
       ret1.push_back(ret2[i]);
    return ret1;
}

int ordem(int a, int n) {
    vector<int> v = div(tot(n));
    for (int i : v) if (expo(a, i, n) == 1) return i;
    return -1;
}
```

## 4.23 Pollard's Rho Alg

```
// Usa o algoritmo de deteccao de ciclo de Floyd
// com uma otimizacao na qual o gcd eh acumulado
// A fatoracao nao sai necessariamente ordenada
// O algoritmo rho encontra um fator de n,
// e funciona muito bem quando n possui um fator pequeno
//
// Complexidades (considerando mul constante):
// rho - esperado O(n^(1/4)) no pior caso
// fact - esperado menos que O(n^(1/4) log(n)) no pior caso
```

```
ll mul(ll a, ll b, ll m) {
    ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
    return ret < 0 ? ret+m : ret;
}

ll pow(ll x, ll y, ll m) {
    if (!y) return 1;
    ll ans = pow(mul(x, x, m), y/2, m);
    return y%2 ? mul(x, ans, m) : ans;
}

bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;

    ll r = __builtin_ctzll(n - 1), d = n >> r;
    for (int a : {2, 325, 9375, 28178, 450775, 9780504,
       795265022}) {
        ll x = pow(a, d, n);
        if (x == 1 or x == n - 1 or a % n == 0) continue;

        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) break;
        }
        if (x != n - 1) return 0;
    }
    return 1;
}

ll rho(ll n) {
    if (n == 1 or prime(n)) return n;
    auto f = [n](ll x) {return mul(x, x, n) + 1;};

    ll x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
    while (t % 40 != 0 or gcd(prd, n) == 1) {
        if (x==y) x = ++x0, y = f(x);
        q = mul(prd, abs(x-y), n);
        if (q != 0) prd = q;
        x = f(x), y = f(f(y)), t++;
    }
    return gcd(prd, n);
}

vector<ll> fact(ll n) {
    if (n == 1) return {};
    if (prime(n)) return {n};
    ll d = rho(n);
    vector<ll> l = fact(d), r = fact(n / d);
    l.insert(l.end(), r.begin(), r.end());
    return l;
}
```

## 4.24 Eliminacao Gaussiana Z2

```
// D eh dimensao do espaco vetorial
// add(v) - adiciona o vetor v na base (retorna se ele jah
   pertencia ao span da base)
// coord(v) - retorna as coordenadas (c) de v na base atual
   (basis^T.c = v)
// recover(v) - retorna as coordenadas de v nos vetores na
   ordem em que foram inseridos
// coord(v).first e recover(v).first - se v pertence ao span
//
// Complexidade:
// add, coord, recover: O(D^2 / 64)

template<int D> struct Gauss_z2 {
    bitset<D> basis[D], keep[D];
    int rk, in;
    vector<int> id;

    Gauss_z2 () : rk(0), in(-1), id(D, -1) {};

    bool add(bitset<D> v) {
        in++;
        bitset<D> k;
        for (int i = D - 1; i >= 0; i--) if (v[i]) {
            if (basis[i][i]) v ^= basis[i], k ^= keep[i];
            else {
```

```
                k[i] = true, id[i] = in, keep[i] = k;
                basis[i] = v, rk++;
                return true;
            }
        }
        return false;
    }
    pair<bool, bitset<D>> coord(bitset<D> v) {
        bitset<D> c;
        for (int i = D - 1; i >= 0; i--) if (v[i]) {
            if (basis[i][i]) v ^= basis[i], c[i] = true;
            else return {false, bitset<D>()};
        }
        return {true, c};
    }
    pair<bool, vector<int>> recover(bitset<D> v) {
        auto [span, bc] = coord(v);
        if (not span) return {false, {}};
        bitset<D> aux;
        for (int i = D - 1; i >= 0; i--) if (bc[i]) aux ^=
            keep[i];
        vector<int> oc;
        for (int i = D - 1; i >= 0; i--) if (aux[i])
            oc.push_back(id[i]);
        return {true, oc};
    }
};
```

### 4.25 Algoritmo de Euclides

```
// O(log(min(a, b)))

int mdc(int a, int b) {
    return !b ? a : mdc(b, a % b);
}
```

# 5 DP

## 5.1 SOS DP

```
// O(n 2^n)

// soma de sub-conjunto
vector<ll> sos_dp(vector<ll> f) {
    int N = __builtin_ctz(f.size());
    assert((1<<N) == f.size());

    for (int i = 0; i < N; i++) for (int mask = 0; mask <
        (1<<N); mask++)
        if (mask>>i&1) f[mask] += f[mask^(1<<i)];
    return f;
}

// soma de super-conjunto
vector<ll> sos_dp(vector<ll> f) {
    int N = __builtin_ctz(f.size());
    assert((1<<N) == f.size());

    for (int i = 0; i < N; i++) for (int mask = 0; mask <
        (1<<N); mask++)
        if (~mask>>i&1) f[mask] += f[mask^(1<<i)];
    return f;
}
```

## 5.2 Longest Common Subsequence

```
// Computa a LCS entre dois arrays usando
// o algoritmo de Hirschberg para recuperar
//
// O(n*m), O(n+m) de memoria

int lcs_s[MAX], lcs_t[MAX];
int dp[2][MAX];

// dp[0][j] = max lcs(s[li...ri], t[lj, lj+j])
```

```
void dp_top(int li, int ri, int lj, int rj) {
    memset(dp[0], 0, (rj-lj+1)*sizeof(dp[0][0]));
    for (int i = li; i <= ri; i++) {
        for (int j = rj; j >= lj; j--)
            dp[0][j - lj] = max(dp[0][j - lj],
            (lcs_s[i] == lcs_t[j]) + (j > lj ? dp[0][j-1 -
               lj] : 0));
        for (int j = lj+1; j <= rj; j++)
            dp[0][j - lj] = max(dp[0][j - lj], dp[0][j-1
               -lj]);
    }
}

// dp[1][j] = max lcs(s[li...ri], t[lj+j, rj])
void dp_bottom(int li, int ri, int lj, int rj) {
    memset(dp[1], 0, (rj-lj+1)*sizeof(dp[1][0]));
    for (int i = ri; i >= li; i--) {
        for (int j = lj; j <= rj; j++)
            dp[1][j - lj] = max(dp[1][j - lj],
            (lcs_s[i] == lcs_t[j]) +  (j < rj ? dp[1][j+1 -
               lj] : 0));
        for (int j = rj-1; j >= lj; j--)
            dp[1][j - lj] = max(dp[1][j - lj], dp[1][j+1 -
               lj]);
    }
}

void solve(vector<int>& ans, int li, int ri, int lj, int rj)
   {
    if (li == ri){
        for (int j = lj; j <= rj; j++)
            if (lcs_s[li] == lcs_t[j]){
                ans.push_back(lcs_t[j]);
                break;
            }
        return;
    }
    if (lj == rj){
        for (int i = li; i <= ri; i++){
            if (lcs_s[i] == lcs_t[lj]){
                ans.push_back(lcs_s[i]);
                break;
            }
        }
        return;
    }
    int mi = (li+ri)/2;
    dp_top(li, mi, lj, rj), dp_bottom(mi+1, ri, lj, rj);

    int j_ = 0, mx = -1;

    for (int j = lj-1; j <= rj; j++) {
        int val = 0;
        if (j >= lj) val += dp[0][j - lj];
        if (j < rj) val += dp[1][j+1 - lj];

        if (val >= mx) mx = val, j_ = j;
    }
    if (mx == -1) return;
    solve(ans, li, mi, lj, j_), solve(ans, mi+1, ri, j_+1,
       rj);
}

vector<int> lcs(const vector<int>& s, const vector<int>& t) {
    for (int i = 0; i < s.size(); i++) lcs_s[i] = s[i];
    for (int i = 0; i < t.size(); i++) lcs_t[i] = t[i];
    vector<int> ans;
    solve(ans, 0, s.size()-1, 0, t.size()-1);
    return ans;
}
```

## 5.3 Divide and Conquer DP

```
// Particiona o array em k subarrays
// minimizando o somatorio das queries
//
// O(k n log n), assumindo quer query(l, r) eh O(1)

ll dp[MAX][2];

void solve(int k, int l, int r, int lk, int rk) {
    if (l > r) return;
```

```
    int m = (l+r)/2, p = -1;
    auto& ans = dp[m][k&1] = LINF;
    for (int i = max(m, lk); i <= rk; i++) {
        int at = dp[i+1][~k&1] + query(m, i);
        if (at < ans) ans = at, p = i;
    }
    solve(k, l, m-1, lk, p), solve(k, m+1, r, p, rk);
}

ll DC(int n, int k) {
    dp[n][0] = dp[n][1] = 0;
    for (int i = 0; i < n; i++) dp[i][0] = LINF;
    for (int i = 1; i <= k; i++) solve(i, 0, n-i, 0, n-i);
    return dp[0][k&1];
}
```

## 5.4 Mochila

```
// Resolve mochila, recuperando a resposta
//
// O(n * cap), O(n + cap) de memoria

int v[MAX], w[MAX]; // valor e peso
int dp[2][MAX_CAP];

// DP usando os itens [l, r], com capacidade = cap
void get_dp(int x, int l, int r, int cap) {
    memset(dp[x], 0, (cap+1)*sizeof(dp[x][0]));
    for (int i = l; i <= r; i++) for (int j = cap; j >= 0;
       j--)
        if (j - w[i] >= 0) dp[x][j] = max(dp[x][j], v[i] +
            dp[x][j - w[i]]);
}

void solve(vector<int>& ans, int l, int r, int cap) {
    if (l == r) {
        if (w[l] <= cap) ans.push_back(l);
        return;
    }
    int m = (l+r)/2;
    get_dp(0, l, m, cap), get_dp(1, m+1, r, cap);
    int left_cap = -1, opt = -INF;
    for (int j = 0; j <= cap; j++)
        if (int at = dp[0][j] + dp[1][cap - j]; at > opt)
            opt = at, left_cap = j;
    solve(ans, l, m, left_cap), solve(ans, m+1, r, cap -
       left_cap);
}

vector<int> knapsack(int n, int cap) {
    vector<int> ans;
    solve(ans, 0, n-1, cap);
    return ans;
}
```

## 5.5 Convex Hull Trick Dinamico

```
// para double, use LINF = 1/.0, div(a, b) = a/b
// update(x) atualiza o ponto de intersecao da reta x
// overlap(x) verifica se a reta x sobrepoe a proxima
// add(a, b) adiciona reta da forma ax + b
// query(x) computa maximo de ax + b para entre as retas
//
// O(log(n)) amortizado por insercao
// O(log(n)) por query

struct Line {
    mutable ll a, b, p;
    bool operator<(const Line& o) const { return a < o.a; }
    bool operator<(ll x) const { return p < x; }
};

struct dynamic_hull : multiset<Line, less<>> {
    ll div(ll a, ll b) {
        return a / b - ((a ^ b) < 0 and a % b);
    }

    void update(iterator x) {
        if (next(x) == end()) x->p = LINF;
        else if (x->a == next(x)->a) x->p = x->b >=
```

```
            next(x)->b ? LINF : -LINF;
        else x->p = div(next(x)->b - x->b, x->a -
            next(x)->a);
    }

    bool overlap(iterator x) {
        update(x);
        if (next(x) == end()) return 0;
        if (x->a == next(x)->a) return x->b >= next(x)->b;
        return x->p >= next(x)->p;
    }

    void add(ll a, ll b) {
        auto x = insert({a, b, 0});
        while (overlap(x)) erase(next(x)), update(x);
        if (x != begin() and !overlap(prev(x))) x = prev(x),
            update(x);
        while (x != begin() and overlap(prev(x)))
             x = prev(x), erase(next(x)), update(x);
    }

    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.a * x + l.b;
    }
};
```

## 5.6 Convex Hull Trick (Rafael)

```
// linear

struct CHT {
    int it;
    vector<ll> a, b;
    CHT():it(0){}
    ll eval(int i, ll x){
        return a[i]*x + b[i];
    }
    bool useless(){
        int sz = a.size();
        int r = sz-1, m = sz-2, l = sz-3;
        return  (b[l] - b[r])*(a[m] - a[l]) <
            (b[l] - b[m])*(a[r] - a[l]);
    }
    void add(ll A, ll B){
        a.push_back(A); b.push_back(B);
        while (!a.empty()){
            if ((a.size() < 3) || !useless()) break;
            a.erase(a.end() - 2);
            b.erase(b.end() - 2);
        }
    }
    ll get(ll x){
        it = min(it, int(a.size()) - 1);
        while (it+1 < a.size()){
            if (eval(it+1, x) > eval(it, x)) it++;
            else break;
        }
        return eval(it, x);
    }
};
```

# 6 Strings

## 6.1 Aho-corasick - Automato

```
// query retorna o numero de matches sem overlap
//
// insert - O(|s|)
// build - O(n * SIGMA), onde n = somatorio dos tamanhos das
   strings

namespace aho {
    const vector<pair<char, char>> vt = {
        {'a', 'z'},
        {'A', 'Z'},
        {'0', '9'}
    };//example of alphabet
```

