Notebook - Maratona de Programação

Posso mandar um WA?

Contents

1 Geometria

1.1 Linear Transformation

```cpp
// Apply linear transformation (p -> q) to r.
point linear_transformation(point p0, point p1, point q0, point q1, point r) {
    point dp = p1-p0, dq = q1-q0, num((dp^dq), (dp^dq));
    return q0 + point((r-p0)^(num), (r-p0)*(num))/(dp*dp);
}
```

1.2 Inside Polygon

```cpp
// Convex O(logn)

bool insideT(point a, point b, point c, point e){
    int x = ccw(a, b, e);
    int y = ccw(b, c, e);
    int z = ccw(c, a, e);
    return !((x==1 or y==1 or z==1) and (x==-1 or y==-1 or z==-1));
}

bool inside(vp &p, point e){ // ccw
    int l=2, r=(int)p.size()-1;
    while(l<r){
        int mid = (l+r)/2;
        if(ccw(p[0], p[mid], e) == 1)
            l=mid+1;
        else{
            r=mid;
        }
    }
    // bordo
    // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)==0) return false;
    // if(r==2 and ccw(p[0], p[1], e)==0) return false;
    // if(ccw(p[r], p[r-1], e)==0) return false;
    return insideT(p[0], p[r-1], p[r], e);
}


// Any O(n)

int inside(vp &p, point pp){
    // 1 - inside / 0 - boundary / -1 - outside
    int n = p.size();
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(line({p[i], p[j]}).inside_seg(pp))
            return 0;
    }
    int inter = 0;
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p[i], p[j], pp)==1)
            inter++; // up
        else if(p[j].x <= pp.x and pp.x < p[i].x and ccw(p[i], p[j], pp)==-1)
            inter++; // down
    }

    if(inter%2==0) return -1; // outside
    else return 1; // inside
}
```

1.3 Delaunay

```cpp
typedef long long ll;

bool ge(const ll& a, const ll& b) { return a >= b; }
bool le(const ll& a, const ll& b) { return a <= b; }
bool eq(const ll& a, const ll& b) { return a == b; }
bool gt(const ll& a, const ll& b) { return a > b; }
bool lt(const ll& a, const ll& b) { return a < b; }
int sgn(const ll& a) { return a >= 0 ? a ? 1 : 0 : -1; }

struct pt {
    ll x, y;
    pt() { }
    pt(ll _x, ll _y) : x(_x), y(_y) { }
    pt operator-(const pt& p) const {
        return pt(x - p.x, y - p.y);
    }
    ll cross(const pt& p) const {
        return x * p.y - y * p.x;
    }
    ll cross(const pt& a, const pt& b) const {
        return (a - *this).cross(b - *this);
    }
    ll dot(const pt& p) const {
        return x * p.x + y * p.y;
    }
    ll dot(const pt& a, const pt& b) const {
        return (a - *this).dot(b - *this);
    }
    ll sqrLength() const {
        return this->dot(*this);
    }
    bool operator==(const pt& p) const {
        return eq(x, p.x) && eq(y, p.y);
    }
};

const pt inf_pt = pt(1e18, 1e18);

struct QuadEdge {
    pt origin;
    QuadEdge* rot = nullptr;
    QuadEdge* onext = nullptr;
    bool used = false;
    QuadEdge* rev() const {
        return rot->rot;
    }
    QuadEdge* lnext() const {
        return rot->rev()->onext->rot;
    }
    QuadEdge* oprev() const {
        return rot->onext->rot;
    }
    pt dest() const {
        return rev()->origin;
    }
};

QuadEdge* make_edge(pt from, pt to) {
    QuadEdge* e1 = new QuadEdge;
    QuadEdge* e2 = new QuadEdge;
    QuadEdge* e3 = new QuadEdge;
    QuadEdge* e4 = new QuadEdge;
    e1->origin = from;
    e2->origin = to;
    e3->origin = e4->origin = inf_pt;
    e1->rot = e3;
    e2->rot = e4;
    e3->rot = e2;
    e4->rot = e1;
    e1->onext = e1;
    e2->onext = e2;
    e3->onext = e4;
    e4->onext = e3;
    return e1;
}

void splice(QuadEdge* a, QuadEdge* b) {
    swap(a->onext->rot->onext, b->onext->rot->onext);
    swap(a->onext, b->onext);
}

void delete_edge(QuadEdge* e) {
    splice(e, e->oprev());
    splice(e->rev(), e->rev()->oprev());
    delete e->rev()->rot;
    delete e->rev();
    delete e->rot;
    delete e;
}

QuadEdge* connect(QuadEdge* a, QuadEdge* b) {
    QuadEdge* e = make_edge(a->dest(), b->origin);
    splice(e, a->lnext());
    splice(e->rev(), b);
    return e;
}

bool left_of(pt p, QuadEdge* e) {
    return gt(p.cross(e->origin, e->dest()), 0);
}

bool right_of(pt p, QuadEdge* e) {
    return lt(p.cross(e->origin, e->dest()), 0);
}

template <class T>
T det3(T a1, T a2, T a3, T b1, T b2, T b3, T c1, T c2, T c3) {
    return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) +
           a3 * (b1 * c2 - c1 * b2);
}

bool in_circle(pt a, pt b, pt c, pt d) {
// If there is __int128, calculate directly.
// Otherwise, calculate angles.
#if defined(__LP64__) || defined(_WIN64)
    __int128 det = -det3<__int128>(b.x, b.y, b.sqrLength(), c.x, c.y,
                                   c.sqrLength(), d.x, d.y, d.sqrLength());
    det += det3<__int128>(a.x, a.y, a.sqrLength(), c.x, c.y, c.sqrLength(), d.x,
                          d.y, d.sqrLength());
    det -= det3<__int128>(a.x, a.y, a.sqrLength(), b.x, b.y, b.sqrLength(), d.x,
                          d.y, d.sqrLength());
    det += det3<__int128>(a.x, a.y, a.sqrLength(), b.x, b.y, b.sqrLength(), c.x,
                          c.y, c.sqrLength());
    return det > 0;
#else
    auto ang = [](pt l, pt mid, pt r) {
        ll x = mid.dot(l, r);
        ll y = mid.cross(l, r);
        long double res = atan2((long double)x, (long double)y);
        return res;
    };
    long double kek = ang(a, b, c) + ang(c, d, a) - ang(b, c, d) - ang(d, a, b);
    if (kek > 1e-8)
        return true;
    else
        return false;
#endif
}

pair<QuadEdge*, QuadEdge*> build_tr(int l, int r, vector<pt>& p) {
    if (r - l + 1 == 2) {
        QuadEdge* res = make_edge(p[l], p[r]);
        return make_pair(res, res->rev());
    }
    if (r - l + 1 == 3) {
        QuadEdge *a = make_edge(p[l], p[l + 1]), *b = make_edge(p[l + 1], p[r]);
        splice(a->rev(), b);
        int sg = sgn(p[l].cross(p[l + 1], p[r]));
        if (sg == 0)
            return make_pair(a, b->rev());
        QuadEdge* c = connect(b, a);
        if (sg == 1)
            return make_pair(a, b->rev());
        else
            return make_pair(c->rev(), c);
    }
    int mid = (l + r) / 2;
    QuadEdge *ldo, *ldi, *rdo, *rdi;
    tie(ldo, ldi) = build_tr(l, mid, p);
    tie(rdi, rdo) = build_tr(mid + 1, r, p);
    while (true) {
        if (left_of(rdi->origin, ldi)) {
            ldi = ldi->lnext();
            continue;
        }
        if (right_of(ldi->origin, rdi)) {
            rdi = rdi->rev()->onext;
            continue;
        }
        break;
    }
    QuadEdge* basel = connect(rdi->rev(), ldi);
    auto valid = [&basel](QuadEdge* e) { return right_of(e->dest(), basel); };
    if (ldi->origin == ldo->origin)
        ldo = basel->rev();
    if (rdi->origin == rdo->origin)
        rdo = basel;
    while (true) {
        QuadEdge* lcand = basel->rev()->onext;
        if (valid(lcand)) {
            while (in_circle(basel->dest(), basel->origin, lcand->dest(),
                             lcand->onext->dest())) {
                QuadEdge* t = lcand->onext;
                delete_edge(lcand);
                lcand = t;
            }
        }
        QuadEdge* rcand = basel->oprev();
        if (valid(rcand)) {
            while (in_circle(basel->dest(), basel->origin, rcand->dest(),
                             rcand->oprev()->dest())) {
                QuadEdge* t = rcand->oprev();
                delete_edge(rcand);
                rcand = t;
            }
        }
        if (!valid(lcand) && !valid(rcand))
            break;
        if (!valid(lcand) ||
            (valid(rcand) && in_circle(lcand->dest(), lcand->origin,
                                       rcand->origin, rcand->dest()))
            basel = connect(rcand, basel->rev());
        else
            basel = connect(basel->rev(), lcand->rev
    ());
    }
    return make_pair(ldo, rdo);
}

vector<tuple<pt, pt, pt>> delaunay(vector<pt> p) {
    sort(p.begin(), p.end(), [](const pt& a, const pt
    & b) {
        return lt(a.x, b.x) || (eq(a.x, b.x) && lt(a.
    y, b.y));
    });
    auto res = build_tr(0, (int)p.size() - 1, p);
    QuadEdge* e = res.first;
    vector<QuadEdge*> edges = {e};
    while (lt(e->onext->dest().cross(e->dest(), e->
    origin), 0))
        e = e->onext;
    auto add = [&p, &e, &edges]() {
        QuadEdge* curr = e;
        do {
            curr->used = true;
            p.push_back(curr->origin);
            edges.push_back(curr->rev());
            curr = curr->lnext();
        } while (curr != e);
    };
    add();
    p.clear();
    int kek = 0;
    while (kek < (int)edges.size()) {
        if (!(e = edges[kek++])->used)
            add();
    }
    vector<tuple<pt, pt, pt>> ans;
    for (int i = 0; i < (int)p.size(); i += 3) {
        ans.push_back(make_tuple(p[i], p[i + 1], p[i
    + 2]));
    }
    return ans;
}
```

1.4 Convex Hull

```
vp convex_hull(const vp P)
{
    sort(P.begin(), P.end());
    vp L, U;
    for(auto p: P){
        while(L.size()>=2 and ccw(L[L.size()-2], L.
    back(), p)!=1)
            L.pop_back();
        L.pb(p);
    }
    reverse(P.begin(), P.end());
    for(auto p: P){
        while(U.size()>=2 and ccw(U[U.size()-2], U.
    back(), p)!=1)
            U.pop_back();
        U.pb(p);
    }
    L.pop_back();
    L.insert(L.end(), U.begin(), U.end()-1);
    return L;
}
```

1.5 Sort By Angle

```
int quarter(point a)
{
    if(a.x>0 and a.y>=0) return 0;
    if(a.x<=0 and a.y>0) return 1;
    if(a.x<0 and a.y<=0) return 2;
    return 3;
}

point c;
bool comp(point a, point b) //ccw
{
    a=a-c;b=b-c;
    int qa = quarter(a);
    int qb = quarter(b);
    if(qa==qb)
        return (a^b)>0;
    else
        return qa<qb;
}

c = center(A);
sort(A.begin(), A.end(), comp);
```

1.6 Minkowski Sum

```
vp mk(const vp &a,const vp &b){
    int i = 0, j = 0;
    for(int k = 0; k < (int)a.size(); k++)if(a[k]<a[i
    ])
        i = k;
    for(int k = 0; k < (int)b.size(); k++)if(b[k]<b[j
    ])
        j = k;

    vp c;
    c.reserve(a.size() + b.size());
    for(int k = 0; k < int(a.size()+b.size()); k++){
        point pt{a[i] + b[j]};
        if((int)c.size() >= 2 and !ccw(c[c.size()-2],
     c.back(), pt))
            c.pop_back();
        c.pb(pt);
        int q = i+1, w = j+1;
        if(q == int(a.size())) q = 0;
        if(w == int(b.size())) w = 0;
        if(ccw(c.back(), a[i]+b[w], a[q]+b[j]) < 0) i
     = q;
        else j = w;
    }

    if(!ccw(c[0], c[(int)c.size()-1], c[(int)c.size()
    -2]))
        c.pop_back();
    if(!ccw(c.back(), c[0], c[1])){
        c[0]=c.back();
        c.pop_back();
    }
    c.shrink_to_fit();

    return c;
}
```

1.7 Tetrahedron Distance3d

```
bool nulo(point a){
    return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0))
    ;
}

ld misto(point p1, point p2, point p3){
    return (p1^p2)*p3;
}

ld dist_pt_face(point p, vp v){
```

```cpp
    assert(v.size()==3);

    point v1 = v[1]-v[0];
    point v2 = v[2]-v[0];
    point n = (v1^v2);

    for(int i=0;i<3;i++){
        point va = p-v[i];
        point vb = v[(i+1)%3]-v[i];
        point ve = vb^n;
        ld d = ve*v[i];
        //se ponto coplanar com um dos lados do prisma (va^vb eh nulo),
        //ele esta dentro do prisma (poderia desconsiderar pois distancia
        //vai ser a msm da distancia do ponto ao segmento)
        if(!nulo(va^vb) and (v[(i+2)%3]*ve>d) ^ (p*ve>d)) return LLINF;
    }

    //se ponto for coplanar ao triangulo (e dentro do triangulo)
    //vai retornar zero corretamente
    return fabs(misto(p-v[0],v1,v2)/norm(n));
}

ld dist_pt_seg(point p, vp li){
    return norm((li[1]-li[0])^(p-li[0]))/norm(li[1]-li[0]);
}

ld dist_line(vp l1, vp l2){
    point n = (l1[1]-l1[0])^(l2[1]-l2[0]);
    if(nulo(n)) //retas paralelas - dist ponto a reta
        return dist_pt_seg(l2[0],l1);

    point o1o2 = l2[0]-l1[0];
    return fabs((o1o2*n)/norm(n));
}
// retas paralelas e intersecao nao nula
ld dist_seg(vp l1, vp l2){

    assert(l2.size()==2);
    assert(l1.size()==2);

    //pontos extremos do segmento
    ld ans = LLINF;
    for(int i=0;i<2;i++)
        for(int j=0;j<2;j++)
            ans = min(ans, norm(l1[i]-l2[j]));

    //verificando distancia de ponto extremo com ponto interno dos segs
    for(int t=0;t<2;t++){
        for(int i=0;i<2;i++){
            bool c=true;
            for(int k=0;k<2;k++){
                point va = l1[i]-l2[k];
                point vb = l2[!k]-l2[k];
                ld ang = atan2(norm((vb^va)), vb*va);
                if(ang>PI/2) c = false;
            }
            if(c)
                ans = min(ans,dist_pt_seg(l1[i],l2));
        }
        swap(l1,l2);
    }

    //ponto interno com ponto interno dos segmentos
    point v1 = l1[1]-l1[0], v2 = l2[1]-l2[0];
    point n = v1^v2;
    if(!nulo(n)){
        bool ok = true;
        for(int t=0;t<2;t++){
            point n2 = v2^n;
            point o1o2 = l2[0]-l1[0];
            ld escalar = (o1o2*n2)/(v1*n2);
            if(escalar<0 or escalar>1) ok = false;
            swap(l1,l2);
            swap(v1,v2);
        }
        if(ok) ans = min(ans,dist_line(l1,l2));
    }

    return ans;
}

ld ver(vector<vp> &vet){
    ld ans = LLINF;
    // vertice - face
    for(int k=0;k<2;k++)
        for(int pt=0;pt<4;pt++)
            for(int i=0;i<4;i++){
                vp v;
                for(int j=0;j<4;j++){
                    if(i!=j) v.pb(vet[!k][j]);
                }
                ans = min(ans, dist_pt_face(vet[k][pt], v));
            }

    // edge - edge
    for(int i1=0;i1<4;i1++)
        for(int j1=0;j1<i1;j1++)
            for(int i2=0;i2<4;i2++)
                for(int j2=0;j2<i2;j2++)
                    ans = min(ans, dist_seg({vet[0][i1], vet[0][j1]},
                                            {vet[1][i2], vet[1][j2]}));

    return ans;
}
```

