ICPC Notebook

MWNWMWNNWMWNWN

Contents

1 Geometry

1.1 Convex Hull

```
// Works with double and integer
// Complexity: O(NlogN)
// c3b176
template <bool UPPER>
vector<point> hull(vector<point> v) {
        vector<point> res;
        if(UPPER) for(auto& p: v) p.y = -p.y;
        sort(all(v));
        for(auto& p: v) {
                if(res.empty()) { res.push_back(p); continue; }
                if(res.back().x == p.x) continue;
                while(res.size() >= 2) {
                        point a = res[res.size()-2], b = res.back();
                        if(!right(a, b, p)) res.pop_back();
                        //to include collinear points
                        //if(right(a, b, p)) res.pop_back();
                        else break;
                }
                res.push_back(p);
        }
        if(UPPER) for(auto &p: res) p.y = -p.y;
        return res;
}
```

1.2 Double Basic

```
constexpr double EPS = 1e-10;

bool zero(double x) {
        return abs(x) <= EPS;
}

// CORNER: point = (0, 0)
struct point {
        double x, y;

        point(double x=0, double y=0): x(x), y(y) {}

        point operator+(point rhs) { return point(x+rhs.x, y+rhs.y); }
        point operator-(point rhs) { return point(x-rhs.x, y-rhs.y); }
        point operator*(double k) { return point(x*k, y*k); }
        point operator/(double k) { return point(x/k, y/k); }
        double operator*(point rhs) { return x*rhs.x + y*rhs.y; }
        double operator^(point rhs) { return x*rhs.y - y*rhs.x; }

        point rotated(point p, point polar) { return point(*this^polar,*this*polar); }
        point rotated(point p, double ang) { return rotated(p, point(sin(ang),cos(ang))); }
        double norm2() { return *this * *this; }
        double norm() { return sqrt(norm2()); }

        bool operator<(const point& rhs) const {
                return x < rhs.x - EPS || (zero(x-rhs.x) && y < rhs.y - EPS);
        }

        bool operator==(const point& rhs) const {
                return zero(x-rhs.x) && zero(y-rhs.y);
        }
};

const point ccw90(1, 0), cw90(-1, 0);

// angular comparison in [0, 2pi)
// smallest is (1, 0)
// CORNER: a || b == (0, 0)
bool ang_cmp(point a, point b) {
        auto quad = [](point p) -> bool {
                // 0 if ang in [0, pi), 1 if in [pi, 2pi)
                return p.y < 0 || (p.y == 0 && p.x < 0);
        };
        using tup = tuple<bool, double>;
        return tup{quad(a), 0} < tup{quad(b), a^b};
}
```

```
double dist2(point p, point q) { // squared distance
    return (p - q)*(p - q);
}

double dist(point p, point q) {
    return sqrt(dist2(p, q));
}

double area2(point a, point b, point c) { // two times signed area of triangle abc
        return (b - a) ^ (c - a);
}

bool left(point a, point b, point c) {
        return area2(a, b, c) > EPS; // counterclockwise
}

bool right(point a, point b, point c) {
        return area2(a, b, c) < -EPS; // clockwise
}

bool collinear(point a, point b, point c) {
        return zero(area2(a,b,c));
}

// CORNER: a || b == (0, 0)
int parallel(point a, point b) {
        if((a ^ b) != 0) return 0;
        return (a.x>0) == (b.x>0) && (a.y > 0) == (b.y > 0) ? 1 : -1;
}

// CORNER: a == b
struct segment {
        point a, b;

        segment(point a=point(), point b=point()): a(a), b(b) {}

        point v() { return b - a; }

};

bool contains(segment r, point p) {
        return r.a==p || r.b==p || parallel(r.a-p,r.b-p) == -1;
}

bool intersects(segment r, segment s) {
        if(contains(r, s.a) || contains(r, s.b) || contains(s, r.a) || contains(s, r.b)) return 1;
        return left(r.a,r.b,s.a) != left(r.a,r.b,s.b) &&
                left(s.a, s.b, r.a) != left(s.a, s.b, r.b);
}

bool parallel(segment r, segment s) {
        return parallel(r.v(), s.v());
}

point line_intersection(segment r, segment s) {
        if(parallel(r, s)) return point(HUGE_VAL, HUGE_VAL);
        point vr = r.v(), vs = s.v();
        double cr = vr ^ r.a, cs = vs ^ s.a;
        return (vs*cr - vr*cs) / (vr ^ vs);
}

point proj(segment r, point p) {
        p = p - r.a;
        point v = r.v();
        return r.a + v*((p*v)/(v*v));
}
```

1.3 Integer Basic

```
#define int long long

bool zero(int x) {
        return x == 0;
}

// CORNER: point = (0, 0)
struct point {
        int x, y;

        point(int x=0, int y=0): x(x), y(y) {}

        point operator+(point rhs) { return point(x+rhs.x, y+rhs.y); }
        point operator-(point rhs) { return point(x-rhs.x, y-rhs.y); }
        int operator*(point rhs) { return x*rhs.x + y*rhs.y; }
        int operator^(point rhs) { return x*rhs.y - y*rhs.x; }

        int norm2() { return *this * *this; }

        using tup = tuple<int, int>;

        bool operator<(const point& rhs) const {
                return tup{x, y} < tup{rhs.x, rhs.y};
        }

        bool operator==(const point& rhs) const {
                return tup{x, y} == tup{rhs.x, rhs.y};
        }
};

// angular comparison in [0, 2pi)
// smallest is (1, 0)
// CORNER: a || b == (0, 0)
bool ang_cmp(point a, point b) {
        auto quad = [](point p) -> bool {
                // 0 if ang in [0, pi), 1 if in [pi, 2pi)
                return p.y < 0 || (p.y == 0 && p.x < 0);
        };
        using tup = tuple<bool, int>;
        return tup{quad(a), 0} < tup{quad(b), a^b};
}

int dist2(point p, point q) { // squared distance
    return (p - q)*(p - q);
}

int area2(point a, point b, point c) { // two times signed area of triangle abc
        return (b - a) ^ (c - a);
}

bool left(point a, point b, point c) {
        return area2(a, b, c) > 0; // counterclockwise
}

bool right(point a, point b, point c) {
        return area2(a, b, c) < 0; // clockwise
}
```

```
bool collinear(point a, point b, point c) {
        return zero(area2(a,b,c));
}

// CORNER: a || b == (0, 0)
int parallel(point a, point b) {
        if((a ^ b) != 0) return 0;
        return (a.x>0) == (b.x>0) && (a.y > 0) == (b.y > 0) ? 1 : -1;
}

// CORNER: a == b
struct segment {
        point a, b;

        segment(point a=point(), point b=point()): a(a), b(b) {}

        point v() { return b - a; }
};

bool contains(segment r, point p) {
        return r.a==p || r.b==p || parallel(r.a-p,r.b-p) == -1;
}

bool intersects(segment r, segment s) {
        if(contains(r, s.a) || contains(r, s.b) || contains(s, r.a) || contains(s, r.b)) return 1;
        return left(r.a,r.b,s.a) != left(r.a,r.b,s.b) &&
                left(s.a, s.b, r.a) != left(s.a, s.b, r.b);
}

bool parallel(segment r, segment s) {
        return parallel(r.v(), s.v());
}
```

