Team Reference Universidad de la Habana: UH Top

ICPC World Finals 2021

Team Members: Lázaro Raúl Iglesias Vera, David Manuel García Aguilera, Denis Gómez Cruz

Contents

1. Data Structures

1.1. Convex Hull Trick.

```
template<typename T>
T cross(complex<T> a, complex<T> b) { return imag(conj(a) * b); }
template<typename T>
T dot(complex<T> a, complex<T> b) { return real(conj(a) * b); }
struct __Query { static bool query; };
bool __Query::query = false;
template<typename T>
struct Point{
	complex<T> p;
	mutable function<const complex<T>*()> succ;
	bool operator<(const Point &rhs) const{
		const complex<T> &q = rhs.p;
		if (!__Query::query){
			if (real(p) != real(q))
				return real(p) < real(q);
			return imag(p) < imag(q);
		}
		const complex<T> *s = succ();
		if (!s) return false;
		return dot(p - *s, q) < 0;
	}
};
template<typename T, int turn>
struct half_hull : public set<Point<T>>{
	using set<Point<T>>::begin;
	using set<Point<T>>::insert;
	using set<Point<T>>::end;
	using set<Point<T>>::lower_bound;
	using set<Point<T>>::empty;
	using set<Point<T>>::erase;
	using typename set<Point<T>>::iterator;
	complex<T> extreme(const complex<T> &p) const{
		assert(!empty() && turn * imag(p) >= 0);
		__Query::query = true;
		auto pos = lower_bound(Point<T>{p});
		__Query::query = false;
		assert(pos != end());
		return pos->p;
	}
	void insert(const complex<T> &p){
		auto y = insert(Point<T>{p}).first;
		if (y == end()) return;
		y->succ = [=] { return next(y) == end() ? nullptr : &next(y)->p; };
		if (bad(y)) { erase(y); return; }
		while (y != begin() && bad(prev(y))) erase(prev(y));
		while (next(y) != end() && bad(next(y))) erase(next(y));
	}
private:
	bool bad(iterator y){
		if (y == begin() || y == end())
			return false;
		auto x = prev(y), z = next(y);
		if (z == end())
			return false;
		return cross(y->p - x->p, z->p - x->p) * turn >= 0;
	}
};
template<typename T>
struct convex_hull_trick{
	void insert(const complex<T> &p){
		lower_hull.insert(p);
		upper_hull.insert(p);
	}
	complex<T> extreme(const complex<T> &p) const{
		if (std::imag(p) < 0)
			return lower_hull.extreme(p);
		return upper_hull.extreme(p);
	}
private:
	half_hull<T, +1> upper_hull;
	half_hull<T, -1> lower_hull;
};
```

1.2. Order Statistics.

```
/* pb_ds_tree allowing:
		order_of_key(x) number of keys less that x
		find_by_order(k) find the k-th key
	Notes: Duplicated keys are not allowed, use pair<key, int> instead */
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
	tree_order_statistics_node_update>;
```

1.3. Treap.

```
// capacity MUST be correct if you use new_node
template<class node, bool persistent, int capacity = 0>
struct treap{
	inline int size(node *u) { return u ? u->nod.sz : 0; }
	inline void push(node *u){
		if (u->lazy()){
			if (u->l) u->l = clone(u->l), u->l->apply(u->lazy);
			if (u->r) u->r = clone(u->r), u->r->apply(u->lazy);
			u->lazy = typename node::lazy_container();
		}
	}
	node* update(node *u){
		u->nod = typename node::node_container(u->key);
		if (u->l) u->nod = node::merge(u->l->nod, u->nod);
```

```cpp
		if (u->r) u->nod = node::merge(u->nod, u->r->nod);
		return u;
	}
	// split for the kth first elements
	pair<node*, node*> split(node* u, int k){
		if (!u) return { u, u };
		u = clone(u); push(u);
		if (size(u->l) >= k){
			auto s = split(u->l, k);
			u->l = s.second;
			return { s.first, update(u) };
		}
		auto s = split(u->r, k - size(u->l) - 1);
		u->r = s.first;
		return { update(u), s.second };
	}
	node* merge(node *u, node *v){
		if (!u || !v) return u ? u : v;
		if (u->prio > v->prio){
			u = clone(u); push(u);
			u->r = merge(u->r, v);
			return update(u);
		}
		v = clone(v); push(v);
		v->l = merge(u, v->l);
		return update(v);
	}
	node* kth(node *u, int k){
		while (u && size(u->l) + 1 != k){
			push(u);
			if (size(u->l) >= k) u = u->l;
			else k -= size(u->l) + 1, u = u->r;
		}
		return u;
	}
	int less(node *u, const typename node::key_container &ky){
		int l = 0;
		while (u){
			push(u);
			if (u->key < ky) l += size(u->l) + 1, u = u->r;
			else u = u->l;
		}
		return l;
	}
	/*int pos(node *u) // require parents (set parent to NULL
	in update and fix child->p){
		int r = size(u->l);
		while (u->p != NULL){
			if (u->p->r == u)
				r += size(u->p->l) + 1;
			u = u->p;
		}
		return r;
	}*/
	node* clone(node *u){return !persistent ? u : new_node(*u);}
	vector<node> nodes;
	treap() { nodes.reserve(capacity); }
	node* new_node(node u){nodes.emplace_back(u);return &nodes.back();}
};
struct node{
	struct key_container{int x;} key;
	struct node_container{ int sz;
		node_container(const key_container &k = {}) : sz(1) {}
	} nod;
	struct lazy_container{ int add;
		bool operator()() { return add != 0; }
		lazy_container(int add = 0) : add(add) {}
	} lazy;
	static node_container merge(const node_container &lhs,
							const node_container &rhs){
		node_container x;
		x.sz = lhs.sz + rhs.sz;
		return x;
	}
	void apply(const lazy_container &p){
		key.x += p.add; lazy.add += p.add;
	}
	node *l, *r;
	int prio;
	node(const key_container &x) : key(x), nod(x){
		l = r = NULL; prio = randint(0, 1'000'000);
	}
};
```

1.4. PST.

```cpp
#define MAXP (ll)(2e6 + 5)
/// up to change
typedef int T;
struct node{ int l, r;T v; };
node pool[MAXP];
int actual;
int next(){
   actual++;
   return actual;
}
struct pst{
   vector<int> versions;
   int n;
   pst() : n(0) {}
   pst(int n) : n(n) {}
   pst(vector<T> &a) : n(a.size()) { versions.push_back(build(0, n - 1, a)); }
   T merge(T v1, T v2) { /* up to code*/ }
   void up(int p, T v) { /* up to code*/ }
   int build(int l, int r, vector<T> &a){
      int ans = next();
      if (l == r){
         pool[ans].v = a[l];
         return ans;
      }
      int mid = (l + r) >> 1;
      pool[ans].l = build(l, mid, a);
      pool[ans].r = build(mid + 1, r, a);
      pool[ans].v = merge(pool[pool[ans].l].v, pool[pool[ans].r].v);
      return ans;
   }
   int build(int l, int r, T *a){
      int ans = next();
```

```
        if (l == r){
            pool[ans].v = a[l];
            return ans;
        }
        int mid = (l + r) >> 1;
        pool[ans].l = build(l, mid, a);
        pool[ans].r = build(mid + 1, r, a);
        pool[ans].v = merge(pool[pool[ans].l].v, pool[pool[ans].r].v);
        return ans;
    }
    int clone(int p){
        int ans = next();
        pool[ans].l = pool[p].l;
        pool[ans].r = pool[p].r;
        pool[ans].v = pool[p].v;
        return ans;
    }
    void update(int ver, int pos, T v) {
        versions.push_back(update(versions[ver], 0, n - 1, pos, v));
    }
    void update(int pos, T v) {
        versions.push_back(update(versions.back(), 0, n - 1, pos, v));
        }
    int update(int p, int l, int r, int pos, T v){
        p = clone(p);
        if (r == l){
            up(p, v);
            return p;
        }
        int mid = (l + r) >> 1;
        if (pos <= mid) pool[p].l = update(pool[p].l, l, mid, pos, v);
        else pool[p].r = update(pool[p].r, mid + 1, r, pos, v);
        pool[p].v = merge(pool[pool[p].l].v, pool[pool[p].r].v);
        return p;
    }
    T query(int t, int l, int r) { return query(versions[t], 0, n - 1, l, r); }
    T query(int p, int l, int r, int L, int R){
        if (L <= l && r <= R) return pool[p].v;
        int mid = (l + r) >> 1;
        if (R <= mid) return query(pool[p].l, l, mid, L, R);
        if (L > mid) return query(pool[p].r, mid + 1, r, L, R);
        return merge(query(pool[p].l, l, mid, L, R),
                query(pool[p].r, mid + 1, r, L, R));
    }
};
```