1.8 Numintersectionline

```cpp
int main()
{
    int lim = 1e6;
    Segtree st(lim+100);
    int n, m, y, x, l, r;
    cin >> n >> m;

    int open=-1, close=INF; // open -> check -> close
    vector< pair<int, pii> > sweep;

    ll ans = 0;
    for(int i=0;i<n;i++){ // horizontal
        cin >> y >> l >> r;
        sweep.pb({l, {open, y}});
        sweep.pb({r, {close, y}});
    }
    for(int i=0;i<m;i++){ // vertical
        cin >> x >> l >> r;
        sweep.pb({x, {l, r}});
    }
    sort(sweep.begin(), sweep.end());

    // set<int> on;
    for(auto s: sweep){
        if(s.ss.ff==open){
            st.update(s.ss.ss, 1);
            // on.insert(s.ss.ss);
        }
        else if(s.ss.ff==close){
                st.update(s.ss.ss, -1);
                // on.erase(s.ss.ss);
            }
            else{
                ans += st.query(s.ss.ff, s.ss.ss);
                // auto it1 = on.lower_bound(s.ss.ff);
                // auto it2 = on.upper_bound(s.ss.ss);
                // for(auto it = it1; it!=it2; it++){
                //     intersection -> (s.ff, it);
                // }
            }
        }

    cout << ans << endl;


    return 0;
}
```

1.9 Polygon Diameter

```
double diameter(const vector<point> &p) {
    vector<point> h = convexHull(p);
    int m = h.size();
    if (m == 1)
        return 0;
    if (m == 2)
        return dist(h[0], h[1]);
    int k = 1;
    while (area(h[m - 1], h[0], h[(k + 1) % m]) >
    area(h[m - 1], h[0], h[k]))
        ++k;
    double res = 0;
    for (int i = 0, j = k; i <= k && j < m; i++) {
        res = max(res, dist(h[i], h[j]));
        while (j < m && area(h[i], h[(i + 1) % m], h
    [(j + 1) % m]) > area(h[i], h[(i + 1) % m], h[j]))
    ) {
            res = max(res, dist(h[i], h[(j + 1) % m])
    );
            ++j;
        }
    }
    return res;
}
```

1.10 Polygon Cut Length

```
// Polygon Cut length
ld solve(vp &p, point a, point b){ // ccw
    int n = p.size();
    ld ans = 0;

    for(int i=0;i<n;i++){
        int j = (i+1) % n;

        int signi = ccw(a, b, p[i]);
        int signj = ccw(a, b, p[j]);

        if(signi == 0 and signj == 0){
            if((b-a) * (p[j]-p[i]) > 0){
                ans += param(a, b, p[j]);
                ans -= param(a, b, p[i]);
            }
        }else if(signi <= 0 and signj > 0){
            ans -= param(a, b, inter_line({a, b}, {p[
    i], p[j]})[0]);
        }else if(signi > 0 and signj <= 0){
            ans += param(a, b, inter_line({a, b}, {p[
    i], p[j]})[0]);
        }
    }

    return abs(ans * norm(b-a));
}
```

1.11 Mindistpair

```
ll MinDistPair(vp &vet){
    int n = vet.size();
    sort(vet.begin(), vet.end());
    set<point> s;

    ll best_dist = LLINF;
    int j=0;
    for(int i=0;i<n;i++){
        ll d = ceil(sqrt(best_dist));
        while(j<n and vet[i].x-vet[j].x >= d){
            s.erase(point(vet[j].y, vet[j].x));
            j++;
        }

        auto it1 = s.lower_bound({vet[i].y - d, vet[i
    ].x});
        auto it2 = s.upper_bound({vet[i].y + d, vet[i
    ].x});

        for(auto it=it1; it!=it2; it++){
            ll dx = vet[i].x - it->y;
            ll dy = vet[i].y - it->x;
            if(best_dist > dx*dx + dy*dy){
                best_dist = dx*dx + dy*dy;
                // vet[i] e inv(it)
            }
        }

        s.insert(point(vet[i].y, vet[i].x));
    }
    return best_dist;
}
```

1.12 Rotating Callipers

```
int N;

int sum(int i, int x){
    if(i+x>N-1) return (i+x-N);
    return i+x;
}

ld rotating_callipers(vp &vet){
    N = vet.size();
    ld ans = 0;
    // 2 triangulos (p1, p3, p4) (p1, p2, p3);
    for(int i=0;i<N;i++){ // p1
        int p2 = sum(i, 1); // p2
        int p4 = sum(i, 3); // p4
        for(int j=sum(i, 2);j!=i;j=sum(j, 1)){ // p3
            if(j==p2) p2 = sum(p2, 1);
            while(sum(p2, 1)!=j and areaT(vet[p2],
    vet[i], vet[j]) < areaT(vet[sum(p2, 1)], vet[i],
    vet[j]))
                p2 = sum(p2, 1);
            while(sum(p4, 1)!=i and areaT(vet[p4],
    vet[i], vet[j]) < areaT(vet[sum(p4, 1)], vet[i],
    vet[j]))
                p4 = sum(p4, 1);

            ans = max(ans, area(vet[i], vet[p2], vet[
    j], vet[p4]));
        }
    }

    return ans;
}
```

1.13 Half Plane Intersect

```
// Half plane intersect O(n3)
vp half_plane_intersect(vector<line> &v){
    vp ret;
    int n = v.size();
    for(int i=0; i<n; i++){
        for(int j=i+1; j<n; j++){
            point crs = inter(v[i], v[j]);
            if(crs.x == INF) continue;
            bool bad = 0;
            for(int k=0; k<n; k++)
                if(v[k].eval(crs) < -EPS){
                    bad = 1;
                    break;
                }

            if(!bad) ret.push_back(crs);
        }
    }
    return ret;
}
```

1.14 2d

```
#define vp vector<point>

// typedef ll cod;
// bool eq(cod a, cod b){ return (a==b); }
typedef ld cod;
bool eq(cod a, cod b){ return abs(a - b) <= EPS; }

struct point{
    cod x, y;
    int id;
    point(cod x=0, cod y=0): x(x), y(y){}


    point operator+(const point &o) const{
        return {x+o.x, y+o.y};
    }
    point operator-(const point &o) const{
        return {x-o.x, y-o.y};
    }
    point operator*(cod t) const{
        return {x*t, y*t};
    }
    point operator/(cod t) const{
        return {x/t, y/t};
    }
    cod operator*(const point &o) const{ // dot
        return x * o.x + y * o.y;
    }
    cod operator^(const point &o) const{ // cross
        return x * o.y - y * o.x;
    }
    bool operator<(const point &o) const{
        if(!eq(x, o.x)) return x < o.x;
        return y < o.y;
    }
    bool operator==(const point &o) const{
        return eq(x, o.x) and eq(y, o.y);
    }

};

ld norm(point a){ // Modulo
    return sqrt(a*a);
}

int ccw(point a, point b, point e){ //-1=dir; 0=
    collinear; 1=esq;
    cod tmp = (b-a)^(e-a); // from a to b
    return (tmp > EPS) - (tmp < -EPS);
}

bool nulo(point a){
    return (eq(a.x, 0) and eq(a.y, 0));
}
point rotccw(point p, ld a){
    // a = PI*a/180; // graus
    return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)
    +p.x*sin(a)));
}
point rot90cw(point a) { return point(a.y, -a.x); };
point rot90ccw(point a) { return point(-a.y, a.x); };

ld proj(point a, point b){ // a sobre b
    return a*b/norm(b);
}
ld angle(point a, point b){ // em radianos
    ld ang = a*b / norm(a) / norm(b);
    return acos(max(min(ang, (ld)1), (ld)-1));
}
ld angle_vec(point v){
    // return 180/PI*atan2(v.x, v.y); // graus
    return atan2(v.x, v.y);
}
ld order_angle(point a, point b){ // from a to b ccw
    (a in front of b)
    ld aux = angle(a,b)*180/PI;
    return ((a^b)<=0 ? aux:360-aux);
}
bool angle_less(point a1, point b1, point a2, point
    b2){ // ang(a1,b1) <= ang(a2,b2)
    point p1((a1*b1), abs((a1^b1)));
    point p2((a2*b2), abs((a2^b2)));
    return (p1^p2) <= 0;
}

ld area(vp &p){ // (points sorted)
    ld ret = 0;
    for(int i=2;i<(int)p.size();i++)
        ret += (p[i]-p[0])^(p[i-1]-p[0]);
    return abs(ret/2);
}
ld areaT(point &a, point &b, point &c){
    return abs((b-a)^(c-a))/2.0;
}

point center(vp &A){
    point c = point();
    int len = A.size();
    for(int i=0;i<len;i++)
        c=c+A[i];
    return c/len;
}

point forca_mod(point p, ld m){
    ld cm = norm(p);
    if(cm<EPS) return point();
    return point(p.x*m/cm,p.y*m/cm);
}

point mirror(point m1, point m2, point p){
    // mirror point p around segment m1m2
    point seg = m2-m1;
    ld t0 = ((p-m1)*seg) / (seg*seg);
    point ort = m1 + seg*t0;
    point pm = ort-(p-ort);
    return pm;
}

```

```
ld param(point a, point b, point v){
    // v = t*(b-a) + a // return t;
    // assert(line(a, b).inside_seg(v));
    return ((v-a) * (b-a)) / ((b-a) * (b-a));
}

bool simetric(vector<point> &a){ //ordered
    int n = a.size();
    c = center(a);
    if(n&1) return false;
    for(int i=0;i<n/2;i++)
        if(!collinear(a[i], a[i+n/2], c))
            return false;
    return true;
}



////////////
//  Line  //
////////////

struct line{
    point p1, p2;
    cod a, b, c; // ax+by+c = 0;
    // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
    line(point p1=0, point p2=0): p1(p1), p2(p2){
        a = p1.y-p2.y;
        b = p2.x-p1.x;
        c = -(a*p1.x + b*p1.y);
    }
    line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c)
    {
        // Gera os pontos p1 p2 dados os coeficientes
        // isso aqui eh um lixo mas quebra um galho
    kkkkkk
        if(b==0){
            p1 = point(1, -c/a);
            p2 = point(0, -c/a);
        }else{
            p1 = point(1, (-c-a*1)/b);
            p2 = point(0, -c/b);
        }
    }

    cod eval(point p){
        return a*p.x+b*p.y+c;
    }
    bool inside(point p){
        return eq(eval(p), 0);
    }
    point normal(){
        return point(a, b);
    }

    bool inside_seg(point p){
        return (
            ((p1-p) ^ (p2-p)) == 0 and
            ((p1-p) * (p2-p)) <= 0
        );
    }

};

// be careful with precision error
vp inter_line(line l1, line l2){
    ld det = l1.a*l2.b - l1.b*l2.a;
    if(det==0) return {};
    ld x = (l1.b*l2.c - l1.c*l2.b)/det;
    ld y = (l1.c*l2.a - l1.a*l2.c)/det;
    return {point(x, y)};
}

// segments not collinear
vp inter_seg(line l1, line l2){
    vp ans = inter_line(l1, l2);
    if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.
    inside_seg(ans[0]))
        return {};
    return ans;
}

ld dist_seg(point p, point a, point b){ // point -
    seg
    if(((p-a)*(b-a)) < EPS) return norm(p-a);
    if(((p-b)*(a-b)) < EPS) return norm(p-b);
    return abs((p-a)^(b-a))/norm(b-a);
}

ld dist_line(point p, line l){ // point - line
    return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
}

line bisector(point a, point b){
    point d = (b-a)*2;
    return line(d.x, d.y, a*a - b*b);
}

line perpendicular(line l, point p){ // passes
    through p
    return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
}


////////////
// Circle //
////////////

struct circle{
    point c; cod r;
    circle() : c(0, 0), r(0){}
    circle(const point o) : c(o), r(0){}
    circle(const point a, const point b){
        c = (a+b)/2;
        r = norm(a-c);
    }
    circle(const point a, const point b, const point
    cc){
        c = inter_line(bisector(a, b), bisector(b, cc
    ));
        r = norm(a-c);
    }
    bool inside(const point &a) const{
        return norm(a - c) <= r + EPS;
    }
};

pair<point, point> getTangentPoint(circle cr, point p
    ) {
    ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
    point p1 = rotccw(cr.c-p, -theta);
    point p2 = rotccw(cr.c-p, theta);
    assert(d1 >= cr.r);
    p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
    p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
    return {p1, p2};
}


circle incircle(point p1, point p2, point p3){
    ld m1 = norm(p2-p3);
    ld m2 = norm(p1-p3);
    ld m3 = norm(p1-p2);
    point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
    ld s = 0.5*(m1+m2+m3);
    ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
```

```cpp
    return circle(c, r);
}

circle circumcircle(point a, point b, point c) {
    circle ans;
    point u = point((b-a).y, -(b-a).x);
    point v = point((c-a).y, -(c-a).x);
    point n = (c-b)*0.5;
    ld t = (u^n)/(v^u);
    ans.c = ((a+c)*0.5) + (v*t);
    ans.r = norm(ans.c-a);
    return ans;
}

vp inter_circle_line(circle C, line L){
    point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.p1)*(ab) / (ab*ab));
    ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s / (ab*ab);
    if (h2 < -EPS) return {};
    if (eq(h2, 0)) return {p};
    point h = (ab/norm(ab)) * sqrt(h2);
    return {p - h, p + h};
}

vp inter_circle(circle C1, circle C2){
    if(C1.c == C2.c) { assert(C1.r != C2.r); return {}; }
    point vec = C2.c - C1.c;
    ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r;
    ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 = C1.r*C1.r - p*p*d2;
    if (sum*sum < d2 or dif*dif > d2) return {};
    point mid = C1.c + vec*p, per = point(-vec.y, vec.x) * sqrt(max((ld)0, h2) / d2);
    if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
    return {mid + per, mid - per};
}

// minimum circle cover O(n) amortizado
circle min_circle_cover(vector<point> v){
    random_shuffle(v.begin(), v.end());
    circle ans;
    int n = v.size();
    for(int i=0;i<n;i++) if(!ans.inside(v[i])){
        ans = circle(v[i]);
        for(int j=0;j<i;j++) if(!ans.inside(v[j])){
            ans = circle(v[i], v[j]);
            for(int k=0;k<j;k++) if(!ans.inside(v[k])){
                ans = circle(v[i], v[j], v[k]);
            }
        }
    }
    return ans;
}
```