```
    void fix(char &c){
        int acc = 0;
        for (auto p : vt){
            if (p.first <= c && c <= p.second){
                c = c - p.first + acc;
                return;
            }
            acc += p.second - p.first + 1;
        }
    }
    void unfix(char &c){
        int acc = 0;
        for (auto p : vt){
            int next_acc = acc + p.second - p.first;
            if (acc <= c && c <= next_acc){
                c = p.first + c - acc;
                return;
            }
            acc = next_acc + 1;
        }
    }
    void fix(string &s){ for (char &c : s) fix(c); }
    void unfix(string &s){ for (char &c : s) unfix(c); }


    const int SIGMA = 70;//fix(vt.back().second) + 1;
    const int MAXN = 1e5+10;

    int to[MAXN][SIGMA];
    int link[MAXN], end[MAXN];
    int idx;
    void init(){
#warning dont forget to init before inserting strings
        memset(to, 0, sizeof to);
        idx = 1;
    }
    void insert(string &s){
        fix(s);
        int v =  0;
        for (char c : s){
            int &w = to[v][c];
            if (!w) w = idx++;
            v = w;
        }
        end[v] = 1;
    }
    void build(){
#warning dont forget to build after inserting strings
        queue<int> q;
        q.push(0);
        while (!q.empty()){
            int cur = q.front(); q.pop();
            int l = link[cur];
            end[cur] |= end[l];
            for (int i = 0; i < SIGMA; i++){
                int &w = to[cur][i];
                if (w){
                    link[w] = ((cur != 0) ? to[l][i] : 0);
                    q.push(w);
                }
                else w = to[l][i];
            }
        }
    }
    int query(string &s){
        fix(s);
        int v = 0;
        int counter = 0;
        for (char c : s){
            v = to[v][c];
            if (end[v]) {
                counter++;
                v = 0;
            }
        }
        return counter;
    }
}
```

## 6.2 Min/max suffix/cyclic shift

```
// Computa o indice do menor/maior sufixo/cyclic shift
// da string, lexicograficamente
//
// O(n)

template<typename T> int max_suffix(T s, bool mi = false) {
    s.push_back(*min_element(s.begin(), s.end())-1);
    int ans = 0;
    for (int i = 1; i < s.size(); i++) {
        int j = 0;
        while (ans+j < i and s[i+j] == s[ans+j]) j++;
        if (s[i+j] > s[ans+j]) {
            if (!mi or i != s.size()-2) ans = i;
        } else if (j) i += j-1;
    }
    return ans;
}

template<typename T> int min_suffix(T s) {
    for (auto& i : s) i *= -1;
    s.push_back(*max_element(s.begin(), s.end())+1);
    return max_suffix(s, true);
}

template<typename T> int max_cyclic_shift(T s) {
    int n = s.size();
    for (int i = 0; i < n; i++) s.push_back(s[i]);
    return max_suffix(s);
}

template<typename T> int min_cyclic_shift(T s) {
    for (auto& i : s) i *= -1;
    return max_cyclic_shift(s);
}
```

## 6.3 String hashing - modulo 2^61 - 1

```
// Usa modulo 2^61 - 1 ~ 2e18
// Eh quase duas vezes mais lento
//
// Complexidades:
// build - O(|s|)
// operator() - O(1)

const ll MOD = (1ll<<61)-1;

ll mulmod(ll a, ll b) {
    const static ll LOWER = (1ll<<30)-1, GET31 = (1ll<<31)-1;
    ll l1 = a&LOWER, h1 = a>>30, l2 = b&LOWER, h2 = b>>30;
    ll m = l1*h2 + l2*h1, h = h1*h2;
    ll ans = l1*l2 + (h>>1) + ((h&1)<<60) + (m>>31) +
        ((m&GET31)<<30) + 1;
    ans = (ans&MOD) + (ans>>61);
    ans = (ans&MOD) + (ans>>61);
    return ans-1;
}

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

ll uniform(ll l, ll r) {
    uniform_int_distribution<ll> uid(l, r);
    return uid(rng);
}

struct str_hash {
    static ll P;
    int n;
    string s;
    vector<ll> h, power;
    str_hash(string s_) : n(s_.size()), s(s_), h(n),
        power(n) {
        power[0] = 1;
        for (int i = 1; i < n; i++) power[i] =
            mulmod(power[i-1], P);
        h[0] = s[0];
        for (int i = 1; i < n; i++) h[i] = (mulmod(h[i-1],
            P) + s[i]) % MOD;
    }
    ll operator()(int i, int j) { // retorna hash da
        substring s[i..j]
        if (!i) return h[j];
```

```
        ll ret = h[j] - mulmod(h[i-1], power[j-i+1]);
        return ret < 0 ? ret+MOD : ret;
    }
};
ll str_hash::P = uniform(27, MOD-1);
// primeiro parametro deve ser maior que o tamanho do
   alfabeto
```

## 6.4 Manacher

```
// manacher recebe um vetor de T e retorna o vetor com
   tamanho dos palindromos
// ret[2*i] = tamanho do maior palindromo centrado em i
// ret[2*i+1] = tamanho maior palindromo centrado em i e i+1
//
// Complexidades:
// manacher - O(n)
// palindrome - <O(n), O(1)>
// pal_end - O(n)

template<typename T> vector<int> manacher(const T& s) {
    int l = 0, r = -1, n = s.size();
    vector<int> d1(n), d2(n);
    for (int i = 0; i < n; i++) {
        int k = i > r ? 1 : min(d1[l+r-i], r-i);
        while (i+k < n && i-k >= 0 && s[i+k] == s[i-k]) k++;
        d1[i] = k--;
        if (i+k > r) l = i-k, r = i+k;
    }
    l = 0, r = -1;
    for (int i = 0; i < n; i++) {
        int k = i > r ? 0 : min(d2[l+r-i+1], r-i+1); k++;
        while (i+k <= n && i-k >= 0 && s[i+k-1] == s[i-k])
            k++;
        d2[i] = --k;
        if (i+k-1 > r) l = i-k, r = i+k-1;
    }
    vector<int> ret(2*n-1);
    for (int i = 0; i < n; i++) ret[2*i] = 2*d1[i]-1;
    for (int i = 0; i < n-1; i++) ret[2*i+1] = 2*d2[i+1];
    return ret;
}

// verifica se a string s[i..j] eh palindromo
template<typename T> struct palindrome {
    vector<int> man;

    palindrome(const T& s) : man(manacher(s)) {}
    bool query(int i, int j) {
        return man[i+j] >= j-i+1;
    }
};

// tamanho do maior palindromo que termina em cada posicao
template<typename T> vector<int> pal_end(const T& s) {
    vector<int> ret(s.size());
    palindrome<T> p(s);
    ret[0] = 1;
    for (int i = 1; i < s.size(); i++) {
        ret[i] = min(ret[i-1]+2, i+1);
        while (!p.query(i-ret[i]+1, i)) ret[i]--;
    }
    return ret;
}
```

## 6.5 Trie

```
// trie T() constroi uma trie para o alfabeto das letras
   minusculas
// trie T(tamanho do alfabeto, menor caracter) tambem pode
   ser usado
//
// T.insert(s) - O(|s|*sigma)
// T.erase(s) - O(|s|)
// T.find(s) retorna a posicao, 0 se nao achar - O(|s|)
// T.count_pref(s) numero de strings que possuem s como
   prefixo - O(|s|)
//
// Nao funciona para string vazia
```

```cpp
struct trie {
    vector<vector<int>> to;
    vector<int> end, pref;
    int sigma; char norm;
    trie(int sigma_=26, char norm_='a') : sigma(sigma_),
        norm(norm_) {
        to = {vector<int>(sigma)};
        end = {0}, pref = {0};
    }
    void insert(string s) {
        int x = 0;
        for(auto c : s) {
            int &nxt = to[x][c-norm];
            if(!nxt) {
                nxt = to.size();
                to.push_back(vector<int>(sigma));
                end.push_back(0), pref.push_back(0);
            }
            x = nxt, pref[x]++;
        }
        end[x]++;
    }
    void erase(string s) {
        int x = 0;
        for(char c : s) {
            int &nxt = to[x][c-norm];
            x = nxt, pref[x]--;
            if(!pref[x]) nxt = 0;
        }
        end[x]--;
    }
    int find(string s) {
        int x = 0;
        for(auto c : s) {
            x = to[x][c-norm];
            if(!x) return 0;
        }
        return x;
    }
    int count_pref(string s) {
        return pref[find(s)];
    }
};
```

## 6.6 String hashing

```cpp
// Para evitar colisao: testar mais de um
// mod; so comparar strings do mesmo tamanho
// ex : str_hash<1e9+7> h(s);
//      ll val = h(10, 20);
//
// Complexidades:
// build - O(|s|)
// operator() - O(1)

mt19937 rng((int)
    chrono::steady_clock::now().time_since_epoch().count());

int uniform(int l, int r) {
    uniform_int_distribution<int> uid(l, r);
    return uid(rng);
}

template<int MOD> struct str_hash {
    static int P;
    int n;
    string s;
    vector<ll> h, power;
    str_hash(string s_) : n(s_.size()), s(s_), h(n),
        power(n) {
        power[0] = 1;
        for (int i = 1; i < n; i++) power[i] = power[i-1]*P
            % MOD;
        h[0] = s[0];
        for (int i = 1; i < n; i++) h[i] = (h[i-1]*P + s[i])
            % MOD;
    }
    ll operator()(int i, int j) { // retorna hash da
        substring s[i..j]
        if (!i) return h[j];
        ll ret = h[j] - h[i-1]*power[j-i+1] % MOD;
        return ret < 0 ? ret+MOD : ret;
```

```
    }
};
template<int MOD> int str_hash<MOD>::P = uniform(27, MOD-1);
// primeiro parametro deve ser maior que o tamanho do
   alfabeto
```

## 6.7 eertree

```
// Constroi a eertree, caractere a caractere
// Inicializar com a quantidade de caracteres maxima
// size() retorna a quantidade de substrings pal. distintas
// depois de chamar propagate(), cada substring palindromica
// ocorre qt[i] vezes. O propagate() retorna o numero de
// substrings pal. com repeticao
//
// O(n) amortizado, considerando alfabeto O(1)

struct eertree {
    vector<vector<int>> t;
    int n, last, sz;
    vector<int> s, len, link, qt;

    eertree(int N) {
        t = vector(N+2, vector(26, int()));
        s = len = link = qt = vector<int>(N+2);
        s[0] = -1;
        link[0] = 1, len[0] = 0, link[1] = 1, len[1] = -1;
        sz = 2, last = 0, n = 1;
    }

    void add(char c) {
        s[n++] = c -= 'a';
        while (s[n-len[last]-2] != c) last = link[last];
        if (!t[last][c]) {
            int prev = link[last];
            while (s[n-len[prev]-2] != c) prev = link[prev];
            link[sz] = t[prev][c];
            len[sz] = len[last]+2;
            t[last][c] = sz++;
        }
        qt[last = t[last][c]]++;
    }
    int size() { return sz-2; }
    ll propagate() {
        ll ret = 0;
        for (int i = n; i > 1; i--) {
            qt[link[i]] += qt[i];
            ret += qt[i];
        }
        return ret;
    }
};
```

## 6.8 Suffix Array Dinamico

```
// Mantem o suffix array, lcp e rank de uma string,
// premitindo push_front e pop_front
// O operador [i] return um par com sa[i] e lcp[i]
// lcp[i] tem o lcp entre sa[i] e sa[i-1] (lcp[0] = 0)
//
// Complexidades:
// Construir sobre uma string de tamanho n: O(n log n)
// push_front e pop_front: O(log n) amortizado

struct dyn_sa {
    struct node {
        int sa, lcp;
        node *l, *r, *p;
        int sz, mi;
        node(int sa_, int lcp_, node* p_) : sa(sa_),
           lcp(lcp_),
            l(NULL), r(NULL), p(p_), sz(1), mi(lcp) {}
        void update() {
            sz = 1, mi = lcp;
            if (l) sz += l->sz, mi = min(mi, l->mi);
            if (r) sz += r->sz, mi = min(mi, r->mi);
        }
    };

    node* root;
```

```
vector<ll> tag; // tag of a suffix (reversed id)
string s; // reversed

dyn_sa() : root(NULL) {}
dyn_sa(string s_) : dyn_sa() {
    reverse(s_.begin(), s_.end());
    for (char c : s_) push_front(c);
}
~dyn_sa() {
    vector<node*> q = {root};
    while (q.size()) {
        node* x = q.back(); q.pop_back();
        if (!x) continue;
        q.push_back(x->l), q.push_back(x->r);
        delete x;
    }
}

int size(node* x) { return x ? x->sz : 0; }
int mirror(int i) { return s.size()-1 - i; }
bool cmp(int i, int j) {
    if (s[i] != s[j]) return s[i] < s[j];
    if (i == 0 or j == 0) return i < j;
    return tag[i-1] < tag[j-1];
}
void fix_path(node* x) { while (x) x->update(), x =
   x->p; }
void flatten(vector<node*>& v, node* x) {
    if (!x) return;
    flatten(v, x->l);
    v.push_back(x);
    flatten(v, x->r);
}
void build(vector<node*>& v, node*& x, node* p, int L,
   int R, ll l, ll r) {
    if (L > R) return void(x = NULL);
    int M = (L+R)/2;
    ll m = (l+r)/2;
    x = v[M];
    x->p = p;
    tag[x->sa] = m;
    build(v, x->l, x, L, M-1, l, m-1), build(v, x->r, x,
       M+1, R, m+1, r);
    x->update();
}
void fix(node*& x, node* p, ll l, ll r) {
    if (3*max(size(x->l), size(x->r)) <= 2*size(x))
       return x->update();
    vector<node*> v;
    flatten(v, x);
    build(v, x, p, 0, v.size()-1, l, r);
}
node* next(node* x) {
    if (x->r) {
        x = x->r;
        while (x->l) x = x->l;
        return x;
    }
    while (x->p and x->p->r == x) x = x->p;
    return x->p;
}
node* prev(node* x) {
    if (x->l) {
        x = x->l;
        while (x->r) x = x->r;
        return x;
    }
    while (x->p and x->p->l == x) x = x->p;
    return x->p;
}

int get_lcp(node* x, node* y) {
    if (!x or !y) return 0; // change defaut value here
    if (s[x->sa] != s[y->sa]) return 0;
    if (x->sa == 0 or y->sa == 0) return 1;
    return 1 + query(mirror(x->sa-1), mirror(y->sa-1));
}
void add_suf(node*& x, node* p, int id, ll l, ll r) {
    if (!x) {
        x = new node(id, 0, p);
        node *prv = prev(x), *nxt = next(x);
        int lcp_cur = get_lcp(prv, x), lcp_nxt =
           get_lcp(x, nxt);
        if (nxt) nxt->lcp = lcp_nxt, fix_path(nxt);
```