2. Geometry

2.1. Antipodal Points.

```
vector<pair<int, int>> antipodal(const polygon &P){
      vector<pair<int, int>> ans;
      int n = P.size();
      if (P.size() == 2)
            ans.push_back({ 0, 1 });
      if (P.size() < 3)
            return ans;
      int q0 = 0;
      while (abs(area2(P[n - 1], P[0], P[NEXT(q0)]))
            > abs(area2(P[n - 1], P[0], P[q0]))) ++q0;
      for (int q = q0, p = 0; q != 0 && p <= q0; ++p){
            ans.push_back({ p, q });
            while (abs(area2(P[p], P[NEXT(p)], P[NEXT(q)]))
                        > abs(area2(P[p], P[NEXT(p)], P[q]))){
                  q = NEXT(q);
                  if (p != q0 || q != 0) ans.push_back({ p, q });
                  else return ans;
            }
            if (abs(area2(P[p], P[NEXT(p)], P[NEXT(q)]))
                        == abs(area2(P[p], P[NEXT(p)], P[q]))){
                  if (p != q0 || q != n - 1) ans.push_back({ p, NEXT(q) });
                  else ans.push_back({ NEXT(p), q });
            }
      }
      return ans;
}
```

2.2. Basics Complex.

```
typedef complex<double> point;
typedef vector<point> polygon;
#define NEXT(i) (((i) + 1) % n)
struct circle { point p; double r; };
struct line { point p, q; };
using segment = line;
const double eps = 1e-9;
// fix comparations on doubles with this two functions
int sign(double x) { return x < -eps ? -1 : x > eps; }
int dblcmp(double x, double y) { return sign(x - y); }
double dot(point a, point b) { return real(conj(a) * b); }
double cross(point a, point b) { return imag(conj(a) * b); }
double area2(point a, point b, point c) { return cross(b - a, c - a); } // cross
// where is c with respect to a->b
int ccw(point a, point b, point c){
      b -= a; c -= a;
      if (cross(b, c) > 0) return +1; // counter clockwise
      if (cross(b, c) < 0) return -1; // clockwise
      if (dot(b, c) < 0) return +2; // c--a--b on line
      if (dot(b, b) < dot(c, c)) return -2; // a--b--c on line
      return 0;
}
namespace std{
      bool operator<(point a, point b){
            if (a.real() != b.real())
                  return a.real() < b.real();
            return a.imag() < b.imag();
      }
```

```
}
// returns the angle abc (cos(x) = Va * Vb / |Va| * |Vb|)
double angle(point a, point b, point c) {
   a -= b, c -= b;
   double ang = (double)dot(a, c) / (sqrtl(norm(a)) * sqrtl(norm(c)));
   return acos(max(min(ang, 1.0), -1.0));
}
// contrary clock side direction
pair<double, double> rotate(double x, double y, double ang){
      ang = (acos(-1.0) * ang) / 180.0;
      return { x * cos(ang) - y * sin(ang), x * sin(ang) + y * cos(ang) };
}
// contrary clock side direction
point rotate(point x, ld ang){
      ang = (acos(-1.0) * ang) / 180.0;
      return x * polar(1.0, ang);//ang in radians...
}
int qua(point x){
      if (x.real() > 0 && x.imag() >= 0) return 0;
      if (x.real() <= 0 && x.imag() > 0) return 1;
      if (x.real() < 0 && x.imag() <= 0) return 2;
      return 3;
}
// assert((0, 0) not in v)
sort(v.begin(), v.end(), [](const point &x, const point &y){
      int qx = qua(x);
      int qy = qua(y);
      if (qx != qy) return qx < qy;
      //if (cross(x, y) == 0) return norm(x) < norm(y);
      return cross(x, y) > 0;
});
```

2.3. Circle.

```
// circle-circle intersection
vector<point> intersect(circle C, circle D){
      double d = abs(C.p - D.p);
      if (sign(d - C.r - D.r) > 0) return {}; // too far
      if (sign(d - abs(C.r - D.r)) < 0) return {}; // too close
      double a = (C.r*C.r - D.r*D.r + d*d) / (2*d);
      double h = sqrt(C.r*C.r - a*a);
      point v = (D.p - C.p) / d;
      if (sign(h) == 0) return {C.p + v*a}; // touch
      return {C.p + v*a + point(0,1)*v*h, // intersect
                  C.p + v*a - point(0,1)*v*h};
}
// circle-line intersection
vector<point> intersect(line L, circle C){
      point u = L.p - L.q, v = L.p - C.p;
      double a = dot(u, u), b = dot(u, v), c = dot(v, v) - C.r*C.r;
      double det = b*b - a*c;
      if (sign(det) < 0) return {}; // no solution
      if (sign(det) == 0) return {L.p - b/a*u}; // touch
      return {L.p + (-b + sqrt(det))/a*u,
                  L.p + (-b - sqrt(det))/a*u};
}
// circle tangents through point
vector<point> tangent(point p, circle C){
      double sin2 = C.r*C.r/norm(p - C.p);
      if (sign(1 - sin2) < 0) return {};
      if (sign(1 - sin2) == 0) return {p};
      point z(sqrt(1 - sin2), sqrt(sin2));
      return {p + (C.p - p)*conj(z), p + (C.p - p)*z};
}
bool incircle(point a, point b, point c, point p){
      a -= p; b -= p; c -= p;
      return norm(a) * cross(b, c)
                  + norm(b) * cross(c, a)
                  + norm(c) * cross(a, b) >= 0;
                  // < : inside, = cocircular, > outside
}
point three_point_circle(point a, point b, point c){
      point x = 1.0 / conj(b - a), y = 1.0 / conj(c - a);
      return (y - x) / (conj(x) * y - x * conj(y)) + a;
}
// Get the center of the circle with minimum ratio that enclose all points
circle min_enclosing_circle(vector<point> P){
      int n = P.size();
      shuffle(P.begin(), P.end(), random_device{});
      double r = 0.0; point p = P[0];
      for (int i = 1; i < n; ++i)
            if (dblcmp(abs(P[i] - p), r) > 0){
                  r = abs(P[0] - P[i]) * 0.5;
                  p = (P[0] + P[i]) * 0.5;
                  for (int j = 1; j < i; ++j)
                        if (dblcmp(abs(P[j] - p), r) > 0){
                              r = abs(P[i] - P[j]) * 0.5;
                              p = (P[i] + P[j]) * 0.5;
                              for (int k = 0; k < j; ++k)
                                    if (dblcmp(abs(P[k] - p), r) > 0){
                                          p=three_point_circle(P[i],P[j],P[k]);
                                          r = abs(p - P[i]);
                                    }
                        }
            }
      return {r, p};
}
```

2.4. Closest pair Points.

```
double closest_pair_points(vector<point> &P){
      auto cmp = [](point a, point b){
            return make_pair(a.imag(), a.real())
                        < make_pair(b.imag(), b.real());};
      int n = P.size();
      sort(P.begin(), P.end());
      set<point, decltype(cmp)> S(cmp);
      const double oo = 1e9; // adjust
      double ans = oo;
      for (int i = 0, ptr = 0; i < n; ++i){
            while (ptr < i && abs(P[i].real() - P[ptr].real()) >= ans)
                  S.erase(P[ptr++]);
```

```
            auto lo = S.lower_bound(point(-oo, P[i].imag() - ans - eps));
            auto hi = S.upper_bound(point(-oo, P[i].imag() + ans + eps));
            for (decltype(lo) it = lo; it != hi; ++it)
                  ans = min(ans, abs(P[i] - *it));
            S.insert(P[i]);
      }
      return ans;
}
```