1.15 Intersect Polygon

```cpp
bool intersect(vector<point> A, vector<point> B) // Ordered ccw
{
    for(auto a: A)
        if(inside(B, a))
            return true;
    for(auto b: B)
        if(inside(A, b))
            return true;

    if(inside(B, center(A)))
        return true;

    return false;
}
```

1.16 3d

```cpp
// typedef int cod;
// bool eq(cod a, cod b){ return (a==b); }

#define vp vector<point>
typedef ld cod;
bool eq(cod a, cod b){ return fabs(a - b) <= EPS; }

struct point
{
    cod x, y, z;
    point(cod x=0, cod y=0, cod z=0): x(x), y(y), z(z){}

    point operator+(const point &o) const{
        return {x+o.x, y+o.y, z+o.z};
    }
    point operator-(const point &o) const{
        return {x-o.x, y-o.y, z-o.z};
    }
    point operator*(cod t) const{
        return {x*t, y*t, z*t};
    }
    point operator/(cod t) const{
        return {x/t, y/t, z/t};
    }
    bool operator==(const point &o) const{
        return eq(x, o.x) and eq(y, o.y) and eq(z, o.z);
    }
    cod operator*(const point &o) const{ // dot
        return x*o.x + y*o.y + z*o.z;
    }
    point operator^(const point &o) const{ // cross
        return point(y*o.z - z*o.y,
                     z*o.x - x*o.z,
                     x*o.y - y*o.x);
    }
};

ld dist(point a, point b){
    return sqrt((a-b)*(a-b));
}
bool nulo(point a){
    return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0));
}

ld norm(point a){ // Modulo
    return sqrt(a*a);
}
ld proj(point a, point b){ // a sobre b
    return (a*b)/norm(b);
}
ld angle(point a, point b){ // em radianos
    return acos((a*b) / norm(a) / norm(b));
}

cod triple(point a, point b, point c){
    return (a * (b^c)); // Area do paralelepipedo
}

point normilize(point a){
    return a/norm(a);
}

struct plane{
    cod a, b, c, d;
    point p1, p2, p3;
    plane(point p1=0, point p2=0, point p3=0): p1(p1), p2(p2), p3(p3){
        point aux = (p1-p3)^(p2-p3);
        a = aux.x; b = aux.y; c = aux.z;
        d = -a*p1.x - b*p1.y - c*p1.z;
    }
    plane(point p, point normal){
        normal = normilize(normal);
        a = normal.x; b = normal.y; c = normal.z;
        d = -(p*normal);
    }

    // ax+by+cz+d = 0;
    cod eval(point &p){
        return a*p.x + b*p.y + c*p.z + d;
    }
};

cod dist(plane pl, point p){
    return fabs(pl.a*p.x + pl.b*p.y + pl.c*p.z + pl.d) / sqrt(pl.a*pl.a + pl.b*pl.b + pl.c*pl.c);
}

point rotate(point v, point k, ld theta){
    // Rotaciona o vetor v theta graus em torno do eixo k
    // theta *= PI/180; // graus
    return (
        v*cos(theta)) +
        ((k^v)*sin(theta)) +
        (k*(k*v))*(1-cos(theta)
    );
}
```

2 Algoritmos

2.1 Mst Xor

```cpp
// omg why just 2 seconds
#include <bits/stdc++.h>
// #define int long long
#define ff first
#define ss second
#define ll long long
#define ld long double
#define pb push_back
#define eb emplace_back
#define pii pair<int, int>
#define pll pair<ll, ll>
#define ti tuple<int, int, int>
#define vi vector<int>
#define vl vector<ll>
#define vii vector<pii>
#define sws ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);
#define endl '\n'
#define teto(a, b) (((a)+(b)-1)/(b))
#define all(x) x.begin(), x.end()
#define forn(i, n) for(int i = 0; i < (int)n; i++)
#define forne(i, a, b) for(int i = a; i <= b; i++)
#define dbg(msg, var) cerr << msg << " " << var << endl;

using namespace std;

const int MAX = 6e6+10;
const ll MOD = 1e9+7;
const int INF = 0x3f3f3f3f;
const ll LLINF = 0x3f3f3f3f3f3f3f3f;
const ld EPS = 1e-6;
const ld PI = acos(-1);

// End Template //

const int N = 2e5+10;

struct DSU {
    int n;
    map<int, int> parent;
    map<int, vi> comp;

    int find(int v) {
        if(v==parent[v])
            return v;
        return parent[v]=find(parent[v]);
    }

    void join(int a, int b) {
        a = find(a);
        b = find(b);
        if(a!=b) {
            if((int)comp[a].size()<(int)comp[b].size())
                swap(a, b);

            for(auto v: comp[b])
                comp[a].pb(v);
            comp[b].clear();
            parent[b]=a;
        }

    }
};

int trie[MAX][2];
set<int> idx[MAX];
int finish[MAX];
int nxt = 1;

void add(int s){
    int node = 0;
    for(int i=30;i>=0;i--){
        bool c = (s & (1<<i));
        if(trie[node][c] == 0)
            node = trie[node][c] = nxt++;
        else
            node = trie[node][c];
        finish[node]++;
    }
}

void remove(int s){
    int node = 0;
    for(int i=30;i>=0;i--){
        bool c = (s & (1<<i));
        node = trie[node][c];
        finish[node]--;
    }
}

int min_xor(int s){
    int node = 0;
    int ans = 0;
    for(int i=30;i>=0;i--){
        bool c = (s & (1<<i));
        if(finish[trie[node][c]] != 0)
            node = trie[node][c];
        else{
            ans ^= 1 << i;
            node = trie[node][!c];
        }
    }
    return ans;
}

```

```cpp

int32_t main()
{sws;

    int n;
    cin >> n;
    vi x(n);
    for(int i=0;i<n;i++)
        cin >> x[i];

    sort(x.begin(), x.end());
    x.erase(unique(x.begin(), x.end()), x.end());
    n = x.size();

    DSU dsu;

    ll mstsum = 0;

    vi pais;
    for(int i=0;i<n;i++){
        add(x[i]);
        dsu.parent[x[i]] = x[i];
        dsu.comp[x[i]].pb(x[i]);
        pais.pb(x[i]);
    }

    while((int)pais.size()!=1){
        vector<ti> edges;
        for(auto p: pais){
            vi &nodes = dsu.comp[p];
            // erase
            for(auto u: nodes) remove(u);

            // query
            ti ed = {LLINF, 0, 0};
            for(auto u: nodes){
                int xr = min_xor(u);
                ed = min(ed, {xr, u, xr^u});
            }
            edges.pb(ed);

            // add back
            for(auto u: nodes) add(u);
        }

        for(auto [xr, u, v]: edges){
            if(dsu.find(u)!=dsu.find(v)){
                // u, v -> mst
                // cout << "mst = " << u << " " << v
    << endl;
                mstsum += xr;
                dsu.join(u, v);
            }
        }
        vi pais2;
        for(auto p: pais)
            if(p==dsu.find(p))
                pais2.pb(p);
        swap(pais, pais2);
    }

    cout << mstsum << endl;


    return 0;
}
```

2.2 Cdq

```cpp
// LIS 3D problem

struct Segtree{
    vi t;
    int n;

    Segtree(int n){
        this->n = n;
        t.assign(2*n, 0);
    }

    int merge(int a, int b){
        return max(a, b);
    }

    void build(){
        for(int i=n-1;i>0;i--)
            t[i] = merge(t[i<<1], t[i<<1|1]);
    }

    int query(int l, int r){
        int resl = -INF, resr = -INF;
        for(l+=n, r+=n+1; l<r; l>>=1, r>>=1){
            if(l&1) resl = merge(resl, t[l++]);
            if(r&1) resr = merge(t[--r], resr);
        }
        return merge(resl, resr);
    }

    void update(int p, int value){
        p+=n;
        for(t[p]=max(t[p], value); p >>= 1;)
            t[p] = merge(t[p<<1], t[p<<1|1]);
    }
};

struct point{
    int x, y, z, id;
    bool left;
    point(int x=0, int y=0, int z=0): x(x), y(y), z(z
    ){
        left = false;
    }
    bool operator<(point &o){
        if(x != o.x) return x < o.x;
        if(y != o.y) return y > o.y;
        return z < o.z;
    }
};


void cdq(int l, int r, vector<point> &a, vi &dp){
    if(l==r) return;

    int mid = (l+r) / 2;

    cdq(l, mid, a, dp);

    // compress z
    set<int> uz; map<int, int> idz;
    for(int i=l;i<=r;i++) uz.insert(a[i].z);
    int id = 0;
    for(auto z: uz) idz[z] = id++;

    vector<point> tmp;
    for(int i=l;i<=r;i++){
        tmp.pb(a[i]);
        tmp.back().x = 0;
        tmp.back().z = idz[tmp.back().z];
        if(i<=mid)
            tmp.back().left = true;
    }

    Segtree st(id);

    sort(tmp.rbegin(), tmp.rend());

    for(auto t: tmp){
        if(t.left){
            st.update(t.z, dp[t.id]);
        }else{
            dp[t.id] = max(dp[t.id], st.query(0, t.z-1)+1);
        }
    }

    cdq(mid+1, r, a, dp);
}


int32_t main()
{sws;

    int n; cin >> n;

    vector<point> vet(n);
    for(int i=0;i<n;i++){
        cin >> vet[i].x >> vet[i].y >> vet[i].z;
    }

    sort(vet.begin(), vet.end());

    for(int i=0;i<n;i++)
        vet[i].id = i;

    vi dp(n, 1);

    cdq(0, n-1, vet, dp);

    int ans = 0;
    for(int i=0;i<n;i++)
        ans = max(ans, dp[i]);

    cout << ans << endl;


    return 0;
}
```

2.3 Histogram Rectangle

```cpp
ll bestRectangle(vi hist){
    int n = hist.size();
    stack<ll> s;
    s.push(-1);
    ll ans = hist[0];
    vl left_smaller(n, -1), right_smaller(n, n);
    for(int i=0;i<n;i++){
        while(!s.empty() and s.top()!=-1 and hist[s.top()]>hist[i]){
            right_smaller[s.top()] = i;
            s.pop();
        }
        if(i>0 and hist[i]==hist[i-1])
            left_smaller[i] = left_smaller[i-1];
        else
            left_smaller[i] = s.top();
        s.push(i);
    }

    for(int j=0;j<n;j++){
        ll area = hist[j]*(right_smaller[j]-left_smaller[j]-1);
        ans = max(ans, area);
    }
    return ans;
}
```

3 Misc

3.1 Rand

```cpp
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> distribution(1,n);

num = distribution(rng); // num no range [1, n]
shuffle(vec.begin(), vec.end(), rng); // shuffle


ull mix(ull o){
    o+=0x9e3779b97f4a7c15;
    o=(o^(o>>30))*0xbf58476d1ce4e5b9;
    o=(o^(o>>27))*0x94d049bb133111eb;
    return o^(o>>31);
}
ull hash(pii a) {return mix(a.first ^ mix(a.second));}
```

3.2 Bitwise

```cpp
// Bitwise
    #pragma GCC target("popcnt")
    unsigned char a = 5, b = 9; // a = (00000101), b = (00001001)

    AND -          a&b   // The result is 00000001 (1)
    OR -           a|b   // The result is 00001101 (13)
    XOR -          a^b   // The result is 00001100 (12)
    NOT -          ~a    // The result is 11111010 (250)
    Left shift -   b<<1  // The result is 00010010 (18)
    Right shift -  b>>1  // The result is 00000100 (4)

    // Exchange two int variables

        a^=b;
        b^=a;
        a^=b;

    // Even or Odd

        (x & 1)? printf("Odd"): printf("Even");

    // Turn on the j-th bit

        int S = 34; //(100010)
        int j = 3;

        S = S | (1<<j);

    // Turn off the j-th bit

        int S = 42; //(101010)
        int j = 1;

        S &= ~(1<<j)

        S == 40 //(101000)

    // Check the j-th element

        int S = 42; //(101010)
        int j = 3;

        T = S & (1<<j); // T = 0

    // Least significant bit (lsb)

        int lsb(int x){ return x&-x; }

    // Exchange o j-th element

        S ^= (1<<j)

    // Position of the first bit on

        T = (S & (-S))
        T -> 4 bit ligado //(1000)

    // Most significant digit of N

        double K = log10(N);
        K = K - floor(K);
        int X = pow(10, K);

    // Number of digits in N

        X =floor(log10(N)) + 1;

    // Power of two

        bool isPowerOfTwo(int x){ return x && (!(x&(x
    -1))); }

    // Turn off the first bit 1
        m = m & (m-1);

    // Built-in functions

        // Number of bits 1
        __builtin_popcount()
        __builtin_popcountll()

        // Number of leading zeros
        __builtin_clz()
        __builtin_clzll()

        // Number of trailing zeros
        __builtin_ctz()
        __builtin_ctzll()

    // floor(log2(x))

        int flog2(int x){ return 32-1-__builtin_clz(x
    ); }

        int flog2ll(ll x){ return 64-1-
    __builtin_clzll(x); }
```

3.3 Template

```
#include <bits/stdc++.h>
using namespace std;

using ll = long long;
using ii = pair<int, int>;
using vi = vector<int>;
#define pb emplace_back
#define mp make_pair

const ll MOD = 998'244'353;
const int MAX = 2e5 + 5;
```

3.4 Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

#include <ext/pb_ds/detail/standard_policies.hpp>

using namespace __gnu_pbds; // or pb_ds;

template<typename T, typename B = null_type>
using ordered_set = tree<T, B, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
// find_by_order / order_of_key
```

4 Math

4.1 Randommod

```
int randommod() {
    auto primo = [](int num) {
        for(int i = 2; i*i <= num; i++) {
            if(num%i == 0) return false;
        }
        return true;
    };
    uniform_int_distribution<int> distribution
    (1000000007, 1500000000);
    int num = distribution(rng);
    while(!primo(num)) num++;
    return num;
}
```

4.2 Division Trick

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

4.3 Inverso Mult

```
// gcd(a, m) = 1 para existir solucao
// ax + my = 1, ou a*x = 1 (mod m)
ll inv(ll a, ll m) { // com gcd
    ll x, y;
    gcd(a, m, x, y);
    return (((x % m) +m) %m);
}

ll inv(ll a, ll phim) { // com phi(m), se m for primo
     entao phi(m) = p-1
    ll e = phim-1;
    return fexp(a, e);
}
```

4.4 Crt