```
        x->lcp = lcp_cur;
        tag[id] = (l+r)/2;
        x->update();
        return;
    }
    if (cmp(id, x->sa)) add_suf(x->l, x, id, l,
       tag[x->sa]-1);
    else add_suf(x->r, x, id, tag[x->sa]+1, r);
    fix(x, p, l, r);
}
void push_front(char c) {
    s += c;
    tag.push_back(-1);
    add_suf(root, NULL, s.size() - 1, 0, 1e18);
}

void rem_suf(node*& x, int id) {
    if (x->sa != id) {
        if (tag[id] < tag[x->sa]) return rem_suf(x->l,
           id);
        return rem_suf(x->r, id);
    }
    node* nxt = next(x);
    if (nxt) nxt->lcp = min(nxt->lcp, x->lcp),
       fix_path(nxt);

    node *p = x->p, *tmp = x;
    if (!x->l or !x->r) {
        x = x->l ? x->l : x->r;
        if (x) x->p = p;
    } else {
        for (tmp = x->l, p = x; tmp->r; tmp = tmp->r) p
           = tmp;
        x->sa = tmp->sa, x->lcp = tmp->lcp;
        if (tmp->l) tmp->l->p = p;
        if (p->l == tmp) p->l = tmp->l;
        else p->r = tmp->l;
    }
    fix_path(p);
    delete tmp;
}
void pop_front() {
    if (!s.size()) return;
    s.pop_back();
    rem_suf(root, s.size());
    tag.pop_back();
}

int query(node* x, ll l, ll r, ll a, ll b) {
    if (!x or tag[x->sa] == -1 or r < a or b < l) return
       s.size();
    if (a <= l and r <= b) return x->mi;
    int ans = s.size();
    if (a <= tag[x->sa] and tag[x->sa] <= b) ans =
       min(ans, x->lcp);
    ans = min(ans, query(x->l, l, tag[x->sa]-1, a, b));
    ans = min(ans, query(x->r, tag[x->sa]+1, r, a, b));
    return ans;
}
int query(int i, int j) { // lcp(s[i..], s[j..])
    if (i == j) return s.size() - i;
    ll a = tag[mirror(i)], b = tag[mirror(j)];
    int ret = query(root, 0, 1e18, min(a, b)+1, max(a,
       b));
    return ret;
}
// optional: get rank[i], sa[i] and lcp[i]
int rank(int i) {
    i = mirror(i);
    node* x = root;
    int ret = 0;
    while (x) {
        if (tag[x->sa] < tag[i]) {
            ret += size(x->l)+1;
            x = x->r;
        } else x = x->l;
    }
    return ret;
}
pair<int, int> operator[](int i) {
    node* x = root;
    while (1) {
        if (i < size(x->l)) x = x->l;
        else {
```

```
                i -= size(x->l);
                if (!i) return {mirror(x->sa), x->lcp};
                i--, x = x->r;
            }
        }
    }
};
```

## 6.9 KMP

```
// mathcing(s, t) retorna os indices das ocorrencias
// de s em t
// autKMP constroi o automato do KMP
//
// Complexidades:
// pi - O(n)
// match - O(n + m)
// construir o automato - O(|sigma|*n)
// n = |padrao| e m = |texto|

template<typename T> vector<int> pi(T s) {
    vector<int> p(s.size());
    for (int i = 1, j = 0; i < s.size(); i++) {
        while (j and s[j] != s[i]) j = p[j-1];
        if (s[j] == s[i]) j++;
        p[i] = j;
    }
    return p;
}

template<typename T> vector<int> matching(T& s, T& t) {
    vector<int> p = pi(s), match;
    for (int i = 0, j = 0; i < t.size(); i++) {
        while (j and s[j] != t[i]) j = p[j-1];
        if (s[j] == t[i]) j++;
        if (j == s.size()) match.push_back(i-j+1), j =
            p[j-1];
    }
    return match;
}

struct KMPaut : vector<vector<int>> {
    KMPaut(){}
    KMPaut (string& s) : vector<vector<int>>(26,
        vector<int>(s.size()+1)) {
        vector<int> p = pi(s);
        auto& aut = *this;
        aut[s[0]-'a'][0] = 1;
        for (char c = 0; c < 26; c++)
            for (int i = 1; i <= s.size(); i++)
                aut[c][i] = s[i]-'a' == c ? i+1 :
                    aut[c][p[i-1]];
    }
};
```

## 6.10 Suffix Array - O(n)

```
// Rapidao
// Computa o suffix array em 'sa', o rank em 'rnk'
// e o lcp em 'lcp'
// query(i, j) retorna o LCP entre s[i..n-1] e s[j..n-1]
//
// Complexidades
// O(n) para construir
// query - O(1)

template<typename T> struct rmq {
    vector<T> v;
    int n; static const int b = 30;
    vector<int> mask, t;

    int op(int x, int y) { return v[x] <= v[y] ? x : y; }
    int msb(int x) { return
        __builtin_clz(1)-__builtin_clz(x); }
    int small(int r, int sz = b) { return
        r-msb(mask[r]&((1<<sz)-1)); }
    rmq() {}
    rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n),
        t(n) {
        for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
```

```
            at = (at<<1)&((1<<b)-1);
            while (at and op(i-msb(at&-at), i) == i) at ^=
                at&-at;
        }
        for (int i = 0; i < n/b; i++) t[i] = small(b*i+b-1);
        for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0;
            i+(1<<j) <= n/b; i++)
            t[n/b*j+i] = op(t[n/b*(j-1)+i],
                t[n/b*(j-1)+i+(1<<(j-1))]);
    }
    int index_query(int l, int r) {
        if (r-l+1 <= b) return small(r, r-l+1);
        int x = l/b+1, y = r/b-1;
        if (x > y) return op(small(l+b-1), small(r));
        int j = msb(y-x+1);
        int ans = op(small(l+b-1), op(t[n/b*j+x],
            t[n/b*j+y-(1<<j)+1]));
        return op(ans, small(r));
    }
    T query(int l, int r) { return v[index_query(l, r)]; }
};

struct suffix_array {
    string s;
    int n;
    vector<int> sa, cnt, rnk, lcp;
    rmq<int> RMQ;

    bool cmp(int a1, int b1, int a2, int b2, int a3=0, int
        b3=0) {
        return a1 != b1 ? a1 < b1 : (a2 != b2 ? a2 < b2 : a3
            < b3);
    }
    template<typename T> void radix(int* fr, int* to, T* r,
        int N, int k) {
        cnt = vector<int>(k+1, 0);
        for (int i = 0; i < N; i++) cnt[r[fr[i]]]++;
        for (int i = 1; i <= k; i++) cnt[i] += cnt[i-1];
        for (int i = N-1; i+1; i--) to[--cnt[r[fr[i]]]] =
            fr[i];
    }
    void rec(vector<int>& v, int k) {
        auto &tmp = rnk, &m0 = lcp;
        int N = v.size()-3, sz = (N+2)/3, sz2 = sz+N/3;
        vector<int> R(sz2+3);
        for (int i = 1, j = 0; j < sz2; i += i%3) R[j++] = i;

        radix(&R[0], &tmp[0], &v[0]+2, sz2, k);
        radix(&tmp[0], &R[0], &v[0]+1, sz2, k);
        radix(&R[0], &tmp[0], &v[0]+0, sz2, k);

        int dif = 0;
        int l0 = -1, l1 = -1, l2 = -1;
        for (int i = 0; i < sz2; i++) {
            if (v[tmp[i]] != l0 or v[tmp[i]+1] != l1 or
                v[tmp[i]+2] != l2)
                l0 = v[tmp[i]], l1 = v[tmp[i]+1], l2 =
                    v[tmp[i]+2], dif++;
            if (tmp[i]%3 == 1) R[tmp[i]/3] = dif;
            else R[tmp[i]/3+sz] = dif;
        }

        if (dif < sz2) {
            rec(R, dif);
            for (int i = 0; i < sz2; i++) R[sa[i]] = i+1;
        } else for (int i = 0; i < sz2; i++) sa[R[i]-1] = i;

        for (int i = 0, j = 0; j < sz2; i++) if (sa[i] < sz)
            tmp[j++] = 3*sa[i];
        radix(&tmp[0], &m0[0], &v[0], sz, k);
        for (int i = 0; i < sz2; i++)
            sa[i] = sa[i] < sz ? 3*sa[i]+1 : 3*(sa[i]-sz)+2;

        int at = sz2+sz-1, p = sz-1, p2 = sz2-1;
        while (p >= 0 and p2 >= 0) {
            if ((sa[p2]%3==1 and cmp(v[m0[p]], v[sa[p2]],
                R[m0[p]/3],
                R[sa[p2]/3+sz])) or (sa[p2]%3==2 and
                    cmp(v[m0[p]], v[sa[p2]],
                v[m0[p]+1], v[sa[p2]+1], R[m0[p]/3+sz],
                    R[sa[p2]/3+1])))
                sa[at--] = sa[p2--];
            else sa[at--] = m0[p--];
        }
```

```
    while (p >= 0) sa[at--] = m0[p--];
    if (N%3==1) for (int i = 0; i < N; i++) sa[i] =
        sa[i+1];
}

suffix_array(const string& s_) : s(s_), n(s.size()),
   sa(n+3),
        cnt(n+1), rnk(n), lcp(n-1) {
    vector<int> v(n+3);
    for (int i = 0; i < n; i++) v[i] = i;
    radix(&v[0], &rnk[0], &s[0], n, 256);
    int dif = 1;
    for (int i = 0; i < n; i++)
        v[rnk[i]] = dif += (i and s[rnk[i]] !=
            s[rnk[i-1]]);
    if (n >= 2) rec(v, dif);
    sa.resize(n);

    for (int i = 0; i < n; i++) rnk[sa[i]] = i;
    for (int i = 0, k = 0; i < n; i++, k -= !!k) {
        if (rnk[i] == n-1) {
            k = 0;
            continue;
        }
        int j = sa[rnk[i]+1];
        while (i+k < n and j+k < n and s[i+k] == s[j+k])
            k++;
        lcp[rnk[i]] = k;
    }
    RMQ = rmq<int>(lcp);
}

int query(int i, int j) {
    if (i == j) return n-i;
    i = rnk[i], j = rnk[j];
    return RMQ.query(min(i, j), max(i, j)-1);
}
pair<int, int> next(int L, int R, int i, char c) {
    int l = L, r = R+1;
    while (l < r) {
        int m = (l+r)/2;
        if (i+sa[m] >= n or s[i+sa[m]] < c) l = m+1;
```

```
            else r = m;
        }
        if (l == R+1 or s[i+sa[l]] > c) return {-1, -1};
        L = l;

        l = L, r = R+1;
        while (l < r) {
            int m = (l+r)/2;
            if (i+sa[m] >= n or s[i+sa[m]] <= c) l = m+1;
            else r = m;
        }
        R = l-1;
        return {L, R};
}
// quantas vezes 't' ocorre em 's' - O(|t| log n)
int count_substr(string& t) {
    int L = 0, R = n-1;
    for (int i = 0; i < t.size(); i++) {
        tie(L, R) = next(L, R, i, t[i]);
        if (L == -1) return 0;
    }
    return R-L+1;
}

// exemplo de f que resolve o problema
//
   https://codeforces.com/edu/course/2/lesson/2/5/practice/con
ll f(ll k) { return k*(k+1)/2; }

ll dfs(int L, int R, int p) { // dfs na suffix tree
   chamado em pre ordem
    int ext = L != R ? RMQ.query(L, R-1) : n - sa[L];

    // Tem 'ext - p' substrings diferentes que ocorrem
       'R-L+1' vezes
    // O LCP de todas elas eh 'ext'
    ll ans = (ext-p)*f(R-L+1);

    // L eh terminal, e folha sse L == R
    if (sa[L]+ext == n) L++;