2.5. Contains.

```
// Determine the position of a point relative to a polygon.
// (1) General Polygon
// (2) Convex Polygon
enum { OUT, ON, IN };
int contains(const polygon &P, const point &p){
   bool in = false;
   for (int i = 0, n = P.size(); i < n; ++i){
      point a = P[i] - p, b = P[NEXT(i)] - p;
      if (imag(a) > imag(b)) swap(a, b);
      if (imag(a) <= 0 && 0 < imag(b))
         if (cross(a, b) < 0) in = !in;
      if (cross(a, b) == 0 && dot(a, b) <= 0)
         return ON;
   }
   return in ? IN : OUT;
}
struct convex_container{
   polygon pol;
   // Polygon MUST be in counter clockwise order
   convex_container(polygon p) : pol(p){
      int pos = 0;
      for (int i = 1; i < p.size(); ++i)
         if (p[i].imag() < p[pos].imag() ||
            (p[i].imag() == p[pos].imag() && p[i].real() < p[pos].real()))
               pos = i;
      rotate(pol.begin(), pol.begin() + pos, pol.end());
   }
   bool contains(point p){
      point c = pol[0];
      if (p.imag() < c.imag() || cross(pol.back() - c, p - c) > 0)
         return false;
      int lo = 1, hi = pol.size() - 1;
      while (lo + 1 < hi){
         int m = (lo + hi) / 2;
         if (cross(pol[m] - c, p - c) >= 0) lo = m;
         else hi = m;
      }
      return abs(cross(c, pol[lo], pol[lo + 1]))
            - abs(cross(p, c, pol[lo]))
               - abs(cross(p, c, pol[lo + 1]))
                  - abs(cross(p, pol[lo], pol[lo + 1])) == 0;
   }
};
```

2.6. Line Segment Intersections.

```
bool intersectLL(const line &l, const line &m){
      return abs(cross(l.q - l.p, m.q - m.p)) > eps || // non-parallel
                  abs(cross(l.q - l.p, m.p - l.p)) < eps; // same line
}
bool intersectLS(const line &l, const segment &s){
      return cross(l.q - l.p, s.p - l.p) * // s[0] is left of l
                  cross(l.q - l.p, s.q - l.p) < eps; // s[1] is right of l
}
bool intersectLP(const line &l, const point &p){
      return abs(cross(l.q - p, l.p - p)) < eps;
}
bool intersectSS(const segment &s, const segment &t){
      return ccw(s.p, s.q, t.p) * ccw(s.p, s.q, t.q) <= 0
                  && ccw(t.p, t.q, s.p) * ccw(t.p, t.q, s.q) <= 0;
}
bool intersectSP(const segment &s, const point &p){
      return abs(s.p - p) + abs(s.q - p) - abs(s.q - s.p) < eps;
      // triangle inequality
      return min(real(s.p), real(s.q)) <= real(p)
                  && real(p) <= max(real(s.p), real(s.q))
                  && min(imag(s.p), imag(s.q)) <= imag(p)
                  && imag(p) <= max(imag(s.p), imag(s.q))
                  && cross(s.p - p, s.q - p) == 0;
}
point projection(const line &l, const point &p){
      double t = dot(p - l.p, l.p - l.q) / norm(l.p - l.q);
      return l.p + t * (l.p - l.q);
}
point reflection(const line &l, const point &p){
      return p + 2.0 * (projection(l, p) - p);
}
double distanceLP(const line &l, const point &p){
      return abs(p - projection(l, p));
}
double distanceLL(const line &l, const line &m){
      return intersectLL(l, m) ? 0 : distanceLP(l, m.p);
}
double distanceLS(const line &l, const line &s){
      if (intersectLS(l, s)) return 0;
      return min(distanceLP(l, s.p), distanceLP(l, s.q));
}
double distanceSP(const segment &s, const point &p){
      const point r = projection(s, p);
      if (intersectSP(s, r)) return abs(r - p);
      return min(abs(s.p - p), abs(s.q - p));
}
double distanceSS(const segment &s, const segment &t){
      if (intersectSS(s, t)) return 0;
      return min(min(distanceSP(s, t.p), distanceSP(s, t.q)),
                  min(distanceSP(t, s.p), distanceSP(t, s.q)));
}
point crosspoint(const line &l, const line &m){
      double A = cross(l.q - l.p, m.q - m.p);
      double B = cross(l.q - l.p, l.q - m.p);
```

```cpp
      if (abs(A) < eps && abs(B) < eps)
            return m.p; // same line
      if (abs(A) < eps)
            assert(false); // !!!PRECONDITION NOT SATISFIED!!!
      return m.p + B / A * (m.q - m.p);
}
point intersect(line l1, line l2){
   if(l1.b == 0 && l2.b == 0)
      return {oo, oo};
   if(l2.b == 0) swap(l1, l2);
   if(l1.b == 0){
      fraction x = -l1.c / l1.a;
      fraction y = (-l2.c - l2.a * x) / l2.b;
      return {x, y};
   }
   l1.a = l1.a / l1.b * l2.b;
   l1.c = l1.c / l1.b * l2.b;
   l1.b = l2.b;
   line l3 = {l1.a - l2.a, 0, l1.c - l2.c};
   if(l3.a == 0) return {oo, oo};
   fraction x = -l3.c / l3.a;
   fraction y = (-l2.c - l2.a * x) / l2.b;
   return {x, y};
}
```

2.7. Minkowski.

```cpp
// The sum of Minkowski of two sets A and B is the set C =
// { a + b : a e A,b e B }. // each element in a set is a vector
// It can be proven that if A and B are convex polygons then C
// will also be a convex polygon.
// Minkowski sum of two convex polygons. O(n + m)
// Note: Polygons MUST be counterclockwise
polygon minkowski(const polygon &ps, const polygon &qs){
      vector<point> rs;
      int i = distance(ps.begin(), min_element(ps.begin(), ps.end()));
      int j = distance(qs.begin(), min_element(qs.begin(), qs.end()));
      do{
            rs.emplace_back(ps[i] + qs[j]);
            int in = i + 1, jn = j + 1;
            if (in == ps.size()) in = 0;
            if (jn == qs.size()) jn = 0;
            int s = sign(cross(ps[i] - ps[in], qs[j] - qs[jn]));
            if (s >= 0) i = in;
            if (s <= 0) j = jn;
      }
      while (rs[0] != ps[i] + qs[j]);
      return rs;
}
```

2.8. Rectangle union.

```cpp
// Note:Rectangle contains coordinates of two opposite corners(xl<=xh,yl<=yh)
// Complexity: O(n log n)
typedef long long ll;
struct rectangle{ ll xl, yl, xh, yh;};
ll rectangle_area(vector<rectangle> &rs){
      vector<ll> ys; // coordinate compression
      for (auto r : rs){ ys.push_back(r.yl); ys.push_back(r.yh); }
      sort(ys.begin(), ys.end());
      ys.erase(unique(ys.begin(), ys.end()), ys.end());
      int n = ys.size(); // measure tree
      vector<ll> C(8 * n), A(8 * n);
      function<void(int, int, int, int, int, int)> aux =
                  [&](int a, int b, int c, int l, int r, int k){
                        if ((a = max(a,l)) >= (b = min(b,r))) return;
                        if (a == l && b == r) C[k] += c;
                        else{
                              aux(a, b, c, l, (l+r)/2, 2*k+1);
                              aux(a, b, c, (l+r)/2, r, 2*k+2);
                        }
                        if (C[k]) A[k] = ys[r] - ys[l];
                        else A[k] = A[2*k+1] + A[2*k+2];
                  };
      struct event{ ll x, l, h, c;}; // plane sweep
      vector<event> es;
      for (auto r : rs){
            int l = lower_bound(ys.begin(), ys.end(), r.yl) - ys.begin();
            int h = lower_bound(ys.begin(), ys.end(), r.yh) - ys.begin();
            es.push_back({ r.xl, l, h, +1 });
            es.push_back({ r.xh, l, h, -1 });
      }
      sort(es.begin(), es.end(), [](event a, event b)
                  {return a.x != b.x ? a.x < b.x : a.c > b.c;});
      ll area = 0, prev = 0;
      for (auto &e : es){
            area += (e.x - prev) * A[0];
            prev = e.x;
            aux(e.l, e.h, e.c, 0, n, 0);
      }
      return area;
}
```

2.9. Polygon Area.

```cpp
//Double of the signed area of a polygon
double area2(const polygon &P){
      double A = 0;
      for (int i = 0, n = P.size(); i < n; ++i)
            A += cross(P[i], P[NEXT(i)]);
      return A;
}
```

2.10. Mass Center.

```
/*
	Centroid of a (possibly nonconvex) polygon
	Coordinates must be listed in a cw or ccw.

	Tested: SPOJ STONE
	Complexity: O(n)
*/

point centroid(const polygon &P)
{
	point c(0, 0);
	double scale = 3.0 * area2(P); // area2 = 2 * polygon_area
	for (int i = 0, n = P.size(); i < n; ++i)
	{
		int j = NEXT(i);
		c = c + (P[i] + P[j]) * (cross(P[i], P[j]));
	}
	return c / scale;
}
```