```
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);
    }
};
```

4.5 Gaussxor

```
struct Gauss {
    array<ll, LOG_MAX> vet;
    int size;
    Gauss() : size(0) {
        fill(vet.begin(), vet.end(), 0);
    }
    Gauss(vector<ll> vals) : size(0) {
        fill(vet.begin(), vet.end(), 0);
        for(ll val : vals) add(val);
    }
    bool add(ll val) {
        for(int i = LOG_MAX-1; i >= 0; i--) if(val &
    (1LL << i)) {
            if(vet[i] == 0) {
                vet[i] = val;
                size++;
                return true;
            }
            val ^= vet[i];
        }
        return false;
    }
};
```

4.6 Pollard Rho

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

ll pow(ll a, ll b, ll m) {
    ll ans = 1;
    for (; b > 0; b /= 2ll, a = mul(a, a, m)) {
        if (b % 2ll == 1)
            ans = mul(ans, a, m);
    }
    return 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;
}
```

4.7 Fast Exponentiaion

```
ll fexp(ll b, ll e, ll mod) {
    ll res = 1;
    b %= mod;
    while(e){
        if(e & 1LL)
            res = (res * b) % mod;
        e = e >> 1LL;
        b = (b * b) % mod;
    }
    return res;
}
```

4.8 Linear Diophantine Equation

```
// Linear Diophantine Equation
int gcd(int a, int b, int &x, int &y)
{
    if (a == 0)
    {
        x = 0; y = 1;
        return b;
    }
    int x1, y1;
    int d = gcd(b%a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}

bool find_any_solution(int a, int b, int c, int &x0,
    int &y0, int &g)
{
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g)
        return false;

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

//  All solutions
//  x = x0 + k*b/g
//  y = y0 - k*a/g
```

4.9 Miller Habin

```
ll mul(ll a, ll b, ll m) {
    return (a*b-ll(a*(long double)b/m+0.5)*m+m)%m;
```

```cpp
}

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

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

    ll d = n - 1;
    int r = 0;
    while (d % 2 == 0) {
        r++;
        d /= 2;
    }

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

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

4.10 Fft Simple

```cpp
struct num{
    ld a {0.0}, b {0.0};
    num(){}
    num(ld na) : a{na}{}
    num(ld na, ld nb) : a{na}, b{nb} {}
    const num operator+(const num &c) const{
        return num(a + c.a, b + c.b);
    }
    const num operator-(const num &c) const{
        return num(a - c.a, b - c.b);
    }
    const num operator*(const num &c) const{
        return num(a*c.a - b*c.b, a*c.b + b*c.a);
    }
    const num operator/(const int &c) const{
         return num(a/c, b/c);
    }
};

void fft(vector<num> &a, bool invert){
    int n = a.size();
    for(int i=1,j=0;i<n;i++){
        int bit = n>>1;
        for(; j&bit; bit>>=1)
            j^=bit;
        j^=bit;
        if(i<j)
            swap(a[i], a[j]);
    }
    for(int len = 2; len <= n; len <<= 1){
        ld ang = 2 * PI / len * (invert ? -1 : 1);
        num wlen(cos(ang), sin(ang));
        for(int i=0;i<n;i+=len){
            num w(1);
            for (int j=0;j<len/2;j++){
                num u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w = w * wlen;
            }
        }
    }
    if(invert)
      for(num &x: a)
         x = x/n;

}

vl multiply(vi const& a, vi const& b){
    vector<num> fa(a.begin(), a.end());
    vector<num> fb(b.begin(), b.end());
    int n = 1;
    while(n < int(a.size() + b.size()) )
        n <<= 1;
    fa.resize(n);
    fb.resize(n);
    fft(fa, false);
    fft(fb, false);
    for(int i=0;i<n;i++)
        fa[i] = fa[i]*fb[i];
    fft(fa, true);
    vl result(n);
    for(int i=0;i<n;i++)
        result[i] = round(fa[i].a);
    while(result.back()==0) result.pop_back();
    return result;
}
```

4.11 Fft Tourist

```cpp
struct num{
    ld x, y;
    num() { x = y = 0; }
    num(ld x, ld 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}};
vi rev = {0, 1};

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){
        ld angle = 2*PI / (1 << (base + 1));
        for(int i = 1 << (base - 1); i < (1 << base); i++){
            roots[i << 1] = roots[i];
            ld 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;
vi multiply(vi &a, vi &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++){
        int x = (i < (int) a.size() ? a[i] : 0);
        int 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);
    vi res(need);
    for(int i = 0; i < need; i++)
        res[i] = fa[i].x + 0.5;

    return res;
}


vi multiply_mod(vi &a, vi &b, int m, int eq = 0){
    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++){
        int 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++){
            int 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);
    }
    ld ratio = 0.25 / sz;
    num r2(0, -1);
    num r3(ratio, 0);
    num r4(0, -ratio);
    num 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);
    vi 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;
}




int main()
{sws;

    //FFT
    vi fx{1, 2, 3}; // 1+2x+3x^2
    vi gx{4, 5}; // 4+5x
    vi res;

    res = multiply(fx,gx); //4 + 13x + 22x^2 + 15x^3

    return 0;

}
```

4.12 Matrix Exponentiation

```
struct Matrix {
    vector<vl> m;
    int r, c;

    Matrix(vector<vl> mat) {
        m = mat;
        r = mat.size();
        c = mat[0].size();
    }

    Matrix(int row, int col, bool ident=false) {
        r = row; c = col;
        m = vector<vl>(r, vl(c, 0));
        if(ident) {
            for(int i = 0; i < min(r, c); i++) {
                m[i][i] = 1;
            }
        }
    }

    Matrix operator*(const Matrix &o) const {
        assert(c == o.r); // garantir que da pra multiplicar
        vector<vl> res(r, vl(o.c, 0));

        for(int i = 0; i < r; i++) {
            for(int k = 0; k < c; k++) {
                for(int j = 0; j < o.c; j++) {
                    res[i][j] = (res[i][j] + m[i][k]*o.m[k][j]) % MOD;
                }
            }
        }

        return Matrix(res);
    }
};

Matrix fexp(Matrix b, int e, int n) {
    if(e == 0) return Matrix(n, n, true); // identidade
    Matrix res = fexp(b, e/2, n);
    res = (res * res);
    if(e%2) res = (res * b);

    return res;
}
```

4.13 Mulmod

```
ll mulmod(ll a, ll b) {
    if(a == 0) {
        return 0LL;
    }
    if(a%2 == 0) {
        ll val = mulmod(a/2, b);
        return (val + val) % MOD;
    }
    else {
        ll val = mulmod((a-1)/2, b);
        val = (val + val) % MOD;
        return (val + b) % MOD;
    }
}
```

4.14 Raiz Primitiva

```
ll fexp(ll b, ll e, ll mod) {
    if(e == 0) return 1LL;
    ll res = fexp(b, e/2LL, mod);
    res = (res*res)%mod;
    if(e%2LL)
        res = (res*b)%mod;

    return res%mod;
}

vl fatorar(ll n) { // fatora em primos
    vl fat;
    for(int i = 2; i*i <= n; i++) {
        if(n%i == 0) {
            fat.pb(i);
            while(n%i == 0)
                n /= i;
        }
    }
    return fat;
}

// O(log(n) ^ 2)
bool raiz_prim(ll a, ll mod, ll phi, vl fat) {
    if(__gcd(a, mod) != 1 or fexp(a, phi/2, mod) == 1) // phi de euler sempre eh PAR
        return false;

    for(auto f : fat) {
        if(fexp(a, phi/f, mod) == 1)
            return false;
    }

    return true;
}

// mods com raizes primitivas: 2, 4, p^k, 2*p^k, p eh primo impar, k inteiro --- O(n log^2(n))
ll achar_raiz(ll mod, ll phi) {
    if(mod == 2) return 1;
    vl fat, elementos;
    fat = fatorar(phi);

    for(ll i = 2; i <= mod-1; i++) {
        if(raiz_prim(i, mod, phi, fat))
            return i;
    }

    return -1; // retorna -1 se nao existe
}

vl todas_raizes(ll mod, ll phi, ll raiz) {
    vl raizes;
    if(raiz == -1) return raizes;
    ll r = raiz;
    for(ll i = 1; i <= phi-1; i++) {
        if(__gcd(i, phi) == 1) {
            raizes.pb(r);
        }
        r = (r * raiz) % mod;
    }

    return raizes;
}
```

4.15 Bigmod

```
ll mod(string a, ll p) {
    ll res = 0, b = 1;
    reverse(all(a));

    for(auto c : a) {
        ll tmp = (((ll)c-'0')*b) % p;
        res = (res + tmp) % p;

        b = (b * 10) % p;
    }

    return res;
}
```

4.16 Berlekamp Massey

```

#define SZ 233333

ll qp(ll a,ll b)
{
    ll x=1; a%=MOD;
    while(b)
    {
        if(b&1) x=x*a%MOD;
        a=a*a%MOD; b>>=1;
    }
    return x;
}
namespace linear_seq {

inline vector<int> BM(vector<int> x)
{
    //ls: (shortest) relation sequence (after filling
     zeroes) so far
    //cur: current relation sequence
    vector<int> ls,cur;
    //lf: the position of ls (t')
    //ldt: delta of ls (v')
    int lf=0,ldt=0;
    for(int i=0;i<int(x.size());++i)
    {
        ll t=0;
        //evaluate at position i
        for(int j=0;j<int(cur.size());++j)
            t=(t+x[i-j-1]*(ll)cur[j])%MOD;
        if((t-x[i])%MOD==0) continue; //good so far
        //first non-zero position
        if(!cur.size())
        {
            cur.resize(i+1);
            lf=i; ldt=(t-x[i])%MOD;
            continue;
        }
        //cur=cur-c/ldt*(x[i]-t)
        ll k=-(x[i]-t)*qp(ldt,MOD-2)%MOD/*1/ldt*/;
        vector<int> c(i-lf-1); //add zeroes in front
        c.pb(k);
        for(int j=0;j<int(ls.size());++j)
            c.pb(-ls[j]*k%MOD);
        if(c.size()<cur.size()) c.resize(cur.size());
        for(int j=0;j<int(cur.size());++j)
            c[j]=(c[j]+cur[j])%MOD;
        //if cur is better than ls, change ls to cur
        if(i-lf+(int)ls.size()>=(int)cur.size())
            ls=cur,lf=i,ldt=(t-x[i])%MOD;
        cur=c;
    }
    for(int i=0;i<int(cur.size());++i)
        cur[i]=(cur[i]%MOD+MOD)%MOD;
    return cur;
}
int m; //length of recurrence
//a: first terms
//h: relation
ll a[SZ],h[SZ],t_[SZ],s[SZ],t[SZ];
//calculate p*q mod f
inline void mull(ll*p,ll*q)
{
    for(int i=0;i<m+m;++i) t_[i]=0;
    for(int i=0;i<m;++i) if(p[i])
        for(int j=0;j<m;++j)
            t_[i+j]=(t_[i+j]+p[i]*q[j])%MOD;
    for(int i=m+m-1;i>=m;--i) if(t_[i])
        //miuns t_[i]x^{i-m}(x^m-\sum_{j=0}^{m-1} x^{
    m-j-1}h_j)
        for(int j=m-1;~j;--j)
            t_[i-j-1]=(t_[i-j-1]+t_[i]*h[j])%MOD;
    for(int i=0;i<m;++i) p[i]=t_[i];
}
inline ll calc(ll K)
{
    for(int i=m;~i;--i)
        s[i]=t[i]=0;
    //init
    s[0]=1; if(m!=1) t[1]=1; else t[0]=h[0];
    //binary-exponentiation
    while(K)
    {
        if(K&1) mull(s,t);
        mull(t,t); K>>=1;
    }
    ll su=0;
    for(int i=0;i<m;++i) su=(su+s[i]*a[i])%MOD;
    return (su%MOD+MOD)%MOD;
}
inline int work(vector<int> x,ll n)
{
    if(n<int(x.size())) return x[n];
    vector<int> v=BM(x); m=v.size(); if(!m) return 0;
    for(int i=0;i<m;++i) h[i]=v[i],a[i]=x[i];
    return calc(n);
}

}
using linear_seq::work;
```

4.17 Double Gcd

```
ld gcdf(ld a, ld b){
    if(a<b) return gcdf(b, a);

    if(fabs(b)<EPS)
        return a;
    else
        return (gcdf(b, a - floor(a/b)*b));
}
```

4.18 Totient

```
// phi(p^k) = (p^(k-1))*(p-1) com p primo
// O(sqrt(m))
ll phi(ll m){
    ll res = m;
    for(ll d=2;d*d<=m;d++){
        if(m % d == 0){
            res = (res/d)*(d-1);
            while(m%d == 0)
                m /= d;
        }
    }
    if(m > 1) {
        res /= m;
        res *= (m-1);
    }
    return res;
}

```

```cpp
// modificacao do crivo, O(n*log(log(n)))
vl phi_to_n(ll n){
    vector<bool> isprime(n+1, true);
    vl tot(n+1);
    tot[0] = 0; tot[1] = 1;
    for(ll i=1;i<=n; i++){
        tot[i] = i;
    }

    for(ll p=2;p<=n;p++){
        if(isprime[p]){
            tot[p] = p-1;
            for(ll i=p+p;i<=n;i+=p){
                isprime[i] = false;
                tot[i] = (tot[i]/p)*(p-1);
            }
        }
    }
    return tot;
}
```

4.19 Kitamasa

```cpp
using poly = vector<mint>; // mint = int mod P with
    operators +, - and *
inline int len(const poly& a) { return a.size(); } //
     get rid of the annoying "hey a.size() is
    unsigned" warning

poly pmul(const poly& a, const poly& b) {
    poly c(len(a) + len(b) - 1, 0);
    for (int i = 0; i < len(a); i++)
        for (int j = 0; j < len(b); j++)
            c[i+j] = c[i+j] + a[i] * b[j];
    return c;
}

// only works if b.back() == 1
poly pmod(const poly& a, const poly& b) {
    poly c(a.begin(), a.end());
    for (int i = len(c) - 1; i >= len(b) - 1; i--) {
        int k = i - (len(b) - 1); // index of the
    quotient term
        for (int j = 0; j < len(b); j++)
            c[j+k] = c[j+k] - c[i] * b[j];
    }
    c.resize(len(b) - 1);
    return c;
}

poly ppwr(poly x, ll e, poly f) {
    poly ans = { 1 };
    for (; e > 0; e /= 2) {
        if (e & 1) ans = pmod(pmul(ans, x), f);
        x = pmod(pmul(x, x), f);
    }
    return ans;
}