    /* se for um SA de varias strings separadas como
```

```
            s#t$u&, usar no lugar do if de cima
             (separadores < 'a', diferentes e inclusive no
                final)
        while (L <= R && (sa[L]+ext == n || s[sa[L]+ext] <
            'a')) {
            L++;
        } */

        while (L <= R) {
            int idx = L != R ? RMQ.index_query(L, R-1) : -1;
            if (idx == -1 or lcp[idx] != ext) idx = R;

            ans += dfs(L, idx, ext);
            L = idx+1;
        }
        return ans;
    }

    // sum over substrings: computa, para toda substring t
       distinta de s,
    // \sum f(# ocorrencias de t em s) - O (n)
    ll sos() { return dfs(0, n-1, 0); }
};
```

## 6.11 Aho-corasick

```
// query retorna o somatorio do numero de matches de
// todas as stringuinhas na stringona
//
// insert - O(|s| * log(SIGMA))
// build - O(n * SIGMA), onde n = somatorio dos tamanhos das
   strings
// query - O(|s|)

namespace aho {
    map<char, int> to[MAX];
    int link[MAX], idx, term[MAX], exit[MAX], sobe[MAX];

    void insert(string& s) {
        int at = 0;
        for (char c : s) {
            auto it = to[at].find(c);
            if (it == to[at].end()) at = to[at][c] = ++idx;
            else at = it->second;
        }
        term[at]++, sobe[at]++;
    }
#warning nao esquece de chamar build() depois de inserir
    void build() {
        queue<int> q;
        q.push(0);
        link[0] = exit[0] = -1;
        while (q.size()) {
            int i = q.front(); q.pop();
            for (auto [c, j] : to[i]) {
                int l = link[i];
                while (l != -1 and !to[l].count(c)) l =
                    link[l];
                link[j] = l == -1 ? 0 : to[l][c];
                exit[j] = term[link[j]] ? link[j] :
                    exit[link[j]];
                if (exit[j]+1) sobe[j] += sobe[exit[j]];
                q.push(j);
            }
        }
    }
    int query(string& s) {
        int at = 0, ans = 0;
        for (char c : s){
            while (at != -1 and !to[at].count(c)) at =
                link[at];
            at = at == -1 ? 0 : to[at][c];
            ans += sobe[at];
        }
        return ans;
    }
}
```

## 6.12 Suffix Array - O(n log n)

```
// kasai recebe o suffix array e calcula lcp[i],
// o lcp entre s[sa[i],...,n-1] e s[sa[i+1],..,n-1]
//
// Complexidades:
// suffix_array - O(n log(n))
// kasai - O(n)

vector<int> suffix_array(string s) {
    s += "$";
    int n = s.size(), N = max(n, 260);
    vector<int> sa(n), ra(n);
    for(int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];

    for(int k = 0; k < n; k ? k *= 2 : k++) {
        vector<int> nsa(sa), nra(n), cnt(N);

        for(int i = 0; i < n; i++) nsa[i] = (nsa[i]-k+n)%n,
            cnt[ra[i]]++;
        for(int i = 1; i < N; i++) cnt[i] += cnt[i-1];
        for(int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]]]] =
            nsa[i];

        for(int i = 1, r = 0; i < n; i++) nra[sa[i]] = r +=
            ra[sa[i]] !=
             ra[sa[i-1]] or ra[(sa[i]+k)%n] !=
                ra[(sa[i-1]+k)%n];
        ra = nra;
        if (ra[sa[n-1]] == n-1) break;
    }
    return vector<int>(sa.begin()+1, sa.end());
}

vector<int> kasai(string s, vector<int> sa) {
    int n = s.size(), k = 0;
    vector<int> ra(n), lcp(n);
    for (int i = 0; i < n; i++) ra[sa[i]] = i;

    for (int i = 0; i < n; i++, k -= !!k) {
        if (ra[i] == n-1) { k = 0; continue; }
        int j = sa[ra[i]+1];
        while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
        lcp[ra[i]] = k;
    }
    return lcp;
}
```

## 6.13 Algoritmo Z

```
// Complexidades:
// z - O(|s|)
// match - O(|s| + |p|)

vector<int> get_z(string s) {
    int n = s.size();
    vector<int> z(n, 0);

    // intervalo da ultima substring valida
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        // estimativa pra z[i]
        if (i <= r) z[i] = min(r - i + 1, z[i - l]);
        // calcula valor correto
        while (i + z[i] < n and s[z[i]] == s[i + z[i]])
            z[i]++;
        // atualiza [l, r]
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }

    return z;
}

// quantas vezes p aparece em s
int match(string s, string p) {
    int n = s.size(), m = p.size();
    vector<int> z = get_z(p + s);

    int ret = 0;
    for (int i = m; i < n + m; i++)
        if (z[i] >= m) ret++;

    return ret;
}
```

## 6.14 Automato de Sufixo

```
// Automato que aceita os sufixos de uma string
// Todas as funcoes sao lineares

namespace sam {
	int cur, sz, len[2*MAX], link[2*MAX], acc[2*MAX];
	int nxt[2*MAX][26];

	void add(int c) {
		int at = cur;
		len[sz] = len[cur]+1, cur = sz++;
		while (at != -1 and !nxt[at][c]) nxt[at][c] = cur,
			at = link[at];
		if (at == -1) { link[cur] = 0; return; }
		int q = nxt[at][c];
		if (len[q] == len[at]+1) { link[cur] = q; return; }
		int qq = sz++;
		len[qq] = len[at]+1, link[qq] = link[q];
		for (int i = 0; i < 26; i++) nxt[qq][i] = nxt[q][i];
		while (at != -1 and nxt[at][c] == q) nxt[at][c] =
			qq, at = link[at];
		link[cur] = link[q] = qq;
	}
	void build(string& s) {
		cur = 0, sz = 0, len[0] = 0, link[0] = -1, sz++;
		for (auto i : s) add(i-'a');
		int at = cur;
		while (at) acc[at] = 1, at = link[at];
	}

	// coisas que da pra fazer:
	ll distinct_substrings() {
		ll ans = 0;
		for (int i = 1; i < sz; i++) ans += len[i] -
			len[link[i]];
		return ans;
	}
	string longest_common_substring(string& S, string& T) {
		build(S);
		int at = 0, l = 0, ans = 0, pos = -1;
		for (int i = 0; i < T.size(); i++) {
			while (at and !nxt[at][T[i]-'a']) at = link[at],
				l = len[at];
			if (nxt[at][T[i]-'a']) at = nxt[at][T[i]-'a'],
				l++;
			else at = 0, l = 0;
			if (l > ans) ans = l, pos = i;
		}
		return T.substr(pos-ans+1, ans);
	}
	ll dp[2*MAX];
	ll paths(int i) {
		auto& x = dp[i];
		if (x) return x;
		x = 1;
		for (int j = 0; j < 26; j++) if (nxt[i][j]) x +=
			paths(nxt[i][j]);
		return x;
	}
	void kth_substring(int k, int at=0) { // k=1 : menor
		substring lexicog.
		for (int i = 0; i < 26; i++) if (k and nxt[at][i]) {
			if (paths(nxt[at][i]) >= k) {
				cout << char('a'+i);
				kth_substring(k-1, nxt[at][i]);
				return;
			}
			k -= paths(nxt[at][i]);
		}
	}
};
```

# 7 Primitivas

## 7.1 Aritmetica Modular

```
// O mod tem q ser primo

template<int p> struct mod_int {
	ll pow(ll b, ll e) {
```

```cpp
        if (e == 0) return 1;
        ll r = pow(b*b%p, e/2);
        if (e%2 == 1) r = (r*b)%p;
        return r;
    }
    ll inv(ll b) { return pow(b, p-2); }

    using m = mod_int;
    int v;
    mod_int() : v(0) {}
    mod_int(ll v_) {
        if (v_ >= p || v_ <= -p) v_ %= p;
        if (v_ < 0) v_ += p;
        v = v_;
    }
    m& operator+=(const m &a) {
        v += a.v;
        if (v >= p) v -= p;
        return *this;
    }
    m& operator-=(const m &a) {
        v -= a.v;
        if (v < 0) v += p;
        return *this;
    }
    m& operator*=(const m &a) {
        v = (v*ll(a.v))%p;
        return *this;
    }
    m& operator/=(const m &a) {
        v = (v*inv(a.v))%p;
        return *this;
    }
    m operator-(){ return m(-v); }
    m& operator^=(ll e) {
        if (e < 0){
            v = inv(v);
            e = -e;
        }
        v = pow(v, e%(p-1));
        return *this;
    }
    bool operator==(const m &a) { return v == a.v; }
    bool operator!=(const m &a) { return v != a.v; }

    friend istream &operator>>(istream &in, m& a) {
        ll val; in >> val;
        a = m(val);
        return in;
    }
    friend ostream &operator<<(ostream &out, m a) {
        return out << a.v;
    }
    friend m operator+(m a, m b) { return a+=b; }
    friend m operator-(m a, m b) { return a-=b; }
    friend m operator*(m a, m b) { return a*=b; }
    friend m operator/(m a, m b) { return a/=b; }
    friend m operator^(m a, ll e) { return a^=e; }

    static m rt(bool f, int n, int N){
        if (p == 998244353){
            m r(102292); // an element of order N
            int ord = (1 << 23);
            while (ord != N){
                r = r*r;
                ord /= 2;
            }
            if (f) r = r^(-1);
            return r^(N/n);
        }
        return -1;
    }
};

typedef mod_int<(int)1e9+7> mint;
```

## 7.2 Primitivas Geometricas Inteiras

```cpp
#define sq(x) ((x)*(ll)(x))

struct pt { // ponto
    int x, y;
```

```
    pt(int x_ = 0, int y_ = 0) : x(x_), y(y_) {}
    bool operator < (const pt p) const {
        if (x != p.x) return x < p.x;
        return y < p.y;
    }
    bool operator == (const pt p) const {
        return x == p.x and y == p.y;
    }
    pt operator + (const pt p) const { return pt(x+p.x,
        y+p.y); }
    pt operator - (const pt p) const { return pt(x-p.x,
        y-p.y); }
    pt operator * (const int c) const { return pt(x*c, y*c);
        }
    ll operator * (const pt p) const { return x*(ll)p.x +
        y*(ll)p.y; }
    ll operator ^ (const pt p) const { return x*(ll)p.y -
        y*(ll)p.x; }
    friend istream& operator >> (istream& in, pt& p) {
        return in >> p.x >> p.y;
    }
};

struct line { // reta
    pt p, q;
    line() {}
    line(pt p_, pt q_) : p(p_), q(q_) {}
    friend istream& operator >> (istream& in, line& r) {
        return in >> r.p >> r.q;
    }
};

// PONTO & VETOR

ll dist2(pt p, pt q) { // quadrado da distancia
    return sq(p.x - q.x) + sq(p.y - q.y);
}

ll sarea2(pt p, pt q, pt r) { // 2 * area com sinal
    return (q-p)^(r-q);
}

bool col(pt p, pt q, pt r) { // se p, q e r sao colin.
    return sarea2(p, q, r) == 0;
}

int paral(pt u, pt v) { // se u e v sao paralelos
    if (u^v) return 0;
    if ((u.x > 0) == (v.x > 0) and (u.y > 0) == (v.y > 0))
        return 1;
    return -1;
}

bool ccw(pt p, pt q, pt r) { // se p, q, r sao ccw
    return sarea2(p, q, r) > 0;
}

int quad(pt p) { // quadrante de um ponto
    return (p.x<0)^3*(p.y<0);
}

bool compare_angle(pt p, pt q) { // retorna se ang(p) <
    ang(q)
    if (quad(p) != quad(q)) return quad(p) < quad(q);
    return ccw(q, pt(0, 0), p);
}

pt rotate90(pt p) { // rotaciona 90 graus
    return pt(-p.y, p.x);
}

// RETA

bool paraline(line r, line s) { // se r e s sao paralelas
    return paral(r.p - r.q, s.p - s.q);
}

bool isinseg(pt p, line r) { // se p pertence ao seg de r
    if (p == r.p or p == r.q) return 1;
    return paral(p - r.p, p - r.q) == -1;
}

bool interseg(line r, line s) { // se o seg de r intersecta
    o seg de s
```

```
    if (isinseg(r.p, s) or isinseg(r.q, s)
        or isinseg(s.p, r) or isinseg(s.q, r)) return 1;

    return ccw(r.p, r.q, s.p) != ccw(r.p, r.q, s.q) and
            ccw(s.p, s.q, r.p) != ccw(s.p, s.q, r.q);
}

int segpoints(line r) { // numero de pontos inteiros no
   segmento
    return 1 + __gcd(abs(r.p.x - r.q.x), abs(r.p.y - r.q.y));
}

double get_t(pt v, line r) { // retorna t tal que t*v
   pertence a reta r
    return (r.p^r.q) / (double) ((r.p-r.q)^v);
}

// POLIGONO

// quadrado da distancia entre os retangulos a e b (lados
   paralelos aos eixos)
// assume que ta representado (inferior esquerdo, superior
   direito)
ll dist2_rect(pair<pt, pt> a, pair<pt, pt> b) {
    int hor = 0, vert = 0;
    if (a.second.x < b.first.x) hor = b.first.x - a.second.x;
    else if (b.second.x < a.first.x) hor = a.first.x -
       b.second.x;
    if (a.second.y < b.first.y) vert = b.first.y -
       a.second.y;
    else if (b.second.y < a.first.y) vert = a.first.y -
       b.second.y;
    return sq(hor) + sq(vert);
}

ll polarea2(vector<pt> v) { // 2 * area do poligono
    ll ret = 0;
    for (int i = 0; i < v.size(); i++)
        ret += sarea2(pt(0, 0), v[i], v[(i + 1) % v.size()]);
    return abs(ret);
}