2.11. Convex Cut.

```
// Cut a convex polygon by a line and
// return the part to the left of the line
// Complexity: O(n)
polygon convex_cut(const polygon &P, const line &l){
	polygon Q;
	for (int i = 0, n = P.size(); i < n; ++i){
		point A = P[i], B = P[(i + 1) % n];
		if (ccw(l.p, l.q, A) != -1) Q.push_back(A);
		if (ccw(l.p, l.q, A) * ccw(l.p, l.q, B) < 0)
			Q.push_back(crosspoint((line){ A, B }, l));
	}
	return Q;
}
```

2.12. Convex Hull.

```
vector<point> convex_hull(vector<point> v){
	int n = v.size(), k = 0;
	vector<point> ch(2 * n);
	sort(v.begin(), v.end(), cmp);
   for (ll i = k = 0; i < n; ch[k++] = v[i++])
	while (k > 1 && cross(ch[k - 2], ch[k - 1], v[i]) <= 0) --k;
   for (ll i = n - 2, t = k; i >= 0; ch[k++] = v[i--])
	while (k > t && cross(ch[k - 2], ch[k - 1], v[i]) <= 0) --k;
   ch.resize(k - (k > 1));
	return ch;
}
```

2.13. Polygon Width.

```
// Compute the width (minimum) of a convex polygon
// Complexity: O(n)
const int oo = 1e9; // adjust
double check(int a, int b, int c, int d, const polygon &P){
	for (int i = 0; i < 4 && a != c; ++i){
		if (i == 1) swap(a, b);
		else swap(c, d);
	}
	if (a == c){ // a admits a support line parallel to bd
		double A = abs(area2(P[a], P[b], P[d]));
		// double of the triangle area
		double base = abs(P[b] - P[d]);
		// base of the triangle abd
		return A / base;
	}
	return oo;
}
double polygon_width(const polygon &P){
	if (P.size() < 3)
		return 0;
	auto pairs = antipodal(P);
	double best = oo;
	int n = pairs.size();
	for (int i = 0; i < n; ++i){
		double tmp = check(pairs[i].first, pairs[i].second,
				pairs[NEXT(i)].first, pairs[NEXT(i)].second, P);
		best = min(best, tmp);
	}
	return best;
}
```

2.14. Semiplane Intersection.

```
// Check whether there is a point in the intersection of
// several semi-planes. If p lies in the border of some
// semiplane it is considered to belong to the semiplane.
// Complexity:
// O(n**2)
// Expected running time: O(n)
```

```
bool intersect(vector<line> semiplane){
   function<bool(line&,point&)> side = [](line &l, point &p){
      // IMPORTANT: point p belongs to semiplane defined by l
      // iff p it's clockwise respect to directed segment <l.p, l.q>
      // i.e. (non negative cross product)
      return cross( l.q - l.p, p - l.p ) >= 0;
   };
   function<bool(line&, line&, point&)> crosspoint = []
   (const line &l, const line &m, point &x){
      double A = cross(l.q - l.p, m.q - m.p);
      double B = cross(l.q - l.p, l.q - m.p);
      if (abs(A) < eps) return false;
      x = m.p + B / A * (m.q - m.p);
      return true;
   };
   int n = (int)semiplane.size();
   random_shuffle(semiplane.begin(), semiplane.end());
   point cent(0, 1e9);
   for (int i = 0; i < n; ++i){
      line &S = semiplane[i];
      if (side(S, cent)) continue;
      point d = S.q - S.p; d /= abs( d );
      point A = S.p - d * 1e8, B = S.p + d * 1e8;
      for (int j = 0; j < i; ++j){
         point x;
         line &T = semiplane[j];
         if (crosspoint(T, S, x)){
            int cnt = 0;
            if (!side(T, A)){
               A = x;
               cnt++;
            }
            if (!side(T, B)){
               B = x;
               cnt++;
            }
            if (cnt == 2)
               return false;
         }
         else{
            if (!side(T, A)) return false;
         }
      }
      if (imag(B) > imag(A)) swap(A, B);
      cent = A;
   }
   return true;
}
```

3. Graph

3.1. Articulation Points.

```
void tarjan(const vector<vector<int>> &adj){
   int n = adj.size(), t;
   vector<int> num(n), low(n), S;
   vector<bool> arts(n);
   vector<pair<int, int>> bridges;
   vector<vector<int>> comps; // biconnected components
   function<void(int, int)> dfs = [&](int u, int p){
      num[u] = low[u] = ++t;
      S.push_back(u);
      for (int v : adj[u])
         if (v != p){
            if (!num[v]){
               dfs(v, u);
               low[u] = min(low[u], low[v]);
               if (num[u] <= low[v]){
                  if (num[u] != 1 || num[v] > 2)
                        arts[u] = true;
                  for (comps.push_back({u});
                        comps.back().back() != v;
                        S.pop_back())
                     comps.back().push_back(S.back());
               }
               if (num[u] < low[v]) bridges.push_back({ u, v });
            }
            else low[u] = min(low[u], num[v]);
         }
   };
   for (int u = 0; u < n; ++u)
      if (!num[u]) t = 0, dfs(u, -1);
}
```

3.2. Bipartite Matching.

```
// Bipartite Matching
// Complexity: O(E sqrt V)
// ** using only dfs marking visited nodes and cleaning
// them in bfs(in O(1)) might be faster **
struct hopcroft_karp{
      int n, m;
      vector<vector<int>> adj;
      vector<int> mu, mv, level, que;
      // u is matched with mu[u] and v with mv[v], -1 if no match
      hopcroft_karp(int n, int m) : n(n), m(m), adj(n),
            mu(n, -1), mv(m, -1), level(n), que(n) {}
      void add_edge(int u, int v) { adj[u].push_back(v); }
      void bfs(){
            int qf = 0, qt = 0;
            for (int u = 0; u < n; ++u){
                  if (mu[u] == -1) que[qt++] = u, level[u] = 0;
                  else level[u] = -1;
            }
            for (; qf < qt; ++qf){
                  int u = que[qf];
```

```
                for (auto w : adj[u]){
                    int v = mv[w];
                    if (v != -1 && level[v] == -1)
                        que[qt++] = v, level[v] = level[u] + 1;
                }
            }
        }
        bool dfs(int u){
            for (auto w : adj[u]){
                int v = mv[w];
                if (v == -1 || (level[v] == level[u] + 1 && dfs(v)))
                // split this line for speed up :)
                    return mu[u] = w, mv[w] = u, true;
            }
            return false;
        }
        int max_matching(){
            int match = 0;
            for (int c = 1; bfs(), c; match += c)
                for (int u = c = 0; u < n; ++u)
                    if (mu[u] == -1) c += dfs(u);
            return match;
        }
        pair<vector<int>, vector<int>> min_vertex_cover(){
            max_matching();
            vector<int> L, R, inR(m);
            for (int u = 0; u < n; ++u){
                if (level[u] == -1) L.push_back(u);
                else if (mu[u] != -1) inR[mu[u]] = true;
            }
            for (int v = 0; v < m; ++v)
                if (inR[v]) R.push_back(v);
            return { L, R };
        }
};
```

3.3. Centroid Decomposition.

```
vector<int> g[MAXN], nodes;
bool mk[MAXN]; //marks centroid taken before
int sz[MAXN], mx[MAXN];
void dfs1(int u, int p){
    sz[u] = 1;
    nodes.push_back(u);
    for (auto v : g[u])
        if (!mk[v]){
            dfs1(v, u);
            sz[u] += sz[v];
            mx[u] = max(sz[v], mx[u]);
        }
}
void dfs2(int u, int p){
    nodes.push_back(u);
    for (auto v : g[u])
        if (!mk[v]) dfs2(v, u);
}
void solve(int u){
    dfs1(u, -1);
    int cent = -1, cant = nodes.size() / 2;
    for (auto v : nodes){
        if (cent == -1 && sz[v] >= cant && mx[v] <= cant) cent = v;
        sz[v] = mx[v] = 0;
    }
    nodes.clear();
    mk[cent] = 1;
    for (auto v : g[cent]){
        if (mk[v]) continue;
        dfs2(v, cent);
        for (auto y : nodes) ;
        nodes.clear();
    }
    for (auto v : g[cent]) if (!mk[v]) solve(v);
}
```