// values = { A0, A1, ..., An }. recurrence = C0 × A0
     + C1 × A1 + ... + Cn × An generates A{n+1}
mint kitamasa(const poly& values, const poly&
    recurrence, ll n) {
    poly f(len(recurrence) + 1);
    f.back() = 1;
    for (int i = 0; i < len(recurrence); i++)
        f[i] = mint(0) - recurrence[i];

    auto d = ppwr(poly{0, 1}, n, f); // x^N mod f(x)

    mint ans = 0;
    for (int i = 0; i < len(values); i++)
        ans = ans + d[i] * values[i];
    return ans;
}
```

4.20 Mobius

```cpp
vi mobius(int n) {
    // g(n) = sum{f(d)} => f(n) = sum{mu(d)*g(n/d)}
    vi mu(n+1);
    mu[1] = 1; mu[0] = 0;
    for(int i = 1; i <= n; i++)
        for(int j = i + i; j <= n; j += i)
            mu[j] -= mu[i];

    return mu;
}
```

5 Grafos

5.1 Ford

```cpp
const int N = 2000010;

struct Ford {
    struct Edge {
        int to, f, c;
    };

    int vis[N];
    vector<int> adj[N];
    vector<Edge> edges;
    int cur = 0;

    void addEdge(int a, int b, int cap, int rcap) {
        Edge e;
        e.to = b; e.c = cap; e.f = 0;
        edges.pb(e);
        adj[a].pb(cur++);

        e = Edge();
        e.to = a; e.c = rcap; e.f = 0;
        edges.pb(e);
        adj[b].pb(cur++);
    }

    int dfs(int s, int t, int f, int tempo) {
        if(s == t)
            return f;
        vis[s] = tempo;

        for(int e : adj[s]) {
            if(vis[edges[e].to] < tempo and (edges[e
    ].c - edges[e].f) > 0) {
                if(int a = dfs(edges[e].to, t, min(f,
     edges[e].c-edges[e].f) , tempo)) {
                    edges[e].f += a;
                    edges[e^1].f -= a;
                    return a;
                }
            }
        }
        return 0;
    }

    int flow(int s, int t) {
        int mflow = 0, tempo = 1;
        while(int a = dfs(s, t, INF, tempo)) {
            mflow += a;
            tempo++;
        }
        return mflow;
    }
};
```

5.2 2sat

```cpp
vector<int> g[MAX], gt[MAX], S; int vis[MAX], cor[MAX];

int val(int n, bool tvalue) {
    if(tvalue) return 2*n;
    return 2*n +1;
}

void dfs(int u) {
    vis[u] = 1; for(int v : g[u]) if(!vis[v]) dfs(v);
    S.push_back(u);
}

void dfst(int u, int e) {
    cor[u] = e;
    for(int v : gt[u]) if(!cor[v]) dfst(v, e);
}

void kosaraju(int n) {
    for(int i = 0; i <= n; i++) if(!vis[i]) dfs(i);
    for(int i = 0; i <= n; i++) for(int j : g[i])
        gt[j].push_back(i);
    int e = 0; reverse(S.begin(), S.end());
    for(int u : S) if(!cor[u]) dfst(u, ++e);
}

// antes de chamar essa funcao, colocar as arestas do grafo
bool solve(int n, vi &res) {
    kosaraju(2*n); // MAX > 2*N
    vi r;

    forn(i, n) {
        int t = val(i, true), f = val(i, false);
        if(cor[t] == cor[f]) {
            return false;
        }
        else {
            if(cor[t] > cor[f])
                r.pb(1);
            else
                r.pb(0);
        }
    }
    swap(r, res);
    return true;
}
```

5.3 Kahn

```cpp
vi g[MAX];
int in[MAX], cor[MAX];
void kahn(int n) {
    int label = 1;
    priority_queue<int, vector<int>, greater<int>> pq; // trocar por queue para O(n)
    for(int i = 1; i <= n; i++) {
        if(in[i] == 0) {
            pq.push(i);
        }
    }

    while(pq.size()) {
        int u = pq.top(); pq.pop();
        cor[u] = label++;
        for(auto prox : g[u]) {
            in[prox]--;
            if(in[prox] == 0) {
                pq.push(prox);
            }
        }
    }
}
```

5.4 Hungarian

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

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

5.5 Dfs Tree

```cpp
int desce[MAX], sobe[MAX], vis[MAX], h[MAX];
int backedges[MAX], pai[MAX];

// backedges[u] = backedges que comecam embaixo de (ou =) u e sobem pra cima de u; backedges[u] == 0 => u eh ponte
void dfs(int u, int p) {
    if(vis[u]) return;
    pai[u] = p;
    h[u] = h[p]+1;
    vis[u] = 1;

    for(auto v : g[u]) {
        if(p == v or vis[v]) continue;
        dfs(v, u);
        backedges[u] += backedges[v];
    }
    for(auto v : g[u]) {
        if(h[v] > h[u]+1)
            desce[u]++;
        else if(h[v] < h[u]-1)
            sobe[u]++;
    }
    backedges[u] += sobe[u] - desce[u];
}
```

5.6 Lca

```cpp
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)]];
    }
}
```

5.7 Hld Aresta

```cpp
vector<vector<pair<int, int>>> g(MAX, vector<pair<int,int>>());
vi in(MAX), inv(MAX), sz(MAX);
vi peso(MAX), pai(MAX);
vi head(MAX), tail(MAX), h(MAX);

int tin;

void dfs(int u, int p=-1, int depth=0){
    sz[u] = 1; h[u] = depth;
    for(auto &i: g[u]) if(i.ff != p){
        auto [v, w] = i;
        dfs(v, u, depth+1);
        pai[v] = u; sz[u] += sz[v]; peso[v] = w;
        if (sz[v] > sz[g[u][0].ff] or g[u][0].ff == p) swap(i, g[u][0]);
    }
}
void build_hld(int u, int p = -1) {
    v[in[u] = tin++] = peso[u]; tail[u] = u;
    inv[tin-1] = u;
    for(auto &i: g[u]) if(i.ff != p) {
        int v = i.ff;
        head[v] = (i == g[u][0] ? head[u] : v);
        build_hld(v, u);
    }
    if(g[u].size() > 1) tail[u] = tail[g[u][0].ff];
}
void init_hld(int root = 0) {
    dfs(root);
    tin = 0;
    build_hld(root);
    build();
}
void reset(){
    g.assign(MAX, vector<pair<int,int>>());
    in.assign(MAX, 0), sz.assign(MAX, 0);
    peso.assign(MAX, 0), pai.assign(MAX, 0);
    head.assign(MAX, 0); tail.assign(MAX, 0);
    h.assign(MAX, 0); inv.assign(MAX, 0);

    t.assign(4*MAX, 0); v.assign(MAX, 0);
    lazy.assign(4*MAX, 0);
}
ll query_path(int a, int b) {
    if (a == b) return 0;
    if(in[a] < in[b]) swap(a, b);

    if(head[a] == head[b]) return query(in[b]+1, in[a]);
    return merge(query(in[head[a]], in[a]), query_path(pai[head[a]], b));
}
void update_path(int a, int b, int x) {
    if (a == b) return;
    if(in[a] < in[b]) swap(a, b);

    if(head[a] == head[b]) return (void)update(in[b]+1, in[a], x);
    update(in[head[a]], in[a], x); update_path(pai[head[a]], b, x);
}
```

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

5.8 Mcmf

```
template <class T = int>
class MCMF {
public:
    struct Edge {
        Edge(int a, T b, T c) : to(a), cap(b), cost(c) {}
        int to;
        T cap, cost;
    };

    MCMF(int size) {
        n = size;
        edges.resize(n);
        pot.assign(n, 0);
        dist.resize(n);
        visit.assign(n, false);
    }

    std::pair<T, T> mcmf(int src, int sink) {
        std::pair<T, T> ans(0, 0);
        if(!SPFA(src, sink)) return ans;
        fixPot();
        // can use dijkstra to speed up depending on the graph
        while(SPFA(src, sink)) {
            auto flow = augment(src, sink);
            ans.first += flow.first;
            ans.second += flow.first * flow.second;
            fixPot();
        }
        return ans;
    }

    void addEdge(int from, int to, T cap, T cost) {
        edges[from].push_back(list.size());
        list.push_back(Edge(to, cap, cost));
        edges[to].push_back(list.size());
        list.push_back(Edge(from, 0, -cost));
    }
private:
    int n;
    std::vector<std::vector<int>> edges;
    std::vector<Edge> list;
    std::vector<int> from;
    std::vector<T> dist, pot;
    std::vector<bool> visit;

    /*bool dij(int src, int sink) {
        T INF = std::numeric_limits<T>::max();
        dist.assign(n, INF);
        from.assign(n, -1);
        visit.assign(n, false);
        dist[src] = 0;
        for(int i = 0; i < n; i++) {
            int best = -1;
            for(int j = 0; j < n; j++) {
                if(visit[j]) continue;
                if(best == -1 || dist[best] > dist[j]) best = j;
            }
            if(dist[best] >= INF) break;
            visit[best] = true;
            for(auto e : edges[best]) {
                auto ed = list[e];
                if(ed.cap == 0) continue;
                T toDist = dist[best] + ed.cost + pot[best] - pot[ed.to];
                assert(toDist >= dist[best]);
                if(toDist < dist[ed.to]) {
                    dist[ed.to] = toDist;
                    from[ed.to] = e;
                }
            }
        }
        return dist[sink] < INF;
    }*/

    std::pair<T, T> augment(int src, int sink) {
        std::pair<T, T> flow = {list[from[sink]].cap, 0};
        for(int v = sink; v != src; v = list[from[v]^1].to) {
            flow.first = std::min(flow.first, list[from[v]].cap);
            flow.second += list[from[v]].cost;
        }
        for(int v = sink; v != src; v = list[from[v]^1].to) {
            list[from[v]].cap -= flow.first;
            list[from[v]^1].cap += flow.first;
        }
        return flow;
    }

    std::queue<int> q;
    bool SPFA(int src, int sink) {
        T INF = std::numeric_limits<T>::max();
        dist.assign(n, INF);
        from.assign(n, -1);
        q.push(src);
        dist[src] = 0;
        while(!q.empty()) {
            int on = q.front();
            q.pop();
            visit[on] = false;
            for(auto e : edges[on]) {
                auto ed = list[e];
                if(ed.cap == 0) continue;
                T toDist = dist[on] + ed.cost + pot[on] - pot[ed.to];
                if(toDist < dist[ed.to]) {
                    dist[ed.to] = toDist;
                    from[ed.to] = e;
                    if(!visit[ed.to]) {
                        visit[ed.to] = true;
                        q.push(ed.to);
                    }
                }
            }
        }
        return dist[sink] < INF;
    }

    void fixPot() {
        T INF = std::numeric_limits<T>::max();
        for(int i = 0; i < n; i++) {
            if(dist[i] < INF) pot[i] += dist[i];
        }
    }
};
```

5.9 Centroid

```
int sz[MAX];
bool erased[MAX];
vi grafo[MAX];

void dfs(int u, int p=-1){
    sz[u] = 1;
    for(int v: grafo[u]) if(v!=p and !erased[v]){
        dfs(v, u);
        sz[u] += sz[v];
    }
}

int centroid(int u, int p=-1, int size=-1){
    if(size==-1) size = sz[u];
    for(int v: grafo[u])
        if(v!=p and !erased[v] and sz[v]>size/2)
            return centroid(v, u, size);
    return u;
}

pii centroids(int u=1){ // idx 1
    dfs(u);
    int c1=centroid(u), c2=c1;
    for(int v: grafo[c1]) if(2*sz[v]==sz[u]) c2=v;
    return {c1, c2};
}
```

5.10 Kosaraju

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

void dfs(int u){
    vis[u] = 1;
    for(auto v: g[u]) if(!vis[v]) dfs(v);
    S.push(u);
}

void scc(int u, int c){
    vis[u] = 1; comp[u] = c;
    for(auto v: gi[u]) if(!vis[v]) scc(v, 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);
    }
}
```

5.11 Dinic

```
const int N = 300;

struct Dinic {
    struct Edge{
        int from, to; ll flow, cap;
    };
    vector<Edge> edge;

    vector<int> g[N];
    int ne = 0;
    int lvl[N], vis[N], pass;
    int qu[N], px[N], qt;

    ll run(int s, int sink, ll minE) {
        if(s == sink) return minE;

        ll ans = 0;

        for(; px[s] < (int)g[s].size(); px[s]++) {
            int e = g[s][ px[s] ];
            auto &v = edge[e], &rev = edge[e^1];
            if(lvl[v.to] != lvl[s]+1 || v.flow >= v.
cap)
                continue;           // v.cap - v.flow
 < lim
            ll tmp = run(v.to, sink,min(minE, v.cap-v
.flow));
            v.flow += tmp, rev.flow -= tmp;
            ans += tmp, minE -= tmp;
            if(minE == 0) break;
        }
        return ans;
    }
    bool bfs(int source, int sink) {
        qt = 0;
        qu[qt++] = source;
        lvl[source] = 1;
        vis[source] = ++pass;
        for(int i = 0; i < qt; i++) {
            int u = qu[i];
            px[u] = 0;
            if(u == sink) return true;
            for(auto& ed : g[u]) {
                auto v = edge[ed];
                if(v.flow >= v.cap || vis[v.to] ==
pass)
                    continue; // v.cap - v.flow < lim
                vis[v.to] = pass;
                lvl[v.to] = lvl[u]+1;
                qu[qt++] = v.to;
            }
        }
        return false;
    }
    ll flow(int source, int sink) {
        reset_flow();
        ll ans = 0;
        //for(lim = (1LL << 62); lim >= 1; lim /= 2)
        while(bfs(source, sink))
            ans += run(source, sink, LLINF);
        return ans;
    }
    void addEdge(int u, int v, ll c, ll rc) {
        Edge e = {u, v, 0, c};
        edge.pb(e);
        g[u].push_back(ne++);

        e = {v, u, 0, rc};
        edge.pb(e);
        g[v].push_back(ne++);
    }
    void reset_flow() {
        for(int i = 0; i < ne; i++)
            edge[i].flow = 0;
        memset(lvl, 0, sizeof(lvl));
        memset(vis, 0, sizeof(vis));
        memset(qu, 0, sizeof(qu));
        memset(px, 0, sizeof(px));
        qt = 0; pass = 0;
    }
};
```