// se o ponto ta dentro do poligono: retorna 0 se ta fora,
// 1 se ta no interior e 2 se ta na borda
int inpol(vector<pt>& v, pt p) { // O(n)
    int qt = 0;
    for (int i = 0; i < v.size(); i++) {
        if (p == v[i]) return 2;
        int j = (i+1)%v.size();
        if (p.y == v[i].y and p.y == v[j].y) {
            if ((v[i]-p)*(v[j]-p) <= 0) return 2;
            continue;
        }
        bool baixo = v[i].y < p.y;
        if (baixo == (v[j].y < p.y)) continue;
        auto t = (p-v[i])^(v[j]-v[i]);
        if (!t) return 2;
        if (baixo == (t > 0)) qt += baixo ? 1 : -1;
    }
    return qt != 0;
}

vector<pt> convex_hull(vector<pt> v) { // convex hull - O(n
   log(n))
    if (v.size() <= 1) return v;
    vector<pt> l, u;
    sort(v.begin(), v.end());
    for (int i = 0; i < v.size(); i++) {
        while (l.size() > 1 and !ccw(l[l.size()-2],
           l.back(), v[i]))
            l.pop_back();
        l.push_back(v[i]);
    }
    for (int i = v.size() - 1; i >= 0; i--) {
        while (u.size() > 1 and !ccw(u[u.size()-2],
           u.back(), v[i]))
            u.pop_back();
        u.push_back(v[i]);
    }
    l.pop_back(); u.pop_back();
    for (pt i : u) l.push_back(i);
    return l;
}
```

```
ll interior_points(vector<pt> v) { // pontos inteiros dentro
    de um poligono simples
    ll b = 0;
    for (int i = 0; i < v.size(); i++)
        b += segpoints(line(v[i], v[(i+1)%v.size()])) - 1;
    return (polarea2(v) - b) / 2 + 1;
}

struct convex_pol {
    vector<pt> pol;

    convex_pol(vector<pt> v) : pol(convex_hull(v)) {}
    bool is_inside(pt p) { // se o ponto ta dentro do hull -
        O(log(n))
        if (pol.size() == 1) return p == pol[0];
        int l = 1, r = pol.size();
        while (l < r) {
            int m = (l+r)/2;
            if (ccw(p, pol[0], pol[m])) l = m+1;
            else r = m;
        }
        if (l == 1) return isinseg(p, line(pol[0], pol[1]));
        if (l == pol.size()) return false;
        return !ccw(p, pol[l], pol[l-1]);
    }
};

bool operator <(const line& a, const line& b) { //
    comparador pra reta
    // assume que as retas tem p < q
    pt v1 = a.q - a.p, v2 = b.q - b.p;
    bool b1 = compare_angle(v1, v2), b2 = compare_angle(v2,
        v1);
    if (b1 or b2) return b1;
    return ccw(a.p, a.q, b.p); // mesmo angulo
}
bool operator ==(const line& a, const line& b) {
    return !(a < b) and !(b < a);
}

// comparador pro set pra fazer sweep line com segmentos
struct cmp_sweepline {
    bool operator () (const line& a, const line& b) const {
        // assume que os segmentos tem p < q
        if (a.p == b.p) return ccw(a.p, a.q, b.q);
        if (a.p.x != a.q.x and (b.p.x == b.q.x or a.p.x <
            b.p.x))
            return ccw(a.p, a.q, b.p);
        return ccw(a.p, b.q, b.p);
    }
};

// comparador pro set pra fazer sweep angle com segmentos
pt dir;
struct cmp_sweepangle {
    bool operator () (const line& a, const line& b) const {
        return get_t(dir, a) < get_t(dir, b);
    }
};
```

## 7.3 Primitivas de Polinomios

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

using namespace std;
namespace algebra {
    const int inf = 1e9;
    const int magic = 500; // threshold for sizes to run the
        naive algo

    namespace fft {
        const int maxn = 1 << 18;

        typedef double ftype;
        typedef complex<ftype> point;

        point w[maxn];
        const ftype pi = acos(-1);
        bool initiated = 0;
        void init() {
            if(!initiated) {
                for(int i = 1; i < maxn; i *= 2) {
```

```
                for(int j = 0; j < i; j++) {
                    w[i + j] = polar(ftype(1), pi * j /
                        i);
                }
            }
            initiated = 1;
        }
    }
    template<typename T>
        void fft(T *in, point *out, int n, int k = 1) {
            if(n == 1) {
                *out = *in;
            } else {
                n /= 2;
                fft(in, out, n, 2 * k);
                fft(in + k, out + n, n, 2 * k);
                for(int i = 0; i < n; i++) {
                    auto t = out[i + n] * w[i + n];
                    out[i + n] = out[i] - t;
                    out[i] += t;
                }
            }
        }

    template<typename T>
        void mul_slow(vector<T> &a, const vector<T> &b) {
            vector<T> res(a.size() + b.size() - 1);
            for(size_t i = 0; i < a.size(); i++) {
                for(size_t j = 0; j < b.size(); j++) {
                    res[i + j] += a[i] * b[j];
                }
            }
            a = res;
        }


    template<typename T>
        void mul(vector<T> &a, const vector<T> &b) {
            if(min(a.size(), b.size()) < magic) {
                mul_slow(a, b);
                return;
            }
            init();
            static const int shift = 15, mask = (1 <<
                shift) - 1;
            size_t n = a.size() + b.size() - 1;
            while(__builtin_popcount(n) != 1) {
                n++;
            }
            a.resize(n);
            static point A[maxn], B[maxn];
            static point C[maxn], D[maxn];
            for(size_t i = 0; i < n; i++) {
                A[i] = point(a[i] & mask, a[i] >> shift);
                if(i < b.size()) {
                    B[i] = point(b[i] & mask, b[i] >>
                        shift);
                } else {
                    B[i] = 0;
                }
            }
            fft(A, C, n); fft(B, D, n);
            for(size_t i = 0; i < n; i++) {
                point c0 = C[i] + conj(C[(n - i) % n]);
                point c1 = C[i] - conj(C[(n - i) % n]);
                point d0 = D[i] + conj(D[(n - i) % n]);
                point d1 = D[i] - conj(D[(n - i) % n]);
                A[i] = c0 * d0 - point(0, 1) * c1 * d1;
                B[i] = c0 * d1 + d0 * c1;
            }
            fft(A, C, n); fft(B, D, n);
            reverse(C + 1, C + n);
            reverse(D + 1, D + n);
            int t = 4 * n;
            for(size_t i = 0; i < n; i++) {
                int64_t A0 = llround(real(C[i]) / t);
                T A1 = llround(imag(D[i]) / t);
                T A2 = llround(imag(C[i]) / t);
                a[i] = A0 + (A1 << shift) + (A2 << 2 *
                    shift);
            }
            return;
        }
}
```

```
template<typename T>
    T bpow(T x, size_t n) {
        return n ? n % 2 ? x * bpow(x, n - 1) : bpow(x *
           x, n / 2) : T(1);
    }
template<typename T>
    T bpow(T x, size_t n, T m) {
        return n ? n % 2 ? x * bpow(x, n - 1, m) % m :
           bpow(x * x % m, n / 2, m) : T(1);
    }
template<typename T>
    T gcd(const T &a, const T &b) {
        return b == T(0) ? a : gcd(b, a % b);
    }
template<typename T>
    T nCr(T n, int r) { // runs in O(r)
        T res(1);
        for(int i = 0; i < r; i++) {
            res *= (n - T(i));
            res /= (i + 1);
        }
        return res;
    }

template<int m>
    struct modular {
        int64_t r;
        modular() : r(0) {}
        modular(int64_t rr) : r(rr) {if(abs(r) >= m) r
           %= m; if(r < 0) r += m;}
        modular inv() const {return bpow(*this, m - 2);}
        modular operator * (const modular &t) const
           {return (r * t.r) % m;}
        modular operator / (const modular &t) const
           {return *this * t.inv();}
        modular operator += (const modular &t) {r +=
           t.r; if(r >= m) r -= m; return *this;}
        modular operator -= (const modular &t) {r -=
           t.r; if(r < 0) r += m; return *this;}
        modular operator + (const modular &t) const
           {return modular(*this) += t;}
        modular operator - (const modular &t) const
```

```
           {return modular(*this) -= t;}
        modular operator *= (const modular &t) {return
           *this = *this * t;}
        modular operator /= (const modular &t) {return
           *this = *this / t;}

        bool operator == (const modular &t) const
           {return r == t.r;}
        bool operator != (const modular &t) const
           {return r != t.r;}

        operator int64_t() const {return r;}
    };
template<int T>
    istream& operator >> (istream &in, modular<T> &x) {
        return in >> x.r;
    }


template<typename T>
    struct poly {
        vector<T> a;

        void normalize() { // get rid of leading zeroes
            while(!a.empty() && a.back() == T(0)) {
                a.pop_back();
            }
        }

        poly(){}
        poly(T a0) : a{a0}{normalize();}
        poly(vector<T> t) : a(t){normalize();}

        poly operator += (const poly &t) {
            a.resize(max(a.size(), t.a.size()));
            for(size_t i = 0; i < t.a.size(); i++) {
                a[i] += t.a[i];
            }
            normalize();
            return *this;
        }
        poly operator -= (const poly &t) {
```

```
        a.resize(max(a.size(), t.a.size()));
        for(size_t i = 0; i < t.a.size(); i++) {
            a[i] -= t.a[i];
        }
        normalize();
        return *this;
    }
    poly operator + (const poly &t) const {return
       poly(*this) += t;}
    poly operator - (const poly &t) const {return
       poly(*this) -= t;}

    poly mod_xk(size_t k) const { // get same
       polynomial mod x^k
        k = min(k, a.size());
        return vector<T>(begin(a), begin(a) + k);
    }
    poly mul_xk(size_t k) const { // multiply by x^k
        poly res(*this);
        res.a.insert(begin(res.a), k, 0);
        return res;
    }
    poly div_xk(size_t k) const { // divide by x^k,
       dropping coefficients
        k = min(k, a.size());
        return vector<T>(begin(a) + k, end(a));
    }
    poly substr(size_t l, size_t r) const { //
       return mod_xk(r).div_xk(l)
        l = min(l, a.size());
        r = min(r, a.size());
        return vector<T>(begin(a) + l, begin(a) + r);
    }
    poly inv(size_t n) const { // get inverse series
       mod x^n
        assert(!is_zero());
        poly ans = a[0].inv();
        size_t a = 1;
        while(a < n) {
            poly C = (ans * mod_xk(2 * a)).substr(a,
               2 * a);
            ans -= (ans * C).mod_xk(a).mul_xk(a);
            a *= 2;
        }
        return ans.mod_xk(n);
    }

    poly operator *= (const poly &t) {fft::mul(a,
       t.a); normalize(); return *this;}
    poly operator * (const poly &t) const {return
       poly(*this) *= t;}

    poly reverse(size_t n, bool rev = 0) const { //
       reverses and leaves only n terms
        poly res(*this);
        if(rev) { // If rev = 1 then tail goes to
           head
            res.a.resize(max(n, res.a.size()));
        }
        std::reverse(res.a.begin(), res.a.end());
        return res.mod_xk(n);
    }

    pair<poly, poly> divmod_slow(const poly &b)
       const { // when divisor or quotient is small
        vector<T> A(a);
        vector<T> res;
        while(A.size() >= b.a.size()) {
            res.push_back(A.back() / b.a.back());
            if(res.back() != T(0)) {
                for(size_t i = 0; i < b.a.size();
                   i++) {
                    A[A.size() - i - 1] -=
                       res.back() * b.a[b.a.size() -
                       i - 1];
                }
            }
            A.pop_back();
        }
        std::reverse(begin(res), end(res));
        return {res, A};
    }

    pair<poly, poly> divmod(const poly &b) const {
```

```
    // returns quotiend and remainder of a mod b
    if(deg() < b.deg()) {
        return {poly{0}, *this};
    }
    int d = deg() - b.deg();
    if(min(d, b.deg()) < magic) {
        return divmod_slow(b);
    }
    poly D = (reverse(d + 1) * b.reverse(d +
        1).inv(d + 1)).mod_xk(d + 1).reverse(d +
        1, 1);
    return {D, *this - D * b};
}

poly operator / (const poly &t) const {return
    divmod(t).first;}
poly operator % (const poly &t) const {return
    divmod(t).second;}
poly operator /= (const poly &t) {return *this =
    divmod(t).first;}
poly operator %= (const poly &t) {return *this =
    divmod(t).second;}
poly operator *= (const T &x) {
    for(auto &it: a) {
        it *= x;
    }
    normalize();
    return *this;
}
poly operator /= (const T &x) {
    for(auto &it: a) {
        it /= x;
    }
    normalize();
    return *this;
}
poly operator * (const T &x) const {return
    poly(*this) *= x;}
poly operator / (const T &x) const {return
    poly(*this) /= x;}

void print() const {
    for(auto it: a) {
        cout << it << ' ';
    }
    cout << endl;
}
T eval(T x) const { // evaluates in single point
    x
    T res(0);
    for(int i = int(a.size()) - 1; i >= 0; i--) {
        res *= x;
        res += a[i];
    }
    return res;
}

T& lead() { // leading coefficient
    return a.back();
}
int deg() const { // degree
    return a.empty() ? -inf : a.size() - 1;
}
bool is_zero() const { // is polynomial zero
    return a.empty();
}
T operator [](int idx) const {
    return idx >= (int)a.size() || idx < 0 ?
        T(0) : a[idx];
}

T& coef(size_t idx) { // mutable reference at
    coefficient
    return a[idx];
}
bool operator == (const poly &t) const {return a
    == t.a;}
bool operator != (const poly &t) const {return a
    != t.a;}

poly deriv() { // calculate derivative
    vector<T> res;
    for(int i = 1; i <= deg(); i++) {
        res.push_back(T(i) * a[i]);
```

```
    }
    return res;
}
poly integr() { // calculate integral with C = 0
    vector<T> res = {0};
    for(int i = 0; i <= deg(); i++) {
        res.push_back(a[i] / T(i + 1));
    }
    return res;
}
size_t leading_xk() const { // Let p(x) = x^k *
   t(x), return k
    if(is_zero()) {
        return inf;
    }
    int res = 0;
    while(a[res] == T(0)) {
        res++;
    }
    return res;
}
poly log(size_t n) { // calculate log p(x) mod
   x^n
    assert(a[0] == T(1));
    return (deriv().mod_xk(n) *
       inv(n)).integr().mod_xk(n);
}
poly exp(size_t n) { // calculate exp p(x) mod
   x^n
    if(is_zero()) {
        return T(1);
    }
    assert(a[0] == T(0));
    poly ans = T(1);
    size_t a = 1;
    while(a < n) {
        poly C = ans.log(2 * a).div_xk(a) -
           substr(a, 2 * a);
        ans -= (ans * C).mod_xk(a).mul_xk(a);
        a *= 2;
    }
    return ans.mod_xk(n);
}
poly pow_slow(size_t k, size_t n) { // if k is
   small
    return k ? k % 2 ? (*this * pow_slow(k - 1,
       n)).mod_xk(n) : (*this *
       *this).mod_xk(n).pow_slow(k / 2, n) :
       T(1);
}
poly pow(size_t k, size_t n) { // calculate
   p^k(n) mod x^n
    if(is_zero()) {
        return *this;
    }
    if(k < magic) {
        return pow_slow(k, n);
    }
    int i = leading_xk();
    T j = a[i];
    poly t = div_xk(i) / j;
    return bpow(j, k) * (t.log(n) *
       T(k)).exp(n).mul_xk(i * k).mod_xk(n);
}
poly mulx(T x) { // component-wise
   multiplication with x^k
    T cur = 1;
    poly res(*this);
    for(int i = 0; i <= deg(); i++) {
        res.coef(i) *= cur;
        cur *= x;
    }
    return res;
}
poly mulx_sq(T x) { // component-wise
   multiplication with x^{k^2}
    T cur = x;
    T total = 1;
    T xx = x * x;
    poly res(*this);
    for(int i = 0; i <= deg(); i++) {
        res.coef(i) *= total;
        total *= cur;
```

```
        cur *= xx;
    }
    return res;
}
vector<T> chirpz_even(T z, int n) { // P(1),
   P(z^2), P(z^4), ..., P(z^2(n-1))
    int m = deg();
    if(is_zero()) {
        return vector<T>(n, 0);
    }
    vector<T> vv(m + n);
    T zi = z.inv();
    T zz = zi * zi;
    T cur = zi;
    T total = 1;
    for(int i = 0; i <= max(n - 1, m); i++) {
        if(i <= m) {vv[m - i] = total;}
        if(i < n) {vv[m + i] = total;}
        total *= cur;
        cur *= zz;
    }
    poly w = (mulx_sq(z) * vv).substr(m, m +
       n).mulx_sq(z);
    vector<T> res(n);
    for(int i = 0; i < n; i++) {
        res[i] = w[i];
    }
    return res;
}
vector<T> chirpz(T z, int n) { // P(1), P(z),
   P(z^2), ..., P(z^(n-1))
    auto even = chirpz_even(z, (n + 1) / 2);
    auto odd = mulx(z).chirpz_even(z, n / 2);
    vector<T> ans(n);
    for(int i = 0; i < n / 2; i++) {
        ans[2 * i] = even[i];
        ans[2 * i + 1] = odd[i];
    }
    if(n % 2 == 1) {
        ans[n - 1] = even.back();
    }
    return ans;
}
template<typename iter>
    vector<T> eval(vector<poly> &tree, int v,
       iter l, iter r) { // auxiliary evaluation
       function
        if(r - l == 1) {
            return {eval(*l)};
        } else {
            auto m = l + (r - l) / 2;
            auto A = (*this % tree[2 *
               v]).eval(tree, 2 * v, l, m);
            auto B = (*this % tree[2 * v +
               1]).eval(tree, 2 * v + 1, m, r);
            A.insert(end(A), begin(B), end(B));
            return A;
        }
    }
vector<T> eval(vector<T> x) { // evaluate
   polynomial in (x1, ..., xn)
    int n = x.size();
    if(is_zero()) {
        return vector<T>(n, T(0));
    }
    vector<poly> tree(4 * n);
    build(tree, 1, begin(x), end(x));
    return eval(tree, 1, begin(x), end(x));
}
template<typename iter>
    poly inter(vector<poly> &tree, int v, iter
       l, iter r, iter ly, iter ry) { //
       auxiliary interpolation function
        if(r - l == 1) {
            return {*ly / a[0]};
        } else {
            auto m = l + (r - l) / 2;
            auto my = ly + (ry - ly) / 2;
            auto A = (*this % tree[2 *
               v]).inter(tree, 2 * v, l, m, ly,
               my);
            auto B = (*this % tree[2 * v +
               1]).inter(tree, 2 * v + 1, m, r,
               my, ry);
```

```
                    return A * tree[2 * v + 1] + B *
                        tree[2 * v];
                }
            }
        };
    template<typename T>
        poly<T> operator * (const T& a, const poly<T>& b) {
            return b * a;
        }

    template<typename T>
        poly<T> xk(int k) { // return x^k
            return poly<T>{1}.mul_xk(k);
        }

    template<typename T>
        T resultant(poly<T> a, poly<T> b) { // computes
            resultant of a and b
            if(b.is_zero()) {
                return 0;
            } else if(b.deg() == 0) {
                return bpow(b.lead(), a.deg());
            } else {
                int pw = a.deg();
                a %= b;
                pw -= a.deg();
                T mul = bpow(b.lead(), pw) * T((b.deg() &
                    a.deg() & 1) ? -1 : 1);
                T ans = resultant(b, a);
                return ans * mul;
            }
        }
    template<typename iter>
        poly<typename iter::value_type> kmul(iter L, iter R)
            { // computes (x-a1)(x-a2)...(x-an) without
            building tree
            if(R - L == 1) {
                return vector<typename
                    iter::value_type>{-*L, 1};
            } else {
                iter M = L + (R - L) / 2;
                return kmul(L, M) * kmul(M, R);
            }
        }
    template<typename T, typename iter>
        poly<T> build(vector<poly<T>> &res, int v, iter L,
            iter R) { // builds evaluation tree for
            (x-a1)(x-a2)...(x-an)
            if(R - L == 1) {
                return res[v] = vector<T>{-*L, 1};
            } else {
                iter M = L + (R - L) / 2;
                return res[v] = build(res, 2 * v, L, M) *
                    build(res, 2 * v + 1, M, R);
            }
        }
    template<typename T>
        poly<T> inter(vector<T> x, vector<T> y) { //
            interpolates minimum polynomial from (xi, yi)
            pairs
            int n = x.size();
            vector<poly<T>> tree(4 * n);
            return build(tree, 1, begin(x),
                end(x)).deriv().inter(tree, 1, begin(x),
                end(x), begin(y), end(y));
        }
};

using namespace algebra;

const int mod = 1e9 + 7;
typedef modular<mod> base;
typedef poly<base> polyn;

using namespace algebra;

signed main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n = 100000;
    polyn a;
    vector<base> x;
    for(int i = 0; i <= n; i++) {
        a.a.push_back(1 + rand() % 100);
```

```
        x.push_back(1 + rand() % (2 * n));
    }
    sort(begin(x), end(x));
    x.erase(unique(begin(x), end(x)), end(x));
    auto b = a.eval(x);
    cout << clock() / double(CLOCKS_PER_SEC) << endl;
    auto c = inter(x, b);
    polyn md = kmul(begin(x), end(x));
    cout << clock() / double(CLOCKS_PER_SEC) << endl;
    assert(c == a % md);
    return 0;
}
```