1.4 Nearest Points

```
// Returns minimum distance SQUARED between two points
// Complexity: O(NlogN)
// 719cd0
template <typename C_T>
C_T nearest_points(vector<point> v) {
        using lim = numeric_limits<C_T>;
        C_T res = lim::max(), sq = sqrt((double)res);
        sort(all(v));
        for(int i=1;i<v.size();i++) if(v[i] == v[i-1]) return 0;
        auto by_y = [](const point& a, const point& b) {
                using tup = tuple<C_T, C_T>;
                return tup{a.y, a.x} < tup{b.y, b.x};
        };
        queue<point> active;
        set<point, decltype(by_y)> pts(by_y);
        for(auto& p: v) {
                while(!active.empty() && p.x-active.front().x > sq) {
                        pts.erase(active.front());
                        active.pop();
                }
                auto it = pts.lower_bound({lim::min(), p.y-sq});
                while(it != pts.end() && it->y <= p.y + sq) {
                        C_T d = dist2(p, *it);
                        if(d < res) {
                                res = d;
                                sq = sqrt((double)res);
                        }
                        it++;
                }
                active.push(p);
                pts.insert(p);
        }
        return res;
}
```

1.5 Shamos Hoey

```
// NAO FUNCIONA BEM PARA DOUBLE
//
// MODIFICADO PARA SEGMENTOS VERTICAIS
// RAZOAVELMENTE TESTADO
//
// TOLERA INTERSECÇÕES NAS EXTREMIDADES DOS SEGMENTOS
// SEGMENTOS NÃO DEVEM SER DEGENERADOS
//
// Checa se existem segmentos que se intersectam
// Complexidade: O(N logN)
// 365cc1
template <typename C_T>
bool shamos_hoey(vector<segment> seg) {
        // create sweep segment events {x, type, seg_id}
        vector<tuple<C_T, bool, int>> ev;
        for(int i=0; i<seg.size(); i++) {
                if(seg[i].b < seg[i].a) swap(seg[i].a, seg[i].b);
                ev.emplace_back(seg[i].a.x, 0, i);
                ev.emplace_back(seg[i].b.x, 1, i);
        }
        sort(all(ev));
        auto cmp = [](segment r, segment s) -> bool {
                if(r.a == s.a) return left(r.a, r.b, s.b);
                bool b = s.a < r.a;
                if(b) swap(r, s);
                if(!zero(r.v().x)) return b^left(r.a, r.b, s.a);
                return b^(r.b.y < s.a.y);
        };
        set<segment, decltype(cmp)> s(cmp);
        for(auto [_, b, id]: ev) {
                segment at = seg[id];
                if(!b) {
                        auto nxt = s.lower_bound(at);
                        if((nxt != s.end() && intersects(*nxt, at))
                                || (nxt != s.begin() && intersects(*(--nxt), at)))
                                        return 1;
                        s.insert(at);
                } else {
                        s.erase(at);
                }
        }
        return 0;
}
```

2 Graphs

2.1 2-SAT

```
// 2-SAT
// Description: Tells if a system is 2-Satisfiable
// Complexity: O(|V| + |E|)
//
// Functions:
//     either (a, b) - (a | b) is true
//     implies (a, b) - (a -> b) is true
//     must (x) - x is true
//     solve () - returns true if the system is possible.
//                ans[] is the answer for each variable.
//
// Details:
//      Not x is equivalente to ~x on this template.
//      Did not test function atMostOne, but it add constraints
//      so that only one of these variables can be true.
// 46e497

struct SCC {
    int N, ti = 0; vector<vector<int>> adj;
    vector<int> disc, comp, st, comps;
    void init(int _N) {
        N = _N;
        adj.resize(N);
        disc.resize(N);
        comp = vector<int>(N,-1);
    }
    void add_edge(int x, int y) { adj[x].push_back(y); }
    int dfs(int x) {
        int low = disc[x] = ++ti; st.push_back(x); // disc[y] != 0 -> in stack
        for (auto y : adj[x]) if (comp[y] == -1) {
            auto b = disc[y] ? : dfs(y); auto &a = low;
            b < a ? a = b, 1 : 0;
        }
        if (low == disc[x]) { // make new SCC, pop off stack until you find x
            comps.push_back(x); for (int y = -1; y != x;)
                comp[y = st.back()] = x, st.pop_back();
        }
        return low;
    }
    void gen() {
        for (int i = 0; i < N; i++) if (!disc[i]) dfs(i);
        reverse(all(comps));
    }
};

struct TwoSAT {
    int N; SCC S; vector<bool> ans;
    void init(int _N) {
        N = _N;
        S.init(2*N);
        ans.resize(N);
    }
    int addVar() {
        return N++;
    }
    void either(int x, int y) {
        x = max(2 * x, -1 - 2 * x), y = max(2 * y, -1 - 2 * y);
        S.add_edge(x ^ 1, y); S.add_edge(y ^ 1, x);
    }
    void implies(int x, int y) {
        either(~x, y);
    }
    void must(int x) {
        either(x, x);
    }
    void atMostOne(const vector<int>& li) {
        if (li.size() <= 1) return;
        int cur = ~li[0];
        for (int i = 2; i < li.size(); i++) {
            int next = addVar();
            either(cur, ~li[i]); either(cur, next);
            either(~li[i], next); cur = ~next;
        }
        either(cur,~li[1]);
    }
    bool solve(int _N = -1) {
        if (_N != -1) N = _N, S.init(2*N);
        S.gen(); reverse(all(S.comps));
        for (int i = 0; i < 2 * N; i += 2)
            if (S.comp[i] == S.comp[i^1]) return 0;
        vector<int> tmp(2 * N); for (auto i : S.comps) if (!tmp[i])
            tmp[i] = 1, tmp[S.comp[i ^ 1]] = -1;
        for(int i = 0; i < N; i++) if (tmp[S.comp[2*i]] == 1) ans[i] = 1;
        return 1;
    }
};
```