5.12 Hld Vertice

```cpp
// Use it together with recursive_segtree
vector<vi> g(MAX, vi());
vi in(MAX), inv(MAX), sz(MAX);
vi peso(MAX), pai(MAX);
vi head(MAX), tail(MAX), h(MAX);

int tin;

void dfs(int u, int p=-1, int depth=0){
    sz[u] = 1; h[u] = depth;
    for(auto &v: g[u]) if(v != p){
        dfs(v, u, depth+1);
        pai[v] = u; sz[u] += sz[v];
        if (sz[v] > sz[g[u][0]] or g[u][0] == p) swap
    (v, g[u][0]);
    }
}
void build_hld(int u, int p = -1) {
    v[in[u] = tin++] = peso[u]; tail[u] = u;
    inv[tin-1] = u;
    for(auto &v: g[u]) if(v != p) {
        head[v] = (v == g[u][0] ? head[u] : v);
        build_hld(v, u);
    }
    if(g[u].size() > 1) tail[u] = tail[g[u][0]];
}
void init_hld(int root = 0) {
    dfs(root);
    tin = 0;
    build_hld(root);
    build();
}
void reset(){
    g.assign(MAX, vi());
    in.assign(MAX, 0), sz.assign(MAX, 0);
    peso.assign(MAX, 0), pai.assign(MAX, 0);
    head.assign(MAX, 0); tail.assign(MAX, 0);
    h.assign(MAX, 0); inv.assign(MAX, 0);

    t.assign(4*MAX, 0); v.assign(MAX, 0);
    lazy.assign(4*MAX, 0);
}
ll query_path(int a, int b) {
    if(in[a] < in[b]) swap(a, b);

    if(head[a] == head[b]) return query(in[b], in[a])
    ;
    return merge(query(in[head[a]], in[a]),
    query_path(pai[head[a]], b));
}
void update_path(int a, int b, int x) {
    if(in[a] < in[b]) swap(a, b);

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

6 Numeric

6.1 Lagrange Interpolation

```cpp
// Lagrange's interpolation O(n^2)
ld interpolate(vii d, ld x){
    ld y = 0;
    int n = d.size();
    for(int i=0;i<n;i++){
        ld yi = d[i].ss;
        for(int j=0;j<n;j++)
            if(j!=i)
                yi = yi*(x - d[j].ff)/(ld)(d[i].ff - d
    [j].ff);

        y += yi;
    }
    return y;
}

// O(n)

template<typename T = mint>
struct Lagrange {
    vector<T> y, den, l, r;
    int n;
    Lagrange(const vector<T>& _y) : y(_y), n(_y.size
    ()) {
        den.resize(n, 0);
        l.resize(n, 0); r.resize(n, 0);

        for (int i = 0; i < n; i++) {
            den[i] = ifac[i] * ifac[n - 1 - i];
            if ((n - 1 - i) % 2 == 1) den[i] = -den[i
    ];
        }
    }

    T eval(T x) {
        l[0] = 1;
        for (int i = 1; i < n; i++)
            l[i] = l[i-1] * (x + -T(i-1));

        r[n - 1] = 1;
        for (int i = n - 2; i >= 0; i--)
            r[i] = r[i+1] * (x + -T(i+1));

        T ans = 0;
        for (int i = 0; i < n; i++) {
            T num = l[i] * r[i];
            ans = ans + y[i] * num * den[i];
        }
        return ans;
    }
};
```

6.2 Newton Raphson

```cpp
// Newton Raphson

ld f(x){ return x*2 + 2; }
ld fd(x){ return 2; } // derivada

ld root(ld x){
    // while(f(x)>EPS)
    for(int i=0;i<20;i++){
        if(fd(x)<EPS)
            x = LLINF;
        else
            x = x - f(x)/fd(x);
    }
    return x;
}
```

6.3 Simpson's Formula

```
inline ld simpson(ld fl, ld fr, ld fmid, ld l, ld r){
    return (fl+fr+4*fmid)*(r-l)/6;
}

ld rsimpson(ld slr, ld fl, ld fr, ld fmid, ld l, ld r
    )
{
    ld mid = (l+r)/2;
    ld fml = f((l+mid)/2), fmr = f((mid+r)/2);
    ld slm = simpson(fl,fmid,fml,l,mid);
    ld smr = simpson(fmid,fr,fmr,mid,r);
    if(fabsl(slr-slm-smr) < EPS) return slm+smr; //
    aprox. good enough
    return rsimpson(slm,fl,fmid,fml,l,mid)+rsimpson(
    smr,fmid,fr,fmr,mid,r);
}

ld integrate(ld l, ld r)
{
    ld mid = (l+r)/2;
    ld fl = f(l), fr = f(r);
    ld fmid = f(mid);
    return rsimpson(simpson(fl,fr,fmid,l,r),fl,fr,
    fmid,l,r);
}
```

7 DP

7.1 Largest Ksubmatrix

```
int n, m;
int a[MAX][MAX];
// Largest K such that exists a block K*K with equal
    numbers
int largestKSubmatrix(){
    int dp[n][m];
    memset(dp, 0, sizeof(dp));

    int result = 0;
    for(int i = 0 ; i < n ; i++){
        for(int j = 0 ; j < m ; j++){
            if(!i or !j)
                dp[i][j] = 1;
            else if(a[i][j] == a[i-1][j] and
                    a[i][j] == a[i][j-1] and
                    a[i][j] == a[i-1][j-1])
                dp[i][j] = min(min(dp[i-1][j], dp[i][
    j-1]),
                               dp[i-1][j-1]) + 1;
            else dp[i][j] = 1;

            result = max(result, dp[i][j]);
        }
    }

    return result;
}
```

7.2 Dp Digitos

```
// dp de quantidade de numeros <= r com ate qt
    digitos diferentes de 0
ll dp(int idx, string& r, bool menor, int qt, vector<
    vector<vi>>& tab) {
    if(qt > 3) return 0;
    if(idx >= r.size()) {
        return 1;
    }
    if(tab[idx][menor][qt] != -1)
        return tab[idx][menor][qt];

    ll res = 0;
    for(int i = 0; i <= 9; i++) {
        if(menor or i <= r[idx]-'0') {
            res += dp(idx+1, r, menor or i < (r[idx]-
'0') , qt+(i>0), tab);
        }
    }

    return tab[idx][menor][qt] = res;
}
```

7.3 Partition Problem

```
// Partition Problem DP O(n2)
bool findPartition(vi &arr){
    int sum = 0;
    int n = arr.size();

    for(int i=0;i<n;i++)
        sum += arr[i];

    if(sum&1) return false;

    bool part[sum/2+1][n+1];

    for(int i=0;i<=n;i++)
        part[0][i] = true;

    for(int i=1;i<=sum/2;i++)
        part[i][0] = false;

    for(int i=1;i<=sum/2;i++){
        for(int j=1;j<=n;j++){
            part[i][j] = part[i][j-1];
            if(i >= arr[j-1])
                part[i][j] |= part[i - arr[j-1]][j
-1];
        }
    }
    return part[sum / 2][n];
}
```

7.4 Aliens

```
// Solves https://codeforces.com/contest/1279/problem
    /F

// dado um vetor de inteiros, escolha k subsegmentos
    disjuntos de soma áxmima
// em vez de rodar a dp[i][k] = melhor soma éat i
    usando k segmentos,
// vc roda uma dp[i] adicionando um custo W toda vez
    que usa um novo subsegmento,
// e faz busca ábinria nesse W pra achar o custo
    ímnimo que usa exatamente K intervalos

ll n, k, L;
pll check(ll w, vl& v){
    vector<pll> dp(n+1);
    dp[0] = {0,0};
    for(int i=1;i<=n;i++){
        dp[i] = dp[i-1];
        dp[i].ff += v[i];
        if(i-L>=0){
            pll t = {dp[i-L].ff + w, dp[i-L].ss + 1};
            dp[i] = min(dp[i], t);
        }
    }
```

```cpp

    return dp[n];
}

ll solve(vl v){
    ll l=-1, r=n+1, ans=-1;
    while(l<=r){
        ll mid = (l+r)/2;
        pll c = check(mid, v);
        if(c.ss <= k){
            r = mid - 1;
            ans = mid;
        }else{
            l = mid + 1;
        }
    }

    pll c = check(ans, v);

    if(ans < 0) return 0;

    // we can simply use k insted of c.ss ~magic~
    return c.ff - ans*k;
}

int32_t main()
{sws;

    string s;
    cin >> n >> k >> L;
    cin >> s;

    vl upper(n+1, 0), lower(n+1, 0);
    for(int i=0;i<n;i++)
        if('A'<= s[i] and s[i] <= 'Z')
            upper[i+1] = 1;
    for(int i=0;i<n;i++)
        if('a'<= s[i] and s[i] <= 'z')
            lower[i+1] = 1;

    cout << min(solve(lower),
                solve(upper)) << endl;

    return 0;
}
```

7.5 Lis

```cpp
multiset<int> S;
for(int i=0;i<n;i++){
    auto it = S.upper_bound(vet[i]); // low for inc
    if(it != S.end())
        S.erase(it);
    S.insert(vet[i]);
}
// size of the lis
int ans = S.size();

//////////////////////////// see that later
// https://codeforces.com/blog/entry/13225?#comment-180208

vi LIS(const vi &elements){
    auto compare = [&](int x, int y) {
        return elements[x] < elements[y];
    };
    set< int, decltype(compare) > S(compare);

    vi previous( elements.size(), -1 );
    for(int i=0; i<int( elements.size() ); ++i){
        auto it = S.insert(i).first;
        if(it != S.begin())
            previous[i] = *prev(it);
        if(*it == i and next(it) != S.end())
            S.erase(next(it));
    }

    vi answer;
    answer.push_back( *S.rbegin() );
    while ( previous[answer.back()] != -1 )
        answer.push_back( previous[answer.back()] );
    reverse( answer.begin(), answer.end() );
    return answer;
}
```

8 Strings

8.1 Manacher

```cpp
// O(n), d1 -> palindromo impar, d2 -> palindromo par (centro da direita)
void manacher(string &s, vi &d1, vi &d2) {
    int n = s.size();
    for(int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
        while(0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
            k++;
        }
        d1[i] = k--;
        if(i + k > r) {
            l = i - k;
            r = i + k;
        }
    }

    for(int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
        while(0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
            k++;
        }
        d2[i] = k--;
        if(i + k > r) {
            l = i - k - 1;
            r = i + k ;
        }
    }
}
```

8.2 Suffix Automaton

```cpp
const int SA = 2*N; // Node 1 is the initial node of the automaton
int last = 1;
int len[SA], link[SA];
array<int, 26> to[SA]; // maybe map<int, int>
int lastID = 1;
void push(int c) {
    int u = ++lastID;
    len[u] = len[last] + 1;

    int p = last;
    last = u; // update last immediately
    for (; p > 0 && !to[p][c]; p = link[p])
        to[p][c] = u;

    if (p == 0) { link[u] = 1; return; }

    int q = to[p][c];
    if (len[q] == len[p] + 1) { link[u] = q; return;
    }

    int clone = ++lastID;
    len[clone] = len[p] + 1;
    link[clone] = link[q];
    link[q] = link[u] = clone;
    to[clone] = to[q];
    for (int pp = p; to[pp][c] == q; pp = link[pp])
        to[pp][c] = clone;
}
```

8.3 Edit Distance

```
int edit_distance(int a, int b, string& s, string& t)
 {
    // indexado em 0, transforma s em t
    if(a == -1) return b+1;
    if(b == -1) return a+1;
    if(tab[a][b] != -1) return tab[a][b];

    int ins = INF, del = INF, mod = INF;
    ins = edit_distance(a-1, b, s, t) + 1;
    del = edit_distance(a, b-1, s, t) + 1;
    mod = edit_distance(a-1, b-1, s, t) + (s[a] != t[
    b]);

    return tab[a][b] = min(ins, min(del, mod));
}
```

8.4 Suffix Array

```
vi suffix_array(string s){
    s.pb('$');
    int n = s.size();

    vi p(n), c(n);
    vector< pair<char, int> > a(n);
    for(int i=0;i<n;i++) a[i] = {s[i], i};
    sort(a.begin(), a.end());

    for(int i=0;i<n;i++) p[i] = a[i].ss;
    c[p[0]]=0;
    for(int i=1;i<n;i++)
        c[p[i]] = c[p[i-1]] + (a[i].ff!=a[i-1].ff);

    int k=0;
    while((1<<k) < n){
        vector< pair<pii, int> > a(n);
        for(int i=0;i<n;i++)
            a[i] = {{c[i], c[(i+(1<<k))%n]}, i};
        sort(a.begin(), a.end());

        for(int i=0;i<n;i++) p[i] = a[i].ss;
        c[p[0]]=0;
        for(int i=1;i<n;i++)
            c[p[i]] = c[p[i-1]] + (a[i].ff!=a[i-1].ff
    );
        k++;
    }
    return p;
}
```

8.5 Lcs

```
string LCSubStr(string X, string Y)
{
    int m = X.size();
    int n = Y.size();

    int result = 0, end;
    int len[2][n];
    int currRow = 0;

    for(int i=0;i<=m;i++){
        for(int j=0;j<=n;j++){
            if(i==0 || j==0)
                len[currRow][j] = 0;
            else if(X[i-1] == Y[j-1]){
                len[currRow][j] = len[1-currRow][j-1]
 + 1;
                if(len[currRow][j] > result){
                    result = len[currRow][j];
                    end = i - 1;
                }
            }
            else
                len[currRow][j] = 0;
        }

        currRow = 1 - currRow;
    }

    if(result==0)
        return string();

    return X.substr(end - result + 1, result);
}
```

8.6 Eertree

```
// heavily based on https://ideone.com/YQX9jv,
// which adamant cites here https://codeforces.com/
    blog/entry/13959?#comment-196033
struct Eertree {
    int s[N];
    int n, last, sz;

    int len[N], link[N];
    int to[N][A];

    Eertree() {
        s[n++] = -1;
        len[1] = -1, link[1] = 1; // "backspace" root
 is 1
        len[0] = 0, link[0] = 1;  // empty root is 0
(to[backspace root][any char] = empty root)
        last = 2;
        sz = 2;
    }

    int get_link(int u) {
        while (s[n - len[u] - 2] != s[n - 1]) u =
link[u];
        return u;
    }

    void push(int c) {
        s[n++] = c;
        int p = get_link(last);
        if (!to[p][c]) {
            int u = ++sz;
            len[u] = len[p] + 2;
            link[u] = to[get_link(link[p])][c]; //
may be 0 (empty), but never 1 (backspace)
            to[p][c] = u;
        }
        last = to[p][c];
    }
};
```

8.7 Aho Corasick

```
// https://github.com/joseleite19/icpc-notebook/blob/
    master/code/string/aho_corasick.cpp
int to[N][A];
```

```cpp
int ne = 2, fail[N], term[N];
void add_string(const char *str, int id){
    int p = 1;
    for(int i = 0; str[i]; i++){
        int ch = str[i] - 'a'; // !
        if(!to[p][ch]) to[p][ch] = ne++;
        p = to[p][ch];
    }
    term[p]++;
}
void init(){
    for(int i = 0; i < ne; i++) fail[i] = 1;
    queue<int> q; q.push(1);
    int u, v; char c;
    while(!q.empty()){
        u = q.front(); q.pop();
        for(int i = 0; i < A; i++){
            if(to[u][i]){
                v = to[u][i]; q.push(v);
                if(u != 1){
                    fail[v] = to[ fail[u] ][i];
                    term[v] += term[ fail[v] ];
                }
            }
            else if(u != 1) to[u][i] = to[ fail[u] ][i];
            else to[u][i] = 1;
        }
    }
}
```

8.8 Kmp