3.4. Dinic.

```
// Maximum Flow
// Complexity: O(n^2 * m) faster in most cases
//  O(min(n^(2/3), m^(1/2)) * m) in networks with unit capacities
//  O(n^(1/2) * m) in bipartite networks
// ** be careful if lb(flow with lower bound) with flow_type **
template <typename C, typename R = C, bool lb = false>
struct dinic{
    typedef C flow_type;
    typedef R result_type;
    static const flow_type oo = std::numeric_limits<flow_type>::max();
    struct edge{
        int src, dst, rev;
        flow_type cap, flow;
        edge(int src, int dst, int rev, flow_type cap,
             flow_type flow) : src(src), dst(dst), rev(rev),
                               cap(cap), flow(flow) {}
    };
    dinic(int n) : adj(n + 2 * lb), que(n + 2 * lb),
                   level(n + 2 * lb), edge_pos(n + 2 * lb) {}
    int add_edge(int src, int dst, flow_type cap,
                 flow_type rcap = 0){ // if lb rcap is low
        adj[src].emplace_back(src, dst, (int)adj[dst].size(),
                              cap, lb ? rcap : 0);
        if (src == dst) adj[src].back().rev++;
        adj[dst].emplace_back(dst, src,
                              (int)adj[src].size() - 1,
                              lb ? 0 : rcap, 0);
        return (int)adj[src].size() - 1 - (src == dst);
    }
    inline bool side_of_S(int u) { return level[u] == -1; }
    result_type max_flow(int source, int sink){
        result_type flow = 0;
        while (true){
            int front = 0, back = 0;
            std::fill(level.begin(), level.end(), -1);
            for (level[que[back++] = sink] = 0;
```

```
                front < back && level[source] == -1;++front){
                int u = que[front];
                for (const edge &e : adj[u])
                    if (level[e.dst] == -1
                        && rev(e).flow < rev(e).cap)
                        level[que[back++] = e.dst] = 1 + level[u];
            }
            if (level[source] == -1) break;
            std::fill(edge_pos.begin(), edge_pos.end(), 0);
            std::function<flow_type(int, flow_type)> find_path
                                    = [&](int from, flow_type res){
                if (from == sink) return res;
                for (int &ept = edge_pos[from];
                     ept < (int)adj[from].size();++ept){
                    edge &e = adj[from][ept];
                    if (e.flow == e.cap
                      || level[e.dst] + 1 != level[from]) continue;
                    flow_type push
                      = find_path(e.dst, std::min(res, e.cap - e.flow));
                    if (push > 0){
                        e.flow += push;
                        rev(e).flow -= push;
                        if (e.flow == e.cap) ++ept;
                        return push;
                    }
                }
                return static_cast<flow_type>(0);
            };
            for (flow_type f; (f = find_path(source, oo)) > 0;) flow += f;
        }
        return flow;
    }
    result_type max_flow_lb(int source, int sink){
        int n = adj.size() - 2;
        vector<flow_type> delta(n + 2);
        for (int u = 0; u < n; ++u)
            for (auto &e : adj[u]){
                delta[u] -= e.flow;
                delta[e.dst] += e.flow;
            }
        result_type sum = 0;
        int s = n, t = n + 1;
        for (int u = 0; u < n; ++u){
            if (delta[u] > 0){
                add_edge(s, u, delta[u], 0);
                sum += delta[u];
            }
            else if (delta[u] < 0) add_edge(u, t, -delta[u], 0);
        }
        add_edge(sink, source, oo, 0);
        if (max_flow(s, t) != sum) return -1; // no solution
        result_type flow = adj[sink].back().flow;
        adj[sink].pop_back();
        adj[source].pop_back();
        return flow + max_flow(source, sink);
    }
    std::vector<std::vector<edge>> adj;
private:
    std::vector<int> que;
    std::vector<int> level;
    std::vector<int> edge_pos;
    inline edge &rev(const edge &e){return adj[e.dst][e.rev];}
};
```

3.5. Eulerian graph.

```
// Euler path undirected (path to use once all edges)
// the degree of all nodes must be even (euler cycle)
// or only exists two odd nodes (euler path)
vector<int> euler_path(const vector<vector<pair<int, int>>> &G, int s = 0){
    int n = G.size(), odd = 0, m = 0;
    for (int i = 0; i < n; ++i){
        odd += G[i].size() & 1;
        m += G[i].size();
    }
    vector<int> path;
    if (odd == 0 || (odd == 2 && (G[s].size() & 1) == 1)){
        vector<int> pos(n);
        vector<bool> mark(m / 2);
        function<void(int)> visit = [&](int u){
            for (int v, id; pos[u] < G[u].size(); ){
                tie(v, id) = G[u][pos[u]++];
                if (!mark[id]){
                    mark[id] = true;
                    visit(v);
                }
            }
            path.push_back(u);
        };
        visit(s);
        reverse(path.begin(), path.end());
        if (path.size() != m / 2 + 1) path.clear();
    }
    return path;
}
// Euler path directed (path to use once all edges)
// the in-degree - out-degree == 0 for all nodes (euler cycle)
// or only exists two nodes with |in-degree - out-degree| == 1 (euler path)
vector<int> euler_path(vector<vector<int>> G, int s = 0){
    int n = G.size(), m = 0;
    vector<int> deg(n);
    for (int u = 0; u < n; ++u){
        m += G[u].size();
        for (auto v : G[u])
            --deg[v]; // in-deg
        deg[u] += G[u].size(); // out-deg
    }

    vector<int> path;
    int k = n - count(deg.begin(), deg.end(), 0);
    if (k == 0 || (k == 2 && deg[s] == 1)){
        function<void(int)> visit = [&](int u){
            while (!G[u].empty()){
                int v = G[u].back();
                G[u].pop_back();
                visit(v);
            }
            path.push_back(u);
```

```
        };
        visit(s);
        reverse(path.begin(), path.end());
            if (path.size() != m + 1) path.clear();
        }
        return path;
    }
```

3.6. Heavy light decomposition.

```
struct heavy_light{
    int heavy[MAXN], root[MAXN], depth[MAXN];
    int pos[MAXN], ipos[MAXN], parent[MAXN],n;
    int dfs(int s, int f, vector<int> *G){
        parent[s] = f, heavy[s] = -1;
        int size = 1, maxSubtree = 0;
        for (auto u : G[s])
            if (u != f){
                depth[u] = depth[s] + 1;
                int subtree = dfs(u, s, G);
                if (subtree > maxSubtree)
                    heavy[s] = u, maxSubtree = subtree;
                size += subtree;
            }
        return size;
    }
    void go(vector<int> *G, int _n){
        n = _n;
        int ROOT = 0;
        depth[ROOT] = 0;
        dfs(ROOT, -1, G);
        vector<pii> nodes;
        for (int i = 0; i < n; i++)
            if (parent[i] == -1 || heavy[parent[i]] != i)
                nodes.push_back(pii(depth[i], i));
        sort(all(nodes));
        for (int ii = 0, currentPos = 0; ii < nodes.size(); ++ii){
            int i = nodes[ii].s;
            for (int u = i; u != -1; u = heavy[u], currentPos++){
                root[u] = i, pos[u] = currentPos, ipos[currentPos] = u;
            }
        }
    }
    int lca(int u, int v, ST<T> &st){
        int ans = oo;
        for (; root[u] != root[v]; v = parent[root[v]]){
            if (depth[root[u]] > depth[root[v]]) swap(u, v);
            ans = min(ans,
                st.operation(1, 0, n - 1, pos[root[v]], pos[v]));
        }
        if (depth[u] > depth[v]) swap(u, v);
        ans = min(ans, st.operation(1, 0, n - 1, pos[u], pos[v]));
        return ans; // LCA at u
    }
    // The kth node (0 indexed) in the path from (u to root)
    int go_up(int u, int k){
        for (; pos[u] - pos[root[u]] < k; u = parent[root[u]])
            k -= pos[u] - pos[root[u]] + 1;
        return ipos[pos[u] - k];
    }
};
```