2.2 Dinic

```
// add_edge(s, t, cap): Adds a directed edge from s to t with capacity cap
// max_flow(s, t): Returns max flow with source s and sink t
//
// Complexity: O(E*V^2). If unit edges only: O(E*sqrt(V))
// 04538b
constexpr int INF = numeric_limits<int>::max();
struct Dinic {
        struct edge {
                int to, cap, flow;
        };

        vector<vector<int>> g;
        vector<int> lvl;
        vector<edge> e;

        Dinic(int sz): g(sz), lvl(sz) {}

        void add_edge(int s, int t, int cap) {
                int id = e.size();
                g[s].push_back(id);
                e.push_back({t, cap, 0});
                g[t].push_back(++id);
                e.push_back({s, cap, cap});
        }

        bool BFS(int s, int t) {
                fill(all(lvl), INF);
                lvl[s] = 0;
                queue<int> q{{s}};
                while(!q.empty() && lvl[t] == INF) {
                        int cur = q.front();
                        q.pop();
                        for(int id: g[cur]) {
                                int prox = e[id].to;
                                if(lvl[prox] != INF || e[id].cap == e[id].flow)
                                        continue;
                                lvl[prox] = lvl[cur] + 1;
                                q.push(prox);
                        }
                }
                return lvl[t] != INF;
        }

        int DFS(int v, int pool, int start[], int t) {
                if(!pool) return 0;
                if(v == t) return pool;
                for(;start[v]<(int)g[v].size();start[v]++) {
                        int id = g[v][start[v]], prox = e[id].to;
                        if(lvl[v]+1 != lvl[prox] || e[id].cap == e[id].flow) continue;
                        int pushed = DFS(prox,min(e[id].cap-e[id].flow,pool),start,t);
                        if(pushed) {
                                e[id].flow += pushed;
                                e[id^1].flow -= pushed;
                                return pushed;
                        }
                }
                return 0;
        }

        int max_flow(int s, int t) {
                int total_flow = 0;
                vector<int> start(g.size());
                while(BFS(s,t)) {
                        fill(all(start), 0);
                        while(int pushed = DFS(s,INF,start.data(),t))
                                total_flow += pushed;
                }
                //reset to initial state
                //for(int i=0;i<e.size();i++) e[i].flow = (i&1) ? e[i].cap : 0;
                return total_flow;
        }
};
```

2.3 Euler Tour

```
/*
    Euler Tour
    Description: Find a path that passes through all edges
    Complexity: O(N + M)

    Details:
        It also works for directed graphs and it is supposed
        that the first vertex is 1.

    bb6db8
*/

template<int SZ, bool directed>
struct Euler {
    int N, M;
    vector< pair<int, int> > adj[SZ], circuit;
    int out[SZ], in[SZ], deg[SZ];
    bool used[SZ], bad;

    void clr() {
        for(int i = 0; i < N; i++) adj[i].clear();
        circuit.clear();
        for(int i = 0; i < N; i++) out[i] = in[i] = deg[i] = 0;
        for(int i = 0; i < M; i++) used[i] = 0;
        N = M = bad = 0;
    }

    void dfs(int pre, int cur) {
        while (adj[cur].size()) {
            pair<int, int> x = adj[cur].back(); adj[cur].pop_back();
            if (used[x.second]) continue;
            used[x.second] = 1; dfs(cur,x.first);
        }
        if (circuit.size() && circuit.back().first != cur) bad = 1;
        circuit.pb({pre,cur}); // generate circuit in reverse order
    }
    void addEdge(int a, int b) {
        if (directed) {
            adj[a].pb({b,M});
            out[a] ++, in[b] ++;
        } else {
            adj[a].pb({b,M}), adj[b].pb({a,M});
            deg[a] ++, deg[b] ++;
        }
        M ++;
    }

    vector<int> solve(int _N) {
        N = _N; // edges only involve vertices from 0 to N-1

        int start = 1;
        for(int i = 1; i <= N; i++) if (deg[i]%2 != 0) return {};
        dfs(-1,start);

        if (circuit.size() != M+1 || bad) return {}; // return empty if no sol
        vector<int> ans;
        for(int i = (int) circuit.size() - 1; i >= 0; i--) ans.pb(circuit[i].second);
        return ans;
    }
};
```

2.4 Hungaro

```
// Find max matching of min/max weight
// set(i, j, weight): add edge from left vertex i to right vertex j
// assign(): returns min/max weight max matching
// Change w_t for edge weight type
//
// Complexity: O(V^3)

constexpr int NONE = numeric_limits<int>::max();

using w_t = double;
constexpr w_t INF = 1e100;
bool zero(w_t x) { return abs(x) < 1e-9; }

// HASH FROM HERE
// e232a9
template <bool MAXIMIZE> struct Hungarian {
        int n, m;
        vector<vector<w_t>> w;
        vector<int> ml, mr; // ml: matched vertexes of left side
        vector<w_t> y, z, d;
        vector<bool> S, T;

        Hungarian(int n, int m): n(n), m(m), w(n, vector<w_t>(m, MAXIMIZE?-INF:INF)),
        ml(n), mr(m), y(n), z(m), d(m), S(n), T(m) {}

        void set(int i, int j, w_t weight) { w[i][j] = MAXIMIZE?weight:-weight; }

        w_t assign() {
                fill(all(ml), NONE); fill(all(mr), NONE);
                for(int i=0;i<n;i++) y[i] = *max_element(all(w[i]));
                fill(all(z), 0);
                for(int i=0;i<n;i++) for(int j=0;j<m;j++) {
                        if(mr[j] == NONE && zero(y[i]+z[j]-w[i][j])) {
                                ml[i] = j; mr[j] = i;
                                break;
                        }
```

```
		}
		auto kuhn = [&](int s, auto&& self) -> bool {
			if(S[s]) return false; S[s] = 1;
			for(int t=0;t<m;t++) if(!T[t]) {
				w_t diff = y[s]+z[t]-w[s][t];
				if(zero(diff)) {
					T[t] = 1;
					if(mr[t] == NONE || self(mr[t], self)) {
						mr[t] = s; ml[s] = t;
						return true;
					}
				} else d[t] = min(d[t], diff);
			}
			return false;
		};
		for(int i=0;i<n;i++) if(ml[i] == NONE) {
			fill(all(d), numeric_limits<w_t>::max());
			while(true) {
				fill(all(S), false); fill(all(T), false);
				if(kuhn(i,kuhn)) break;
				w_t delta = numeric_limits<w_t>::max();
				for(int j=0;j<m;j++) if(!T[j]) delta=min(delta, d[j]);
				for(int s=0;s<n;s++) if(S[s]) y[s] -= delta;
				for(int j=0;j<m;j++) {
					if(T[j]) z[j] += delta;
					else d[j] -= delta;
				}
			}
		}
		w_t res = 0;
		for(int i=0;i<n;i++) res += y[i];
		for(int j=0;j<m;j++) res += z[j];
		return MAXIMIZE?res:-res;
	}
};
```

2.5 LCA

```
// 4d58fb
struct LCA {
	vector<int> t;
	RMQ<pair<int,int>> rmq;

	LCA() {}
	LCA(int sz, vector<int> g[], int root): t(sz) {
		vector<pair<int,int>> tour; tour.reserve(2*sz-1);
		vector<int> prof(sz);
		auto dfs = [&](int v, int dad, auto& self) -> void {
			t[v] = tour.size();
			tour.push_back({prof[v],v});
			for(int p: g[v]) if(p != dad) {
				prof[p] = prof[v]+1;
				self(p,v,self);
				tour.push_back({prof[v],v});
			}
		};
		dfs(root, root, dfs);
		rmq = RMQ<pair<int,int>>(2*sz-1, tour.data());
	}

	int query(int a, int b) {
		if(t[a] > t[b]) swap(a,b);
		return rmq.query(t[a],t[b]).second;
	}
};
```