```cpp
string p;
int neighbor[N];
int walk(int u, char c) { // leader after inputting `c`
    while (u != -1 && (u+1 >= (int)p.size() || p[u + 1] != c)) // leader doesn't match
        u = neighbor[u];
    return p[u + 1] == c ? u+1 : u;
}
void build() {
    neighbor[0] = -1; // -1 is the leftmost state
    for (int i = 1; i < (int)p.size(); i++)
        neighbor[i] = walk(neighbor[i-1], p[i]);
}
```

8.9 Z Func

```cpp
vi Z(string s) {
  int n = s.size();
  vi z(n);
  int x = 0, y = 0;
  for (int i = 1; i < n; i++) {
    z[i] = max(0, min(z[i - x], y - i + 1));
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
      x = i; y = i + z[i]; z[i]++;
    }
  }
  return z;
}
```

8.10 Lcsubseq

```cpp
// Longest Common Subsequence
string lcs(string x, string y){
    int n = x.size(), m = y.size();
    vector<vi> dp(n+1, vi(m+1, 0));

    for(int i=0;i<=n;i++){
        for(int j=0;j<=m;j++){
            if(!i or !j)
                dp[i][j]=0;
            else if(x[i-1] == y[j-1])
                dp[i][j]=dp[i-1][j-1]+1;
            else
                dp[i][j]=max(dp[i-1][j], dp[i][j-1]);
        }
    }

    // int len = dp[n][m];
    string ans="";

    // recover string
    int i = n-1, j = m-1;
    while(i>=0 and j>=0){
        if(x[i] == y[j]){
            ans.pb(x[i]);
            i--; j--;
        }else if(dp[i][j+1]>dp[i+1][j])
            i--;
        else
            j--;
    }

    reverse(ans.begin(), ans.end());

    return ans;
}
```

8.11 Hash

```cpp
struct Hash {
    vector<unordered_set<ll>> h;
    vector<ll> mods = {
        1000000009,1000000021,1000000033,1000000087,1000000093,
        1000000123,1000000181,1000000207,1000000223,1000000241,
    };
    ll p = 31;
    int num;

    Hash(int qt) {
        srand(time(0));
        num = qt;
        h.assign(num, unordered_set<ll>());
        random_shuffle(all(mods));
    }

    ll compute_hash(string const& s, ll p, ll m) {
        ll res = 0, p_pow = 1;

        for(char c : s) {
            res = ( res + (c-'a'+1) * p_pow) % m;
            p_pow = (p_pow * p) % m;
        }
        return res;
    }

    void add(string const& s) {
        forn(i, num) {
            ll value = compute_hash(s, p, mods[i]);
            h[i].insert(value);
        }
    }

    bool query(string const& s) {
        forn(i, num) {
            ll val = compute_hash(s, p, mods[i]);
            if(!h[i].count(val))
                return false;
        }
        return true;
    }
};
```

8.12 Suffix Array Radix

```cpp
void radix_sort(vector<pii>& rnk, vi& ind) {
    auto counting_sort = [](vector<pii>& rnk, vi& ind) {
        int n = ind.size(), maxx = -1;
        for(auto p : rnk) maxx = max(maxx, p.ff);

        vi cnt(maxx+1, 0), pos(maxx+1), ind_new(n);
        for(auto p : rnk) cnt[p.ff]++;
        pos[0] = 0;

        for(int i = 1; i <= maxx; i++) {
            pos[i] = pos[i-1] + cnt[i-1];
        }

        for(auto idx : ind) {
            int val = rnk[idx].ff;
            ind_new[pos[val]] = idx;
            pos[val]++;
        }

        swap(ind, ind_new);
    };

    for(int i = 0; i < (int)rnk.size(); i++) swap(rnk[i].ff, rnk[i].ss);
    counting_sort(rnk, ind);
    for(int i = 0; i < (int)rnk.size(); i++) swap(rnk[i].ff, rnk[i].ss);
    counting_sort(rnk, ind);
}

vi suffix_array(const string& s) {
    int n = s.size();
    vector<pii> rnk(n, mp(0, 0));
    vi ind(n);
    forn(i, n) {
        rnk[i].ff = (s[i] == '$') ? 0 : s[i]-'a'+1; // manter '$' como 0
        ind[i] = i;
    }

    for(int k = 1; k <= n; k = (k << 1)) {
        for(int i = 0; i < n; i++) {
            if(ind[i]+k >= n) {
                rnk[ind[i]].ss = 0;
            }
            else {
                rnk[ind[i]].ss = rnk[ind[i]+k].ff;
            }
        }
        radix_sort(rnk, ind); // sort(all(rnk), cmp) pra n*log(n), cmp com rnk[i] < rnk[j]

        vector<pii> tmp = rnk;
        tmp[ind[0]] = mp(1, 0); // rnk.ff comecar em 1 pois '$' eh o 0
        for(int i = 1; i < n; i++) {
            tmp[ind[i]].ff = tmp[ind[i-1]].ff;
            if(rnk[ind[i]] != rnk[ind[i-1]]) {
                tmp[ind[i]].ff++;
            }
        }
        swap(rnk, tmp);
    }
    return ind;
}

vi lcp_array(const string& s, const vi& sarray) {
    vi inv(s.size());
    for(int i = 0; i < (int)s.size(); i++) {
        inv[sarray[i]] = i;
    }
    vi lcp(s.size());
    int k = 0;
    for(int i = 0; i < (int)s.size()-1; i++) {
        int pi = inv[i];
        if(pi-1 < 0) continue;
        int j = sarray[pi-1];

        while(s[i+k] == s[j+k]) k++;
        lcp[pi] = k;
        k = max(k-1, 0);
    }

    return vi(lcp.begin()+1, lcp.end()); // LCP(i, j) = min(lcp[i], ..., lcp[j-1])
}
```

9 ED

9.1 Sparse Table

```cpp
int logv[MAX+1];
void make_log() {
    logv[1] = 0; // pre-computar tabela de log
    for (int i = 2; i <= MAX; i++)
        logv[i] = logv[i/2] + 1;
}
struct Sparse {
    int n;
    vector<vi> st;

    Sparse(vi& v) {
        n = v.size();
        int k = logv[n];
        st.assign(n+1, vi(k+1, 0));

        forn(i, n) {
            st[i][0] = v[i];
        }

        for(int j = 1; j <= k; j++) {
            for(int i = 0; i + (1 << j) <= n; i++) {
                st[i][j] = f(st[i][j-1], st[i + (1 << (j-1))][j-1]);
            }
        }
    }

    int f(int a, int b) {
        return min(a, b);
    }

    int query(int l, int r) {
        int k = logv[r-l+1];
        return f(st[l][k], st[r - (1 << k) + 1][k]);
    }
};


struct Sparse2d {
    int n, m;
    vector<vector<vi>> st;

    Sparse2d(vector<vi> mat) {
        n = mat.size();
        m = mat[0].size();
        int k = logv[min(n, m)];

        st.assign(n+1, vector<vi>(m+1, vi(k+1)));
        for(int i = 0; i < n; i++)
            for(int j = 0; j < m; j++)
                st[i][j][0] = mat[i][j];

        for(int j = 1; j <= k; j++) {
            for(int x1 = 0; x1 < n; x1++) {
                for(int y1 = 0; y1 < m; y1++) {
                    int delta = (1 << (j-1));
                    if(x1+delta >= n or y1+delta >= m
    ) continue;

                    st[x1][y1][j] = st[x1][y1][j-1];
                    st[x1][y1][j] = f(st[x1][y1][j],
    st[x1+delta][y1][j-1]);
                    st[x1][y1][j] = f(st[x1][y1][j],
    st[x1][y1+delta][j-1]);
                    st[x1][y1][j] = f(st[x1][y1][j],
    st[x1+delta][y1+delta][j-1]);
                }
            }
        }
    }

    // so funciona para quadrados
    int query(int x1, int y1, int x2, int y2) {
        assert(x2-x1+1 == y2-y1+1);
        int k = logv[x2-x1+1];
        int delta = (1 << k);

        int res = st[x1][y1][k];
        res = f(res, st[x2 - delta+1][y1][k]);
        res = f(res, st[x1][y2 - delta+1][k]);
        res = f(res, st[x2 - delta+1][y2 - delta+1][k
    ]);
        return res;
    }

    int f(int a, int b) {
        return a | b;
    }

};
```

9.2 Color Update

```
struct Color{
    set<ti> inter; // l, r, color
    vector<ti> update(int l, int r, int c){
        if(inter.empty()){ inter.insert({l, r, c});
    return {}; }
        vector<ti> removed;
        auto it = inter.lower_bound({l+1, 0, 0});
        it = prev(it);
        while(it != inter.end()){
            auto [l1, r1, c1] = *it;
            if((l<=l1 and l1<=r) or (l<=r1 and r1<=r)
     or (l1<=l and r<=r1)){
                removed.pb({l1, r1, c1});
            }else if(l1 > r)
                break;
            it = next(it);
        }
        for(auto [l1, r1, c1]: removed){
            inter.erase({l1, r1, c1});
            if(l1<l) inter.insert({l1, min(r1, l-1),
    c1});
            if(r<r1) inter.insert({max(l1, r+1), r1,
    c1});
        }
        if(c != 0) inter.insert({l, r, c});
        return removed;
    }

    ti query(int i){
        if(inter.empty()) return {INF, INF, INF};
        return *prev(inter.lower_bound({i+1, 0, 0}));
    }
};
```

9.3 Segtree Pa

```
int N;
vl t(4*MAX, 0);
vl v(MAX, 0);
vector<pll> lazy(4*MAX, {0,0});
// [x, x+y, x+2y...] //

inline ll merge(ll a, ll b){
    return a + b;
}

void build(int l=0, int r=N-1, int no=1){
    if(l == r){ t[no] = v[l]; return; }
    int mid = (l + r) / 2;
    build(l, mid, 2*no);
    build(mid+1, r, 2*no+1);
    t[no] = merge(t[2*no], t[2*no+1]);
}

inline pll sum(pll a, pll b){ return {a.ff+b.ff, a.ss
    +b.ss}; }

inline void prop(int l, int r, int no){
    auto [x, y] = lazy[no];
    if(x==0 and y==0) return;
    ll len = (r-l+1);
    t[no] += (x + x + y*(len-1))*len / 2;
    if(l != r){
        int mid = (l + r) / 2;
        lazy[2*no] = sum(lazy[2*no], lazy[no]);
        lazy[2*no+1] = sum(lazy[2*no+1], {x + (mid-l
    +1)*y, y});
    }
    lazy[no] = {0,0};
}

ll query(int a, int b, int l=0, int r=N-1, int no=1){
    prop(l, r, no);
    if(r<a or b<l) return 0;
    if(a<=l and r<=b) return t[no];
    int mid = (l + r) / 2;
    return merge(
        query(a, b, l, mid, 2*no),
        query(a, b, mid+1, r, 2*no+1)
    );
}

void update(int a, int b, ll x, ll y, int l=0, int r=
    N-1, int no=1){
    prop(l, r, no);
    if(r<a or b<l) return;
    if(a<=l and r<=b){
        lazy[no] = {x, y};
        prop(l, r, no);
        return;
    }
    int mid = (l + r) / 2;
    update(a, b, x, y, l, mid, 2*no);
    update(a, b, x + max((mid-max(l, a)+1)*y, 0LL), y
    , mid+1, r, 2*no+1);
    t[no] = merge(t[2*no], t[2*no+1]);
}
```

9.4 Segtree Iterative Lazy

```cpp
struct Segtree {
    vector<ll> seg, lazy;
    int n, LOG;

    Segtree(int n=0){
        this->n = n;
        LOG = ceil(log2(n));
        seg.assign(2*n, 0);
        lazy.assign(2*n, 0);
    }

    ll merge(ll a, ll b){
        return a + b;
    }
    void poe(int p, ll x, int tam, bool prop=1){
        seg[p] += x*tam;
        if(prop and p < n) lazy[p] += x;
    }
    void sobe(int p){
        for(int tam = 2; p /= 2; tam *= 2){
            seg[p] = merge(seg[2*p], seg[2*p+1]);
            if(lazy[p]!=0)
                poe(p, lazy[p], tam, 0);
        }
    }
    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(){
        for(int i = n-1; i; i--)
            seg[i] = merge(seg[2*i], seg[2*i+1]);
    }
    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 = merge(ret, seg[a]);
            if(b%2 == 0) ret = merge(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);
    }
};
```

9.5 Segtree Recursive

```cpp
int N;
vector<ll> t(4*MAX, 0);
vector<ll> v(MAX, 0);
vector<ll> lazy(4*MAX, 0);

inline ll merge(ll a, ll b){
    return a + b;
}

void build(int l=0, int r=N-1, int no=1){
    if(l == r){ t[no] = v[l]; return; }
    int mid = (l + r) / 2;
    build(l, mid, 2*no);
    build(mid+1, r, 2*no+1);
    t[no] = merge(t[2*no], t[2*no+1]);
}

void prop(int l, int r, int no){
    if(lazy[no] != 0){
        t[no] += lazy[no] * (r-l+1);
        if(l != r){
            lazy[2*no] += lazy[no];
            lazy[2*no+1] += lazy[no];
        }
        lazy[no] = 0;
    }
}

ll query(int a, int b, int l=0, int r=N-1, int no=1){
    prop(l, r, no);
    if(r<a or b<l) return 0;
    if(a<=l and r<=b) return t[no];
    int mid = (l + r) / 2;
    return merge(
        query(a, b, l, mid, 2*no),
        query(a, b, mid+1, r, 2*no+1)
    );
}

void update(int a, int b, ll x, int l=0, int r=N-1,
    int no=1){
    prop(l, r, no);
    if(r<a or b<l) return;
    if(a<=l and r<=b){
        lazy[no] += x;
        prop(l, r, no);
        return;
    }
    int mid = (l + r) / 2;
    update(a, b, x, l, mid, 2*no);
    update(a, b, x, mid+1, r, 2*no+1);
    t[no] = merge(t[2*no], t[2*no+1]);
}
```

9.6 Segtree Maxsubarray

```cpp
// Subarray with maximum sum
struct no{
    ll p, s, t, b; // prefix, suffix, total, best
    no(ll x=0): p(x), s(x), t(x), b(x){}
};

struct Segtree{
    vector<no> t;
    int n;

    Segtree(int n){
        this->n = n;
        t.assign(2*n, no(0));
    }

    no merge(no l, no r){
        no ans;
        ans.p = max(0LL, max(l.p, l.t+r.p));
        ans.s = max(0LL, max(r.s, l.s+r.t));
        ans.t = l.t+r.t;
        ans.b = max(max(l.b, r.b), l.s+r.p);
        return ans;
    }

    void build(){
        for(int i=n-1; i>0; i--)
```

```
            t[i]=merge(t[i<<1], t[i<<1|1]);
    }

    no query(int l, int r){ // idx 0
        no a(0), b(0);
        for(l+=n, r+=n+1; l<r; l>>=1, r>>=1){
            if(l&1)
                a=merge(a, t[l++]);
            if(r&1)
                b=merge(t[--r], b);
        }
        return merge(a, b);
    }

    void update(int p, int value){
        for(t[p+=n] = no(value); p >>= 1;)
            t[p] = merge(t[p<<1], t[p<<1|1]);
    }