3.7. Min Cost Flow.

```
// Maximum flow of minimum cost with potentials
// Complexity: O(min(m^2 n log n, m log n flow))
template<typename T, typename C = T>
struct min_cost_flow{
    struct edge{
        int src, dst;
        T cap, flow;
        C cost;
        int rev;
    };
    int n;
    vector<vector<edge>> adj;
    min_cost_flow(int n) : n(n), adj(n) {}
    void add_edge(int src, int dst, T cap, C cost){
        adj[src].push_back({src, dst, cap, 0, cost,
           (int)adj[dst].size()});
        if (src == dst) adj[src].back().rev++;
        adj[dst].push_back({dst, src, 0, 0, -cost,
             (int)adj[src].size() - 1});
    }
    const C oo = numeric_limits<C>::max();
    vector<C> dist, pot;
    vector<edge*> prev;
    vector<T> curflow;
    void bellman_ford(int s, int t){
        pot.assign(n, oo);
        pot[s] = 0;
        for (int it = 0, change = true; it < n && change; ++it){
            change = false;
            for (int u = 0; u < n; ++u) if (pot[u] != oo) {
                    for (auto &e : adj[u])
                        if (e.flow < e.cap
                            && pot[e.dst] > pot[u] + e.cost) {
                            pot[e.dst] = pot[u] + e.cost;
                            change = true;
                        }
                }
        }
    }
    bool dijkstra(int s, int t){
        dist.assign(n, oo);
        prev.assign(n, nullptr);
        dist[s] = 0;
        curflow[s] = numeric_limits<T>::max();
        using pci = pair<C, int>;
        priority_queue<pci, vector<pci>, greater<pci>> pq;
```

```
		pq.push({ 0, s });
		while (!pq.empty()){
			C d; int u;
			tie(d, u) = pq.top();
			pq.pop();
			if (d != dist[u]) continue;
			for (auto &e : adj[u])
				if (e.flow < e.cap && dist[e.dst] > dist[u]
					+ e.cost + pot[u] - pot[e.dst]){
					dist[e.dst] = dist[u] + e.cost
						+ pot[u] - pot[e.dst];
					prev[e.dst] = &e;
					curflow[e.dst] = min(curflow[u], e.cap - e.flow);
					pq.push({ dist[e.dst], e.dst });
				}
		}
		return dist[t] < oo;
	}
	pair<T, C> max_flow(int s, int t, bool neg_edges = true){
		T flow = 0;
		C cost = 0;
		curflow.assign(n, 0);
		if (neg_edges) bellman_ford(s, t);
		else pot.assign(n, 0);
		while (dijkstra(s, t)){
			for (int u = 0; u < n; ++u)
				if (dist[u] < oo) pot[u] += dist[u];
			T delta = curflow[t];
			flow += delta;
			for (edge *e = prev[t]; e != nullptr; e = prev[e->src]){
				e->flow += delta;
				adj[e->dst][e->rev].flow -= delta;
				cost += delta * e->cost;
			}
		}
		return {flow, cost};
	}
};
```

3.8. 2-SAT.

```
struct satisfiability_twosat{
	satisfiability_twosat(int n) : n(n), imp(2 * n) {}
	inline int neg(int u) const { return ~u; }
	inline void add_implication(int u, int v){
		if (u == v) return;
		imp[u+n].emplace_back(v+n);
		imp[~v+n].emplace_back(~u+n);
	}
	inline void add_clause(int u, int v) { add_implication(neg(u), v); }
	vector<bool> solve() const{
		vector<int> S, B, I(2 * n);
		function<void(int, int&)> dfs = [&](int u, int &t){
			B.push_back(I[u] = S.size());
			S.push_back(u);
			for (int v : imp[u])
				if (!I[v]) dfs(v, t);
				else while (I[v] < B.back()) B.pop_back();
			if (I[u] == B.back())
				for (B.pop_back(), ++t; I[u] < S.size(); S.pop_back())
					I[S.back()] = t;
		};
		for (int u = 0, t = 2 * n; u < 2 * n; ++u)
			if (!I[u]) dfs(u, t);
		vector<bool> value(n);
		for (int i = 0; i < n; ++i)
			if (I[i+n] == I[~i+n]) return vector<bool>();
			else value[i] = I[i+n] < I[~i+n];
		return value;
	}
private:
	int n;
	vector<vector<int>> imp;
};
```

3.9. Strongly connected components.

```
// returns which nodes that belong to each scc
vector<vector<int>> strongly_connected_components(const vector<vector<int>> &g){
	int n = g.size();
	vector<vector<int>> scc;
	vector<int> S, B, I(n, -1);
	function<void(int)> dfs = [&](int u){
		B.push_back(I[u] = S.size());
		S.push_back(u);
		for (int v : g[u]){
			if (!~I[v]) dfs(v);
			else while (I[v] < B.back()) B.pop_back();
		}
		if (I[u] == B.back()){
			scc.push_back({});
			B.pop_back();
			while (I[u] < (int)S.size()){
				scc.back().push_back(S.back());
				I[S.back()] = n + scc.size();
				S.pop_back();
			}
		}
	};
	for (int u = 0; u < n; ++u) if (!~I[u]) dfs(u);
	return scc;
}
```

3.10. Virtual tree.

```
// Compute the lca of two nodes and the distance
// between them
// Compress a subset of k nodes into a tree with
// the same structure
// Notes: mp are only necessary for compress
// after compress every node u is mapped
// to mp[u]
// Complexity: O(n log n) build, O(1) lca,
// O(k log k) compress
struct virtual_tree{
	vector<int> tour, depth, pos, mp;
	vector<vector<int>> table;
	virtual_tree(vector<vector<int>> &adj){
		pos = mp = vector<int>(adj.size());
		function<void(int, int, int)> dfs
			= [&](int u, int p, int d){
			pos[u] = tour.size();
			tour.push_back(u);
			depth.push_back(d);
			for (int v : adj[u])
				if (v != p){
					dfs(v, u, d+1);
					tour.push_back(u);
					depth.push_back(d);
				}
		};
		dfs(0, -1, 0);
		int t = tour.size(), lg = __lg(t);
		table.resize(lg+1, vector<int>(t));
		iota(table[0].begin(), table[0].end(), 0);
		for (int j = 0; j < lg; ++j)
			for (int i = 0; i + (1<<(j+1)) <= t; ++i)
				table[j+1][i] = argmin(table[j][i],
					table[j][i+(1<<j)]);
	}
	inline int argmin(int i, int j)
	{ return depth[i] < depth[j] ? i : j; }
	inline int lca(int u, int v){
		int i = pos[u], j = pos[v];
		if (i > j) swap(i, j);
		int l = __lg(j - i);
		return i == j ? u : tour[argmin(table[l][i],
			table[l][j-(1<<l)])];
	}

	inline int dist(int u, int v){
		return depth[pos[u]] + depth[pos[v]] - 2*depth[pos[lca(u, v)]];
	}
	vector<vector<pair<int, int>>> compress(vector<int> &a){
		auto cmp = [&](const int &x, const int &y)
		{ return pos[x] < pos[y]; };
		auto c = a;
		sort(c.begin(), c.end(), cmp);
		for (int i = 1, sz = c.size(); i < sz; ++i)
			c.push_back(lca(c[i-1], c[i]));
		sort(c.begin(), c.end(), cmp);
		c.erase(unique(c.begin(), c.end()), c.end());
		vector<vector<pair<int, int>>> g(c.size());
		vector<int> s;
		// u become mp[u]
		for (auto &u : c){
			mp[u] = &u-&c[0];
			while (!s.empty() && lca(s.back(), u) != s.back())
				s.pop_back();
			if (!s.empty()){
				int d = dist(s.back(), u);
				g[mp[s.back()]].push_back({ mp[u], d });
				g[mp[u]].push_back({ mp[s.back()], d });
			}
			s.push_back(u);
		}
		return g;
	}
};
```

3.11. Gomory Hu Tree.

```
// Gomory-Hu tree
// Complexity: O(n-1) max-flow call
template<typename flow_type>
struct edge{
	int src, dst;
	flow_type cap;
};
template<typename flow_type>
vector<edge<flow_type>> gomory_hu(dinic<flow_type> &adj){
	int n = adj.n;
	vector<edge<flow_type>> tree;
	vector<int> parent(n);
	for (int u = 1; u < n; ++u){
		tree.push_back({ u, parent[u], adj.max_flow(u, parent[u]) });
		for (int v = u + 1; v < n; ++v)
			if (adj.level[v] == -1 && parent[v] == parent[u])
				parent[v] = u;
	}
	return tree;
}
```