3 Math

3.1 Bit Hacks

```
// iterator through all masks with n bits and m set bits
// use: for(auto it: BitIterator(n,m) { int mask = *it; ... }
// e7a130
struct BitIterator {
        struct Mask {
                uint32_t mask;
                Mask(uint32_t mask): mask(mask) {}
                bool operator!=(const Mask& rhs) const { return mask < rhs.mask; };
                void operator++(){const uint32_t t=mask|(mask-1);mask=(t+1)|(((~t&-~t)-1)>>__builtin_ffs(ma
                uint32_t operator*() const { return mask; }
        };
        const uint32_t n, m;
        BitIterator(const uint32_t n, const uint32_t m): n(n), m(m) {}
        Mask begin() const { return Mask((1<<m)-1); }
        Mask end() const { return Mask((1<<n)); }
};
```

3.2 Convoluções

```
//      fft - Fast Fourier Transform
//      Description: Multiply two polinomial
//      Complexity: O(N logN)

//      Functions:
//          multiply(a, b)
//          multiply_mod(a, b, m) - return answer modulo m

//      Details:
//          For function multiply_mod, any modulo can be used.
//          It is implemented using the technique of dividing
//          in sqrt to use less fft. Function multiply may have
//          precision problems.
//          This code is faster than normal. So you may use it
//          if TL e tight.

// f79fe3
const double PI=acos(-1.0);
namespace fft {
    struct num {
        double x,y;
        num() {x = y = 0;}
        num(double x,double y): x(x), y(y){}
    };
    inline num operator+(num a, num b) {return num(a.x + b.x, a.y + b.y);}
    inline num operator-(num a, num b) {return num(a.x - b.x, a.y - b.y);}
    inline num operator*(num a, num b) {
        return num(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x);
    }
    inline num conj(num a) {return num(a.x, -a.y);}

    int base = 1;
    vector<num> roots={{0,0}, {1,0}};
    vector<ll> rev={0, 1};
    const double PI=acosl(-1.0);

    // always try to increase the base
    void ensure_base(int nbase) {
        if(nbase <= base) return;
        rev.resize(1 << nbase);
        for (int i = 0; i < (1 << nbase); i++)
            rev[i] = (rev[i>>1] >> 1) + ((i&1) << (nbase-1));
        roots.resize(1<<nbase);
        while(base<nbase) {
            double angle = 2*PI / (1<<(base+1));
            for(int i = 1<<(base-1); i < (1<<base); i++) {
                roots[i<<1] = roots[i];
                double angle_i = angle * (2*i+1-(1<<base));
                roots[(i<<1)+1] = num(cos(angle_i),sin(angle_i));
            }
            base++;
        }
    }

    void fft(vector<num> &a,int n=-1) {
        if(n==-1) n=a.size();
        assert((n&(n-1)) == 0);
        int zeros = __builtin_ctz(n);
        ensure_base(zeros);
        int shift = base - zeros;
        for (int i = 0; i < n; i++) {
            if(i < (rev[i] >> shift)) {
                swap(a[i],a[rev[i] >> shift]);
            }
        }
        for(int k = 1; k < n; k <<= 1) {
            for(int i = 0; i < n; i += 2*k) {
                for(int j = 0; j < k; j++) {
                    num z = a[i+j+k] * roots[j+k];
                    a[i+j+k] = a[i+j] - z;
                    a[i+j] = a[i+j] + z;
                }
            }
        }
    }

    vector<num> fa, fb;
    // multiply with less fft by using complex numbers.
    vector<ll> multiply(vector<ll> &a, vector<ll> &b);

    // using the technique of dividing in sqrt to use less fft.
    vector<ll> multiply_mod(vector<ll> &a, vector<ll> &b, ll m, ll eq=0);
    vector<ll> square_mod(vector<ll>&a, ll m);
};

// 16be45
vector<ll> fft::multiply(vector<ll> &a, vector<ll> &b) {
    int need = a.size() + b.size() - 1;

    int nbase = 0;
    while((1 << nbase) < need) nbase++;
    ensure_base(nbase);

    int sz = 1 << nbase;
    if(sz > (int)fa.size()) fa.resize(sz);
    for(int i = 0; i < sz; i++) {
        ll x = (i < (int)a.size() ? a[i] : 0);
        ll y = (i < (int)b.size() ? b[i] : 0);
        fa[i] = num(x, y);
    }

    fft(fa, sz);
    num r(0,-0.25/sz);
    for(int i = 0; i <= (sz>>1); i++) {
        int j = (sz-i) & (sz-1);
        num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
        if(i != j) fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
        fa[i] = z;
    }

    fft(fa, sz);
    vector<ll> res(need);
    for(int i = 0; i < need; i++) res[i] = fa[i].x + 0.5;
    return res;
}

// 4eb347
vector<ll> fft::multiply_mod(vector<ll> &a, vector<ll> &b, ll m, ll eq) {
    int need = a.size() + b.size() - 1;
    int nbase = 0;
    while((1 << nbase) < need) nbase++;
    ensure_base(nbase);
    int sz = 1 << nbase;
    if(sz > (int)fa.size()) fa.resize(sz);
    for(int i = 0; i < (int)a.size(); i++) {
        ll x = (a[i] % m + m) % m;
        fa[i] = num(x & ((1 << 15) - 1), x >> 15);
    }
    fill(fa.begin() + a.size(), fa.begin() + sz, num{0,0});
    fft(fa, sz);
    if(sz > (int)fb.size()) fb.resize(sz);
    if(eq) copy(fa.begin(), fa.begin() + sz, fb.begin());
    else {
        for(int i = 0; i < (int)b.size(); i++) {
            ll x = (b[i] % m + m) % m;
            fb[i] = num(x & ((1 << 15) - 1), x >> 15);
        }
        fill(fb.begin() + b.size(), fb.begin() + sz, num{0,0});
        fft(fb,sz);
    }
    double ratio = 0.25 / sz;
    num r2(0, -1), r3(ratio, 0), r4(0, -ratio), r5(0,1);
    for(int i = 0; i <= (sz>>1); i++) {
        int j = (sz - i) & (sz - 1);
        num a1 = (fa[i] + conj(fa[j]));
        num a2 = (fa[i] - conj(fa[j])) * r2;
        num b1 = (fb[i] + conj(fb[j])) * r3;
        num b2 = (fb[i] - conj(fb[j])) * r4;
        if(i != j) {
            num c1 = (fa[j] + conj(fa[i]));
            num c2 = (fa[j] - conj(fa[i])) * r2;
            num d1 = (fb[j] + conj(fb[i])) * r3;
            num d2 = (fb[j] - conj(fb[i])) * r4;
            fa[i] = c1 * d1 + c2 * d2 * r5;
            fb[i] = c1 * d2 + c2 * d1;
        }
        fa[j] = a1 * b1 + a2 * b2 * r5;
        fb[j] = a1 * b2 + a2 * b1;
    }
    fft(fa, sz); fft(fb, sz);
    vector<ll> res(need);
    for(int i = 0; i < need; i++) {
        ll aa = fa[i].x + 0.5;
        ll bb = fb[i].x + 0.5;
        ll cc = fa[i].y + 0.5;
        res[i] = (aa + ((bb%m) << 15) + ((cc%m) << 30))%m;
    }
    return res;
}

vector<ll> fft::square_mod(vector<ll> &a, ll m) {
    return multiply_mod(a, a, m, 1);
```