## 7.4 Primitivas de matriz - exponenciacao

```
#define MODULAR false
template<typename T> struct matrix : vector<vector<T>> {
    int n, m;

    void print() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) cout <<
               (*this)[i][j] << " ";
            cout << endl;
        }
    }

    matrix(int n_, int m_, bool ident = false) :
            vector<vector<T>>(n_, vector<T>(m_, 0)), n(n_),
               m(m_) {
        if (ident) {
            assert(n == m);
            for (int i = 0; i < n; i++) (*this)[i][i] = 1;
        }
    }
    matrix(const vector<vector<T>>& c) :
       vector<vector<T>>(c),
        n(c.size()), m(c[0].size()) {}
    matrix(const initializer_list<initializer_list<T>>& c) {
        vector<vector<T>> val;
        for (auto& i : c) val.push_back(i);
        *this = matrix(val);
    }

    matrix<T> operator*(matrix<T>& r) {
        assert(m == r.n);
        matrix<T> M(n, r.m);
        for (int i = 0; i < n; i++) for (int k = 0; k < m;
           k++)
            for (int j = 0; j < r.m; j++) {
                T add = (*this)[i][k] * r[k][j];
#if MODULAR
#warning Usar matrix<ll> e soh colocar valores em [0, MOD)
   na matriz!
                M[i][j] += add%MOD;
                if (M[i][j] >= MOD) M[i][j] -= MOD;
#else
                M[i][j] += add;
#endif
            }
        return M;
    }
    matrix<T> operator^(ll e){
        matrix<T> M(n, n, true), at = *this;
        while (e) {
            if (e&1) M = M*at;
            e >>= 1;
            at = at*at;
        }
        return M;
    }
    void apply_transform(matrix M, ll e){
        auto& v = *this;
        while (e) {
            if (e&1) v = M*v;
            e >>= 1;
            M = M*M;
        }
    }
};
```