4. Math

4.1. Bitwise transform.

```
// Notes: if you use mod make sure 0 <= a[i], b[i] < mod when you call convolve
enum bit_op { AND, OR, XOR };
namespace bitwise_transform{
	template<int P, typename T>
	inline void add(T &x, T y){
		x += y;
		if (P != 0 && x >= P) x -= P;
	}
	template<bit_op B, int P, bool inv = false, typename T>
	void transform(T a[], int n){
		for (int len = 1; len < n; len <<= 1)
			for (int i = 0; i < n; i += len << 1)
				for (int j = i; j < i + len; ++j){
					T u = a[j], v = a[j + len];
					if (B == AND) add<P>(a[j], inv ? P-v : v);
					if (B == OR) add<P>(a[j + len], inv ? P-u : u);
					if (B == XOR)
						add<P>(a[j], v),
						add<P>(a[j + len] = u, P-v);
				}
		if (B == XOR && inv){
			int in = pow_mod(n, P-2, P);
			for (int i = 0; i < n; ++i){
				if (P == 0) a[i] /= n;
				else a[i] = (ll)a[i] * in % P;
			}
		}
	}
	template<bit_op B, int P = 0, typename T>
	vector<T> convolve(vector<T> a, vector<T> b){
		int n = max(a.size(), b.size()), sz = 1;
		while (sz < n) sz <<= 1;
		a.resize(sz);
		b.resize(sz);
		transform<B, P>(a.data(), sz);
		transform<B, P>(b.data(), sz);
		for (int i = 0; i < sz; ++i){
			if (P == 0) a[i] *= b[i];
			else a[i] = (ll)a[i] * b[i] % P;
		}
		transform<B, P, true>(a.data(), sz);
		return a;
	}
}
```

4.2. Simplex.

```
// Description:
//  Solve a canonical LP:
//   min. c x
//   s.t. A x <= b
//   x >= 0
const double eps = 1e-9, oo = numeric_limits<double>::infinity();
typedef vector<double> vec;
typedef vector<vec> mat;
double simplexMethodPD(mat &A, vec &b, vec &c){
	int n = c.size(), m = b.size();
	mat T(m + 1, vec(n + m + 1));
	vector<int> base(n + m), row(m);
	for(int j = 0; j < m; ++j){
		for (int i = 0; i < n; ++i) T[j][i] = A[j][i];
		T[j][n + j] = 1;
		base[row[j] = n + j] = 1;
		T[j][n + m] = b[j];
	}
	for (int i = 0; i < n; ++i) T[m][i] = c[i];
	while (1){
		int p = 0, q = 0;
		for (int i = 0; i < n + m; ++i)
			if (T[m][i] <= T[m][p]) p = i;
		for (int j = 0; j < m; ++j)
			if (T[j][n + m] <= T[q][n + m]) q = j;
		double t = min(T[m][p], T[q][n + m]);
		if (t >= -eps){
			vec x(n);
			for (int i = 0; i < m; ++i)
				if (row[i] < n) x[row[i]] = T[i][n + m];
			// x is the solution
			return -T[m][n + m]; // optimal
		}
		if (t < T[q][n + m]){
			// tight on c -> primal update
			for (int j = 0; j < m; ++j)
				if (T[j][p] >= eps)
					if (T[j][p] * (T[q][n + m] - t)
							>= T[q][p] * (T[j][n + m] - t))
						q = j;
			if (T[q][p] <= eps) return oo; // primal infeasible
		}
		else{
			// tight on b -> dual update
			for (int i = 0; i < n + m + 1; ++i)
				T[q][i] = -T[q][i];
			for (int i = 0; i < n + m; ++i)
				if (T[q][i] >= eps)
					if (T[q][i] * (T[m][p] - t)
						>= T[q][p] * (T[m][i] - t))
						p = i;
			if (T[q][p] <= eps) return -oo; // dual infeasible
		}
		for (int i = 0; i < m + n + 1; ++i)
			if (i != p) T[q][i] /= T[q][p];
		T[q][p] = 1; // pivot(q, p)
		base[p] = 1;
		base[row[q]] = 0;
		row[q] = p;
		for (int j = 0; j < m + 1; ++j) if (j != q){
				double alpha = T[j][p];
				for (int i = 0; i < n + m + 1; ++i)
					T[j][i] -= T[q][i] * alpha;
			}
	}
	return oo;
}
```

4.3. Number theoretic transform.

```
// Notes: mod = 2**k * c + 1 should be prime, k >= max_degree
namespace ntt{
	const int mod = 998244353;
	const int root = 5; // primitive root of mod
	int base = 1;
	vector<int> roots;
	void ensure_base(int nbase){
		if (nbase <= base) return;
		roots.resize(nbase);
		for (int mh = base; mh << 1 <= nbase; mh <<= 1){
			int wm = pow_mod(root, (mod - 1) / (mh << 1), mod);
			roots[mh] = 1;
			for (int i = 1; i < mh; ++i)
				roots[i + mh] = (ll)roots[i + mh - 1] * wm % mod;
		}
		base = nbase;
	}
	void fft(int a[], int n, int sign){
		ensure_base(n);
		for (int i = 1, j = 0; i < n - 1; ++i){
			for (int k = n >> 1; (j ^= k) < k; k >>= 1);
			if (i < j) swap(a[i], a[j]);
		}
		for (int m, mh = 1; (m = mh << 1) <= n; mh = m)
			for (int i = 0; i < n; i += m)
				for (int j = i; j < i + mh; ++j){
					int y = (ll)a[j + mh] * roots[j - i + mh] % mod;
					if ((a[j + mh] = a[j] - y) < 0) a[j + mh] += mod;
					if ((a[j] += y) >= mod) a[j] -= mod;
				}
		if (sign < 0){
			int inv = pow_mod(n, mod - 2, mod);
			for (int i = 0; i < n; ++i) a[i] = (ll)a[i] * inv % mod;
			reverse(a + 1, a + n);
		}
	}
	vector<int> convolve(vector<int> x, vector<int> y){
		int n = x.size() + y.size() - 1, sz = 1;
		while (sz < n) sz <<= 1;
		x.resize(sz);
		y.resize(sz);
		fft(x.data(), sz, +1);
		fft(y.data(), sz, +1);
		for (int i = 0; i < sz; ++i)
			x[i] = (ll)x[i] * y[i] % mod;
		fft(x.data(), sz, -1);
		x.resize(n);
		return x;
	}
}
```

4.4. Gauss.

```
const int oo = 0x3f3f3f3f;
const double eps = 1e-9;
int gauss(vector<vector<double>> a, vector<double> &ans){
	int n = (int) a.size();
	int m = (int) a[0].size() - 1;
	vector<int> where(m, -1);
	for (int col = 0, row = 0; col < m && row < n; ++col){
		int sel = row;
		for (int i = row; i < n; ++i)
			if (abs(a[i][col]) > abs(a[sel][col])) sel = i;
		if (abs(a[sel][col]) < eps) continue;
		for (int i = col; i <= m; ++i) swap(a[sel][i], a[row][i]);
		where[col] = row;
		for (int i = 0; i < n; ++i) if (i != row) {
				double c = a[i][col] / a[row][col];
				for (int j = col; j <= m; ++j)
					a[i][j] -= a[row][j] * c;
			}
		++row;
	}
	ans.assign(m, 0);
	for (int i = 0; i < m; ++i)
		if (where[i] != -1) ans[i] = a[where[i]][m] / a[where[i]][i];
	for (int i = 0; i < n; ++i){
		double sum = 0;
		for (int j = 0; j < m; ++j)
			sum += ans[j] * a[i][j];
		if (abs(sum - a[i][m]) > eps) return 0;
	}
	for (int i = 0; i < m; ++i) if (where[i] == -1) return oo;
	return 1;
}
```

4.5. Hungarian.

```
// max weight matching
template<typename T>
T hungarian(const vector<vector<T>> &a){
	int n = a.size(), m = a[0].size(), p, q; // n <= m
	vector<T> fx(n, numeric_limits<T>::min()), fy(m, 0);
	vector<int> x(n, -1), y(m, -1);
	for (int i = 0; i < n; ++i)
		for (int j = 0; j < m; ++j) fx[i] = max(fx[i], a[i][j]);
	for (int i = 0; i < n;){
		vector<int> t(m, -1), s(n + 1, i);
		for (p = q = 0; p <= q && x[i] < 0; ++p)
			for (int k = s[p], j = 0; j < m && x[i] < 0; ++j)
				// be careful with comparison on doubles
				if (fx[k] + fy[j] == a[k][j] && t[j] < 0) {
					s[++q] = y[j], t[j] = k;
					if (s[q] < 0) for (p = j; p >= 0; j = p)
						y[j] = k = t[j], p = x[k], x[k] = j;
				}
```

```
		if (x[i] < 0){
			T d = numeric_limits<T>::max();
			for (int k = 0; k <= q; ++k)
				for (int j = 0; j < m; ++j) if (t[j] < 0)
					d = min(d, fx[s[k]] + fy[j] - a[s[k]][j]);
			for (int j = 0; j < m; ++j)
				fy[j] += (t[j] < 0 ? 0 : d);
			for (int k = 0; k <= q; ++k) fx[s[k]] -= d;
		}
		else ++i;
	}
	T ret = 0;
	for (int i = 0; i < n; ++i) ret += a[i][x[i]];
	return ret;
}
```