3.3 NTT

```
/*
    NTT - Number Theoretic Transform
    Description: Multiply two polinomials in Z_p, for p prime
    Complexity: O(N logN)

    Functions:
        multiply(a, b)

    Details:
        Not all primes can be used and p = 998244353 is the most used prime.
        To multiply it for a general modulus, use 3 different possible primes
        and use Chinese Remainder Theorem to get the answear.

    Possibilities
    { 7340033, 5, 4404020, 1 << 20 },
    { 415236097, 73362476, 247718523, 1 << 22 },
    { 463470593, 428228038, 182429, 1 << 21},
    { 998244353, 15311432, 469870224, 1 << 23 },
    { 918552577, 86995699, 324602258, 1 << 22 }

    98bcfc
*/

namespace NTT {
    const ll mod = 998244353, root = 15311432, root_1 = 469870224, root_pw = 1 << 23;

    ll fastxp(ll n, ll e){
        ll ans = 1, pwr = n;
        while(e){
            if(e%2)  ans = ans * pwr % mod;
            e /= 2;
            pwr = pwr * pwr % mod;
        }
        return ans % mod;
    }


    void fft(vector<ll> & a, bool invert) {
        ll n = a.size();

        for (ll i = 1, j = 0; i < n; i++) {
            ll bit = n >> 1;
            for (; j & bit; bit >>= 1)
                j ^= bit;
            j ^= bit;

            if (i < j) swap(a[i], a[j]);
        }

        for (ll len = 2; len <= n; len <<= 1) {
            ll wlen = invert ? root_1 : root;
            for (ll i = len; i < root_pw; i <<= 1)
                wlen = wlen * wlen % mod;

            for (ll i = 0; i < n; i += len) {
                ll w = 1;
                for (ll j = 0; j < len / 2; j++) {
                    ll u = a[i+j], v = a[i+j+len/2] * w % mod;
                    a[i+j] = u + v < mod ? u + v : u + v - mod;
                    a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
                    w = w * wlen % mod;
                }
            }
        }

        if (invert) {
            ll n_1 = fastxp(n, mod - 2);
            for (ll & x : a) x = x * n_1 % mod;
        }
    }

    vector<ll> multiply(vector<ll> &a, vector<ll> &b) {
        vector<ll> fa(a.begin(), a.end()), fb(b.begin(), b.end());
        ll sz = a.size() + b.size() - 1, n = 1;
        while (n < sz) n <<= 1;

        fa.resize(n), fb.resize(n);
        fft(fa, 0), fft(fb, 0);
        for (ll i = 0; i < fa.size(); i++) fa[i] = fa[i] * fb[i] % mod;

        fft(fa, 1);
        fa.resize(sz);
        return fa;
    }
```

3.4 Coprimes/Mobius

```
/*
    Coprimes
    Description:
        Given a set o integers, calculates the quantity of integers
        in the set coprimes with x.
    Complexity:
        precalc - O(n logn)
        add - O(sigma(N))
        coprime - O(sigma(N))
    Details:
        It uses Mobius Function. To add or remove an integer of the set
        just change sign to +1 or -1.
    e9bba1
*/
struct Coprimes {
    int n;
    vector<ll> U, cnt;
    vector<vector<int>> fat;
    Coprimes () {}
    Coprimes (int n) : n(n), U(n), fat(n), cnt(n) {
        precalc ();
    }
    void precalc () {
        for (int i = 1; i < n; i++) {
            for (int j = i; j < n; j += i) fat[j].pb(i);
            if (i == 1) U[i] = 1;
            else if ((i / fat[i][1]) % fat[i][1] == 0) U[i] = 0;
            else U[i] = -U[i / fat[i][1]];
        }
    }
    void add(int x, int sign){
        for(auto d : fat[x]) cnt[d] += sign;
    }
    int coprimo(int x){
        int quant = 0;
        for(auto d : fat[x]){
            quant += U[d] * cnt[d];
        }
        return quant;
    }
```

3.5 MDC extendido

```
// returns X, Y such that a*X + b*Y = gcd(a, b)
pair<int,int> egcd(int a, int b) {
        if(b == 0) return {1, 0};
        auto [x, y] = egcd(b, a%b);
        return {y, x - y * (a/b)};
}
```

3.6 Base Gauss

```
/*
    Gauss_xor - Gauss elimination mod 2
    Description: Given a set of Vectors of size D,
                 maintains a basis of the set.
    Complexity: check - O(D)
                add - O(D)

    Functions:
        check(mask) - returns true if the mask can be represented
                      by the basis;
        add(mask) - adds mask to the basis if it is independent.

    Details:
        We are assuming the vectors have size D <= 64. For general
        case, you may change ll basis[] for bitset<D> basis[].
    6e102d
*/

const int logN = 30;

struct Gauss_xor {
    ll sz, basis[logN];

    Gauss_xor () {
        sz = 0;
        for(int i = 0; i < logN; i++)
            basis[i] = 0;
    }

    bool check (ll mask) {
        for(int i = 0; i < logN; i++) {
            if(((1ll << i) & mask) == 0) continue;
            if(!basis[i]) return false;
            mask ^= basis[i];
        }
        return true;
    }

    void add (ll mask) {
        for (int i = 0; i < logN; i++) {
            if(!((1ll << i) & mask)) continue;
            if(!basis[i]) {
                sz++;
                basis[i] = mask;
                return;
            }
            mask ^= basis[i];
        }
    }
```

3.7 Gauss Elimination

```
/*
    Gauss elimination - modulo 2

    Description:
        Solves a linear system with n equations and m - 1 variables.
        Is faster duo to the use of bitset.
    Complexity: O(n^2 * m / 32)


    Details:
        Function solve return a boolean indicating if system is possible
        or not. Also, if it is possible, the parameter maintains the answer.
*/
// f27955
struct Gauss_mod2 {
    int n, m;
    array<int, M> pos;
    int rank = 0;
    vector<bitset<M>> a;

    // n equations, m-1 variables, last column is for coefficients
    Gauss_mod2(int n, int m, vector<bitset<M>> &a): n(n), m(m), a(a) {
        pos.fill(-1);
    }

    int solve(bitset<M> &ans) {
        for (int col = 0, row = 0; col < m && row < n; col++) {
            int one = -1;
            for (int i = row; i < n; i++) {
                if (a[i][col]) {
                    one = i;
                    break;
                }
            }

            if (one == -1) { continue; }

            swap(a[one], a[row]);

            pos[col] = row;

            for (int i = row + 1; i < n; i++) {
                if (a[i][col])
                    a[i] ^= a[row];
            }
            ++row, ++rank;
        }

        ans.reset();

        for (int i = m - 1; i >= 0; i--) {
            if (pos[i] == -1) ans[i] = true;
            else {
                int k = pos[i];
                for (int j = i + 1; j < m; j++) if (a[k][j]) ans[i] = ans[i] ^ ans[j];
                ans[i] = ans[i] ^ a[k][m];
            }
        }

        for (int i = rank; i < n; i++) if (a[i][m]) return 0;

        return 1;
    }
};
```