};
```

9.7 Segtree Implicita Lazy

```
struct node{
    pll val;
    ll lazy;
    ll l, r;
    node(){
        l=-1;r=-1;val={0,0};lazy=0;
    }
};

node tree[40*MAX];
int id = 2;
ll N=1e9+10;

pll merge(pll A, pll B){
    if(A.ff==B.ff) return {A.ff, A.ss+B.ss};
    return (A.ff<B.ff ? A:B);
}

void prop(ll l, ll r, int no){
    ll mid = (l+r)/2;
    if(l!=r){
        if(tree[no].l==-1){
            tree[no].l = id++;
            tree[tree[no].l].val = {0, mid-l+1};
        }
        if(tree[no].r==-1){
            tree[no].r = id++;
            tree[tree[no].r].val = {0, r-(mid+1)+1};
        }
        tree[tree[no].l].lazy += tree[no].lazy;
        tree[tree[no].r].lazy += tree[no].lazy;
    }
    tree[no].val.ff += tree[no].lazy;
    tree[no].lazy=0;
}

void update(int a, int b, int x, ll l=0, ll r=2*N, ll
     no=1){
    prop(l, r, no);
    if(a<=l and r<=b){
        tree[no].lazy += x;
        prop(l, r, no);
        return;
    }
    if(r<a or b<l) return;
    int m = (l+r)/2;
    update(a, b, x, l, m, tree[no].l);
    update(a, b, x, m+1, r, tree[no].r);

    tree[no].val = merge(tree[tree[no].l].val, tree[
    tree[no].r].val);
}

pll query(int a, int b, int l=0, int r=2*N, int no=1)
    {
    prop(l, r, no);
    if(a<=l and r<=b) return tree[no].val;
    if(r<a or b<l) return {INF, 0};
    int m = (l+r)/2;
    int left = tree[no].l, right = tree[no].r;

    return tree[no].val = merge(query(a, b, l, m,
    left),
                                query(a, b, m+1, r,
    right));
}
```

9.8 Segtree Iterative

```
// Segment Tree Iterativa - Max

struct Segtree{
    vi t;
    int n;

    Segtree(int n){
        this->n = n;
        t.assign(2*n, 0);
    }

    int merge(int a, int b){
        return max(a, b);
    }

    void build(){
        for(int i=n-1; i>0; i--)
            t[i]=merge(t[i<<1], t[i<<1|1]);
    }

    int query(int l, int r){ // [l, r]
        int resl=-INF, resr=-INF;
        for(l+=n, r+=n+1; l<r; l>>=1, r>>=1){
            if(l&1) resl = merge(resl, t[l++]);
            if(r&1) resr = merge(t[--r], resr);
        }
        return merge(resl, resr);
    }

    void update(int p, int value){
        for(t[p+=n]=value; p >>= 1;)
            t[p] = merge(t[p<<1], t[p<<1|1]);
    }

};
```

9.9 Segtree Implicita

```
// SegTree Implicita O(nlogMAX)

struct node{
    int val;
    int l, r;
    node(int a=0, int b=0, int c=0){
        l=a;r=b;val=c;
    }
};

int idx=2; // 1-> root / 0-> zero element
node t[8600010];
int N;

```

```cpp
int merge(int a, int b){
    return a + b;
}

void update(int pos, int x, int i=1, int j=N, int no=1){
    if(i==j){
        t[no].val+=x;
        return;
    }
    int meio = (i+j)/2;

    if(pos<=meio){
        if(t[no].l==0) t[no].l=idx++;
        update(pos, x, i, meio, t[no].l);
    }
    else{
        if(t[no].r==0) t[no].r=idx++;
        update(pos, x, meio+1, j, t[no].r);
    }

    t[no].val=merge(t[t[no].l].val, t[t[no].r].val);
}

int query(int A, int B, int i=1, int j=N, int no=1){
    if(B<i or j<A)
        return 0;
    if(A<=i and j<=B)
        return t[no].val;

    int mid = (i+j)/2;

    int ansl = 0, ansr = 0;

    if(t[no].l!=0) ansl = query(A, B, i, mid, t[no].l);
    if(t[no].r!=0) ansr = query(A, B, mid+1, j, t[no].r);

    return merge(ansl, ansr);
}
```

9.10 Mergesorttree

```cpp
struct ST { // indexado em 0, O(n * log^2(n) )
    int size;
    vector<vl> v;

    vl f(vl a, vl& b) {
        vl res = a;
        for(auto val : b) {
            res.pb(val);
        }
        sort(all(res));
        return res;
    }

    ST(int n) {
        size = n;
        v.assign(4*size, vl());
    }

    void build(vector<ll>& a, int lx=0, int rx=size-1, int x=1) {
        if(lx==rx) {
            v[x].pb(a[lx]);
            return;
        }
        int m = (lx+rx)/2;
        build(a, lx, m, 2*x);
        build(a, m+1, rx, 2*x+1);
        v[x] = f(v[2*x], v[2*x+1]);
    }

    ll greaterequal(int l, int r, int k, int lx=0, int rx=size-1, int x=1) {
        if(r < lx or l > rx) return 0;
        if(l <= lx and rx <= r) {
            auto it = lower_bound(all(v[x]), k);
            return (v[x].end() - it);
        }
        int m = (lx + rx)/2;
        ll s1 = greaterequal(l, r, k, lx, m, 2*x);
        ll s2 = greaterequal(l, r, k, m+1, rx, 2*x+1);

        return s1 + s2;
    }

};
```

9.11 Segpersistente Mkthnum

```cpp
// kth number in range [l, r] if it was ordered
struct node{
    int val;
    int l, r;
    node(int a=-1, int b=-1, int c=0){
        val=c;l=a;r=b;
    }
};

node tree[8600010]; // 4*nlog(4*n) space = 8600010
int idx=0;

int build(int l, int r){
    if(l==r)
        return idx++;

    int mid = (l+r)/2;

    tree[idx].l = build(l, mid);
    tree[idx].r = build(mid+1, r);

    return idx++;
}

int update(int l, int r, int root, int e){
    if(l>e or r<e)
        return root;
    if(l==e and r==e){
        tree[idx]=node(-1, -1, tree[root].val+1);
        return idx++;
    }
    int mid = (l+r)/2;
    tree[idx]=node(update(l, mid, tree[root].l, e),
                   update(mid+1, r, tree[root].r, e), tree[root].val+1);
    return idx++;
}

int query(int l, int r, int root1, int root2, int k){
    while(l!=r)
    {
        int mid=(l+r)/2;
        if(tree[tree[root2].l].val-tree[tree[root1].l].val>=k)
        {
            r = mid;
            root1 = tree[root1].l;
            root2 = tree[root2].l;
        }else
        {
            l = mid+1;
            k-=tree[tree[root2].l].val-tree[tree[root1].l].val;
            root1 = tree[root1].r;
            root2 = tree[root2].r;
        }
    }
    return l;
}


int main()
{sws;

    int n, m, a, b, k;
    int v[MAX], aux[MAX];
    int root[MAX];

    cin >> n >> m;

    for(int i=0;i<n;i++){
        cin >> v[i]; aux[i]=v[i];
    }

    sort(v, v+n);

    map<int, int> comp;
    for(int i=0, j=0;i<n;i++)
        if(i==0 or v[i]!=v[i-1])
            comp[v[i]]=j++;

    root[0]=build(0, n-1);

    for(int i=1;i<=n;i++)
        root[i] = update(0, n-1, root[i-1], comp[aux[i-1]]);

    for(int i=0;i<m;i++){
        cin >> a >> b >> k;
        cout << v[query(0, n-1, root[a-1], root[b], k)] << endl;
    }

    return 0;
}
```

9.12 Cht

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

9.13 Bit Kth

```
struct FT {
    vector<int> bit;  // indexado em 1
    int n;

    FT(int n) {
        this->n = n + 1;
        bit.assign(n + 1, 0);
    }

    int kth(int x){
        int resp = 0;
        x--;
        for(int i=26;i>=0;i--){
            if(resp + (1<<i) >= n) continue;
            if(bit[resp + (1<<i)] <= x){
                x -= bit[resp + (1<<i)];
                resp += (1<<i);
            }
        }
        return resp + 1;
    }

    void upd(int pos, int val){
        for(int i = pos; i < n; i += (i&-i))
            bit[i] += val;
    }
};
```

9.14 Bit

```
struct FT {
    vi bit;  // indexado em 1
    int n;

    FT(int n) {
        this->n = n+1;
        bit.assign(n+2, 0);
    }

    int sum(int idx) {
        int ret = 0;
        for(++idx; idx > 0; idx -= idx & -idx)
            ret += bit[idx];
        return ret;
    }

    int sum(int l, int r) { // [l, r]
        return sum(r) - sum(l - 1);
    }

    void add(int idx, int delta) {
        for(++idx; idx < n; idx += idx & -idx)
            bit[idx] += delta;
    }
};
```

9.15 Virtual Tree

```cpp
bool initialized = false;
int original_root = 1;
const int E = 2 * N;
vector<int> vt[N]; // virtual tree edges
int in[N], out[N], T, t[E<<1];
void dfs_time(int u, int p = 0) {
    in[u] = ++T;
    t[T + E] = u;
    for (int v : g[u]) if (v != p) {
        dfs_time(v, u);
        t[++T + E] = u;
    }
    out[u] = T;
}

int take(int u, int v) { return in[u] < in[v] ? u : v
    ; }
bool cmp_in(int u, int v) { return in[u] < in[v]; }
void build_st() {
    in[0] = 0x3f3f3f3f;
    for (int i = E-1; i > 0; i--)
        t[i] = take(t[i<<1], t[i<<1|1]);
}

int query(int l, int r) {
    int ans = 0;
    for (l+=E, r+=E; l < r; l>>=1, r>>=1) {
        if (l&1) ans = take(ans, t[l++]);
        if (r&1) ans = take(ans, t[--r]);
    }
    return ans;
}

int get_lca(int u, int v) {
    if (in[u] > in[v]) swap(u, v);
    return query(in[u], out[v]+1);
}

int covers(int u, int v) { // does u cover v?
    return in[u] <= in[v] && out[u] >= out[v];
}

int build_vt(vector<int>& vnodes) {
    assert(initialized);

    sort(all(vnodes), cmp_in);
    int n = vnodes.size();
    for (int i = 0; i < n-1; i++) {
        int u = vnodes[i], v = vnodes[i+1];
        vnodes.push_back(get_lca(u, v));
    }
    sort(all(vnodes), cmp_in);
    vnodes.erase(unique(all(vnodes)), vnodes.end());

    for (int u : vnodes)
        vt[u].clear();

    stack<int> s;
    for (int u : vnodes) {
        while (!s.empty() && !covers(s.top(), u))
            s.pop();
        if (!s.empty()) vt[s.top()].push_back(u);
        s.push(u);
    }
    return vnodes[0]; // root
}

void initialize() {
    initialized = true;
    dfs_time(original_root);
    build_st();
}
```

9.16 Treap

```cpp
// source: https://github.com/victorsenam/caderno/
    blob/master/code/treap.cpp
//const int N = ; typedef int num;
num X[N]; int en = 1, Y[N], sz[N], L[N], R[N];
void calc (int u) { // update node given children
    info
    if(!u) return;
    sz[u] = sz[L[u]] + 1 + sz[R[u]];
    // code here, no recursion
}
void unlaze (int u) {
    if(!u) return;
    // code here, no recursion
}
void split_val(int u, num x, int &l, int &r) { // l
    gets <= x, r gets > x
    unlaze(u); if(!u) return (void) (l = r = 0);
    if(X[u] <= x) { split_val(R[u], x, l, r); R[u] =
    l; l = u; }
    else { split_val(L[u], x, l, r); L[u] = r; r = u;
     }
    calc(u);
}
void split_sz(int u, int s, int &l, int &r) { // l
    gets first s, r gets remaining
    unlaze(u); if(!u) return (void) (l = r = 0);
    if(sz[L[u]] < s) { split_sz(R[u], s - sz[L[u]] -
    1, l, r); R[u] = l; l = u; }
    else { split_sz(L[u], s, l, r); L[u] = r; r = u;
    }
    calc(u);
}
int merge(int l, int r) { // els on l <= els on r
    unlaze(l); unlaze(r); if(!l || !r) return l + r;
    int u;
    if(Y[l] > Y[r]) { R[l] = merge(R[l], r); u = l; }
    else { L[r] = merge(l, L[r]); u = r; }
    calc(u); return u;
}
void init(int n=N-1) { // XXX call before using other
     funcs
    for(int i = en = 1; i <= n; i++) { Y[i] = i; sz[i
    ] = 1; L[i] = R[i] = 0; }
    random_shuffle(Y + 1, Y + n + 1);
}
void insert(int &u, int it){
    unlaze(u);
    if(!u) u = it;
    else if(Y[it] > Y[u]) split_val(u, X[it], L[it],
    R[it]), u = it;
    else insert(X[it] < X[u] ? L[u] : R[u], it);
    calc(u);
}
void erase(int &u, num key){
    unlaze(u);
    if(!u) return;
    if(X[u] == key) u = merge(L[u], R[u]);
    else erase(key < X[u] ? L[u] : R[u], key);
    calc(u);
}
int create_node(num key){
    X[en] = key;
    sz[en] = 1;
    L[en] = R[en] = 0;
    return en++;
}
int query(int u, int l, int r){//0 index
    unlaze(u);
    if(u! or r < 0 or l >= sz[u]) return identity_element;
    if(l <= 0 and r >= sz[u] - 1) return subt_data[u];
    int ans = query(L[u], l, r);
    if(l <= sz[ L[u] ] and sz[ L[u] ] <= r)
        ans = max(ans, st[u]);
    ans = max(ans, query(R[u], l-sz[L[u]]-1, r-sz[L[u]]-1));
    return ans;
}
```

9.17 Minqueue

```
struct MinQ {
    stack<pair<ll,ll>> in;
    stack<pair<ll,ll>> out;

    void add(ll val) {
        ll minimum = in.empty() ? val : min(val, in.top().ss);
        in.push(mp(val, minimum));
    }

    ll pop() {
        if(out.empty()) {
            while(!in.empty()) {
                ll val = in.top().ff;
                in.pop();
                ll minimum = out.empty() ? val : min(val, out.top().ss);
                out.push({val, minimum});
            }
        }
        ll res = out.top().ff;
        out.pop();
        return res;
    }

    ll minn() {
        ll minimum = LLINF;
        if(in.empty() || out.empty())
            minimum = in.empty() ? (ll)out.top().ss : (ll)in.top().ss;
        else
            minimum = min((ll)in.top().ss, (ll)out.top().ss);

        return minimum;
    }

    ll size() {
        return in.size() + out.size();
    }
};
```