## 7.5 Big Integer

```
// Complexidades: (para n digitos)
// Soma, subtracao, comparacao - O(n)
// Multiplicacao - O(n log(n))
// Divisao, resto - O(n^2)

struct bint {
    static const int BASE = 1e9;
    vector<int> v;
    bool neg;

    bint() : neg(0) {}
    bint(int val) : bint() { *this = val; }
    bint(long long val) : bint() { *this = val; }

    void trim() {
        while (v.size() and v.back() == 0) v.pop_back();
        if (!v.size()) neg = 0;
    }

    // converter de/para string | cin/cout
    bint(const char* s) : bint() { from_string(string(s)); }
    bint(const string& s) : bint() { from_string(s); }
    void from_string(const string& s) {
        v.clear(), neg = 0;
        int ini = 0;
        while (ini < s.size() and (s[ini] == '-' or s[ini]
            == '+' or s[ini] == '0'))
            if (s[ini++] == '-') neg = 1;
        for (int i = s.size()-1; i >= ini; i -= 9) {
            int at = 0;
            for (int j = max(ini, i - 8); j <= i; j++) at =
                10*at + (s[j]-'0');
            v.push_back(at);
        }
        if (!v.size()) neg = 0;
    }
    string to_string() const {
        if (!v.size()) return "0";
        string ret;
        if (neg) ret += '-';
        for (int i = v.size()-1; i >= 0; i--) {
            string at = ::to_string(v[i]);
            int add = 9 - at.size();
            if (i+1 < v.size()) for (int j = 0; j < add;
                j++) ret += '0';
            ret += at;
        }
        return ret;
    }
    friend istream& operator>>(istream& in, bint& val) {
        string s; in >> s;
        val = s;
        return in;
    }
    friend ostream& operator<<(ostream& out, const bint&
        val) {
        string s = val.to_string();
        out << s;
        return out;
    }

    // operators
    friend bint abs(bint val) {
        val.neg = 0;
        return val;
    }
    friend bint operator-(bint val) {
        if (val != 0) val.neg ^= 1;
        return val;
    }
    bint& operator=(const bint& val) { v = val.v, neg =
        val.neg; return *this; }
    bint& operator=(long long val) {
        v.clear(), neg = 0;
        if (val < 0) neg = 1, val *= -1;
        for (; val; val /= BASE) v.push_back(val % BASE);
        return *this;
    }
    int cmp(const bint& r) const { // menor: -1 | igual: 0 |
        maior: 1
        if (neg != r.neg) return neg ? -1 : 1;
        if (v.size() != r.v.size()) {
```

```
        int ret = v.size() < r.v.size() ? -1 : 1;
        return neg ? -ret : ret;
    }
    for (int i = int(v.size())-1; i >= 0; i--) {
        if (v[i] != r.v[i]) {
            int ret = v[i] < r.v[i] ? -1 : 1;
            return neg ? -ret : ret;
        }
    }
    return 0;
}
friend bool operator<(const bint& l, const bint& r) {
   return l.cmp(r) == -1; }
friend bool operator>(const bint& l, const bint& r) {
   return l.cmp(r) == 1; }
friend bool operator<=(const bint& l, const bint& r) {
   return l.cmp(r) <= 0; }
friend bool operator>=(const bint& l, const bint& r) {
   return l.cmp(r) >= 0; }
friend bool operator==(const bint& l, const bint& r) {
   return l.cmp(r) == 0; }
friend bool operator!=(const bint& l, const bint& r) {
   return l.cmp(r) != 0; }

bint& operator +=(const bint& r) {
    if (!r.v.size()) return *this;
    if (neg != r.neg) return *this -= -r;
    for (int i = 0, c = 0; i < r.v.size() or c; i++) {
        if (i == v.size()) v.push_back(0);
        v[i] += c + (i < r.v.size() ? r.v[i] : 0);
        if ((c = v[i] >= BASE)) v[i] -= BASE;
    }
    return *this;
}
friend bint operator+(bint a, const bint& b) { return a
   += b; }
bint& operator -=(const bint& r) {
    if (!r.v.size()) return *this;
    if (neg != r.neg) return *this += -r;
    if ((!neg and *this < r) or (neg and r < *this)) {
        *this = r - *this;
        neg ^= 1;
        return *this;
    }
    for (int i = 0, c = 0; i < r.v.size() or c; i++) {
        v[i] -= c + (i < r.v.size() ? r.v[i] : 0);
        if ((c = v[i] < 0)) v[i] += BASE;
    }
    trim();
    return *this;
}
friend bint operator-(bint a, const bint& b) { return a
   -= b; }

// operators de * / %
bint& operator *=(int val) {
    if (val < 0) val *= -1, neg ^= 1;
    for (int i = 0, c = 0; i < v.size() or c; i++) {
        if (i == v.size()) v.push_back(0);
        long long at = (long long) v[i] * val + c;
        v[i] = at % BASE;
        c = at / BASE;
    }
    trim();
    return *this;
}
friend bint operator *(bint a, int b) { return a *= b; }
friend bint operator *(int a, bint b) { return b *= a; }
using cplx = complex<double>;
void fft(vector<cplx>& a, bool f, int N, vector<int>&
   rev) const {
    for (int i = 0; i < N; i++) if (i < rev[i])
       swap(a[i], a[rev[i]]);
    vector<cplx> roots(N);
    for (int n = 2; n <= N; n *= 2) {
        const static double PI = acos(-1);
        for (int i = 0; i < n/2; i++) {
            double alpha = (2*PI*i)/n;
            if (f) alpha = -alpha;
            roots[i] = cplx(cos(alpha), sin(alpha));
        }
        for (int pos = 0; pos < N; pos += n)
            for (int l = pos, r = pos+n/2, m = 0; m <
               n/2; l++, r++, m++) {
```

```
                auto t = roots[m]*a[r];
                a[r] = a[l] - t;
                a[l] = a[l] + t;
            }
        }
        if (!f) return;
        auto invN = cplx(1)/cplx(N);
        for (int i = 0; i < N; i++) a[i] *= invN;
    }
    vector<long long> convolution(const vector<int>& a,
        const vector<int>& b) const {
        vector<cplx> l(a.begin(), a.end()), r(b.begin(),
            b.end());
        int ln = l.size(), rn = r.size(), N = ln+rn+1, n =
            1, log_n = 0;
        while (n <= N) n <<= 1, log_n++;
        vector<int> rev(n);
        for (int i = 0; i < n; i++) {
            rev[i] = 0;
            for (int j = 0; j < log_n; j++) if (i>>j&1)
                rev[i] |= 1 << (log_n-1-j);
        }
        l.resize(n), r.resize(n);
        fft(l, false, n, rev), fft(r, false, n, rev);
        for (int i = 0; i < n; i++) l[i] *= r[i];
        fft(l, true, n, rev);
        vector<long long> ret;
        for (auto& i : l) ret.push_back(round(i.real()));
        return ret;
    }
    vector<int> convert_base(const vector<int>& a, int from,
        int to) const {
        static vector<long long> pot(10, 1);
        if (pot[1] == 1) for (int i = 1; i < 10; i++) pot[i]
            = 10*pot[i-1];
        vector<int> ret;
        long long at = 0;
        int digits = 0;
        for (int i : a) {
            at += i * pot[digits];
            digits += from;
            while (digits >= to) {
                ret.push_back(at % pot[to]);
                at /= pot[to];
                digits -= to;
            }
        }
        ret.push_back(at);
        while (ret.size() and ret.back() == 0)
            ret.pop_back();
        return ret;
    }
    bint operator*(const bint& r) const { // O(n log(n))
        bint ret;
        ret.neg = neg ^ r.neg;
        auto conv = convolution(convert_base(v, 9, 4),
            convert_base(r.v, 9, 4));
        long long c = 0;
        for (auto i : conv) {
            long long at = i+c;
            ret.v.push_back(at % 10000);
            c = at / 10000;
        }
        for (; c; c /= 10000) ret.v.push_back(c%10000);
        ret.v = convert_base(ret.v, 4, 9);
        if (!ret.v.size()) ret.neg = 0;
        return ret;
    }
    bint& operator*=(const bint& r) { return *this = *this *
        r; };
    bint& operator/=(int val) {
        if (val < 0) neg ^= 1, val *= -1;
        for (int i = int(v.size())-1, c = 0; i >= 0; i--) {
            long long at = v[i] + c * (long long) BASE;
            v[i] = at / val;
            c = at % val;
        }
        trim();
        return *this;
    }
    friend bint operator/(bint a, int b) { return a /= b; }
    int operator %=(int val) {
        if (val < 0) val *= -1;
        long long at = 0;
```

```
        for (int i = int(v.size())-1; i >= 0; i--)
            at = (BASE * at + v[i]) % val;
        if (neg) at *= -1;
        return at;
    }
    friend int operator%(bint a, int b) { return a %= b; }
    friend pair<bint, bint> divmod(const bint& a_, const
       bint& b_) { // O(n^2)
        if (a_ == 0) return {0, 0};
        int norm = BASE / (b_.v.back() + 1);
        bint a = abs(a_) * norm;
        bint b = abs(b_) * norm;
        bint q, r;
        for (int i = a.v.size() - 1; i >= 0; i--) {
            r *= BASE, r += a.v[i];
            long long upper = b.v.size() < r.v.size() ?
               r.v[b.v.size()] : 0;
            int lower = b.v.size() - 1 < r.v.size() ?
               r.v[b.v.size() - 1] : 0;
            int d = (upper * BASE + lower) / b.v.back();
            r -= b*d;
            while (r < 0) r += b, d--; // roda O(1) vezes
            q.v.push_back(d);
        }
        reverse(q.v.begin(), q.v.end());
        q.neg = a_.neg ^ b_.neg;
        r.neg = a_.neg;
        q.trim(), r.trim();
        return {q, r / norm};
    }
    bint operator/(const bint& val) { return divmod(*this,
       val).first; }
    bint& operator/=(const bint& val) { return *this = *this
       / val; }
    bint operator%(const bint& val) { return divmod(*this,
       val).second; }
    bint& operator%=(const bint& val) { return *this = *this
       % val; }
};
```

## 7.6 Complex

```
struct cplx{
    double r, i;
    cplx(complex<double> c):r(c.real()), i(c.imag()){}
    cplx() : r(0), i(0){}
    cplx(double r_, double i_ = 0):r(r_), i(i_){}
    double abs(){ return hypot(r, i); }
    double abs2(){ return r*r + i*i; }
    cplx inv() { return cplx(r/abs2(), i/abs2()); }
    cplx& operator+=(cplx a){
        r += a.r; i += a.i;
        return *this;
    }
    cplx& operator-=(cplx a){
        r -= a.r; i -= a.i;
        return *this;
    }
    cplx& operator*=(cplx a){
        double r_ = r*a.r - i*a.i;
        double i_ = r*a.i + i*a.r;
        r = r_;
        i = i_;
        return *this;
    }
    cplx conj(){
        return cplx(r, -i);
    }
    cplx& operator/=(cplx a){
        auto a_ = a.inv();
        return (*this)*=a_;
    }
    cplx operator-(){ return cplx(-r, -i); }
    cplx& operator^=(double e){
        return *this = pow(complex<double>(r, i), e);
    }
    friend ostream &operator<<(ostream &out, cplx a){
        return out << a.r << " + " << a.i << "i";
    }
    friend cplx operator+(cplx a, cplx b){ return a+=b; }
    friend cplx operator-(cplx a, cplx b){ return a-=b; }
    friend cplx operator*(cplx a, cplx b){ return a*=b; }
```

```
    friend cplx operator/(cplx a, cplx b){ return a/=b; }
    friend cplx operator^(cplx a, double e){ return a^=e; }

    //fft
    static int fft_len(int N){
        int n = 1, log_n = 0;
        while (n <= N) { n <<= 1; log_n++; }
        return log_n;
    }
    static cplx rt(bool f, int n, int N){
        const static double PI = acos(-1);
        double alpha = (2*PI)/n;
        if (f) alpha = -alpha;
        return cplx(cos(alpha), sin(alpha));
    }
};
```

## 7.7 Primitivas de fracao

```
// Funciona com o Big Int

template<typename T = int> struct frac {
    T num, den;
    template<class U> frac(U num_ = 0, U den_ = 1) :
       num(num_), den(den_) {
        assert(den != 0);
        if (den < 0) num *= -1, den *= -1;
        T g = gcd(abs(num), den);
        num /= g, den /= g;
    }

    friend bool operator<(const frac& l, const frac& r) {
        return l.num * r.den < r.num * l.den;
    }
    friend frac operator+(const frac& l, const frac& r) {
        return {l.num*r.den + l.den*r.num, l.den*r.den};
    }
    friend frac operator-(const frac& l, const frac& r) {
        return {l.num*r.den - l.den*r.num, l.den*r.den};
    }
    friend frac operator*(const frac& l, const frac& r) {
        return {l.num*r.num, l.den*r.den};
    }
    friend frac operator/(const frac& l, const frac& r) {
        return {l.num*r.den, l.den*r.num};
    }
    friend ostream& operator<<(ostream& out, frac f) {
        out << f.num << '/' << f.den;
        return out;
    }
};
```