4 Strings

4.1 Eertree

```
/*
    Err Tree - Palindromic Tree

    Description: A tree such that each node represents a
                 palindrome of string s. It is possible to append
                 a character.
    Complexity: Amortized O(|s|)
*/

// 0d1a1c
struct palindromic_tree {
    struct node {
        int length, link;
        map<char, int> to;
        node(int length, int link): length(length), link(link) {}
    };
    vector<node> nodes;
    int current;
    palindromic_tree(): current(1) {
        nodes.push_back(node(-1, 0));
        nodes.push_back(node(0, 0));
    }
    void add(int i, string& s) {
        int parent = nodes[current].length == i ? nodes[current].link : current;
        while (s[i - nodes[parent].length - 1] != s[i])
            parent = nodes[parent].link;
        if (nodes[parent].to.find(s[i]) != nodes[parent].to.end()) {
            current = nodes[parent].to[s[i]];
        } else {
            int link = nodes[parent].link;
            while (s[i - nodes[link].length - 1] != s[i])
                link = nodes[link].link;
            link = max(1, nodes[link].to[s[i]]);
            current = nodes[parent].to[s[i]] = nodes.size();
            nodes.push_back(node(nodes[parent].length + 2, link));
        }
    }
    void insert(string& s) {
        current = 1;
        for (int i = 0; i < int(s.size()); i++)
            add(i, s);
    }
};
```

4.2 Hash

```
/*
    String Hashing
    Description: A data structure that transforms a string into a number

    Functions:
        str_hash - Builds the hash in O(|S|)
        operator() - Gives the number representing substring s[l,r] in O(1)

    Details:
            - To use more than one prime, you may use long long, __int128 or array<int>
            - You may easily change it to handle vector<int> instead of string
            - Other large primes: 1000041323, 100663319, 201326611
            - If smaller primes are needed(For instance, need to store the mods in an array):
                - 50331653, 12582917, 6291469, 3145739, 1572869
*/

const long long mod1 = 1000015553, mod2 = 1000028537;

mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count()); // random number generator

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

// e50d39
template<int MOD>
struct str_hash {
    static int P;
    vector<ll> h, p;
    str_hash () {}
    str_hash(string s) : h(s.size()), p(s.size()) {
        p[0] = 1, h[0] = s[0];
        for (int i = 1; i < s.size(); i++)
            p[i] = p[i - 1]*P%MOD, h[i] = (h[i - 1]*P + s[i])%MOD;
    }
    ll operator()(int l, int r) { // retorna hash s[l...r]
        ll hash = h[r] - (l ? h[l - 1]*p[r - l + 1]%MOD : 0);
        return hash < 0 ? hash + MOD : hash;
    }
};
template<int MOD> int str_hash<MOD>::P = uniform(256, MOD - 1); // l > |sigma|

// b31197
struct Hash {
    // Uses 2 primes to better avoid colisions
    str_hash<mod1> H1;
    str_hash<mod2> H2;

    Hash (string s) : H1(str_hash<mod1>(s)), H2(str_hash<mod2>(s)) {}

    ll operator()(int l, int r) {
        ll ret1 = H1(l, r), ret2 = H2(l, r);
        return (ret1 << 30) ^ (ret2);
    }
```

4.3 KMP

```
/*
        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|
*/

// b29648
template <typename T> vector<int> kmp(int sz, const T s[]) {
        vector<int> pi(sz);
        for(int i=1;i<sz;i++) {
                int &j = pi[i];
                for(j=pi[i-1];j>0 && s[i]!=s[j];j=pi[j-1]);
                if(s[i] == s[j]) j++;
        }
        return pi;
};

// c82524
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;
}

// 79bd9e
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]];
        }
```

4.4 Suffix Array

```
/*
        Suffix Array

        Description: Algorithm that sorts the suffixes of a string
        Complexity: O(|s| log(|s|))

    Observation: There is a code below from KTH that is twice as faster.
*/

// c26d8e
struct suffixArray {
    string s;
    vector<int> sa, lcp;
    int n;
    void cSort(int k, vector<int>& ra) {
        int maxi = max(300ll, n);
        vector<int> c(maxi, 0), temp_sa(n);
        for(int i = 0; i < n; i++)
            c[i + k < n ? ra[i + k] : 0]++;
        for(int i = 1; i < maxi; i++)
            c[i] += c[i - 1];
        for(int i = n - 1; i >= 0; i--)
            temp_sa[--c[sa[i] + k < n ? ra[sa[i] + k] : 0]] = sa[i];
        sa.swap(temp_sa);
    }
    suffixArray() {}
    suffixArray(vector<string>& v) {
        for(auto str: v) {
            s += str;
            s += '$';
        }
        n = s.size();
        sa.resize(n);
        vector<int> ra(n), temp_ra(n);
        for(int i = 0; i < n; i++) {
            ra[i] = s[i];
            sa[i] = i;
        }
        for(int k = 1; k < n; k <<= 1) {
            cSort(k, ra);
            cSort(0, ra);
            int r = temp_ra[sa[0]] = 0;
            for(int i = 1; i < n; i++)
                temp_ra[sa[i]] = (ra[sa[i]] == ra[sa[i - 1]]
                                    && ra[sa[i] + k] == ra[sa[i - 1] + k]) ? r : ++r;
            ra.swap(temp_ra);
            if(r == n - 1) break;
        }
    }
    void computeLcp() {
        vector<int> phi(n);
        int k = 0;
        phi[sa[0]] = -1;
        for(int i = 1; i < n; i++)
            phi[sa[i]] = sa[i-1];
        vector<int> plcp(n);
        for(int i = 0; i < n - 1; i++) {
            if(phi[i] == -1) {
                plcp[i] = 0;
                continue;
            }
            while(s[i + k] == s[phi[i] + k]) k++;
            plcp[i] = k;
            k = max(k - 1, 0ll);
        }
        lcp.resize(n);
        for(int i = 0; i < n; i++)
            lcp[i] = plcp[sa[i]];
    }
};

// Suffix Array da KTH
// 528705
struct SuffixArray {
    string s;
    vector<int> sa, lcp;
    SuffixArray () {}
    SuffixArray(vector<string>& v, int lim=256) { // or basic_string<int>
        for(auto str : v) {
            s += str;
            s += '$';
        }
        int n = s.size(), k = 0, a, b;
        vector<int> x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y, iota(all(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j, iota(all(y), n - j);
            for(int i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            for(int i = 0; i < n; i++) ws[x[i]]++;
            for(int i = 1; i < lim; i++) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            for(int i = 1; i < n; i++) a = sa[i - 1], b = sa[i], x[b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
        }
        for(int i = 1; i < n; i++) rank[sa[i]] = i;
        for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
            for (k && k--, j = sa[rank[i] - 1];
                    s[i + k] == s[j + k]; k++);
    }
```

4.5 Z

```
// ad34f9
template <typename T> vector<int> z_alg(int sz, const T s[]) {
        vector<int> ret(sz);
        for(int l=0,r=0,i=1;i<sz;i++) {
                auto expand = [&]() {
                        while(r<sz && s[r-l]==s[r]) r++;
                        ret[i] = r-l;
                };
                if(i >= r) {
                        l=r=i;
                        expand();
                } else {
                        if(ret[i-l] < r-i) ret[i] = ret[i-l];
                        else {
                                l=i;
                                expand();
                        }
                }
        }
        return ret;
};
```

5 DP

5.1 CHT dinamico

```
// CHT - Dynamic Convex Hull Trick
// Description: Maintain the convex hull of some functions
// Complexity:
//     add - O(logN)
//     query - O(logN)

// Functions:
//     add(a, b) - add line (a * x + b) to the convex hull.
//     query (x) - return the maximum value of any line on point x.

// Details:
//     If you want to maintain the bottom convex hull, it is
//     easier to just change the sign. Be careful with overflow
//     on query. Can use __int128 to avoid.
// 978376

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;
    }
```

6 Data Structures

6.1 Implicit Lazy Treap

```
// All operations are O(log N)
// If changes need to be made in lazy propagation,
// see Treap::reverse() and change Treap::no::prop()
//
// Important functions:
// Treap::insert(T val, int idx)
// Treap::erase(int idx)
// Treap::reverse(int l, int r)
// Treap::operator[](int idx)

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

// HASH FROM HERE:
// c018fe
template <typename T>
struct Treap {
        struct no {
                array<no*, 2> c;
                T dat;
                int cnt, h;

                // Example: reverse interval
                bool rev;

                no(T dat=T()): c({0, 0}), dat(dat), cnt(1), h(rng()), rev(0) {}

                // propagate
                void prop() {
                        if(rev) {
                                swap(c[0], c[1]);
                                for(no* x: c) if(x) x->rev ^= !x->rev;
                                rev = 0;
                        }
                }

                // refresh
                no* ref() {
                        cnt = 1;
                        for(no* x: c) if(x) {
                                x->prop();
                                cnt += x->cnt;
                        }
                        return this;
                }

                // left child size
                int l() {
                        return c[0] ? c[0]->cnt : 0;
                }
        };

        int sz;
        no *root;
        unique_ptr<no[]> arena;

        // prealloc: number of new_no() calls that will be made in total
        Treap(int prealloc): sz(0), root(0), arena(new no[prealloc]) {}

        no* new_no(T dat) {
                arena[sz] = no(dat);
                return &arena[sz++];
        }

        int cnt(no* x) { return x ? x->cnt : 0; }

        void merge(array<no*, 2> c, no*& res) {
                if(!c[0] || !c[1]) {
                        res = c[0] ? c[0] : c[1];
                        return;
                }
                for(no* x: c) x->prop();
                int i = c[0]->h < c[1]->h;
                no *l = c[i]->c[!i], *r = c[!i];
                if(i) swap(l, r);
                merge({l, r}, c[i]->c[!i]);
                res = c[i]->ref();
        }

        // left treap has size pos
        void split(no* x, int pos, array<no*, 2>& res, int ra = 0) {
                if(!x) {
                        res.fill(0);
                        return;
                }
                x->prop();
                ra += x->l();
                int i = pos > ra;
                split(x->c[i], pos, res, ra+(i?1:-x->l()));
                x->c[i] = res[!i];
                res[!i] = x->ref();
        }

        // Merges all s and makes them root
        template <int SZ>
```

```
        void merge(array<no*, SZ> s) {
                root = s[0];
                for(int i=1;i<SZ;i++)
                        merge({root, s[i]}, root);
        }

        // Splits root into SZ EXCLUSIVE intervals
        // [0..s[0]), [s[0]..s[1]), [s[1]..s[2])... [s[SZ-1]..end)
        // Example: split<2>({l, r}) gets the exclusive interval [l, r)
        template <int SZ>
        array<no*, SZ> split(array<int, SZ-1> s) {
                array<no*, SZ> res;
                array<no*, 2> aux;
                split(root, s[0], aux);
                res[0] = aux[0]; res[1] = aux[1];
                for(int i=1;i<SZ-1;i++) {
                        split(res[i], s[i]-s[i-1], aux);
                        res[i] = aux[0]; res[i+1] = aux[1];
                }
                root = nullptr;
                return res;
        }

        void insert(T val, int idx) {
                auto s = split<2>({idx});
                merge<3>({s[0], new_no(val), s[1]});
        }

        void erase(int idx) {
                auto s = split<3>({idx, idx+1});
                merge<2>({s[0], s[2]});
        }

        // Inclusive
        void reverse(int l, int r) {
                auto s = split<3>({l, r+1});
                s[1]->rev = !s[1]->rev;
                merge<3>(s);
        }

        T operator[](int idx) {
                no* x = root;
                //assert(0 <= idx && idx < x->cnt);
                x->prop();
                for(int ra = x->l(); ra != idx; ra += x->l()) {
                        if(ra < idx) ra++, x = x->c[1];
                        else ra -= x->l(), x = x->c[0];
                        x->prop();
                }
                return x->dat;
        }
};
```

6.2 MO

```
/*
    MO
    Description:
        Answers queries offline with sqrt decomposition.
    Complexity:
        exec - O(n*sqrt(n)*O(remove / add))
    ed9fec
*/
const int magic = 230;
```

```
struct Query {
    int l, r, idx;
    Query () {}
    Query (int _l, int _r, int _idx) : l(_l), r(_r), idx(_idx) {}
    bool operator < (const Query &o) const {
        return mp(l / magic, r) < mp(o.l / magic, o.r);
    }
};

struct MO {
    int sum;
    MO(vector<ll> &v) : sum(0), v(v), cnt(N), C(N) {}

    void exec(vector<Query> &queries, vector<ll> &answers) {
        answers.resize(queries.size());
        sort(queries.begin(), queries.end());

        int cur_l = 0;
        int cur_r = -1;

        for (Query q : queries) {
            while (cur_l > q.l) {
                cur_l--;
                add(cur_l);
            }
            while (cur_r < q.r) {
                cur_r++;
                add(cur_r);
            }
            while (cur_l < q.l) {
                remove(cur_l);
                cur_l++;
            }
            while (cur_r > q.r) {
                remove(cur_r);
                cur_r--;
            }
            answers[q.idx] = get_answer(cur_l, cur_r);
        }
    }

    void add(int i) {
        sum += v[i];
    }

    void remove(int i) {
        sum -= v[i];
    }

    ll get_answer(int l, int r) {
        return sum;
    }
};
```