## 7.8 Primitivas Geometricas

```
typedef double ld;
const ld DINF = 1e18;
const ld pi = acos(-1.0);
const ld eps = 1e-9;

#define sq(x) ((x)*(x))

bool eq(ld a, ld b) {
    return abs(a - b) <= eps;
}

struct pt { // ponto
    ld x, y;
    pt(ld x_ = 0, ld y_ = 0) : x(x_), y(y_) {}
    bool operator < (const pt p) const {
        if (!eq(x, p.x)) return x < p.x;
        if (!eq(y, p.y)) return y < p.y;
        return 0;
    }
    bool operator == (const pt p) const {
        return eq(x, p.x) and eq(y, p.y);
    }
    pt operator + (const pt p) const { return pt(x+p.x,
       y+p.y); }
    pt operator - (const pt p) const { return pt(x-p.x,
       y-p.y); }
```

```
    pt operator * (const ld c) const { return pt(x*c  , y*c
       ); }
    pt operator / (const ld c) const { return pt(x/c  , y/c
       ); }
    ld operator * (const pt p) const { return x*p.x + y*p.y;
       }
    ld operator ^ (const pt p) const { return x*p.y - y*p.x;
       }
    friend istream& operator >> (istream& in, pt& p) {
        return in >> p.x >> p.y;
    }
};

struct line { // reta
    pt p, q;
    line() {}
    line(pt p_, pt q_) : p(p_), q(q_) {}
    friend istream& operator >> (istream& in, line& r) {
        return in >> r.p >> r.q;
    }
};

// PONTO & VETOR

ld dist(pt p, pt q) { // distancia
    return hypot(p.y - q.y, p.x - q.x);
}

ld dist2(pt p, pt q) { // quadrado da distancia
    return sq(p.x - q.x) + sq(p.y - q.y);
}

ld norm(pt v) { // norma do vetor
    return dist(pt(0, 0), v);
}

ld angle(pt v) { // angulo do vetor com o eixo x
    ld ang = atan2(v.y, v.x);
    if (ang < 0) ang += 2*pi;
    return ang;
}
```

```
ld sarea(pt p, pt q, pt r) { // area com sinal
    return ((q-p)^(r-q))/2;
}

bool col(pt p, pt q, pt r) { // se p, q e r sao colin.
    return eq(sarea(p, q, r), 0);
}

int paral(pt u, pt v) { // se u e v sao paralelos
    if (!eq(u^v, 0)) return 0;
    if ((u.x > eps) == (v.x > eps) and (u.y > eps) == (v.y >
        eps))
        return 1;
    return -1;
}

bool ccw(pt p, pt q, pt r) { // se p, q, r sao ccw
    return sarea(p, q, r) > eps;
}

pt rotate(pt p, ld th) { // rotaciona o ponto th radianos
    return pt(p.x * cos(th) - p.y * sin(th),
            p.x * sin(th) + p.y * cos(th));
}

pt rotate90(pt p) { // rotaciona 90 graus
    return pt(-p.y, p.x);
}

// RETA

bool isvert(line r) { // se r eh vertical
    return eq(r.p.x, r.q.x);
}

bool paraline(line r, line s) { // se r e s sao paralelas
    return paral(r.p - r.q, s.p - s.q);
}

bool isinseg(pt p, line r) { // se p pertence ao seg de r
    if (p == r.p or p == r.q) return 1;
    return paral(p - r.p, p - r.q) == -1;
```

```
}

ld get_t(pt v, line r) { // retorna t tal que t*v pertence a
    reta r
    return (r.p^r.q) / ((r.p-r.q)^v);
}

pt proj(pt p, line r) { // projecao do ponto p na reta r
    if (r.p == r.q) return r.p;
    r.q = r.q - r.p; p = p - r.p;
    pt proj = r.q * ((p*r.q) / (r.q*r.q));
    return proj + r.p;
}

pt inter(line r, line s) { // r inter s
    if (paraline(r, s)) return pt(DINF, DINF);
    r.q = r.q - r.p, s.p = s.p - r.p, s.q = s.q - r.p;
    return r.q * get_t(r.q, s) + r.p;
}

bool interseg(line r, line s) { // se o seg de r intersecta
    o seg de s
    if (isinseg(r.p, s) or isinseg(r.q, s)
        or isinseg(s.p, r) or isinseg(s.q, r)) return 1;

    return ccw(r.p, r.q, s.p) != ccw(r.p, r.q, s.q) and
            ccw(s.p, s.q, r.p) != ccw(s.p, s.q, r.q);
}

ld disttoline(pt p, line r) { // distancia do ponto a reta
    return 2 * abs(sarea(p, r.p, r.q)) / dist(r.p, r.q);
}

ld disttoseg(pt p, line r) { // distancia do ponto ao seg
    if ((r.q - r.p)*(p - r.p) < 0) return dist(r.p, p);
    if ((r.p - r.q)*(p - r.q) < 0) return dist(r.q, p);
    return disttoline(p, r);
}

ld distseg(line a, line b) { // distancia entre seg
    if (interseg(a, b)) return 0;

    ld ret = DINF;
    ret = min(ret, disttoseg(a.p, b));
    ret = min(ret, disttoseg(a.q, b));
    ret = min(ret, disttoseg(b.p, a));
    ret = min(ret, disttoseg(b.q, a));

    return ret;
}

// POLIGONO

// distancia entre os retangulos a e b (lados paralelos aos
    eixos)
// assume que ta representado (inferior esquerdo, superior
    direito)
ld dist_rect(pair<pt, pt> a, pair<pt, pt> b) {
    ld hor = 0, vert = 0;
    if (a.second.x < b.first.x) hor = b.first.x - a.second.x;
    else if (b.second.x < a.first.x) hor = a.first.x -
        b.second.x;
    if (a.second.y < b.first.y) vert = b.first.y -
        a.second.y;
    else if (b.second.y < a.first.y) vert = a.first.y -
        b.second.y;
    return dist(pt(0, 0), pt(hor, vert));
}

ld polarea(vector<pt> v) { // area do poligono
    ld ret = 0;
    for (int i = 0; i < v.size(); i++)
        ret += sarea(pt(0, 0), v[i], v[(i + 1) % v.size()]);
    return abs(ret);
}

// se o ponto ta dentro do poligono: retorna 0 se ta fora,
// 1 se ta no interior e 2 se ta na borda
int inpol(vector<pt>& v, pt p) { // O(n)
    int qt = 0;
    for (int i = 0; i < v.size(); i++) {
        if (p == v[i]) return 2;
        int j = (i+1)%v.size();
        if (eq(p.y, v[i].y) and eq(p.y, v[j].y)) {
```

```
            if ((v[i]-p)*(v[j]-p) < eps) return 2;
            continue;
        }
        bool baixo = v[i].y+eps < p.y;
        if (baixo == (v[j].y+eps < p.y)) continue;
        auto t = (p-v[i])^(v[j]-v[i]);
        if (eq(t, 0)) return 2;
        if (baixo == (t > eps)) qt += baixo ? 1 : -1;
    }
    return qt != 0;
}

bool interpol(vector<pt> v1, vector<pt> v2) { // se dois
    poligonos se intersectam - O(n*m)
    int n = v1.size(), m = v2.size();
    for (int i = 0; i < n; i++) if (inpol(v2, v1[i])) return
        1;
    for (int i = 0; i < n; i++) if (inpol(v1, v2[i])) return
        1;
    for (int i = 0; i < n; i++) for (int j = 0; j < m; j++)
        if (interseg(line(v1[i], v1[(i+1)%n]), line(v2[j],
            v2[(j+1)%m]))) return 1;
    return 0;
}

ld distpol(vector<pt> v1, vector<pt> v2) { // distancia
    entre poligonos
    if (interpol(v1, v2)) return 0;

    ld ret = DINF;

    for (int i = 0; i < v1.size(); i++) for (int j = 0; j <
        v2.size(); j++)
        ret = min(ret, distseg(line(v1[i], v1[(i + 1) %
            v1.size()]),
                    line(v2[j], v2[(j + 1) % v2.size()])));
    return ret;
}

vector<pt> convex_hull(vector<pt> v) { // convex hull - O(n
    log(n))
    if (v.size() <= 1) return v;
```

```
    vector<pt> l, u;
    sort(v.begin(), v.end());
    for (int i = 0; i < v.size(); i++) {
        while (l.size() > 1 and !ccw(l[l.size()-2],
            l.back(), v[i]))
            l.pop_back();
        l.push_back(v[i]);
    }
    for (int i = v.size() - 1; i >= 0; i--) {
        while (u.size() > 1 and !ccw(u[u.size()-2],
            u.back(), v[i]))
            u.pop_back();
        u.push_back(v[i]);
    }
    l.pop_back(); u.pop_back();
    for (pt i : u) l.push_back(i);
    return l;
}

struct convex_pol {
    vector<pt> pol;

    convex_pol(vector<pt> v) : pol(convex_hull(v)) {}
    bool is_inside(pt p) { // se o ponto ta dentro do hull -
        O(log(n))
        if (pol.size() == 1) return p == pol[0];
        int l = 1, r = pol.size();
        while (l < r) {
            int m = (l+r)/2;
            if (ccw(p, pol[0], pol[m])) l = m+1;
            else r = m;
        }
        if (l == 1) return isinseg(p, line(pol[0], pol[1]));
        if (l == pol.size()) return false;
        return !ccw(p, pol[l], pol[l-1]);
    }
};

// CIRCUNFERENCIA

pt getcenter(pt a, pt b, pt c) { // centro da circunf dado 3
    pontos
```

```
    b = (a + b) / 2;
    c = (a + c) / 2;
    return inter(line(b, b + rotate90(a - b)),
            line(c, c + rotate90(a - c)));
}

vector<pt> circ_line_inter(pt a, pt b, pt c, ld r) { //
   intersecao da circunf (c, r) e reta ab
    vector<pt> ret;
    b = b-a, a = a-c;
    ld A = b*b;
    ld B = a*b;
    ld C = a*a - r*r;
    ld D = B*B - A*C;
    if (D < -eps) return ret;
    ret.push_back(c+a+b*(-B+sqrt(D+eps))/A);
    if (D > eps) ret.push_back(c+a+b*(-B-sqrt(D))/A);
    return ret;
}

vector<pt> circ_inter(pt a, pt b, ld r, ld R) { //
   intersecao da circunf (a, r) e (b, R)
    vector<pt> ret;
    ld d = dist(a, b);
    if (d > r+R or d+min(r, R) < max(r, R)) return ret;
    ld x = (d*d-R*R+r*r)/(2*d);
    ld y = sqrt(r*r-x*x);
    pt v = (b-a)/d;
    ret.push_back(a+v*x + rotate90(v)*y);
    if (y > 0) ret.push_back(a+v*x - rotate90(v)*y);
    return ret;
}

bool operator <(const line& a, const line& b) { //
   comparador pra reta
    // assume que as retas tem p < q
    pt v1 = a.q - a.p, v2 = b.q - b.p;
    if (!eq(angle(v1), angle(v2))) return angle(v1) <
       angle(v2);
    return ccw(a.p, a.q, b.p); // mesmo angulo
}
bool operator ==(const line& a, const line& b) {
    return !(a < b) and !(b < a);
}

// comparador pro set pra fazer sweep line com segmentos
struct cmp_sweepline {
    bool operator () (const line& a, const line& b) const {
        // assume que os segmentos tem p < q
        if (a.p == b.p) return ccw(a.p, a.q, b.q);
        if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or
           a.p.x+eps < b.p.x))
            return ccw(a.p, a.q, b.p);
        return ccw(a.p, b.q, b.p);
    }
};

// comparador pro set pra fazer sweep angle com segmentos
pt dir;
struct cmp_sweepangle {
    bool operator () (const line& a, const line& b) const {
        return get_t(dir, a) + eps < get_t(dir, b);
    }
};
```

## 7.9 Primitivas Geometricas 3D

```
typedef double ld;
const ld DINF = 1e18;
const ld pi = acos(-1.0);
const ld eps = 1e-9;

#define sq(x) ((x)*(x))

bool eq(ld a, ld b) {
    return abs(a - b) <= eps;
}

struct pt { // ponto
    ld x, y, z;
    pt(ld x_ = 0, ld y_ = 0, ld z_ = 0) : x(x_), y(y_),
       z(z_) {}
```

```
    bool operator < (const pt p) const {
        if (!eq(x, p.x)) return x < p.x;
        if (!eq(y, p.y)) return y < p.y;
        if (!eq(z, p.z)) return z < p.z;
        return 0;
    }
    bool operator == (const pt p) const {
        return eq(x, p.x) and eq(y, p.y) and eq(z, p.z);
    }
    pt operator + (const pt p) const { return pt(x+p.x,
       y+p.y, z+p.z); }
    pt operator - (const pt p) const { return pt(x-p.x,
       y-p.y, z-p.z); }
    pt operator * (const ld c) const { return pt(x*c  , y*c
       , z*c  ); }
    pt operator / (const ld c) const { return pt(x/c  , y/c
       , z/c  ); }
    ld operator * (const pt p) const { return x*p.x + y*p.y
       + z*p.z; }
    pt operator ^ (const pt p) const { return pt(y*p.z -
       z*p.y, z*p.x - x*p.z, x*p.y - y*p.x); }
};

// converte de coordenadas polares para cartesianas
// (angulos devem estar em radianos)
pt convert(ld rho, ld th, ld phi) {
    return pt(sin(phi) * cos(th), sin(phi) * sin(th),
       cos(phi)) * rho;
}

// distancia
ld dist(pt a, pt b) {
    return sqrt(sq(a.x-b.x) + sq(a.y-b.y) + sq(a.z-b.z));
}

// rotaciona p ao redor do eixo u por um angulo a
pt rotate(pt p, pt u, ld a) {
    u = u / dist(u, pt());
    return u * (u * p) + (u ^ p ^ u) * cos(a) + (u ^ p) *
       sin(a);
}
```

# 8 Extra

## 8.1 fastIO.cpp

```
int read_int() {
    bool minus = false;
    int result = 0;
    char ch;
    ch = getchar();
    while (1) {
        if (ch == '-') break;
        if (ch >= '0' && ch <= '9') break;
        ch = getchar();
    }
    if (ch == '-') minus = true;
    else result = ch-'0';
    while (1) {
        ch = getchar();
        if (ch < '0' || ch > '9') break;
        result = result*10 + (ch - '0');
    }
    if (minus) return -result;
    else return result;
}
```

## 8.2 vimrc

```
set ts=4 si ai sw=4 number mouse=a
syntax on
```

## 8.3 rand.cpp

```
mt19937 rng((int)
   chrono::steady_clock::now().time_since_epoch().count());

int uniform(int l, int r){
    uniform_int_distribution<int> uid(l, r);
```

```
    return uid(rng);
}
```

## 8.4 template.cpp

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

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

int main() { _
    exit(0);
}
```

## 8.5 debug.cpp

```
void debug_out(string s, int line) { cerr << endl; }
template<typename H, typename... T>
void debug_out(string s, int line, H h, T... t) {
    if (s[0] != ',') cerr << "Line(" << line << ") ";
    do { cerr << s[0]; s = s.substr(1);
    } while (s.size() and s[0] != ',');
    cerr << " = " << h;
    debug_out(s, line, t...);
}
#ifdef DEBUG
#define debug(...) debug_out(#__VA_ARGS__, __LINE__,
   __VA_ARGS__)
#else
#define debug(...)
#endif
```

## 8.6 stress.sh

```
make a a2 gen || exit 1
for ((i = 1; ; i++)) do
    ./gen $i > in
    ./a < in > out
    ./a2 < in > out2
    if (! cmp -s out out2) then
        echo "--> entrada:"
        cat in
        echo "--> saida1:"
        cat out
        echo "--> saida2:"
        cat out2
        break;
    fi
    echo $i
done
```

## 8.7 makefile

```
CXX = g++
CXXFLAGS = -fsanitize=address,undefined
   -fno-omit-frame-pointer -g -Wall -Wshadow -std=c++17
   -Wno-unused-result -Wno-sign-compare -Wno-char-subscripts
   #-fuse-ld=gold
```