6.3 NCE

```
// For each i, find nearest l < i < r such that
// op(l, i), op(r, i) = true if they exist
// l = -1, r = v.size() otherwise
//
// Example: nce(v, greater<T>()): for each i returns
// nce[i] = {
//  biggest l < i such that v[l] > v[i]
//  smallest r > i such that v[r] > v[i]
// }
//
// Complexity: O(N)
// 1e83ae
template <typename T, typename OP>
vector<pair<int, int>> nce(vector<T> v, OP op) {
        int n = v.size();
        vector<pair<int, int>> res(n);
        vector<pair<T, int>> st;
        for(int i=0;i<n;i++) {
                while(!st.empty() && !op(st.back().first, v[i]))
                        st.pop_back();
                if(st.empty()) res[i].first = -1;
                else res[i].first = st.back().second;
                st.emplace_back(v[i], i);
        }
        st.clear();
        for(int i=n-1;i>=0;i--) {
                while(!st.empty() && !op(st.back().first, v[i]))
                        st.pop_back();
                if(st.empty()) res[i].second = n;
                else res[i].second = st.back().second;
                st.emplace_back(v[i], i);
        }
        return res;
}
```

6.4 Sparse Table

```
// c2e4d3
template <typename T> struct RMQ {
        vector<vector<T>> dp;
        T ops(T a, T b) { return min(a,b); }
        RMQ() {}
        RMQ(int sz, T v[]) {
                int log = 64-__builtin_clzll(sz);
                dp.assign(log, vector<T>(sz));
                for(int i=0;i<sz;i++) dp[0][i] = v[i];
                for(int l=1;l<log;l++) for(int i=0;i<sz;i++)
                        dp[l][i] = ops(dp[l-1][i],dp[l-1][min(i+(1<<(l-1)), sz-1)]);
        }
        T query(int a, int b)  {
                if(a == b) return dp[0][a];
                int pot = 63-__builtin_clzll(b-a);
                return ops(dp[pot][a], dp[pot][b-(1<<pot)+1]);
        }
};
```

6.5 Search Buckets

```
/*
    search_buckets:
    Data structure that provides two operations on an array:
    1) set array[i] = x
    2) count how many i in [start, end) satisfy array[i] < value
    Both operations take sqrt(N log N) time. Amazingly, because of
    the cache efficiency this is faster than the(log N)^2 algorithm
    until N = 2-5 million.

    ec7ab8
*/
template<typename T>
struct search_buckets {
```

```
// values are just the values in order. buckets are sorted in segments of BUCKET_SIZE (last segment may
int N, BUCKET_SIZE;
vector<T> values, buckets;

search_buckets(const vector<T> &initial = {}) {
    init(initial);
}

int get_bucket_end(int bucket_start) const {
    return min(bucket_start + BUCKET_SIZE, N);
}

void init(const vector<T> &initial) {
    values = buckets = initial;
    N = values.size();
    BUCKET_SIZE = 3 * sqrt(N * log(N + 1)) + 1;
    cerr << "Bucket size: " << BUCKET_SIZE << endl;

    for (int start = 0; start < N; start += BUCKET_SIZE)
        sort(buckets.begin() + start, buckets.begin() + get_bucket_end(start));
}

int bucket_less_than(int bucket_start, T value) const {
    auto begin = buckets.begin() + bucket_start;
    auto end = buckets.begin() + get_bucket_end(bucket_start);
    return lower_bound(begin, end, value) - begin;
}

int less_than(int start, int end, T value) const {
    int count = 0;
    int bucket_start = start - start % BUCKET_SIZE;
    int bucket_end = min(get_bucket_end(bucket_start), end);

    if (start - bucket_start < bucket_end - start) {
        while (start > bucket_start)
            count -= values[--start] < value;
    } else {
        while (start < bucket_end)
            count += values[start++] < value;
    }

    if (start == end)
        return count;

    bucket_start = end - end % BUCKET_SIZE;
    bucket_end = get_bucket_end(bucket_start);

    if (end - bucket_start < bucket_end - end) {
        while (end > bucket_start)
            count += values[--end] < value;
    } else {
        while (end < bucket_end)
            count -= values[end++] < value;
    }

    while (start < end && get_bucket_end(start) <= end) {
        count += bucket_less_than(start, value);
        start = get_bucket_end(start);
    }

    assert(start == end);
    return count;
}
```

```
    int prefix_less_than(int n, T value) const {
        return less_than(0, n, value);
    }

    void modify(int index, T value) {
        int bucket_start = index - index % BUCKET_SIZE;
        int old_pos = bucket_start + bucket_less_than(bucket_start, values[index]);
        int new_pos = bucket_start + bucket_less_than(bucket_start, value);

        if (old_pos < new_pos) {
            copy(buckets.begin() + old_pos + 1, buckets.begin() + new_pos, buckets.begin() + old_pos);
            new_pos--;
            // memmove(&buckets[old_pos], &buckets[old_pos + 1], (new_pos - old_pos) * sizeof(T));
        } else {
            copy_backward(buckets.begin() + new_pos, buckets.begin() + old_pos, buckets.begin() + old_pos +
            // memmove(&buckets[new_pos + 1], &buckets[new_pos], (old_pos - new_pos) * sizeof(T));
        }

        buckets[new_pos] = value;
        values[index] = value;
    }
```

6.6 Seg 2D

```
/*
    Segtree 2D:
    Data structure that makes operation on a grid.

    Complexity:
        build - O(N)
        query - O(logN^2)

    9a865d
*/
struct Node{
    Node(){}
    Node operator +(const Node &o) const{
        return Node ();
    }
};

int n,m;
Node a[MAXN][MAXN], st[2*MAXN][2*MAXN];

Node op(Node a, Node b){
    return a + b;
}

void build(){
    for(int i = 0; i < n; i++) for(int j = 0; j < m; j++)st[i+n][j+m]=a[i][j];
    for(int i = 0; i < n; i++) for(int j=m-1;j;--j)
        st[i+n][j]=op(st[i+n][j<<1],st[i+n][j<<1|1]);
    for(int i=n-1;i;--i) for(int j = 0; j < 2 * m; j++)
        st[i][j]=op(st[i<<1][j],st[i<<1|1][j]);
}
void upd(int x, int y, Node v){
    st[x+n][y+m]=v;
    for(int j=y+m;j>1;j>>=1)st[x+n][j>>1]=op(st[x+n][j],st[x+n][j^1]);
    for(int i=x+n;i>1;i>>=1)for(int j=y+m;j;j>>=1)
        st[i>>1][j]=op(st[i][j],st[i^1][j]);
}

// essa query vai de x0, y0 ate x1 - 1, y1 - 1 !!!
Node query(int x0, int x1, int y0, int y1){
    Node r = Node(infinite, 0, 0); // definir elemento neutro da query!!!
    for(int i0=x0+n,i1=x1+n;i0<i1;i0>>=1,i1>>=1){
        int t[4],q=0;
        if(i0&1)t[q++]=i0++;
        if(i1&1)t[q++]=--i1;
        for(int k = 0; k < q; k++) for(int j0=y0+m,j1=y1+m;j0<j1;j0>>=1,j1>>=1){
            if(j0&1)r=op(r,st[t[k]][j0++]);
            if(j1&1)r=op(r,st[t[k]][--j1]);
        }
    }
    return r;
}
```