4.6. Integrate.

```
// Numerical Integration (Adaptive Gauss--Lobatto formula)
// Description:
// Gauss--Lobatto formula is a numerical integrator
// that is exact for polynomials of degree <= 2n+1.
// Adaptive Gauss--Lobatto recursively decomposes the
// domain and computes integral by using G-L formula.
// Complexity:
// O(#pieces) for a piecewise polynomials.
// In general, it converges in O(1/n^6) for smooth functions.
// For (possibly) non-smooth functions, this is the best integrator.
template <class F>
double integrate(F f, double lo, double hi, double eps = 1e-9){
	const double th = eps / 1e-14; // (= eps / machine_epsilon)
	function<double(double, double, double, double, int)> rec =
		[&](double x0, double x6, double y0, double y6, int d){
		const double a = sqrt(2.0 / 3.0), b = 1.0 / sqrt(5.0);
		double x3 = (x0 + x6) / 2, y3 = f(x3), h = (x6 - x0) / 2;
		double x1 = x3 - a * h, x2 = x3 - b * h, x4 = x3 + b * h, x5 = x3 + a * h;
		double y1 = f(x1), y2 = f(x2), y4 = f(x4), y5 = f(x5);
		double I1 = (y0 + y6 + 5 * (y2 + y4)) * (h / 6);
		double I2 = (77*(y0+y6)+432*(y1+y5)+625*(y2+y4)+672*y3)*(h/1470);
		if (x3 + h == x3 || d > 50) return 0.0;
		if (d > 4 && th + (I1 - I2) == th) return I2; // avoid degeneracy
		return (double)(rec(x0, x1, y0, y1, d + 1) + rec(x1, x2, y1, y2, d + 1)
						+ rec(x2, x3, y2, y3, d + 1) + rec(x3, x4, y3, y4, d + 1)
						+ rec(x4, x5, y4, y5, d + 1) + rec(x5, x6, y5, y6, d + 1));
	};
	return rec(lo, hi, f(lo), f(hi), 0);
}
```

4.7. NTT with arbitrary mod.

```
// Notes: fft_core function doesn't normalize when
//  call it with sign=-1, you must do it be yourself,
//  implementing your own point is much faster
//  ** #define double ld if you have precision issues
// (probably for n around 5e5) **
namespace fft{
	typedef complex<double> point;
	// n must be a power of 2, sign must be +1 or -1
	void fft_core(point a[], int n, int sign = +1){
		const double theta = 8 * sign
				* atan(static_cast<point::value_type>(1.0)) / n;
		for (int i = 0, j = 1; j < n - 1; ++j){
			for (int k = n >> 1; k > (i ^= k); k >>= 1);
			if (j < i) swap(a[i], a[j]);
		}
		for (int m, mh = 1; (m = mh << 1) <= n; mh = m)
			for (int i = 0, irev = 0; i < n; i += m){
				point w = exp(point(0, theta * irev));
				for (int k = n >> 2; k > (irev ^= k); k >>= 1);
				for (int j = i; j < mh + i; ++j){
					int k = j + mh; point x = a[j] - a[k];
					a[j] += a[k]; a[k] = w * x;
				}
			}
	}
	vector<point> convolve(vector<point> &a, vector<point> &b){
		int n = a.size(), m = b.size(); int sum = n + m;
		while (sum != (sum & -sum)) sum += (sum & -sum);
		while (a.size() < sum) a.push_back(point(0, 0));
		while (b.size() < sum) b.push_back(point(0, 0));
		fft_core(a.data(), a.size(), 1);
		fft_core(b.data(), b.size(), 1);
		vector<point> res(sum);
		for (int i = 0; i < sum; i++) res[i] = a[i] * b[i];
		fft_core(res.data(), res.size(), -1);
		for(auto &p : res) p /= res.size();
		return res;
	}
	// mod < 2^31
	vector<int> convolve(const vector<int> &a,const vector<int> &b,int mod){
		int n = a.size() + b.size() - 1;
		for (int k : {1, 2, 4, 8, 16}) n |= (n >> k); ++n;
		vector<point> pa(n), pb(n);
		for (int i = 0; i < n; ++i){
			if (i < a.size())
				pa[i] = point(a[i] >> 15, a[i] & ((1 << 15) - 1));
			if (i < b.size())
				pb[i] = point(b[i] >> 15, b[i] & ((1 << 15) - 1));
		}
		fft_core(pa.data(), n, +1);
		fft_core(pb.data(), n, +1);
		vector<point> c(n), d(n);
		for (int i = 0; i < n; ++i){
			int j = (n - i) & (n - 1);
			point u = (pa[i] + conj(pa[j])) * point(0.5, +0.0);
			point v = (pa[i] - conj(pa[j])) * point(0.0, -0.5);
			point x = (pb[i] + conj(pb[j])) * point(0.5, +0.0);
			point y = (pb[i] - conj(pb[j])) * point(0.0, -0.5);
			c[i] = u * (x + y * point(0, 1));
			d[i] = v * (x + y * point(0, 1));
			c[i] /= n;
			d[i] /= n;
```

```
        }
        fft_core(c.data(), n, -1);
        fft_core(d.data(), n, -1);
        vector<int> ans(a.size() + b.size() - 1);
        for (int i = 0; i < (int)ans.size(); ++i){
            int u = llround(real(c[i])) % mod;
            int v = llround(imag(c[i])) % mod;
            int x = llround(real(d[i])) % mod;
            int y = llround(imag(d[i])) % mod;
            ans[i] = ((ll)u * (1 << 30) % mod
                      + (ll)(v + x) * (1 << 15) % mod + y) % mod;
        }
        return ans;
    }
}
```

4.8. Interpolation.

```
vector<double> interpolate(vector<double> x, vector<double> y, int n){
    vector<double> res(n), temp(n);
    for (int k = 0; k < n - 1; k++) for (int i = k + 1; i < n; i++)
        y[i] = (y[i] - y[k]) / (x[i] - x[k]);
    double last = 0, temp[0] = 1;
    for (int k = 0; k < n; k++) for (int i = 0; i < n; i++){
        res[i] += y[k] * temp[i];
        swap(last, temp[i]);
        temp[i] -= last * x[k];
    }
    return res;
}
```

5. Number Theory

5.1. Diophantine Equation.

```
// returns (d, x, y) such that d = gcd(a, b) = ax + by
ll extended_euclid(ll a, ll b, ll &x, ll &y){
    if (b == 0) { x = 1, y = 0; return a; }
    ll r = extended_euclid(b, a % b, y, x);
    y -= a / b * x;
    return r;
}
// returns (x, y) such that c = ax + by
pair<ll, ll> diophantine_equation(ll a, ll b, ll c){
    ll g, x, y; g = extended_euclid(a, b, x, y);
    ll k = 0; // k e Z
    return { x * c / g + (k * b / g), y * c / g - (k * a / g) };
}
```

5.2. Miller Rabin.

```
bool witness(ll a, ll s, ll d, ll n){
    ll x = pow(a, d, n);
    if (x == 1 || x == n - 1) return 0;
    for (int i = 0; i < s - 1; i++){
        x = mul(x, x, n);
        if (x == 1) return 1;
        if (x == n - 1) return 0;
    }
    return 1;
}
bool miller_rabin(ll n){
    if (n < 2) return 0;
    if (n == 2) return 1;
    if (n % 2 == 0) return 0;
    ll d = n - 1, s = 0;
    while (d % 2 == 0) ++s, d /= 2;
    vector<ll> test = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
    for (ll p : test)
        if (p >= n) break;
        else if (witness(p, s, d, n)) return 0;
    return 1;
}
```

5.3. Pollard Rho.

```
// return not trivial divisor of n
ll pollard_rho(ll n){
    if (!(n & 1)) return 2;
    while (1) {
        ll x = (ll) rand() % n, y = x;
        ll c = rand() % n;
        if (c == 0 || c == 2) c = 1;
        for (int i = 1, k = 2;; i++) {
            x = mul(x, x, n);
            if (x >= c) x -= c;
            else x += n - c;
            if (x == n) x = 0;
            if (x == 0) x = n - 1;
            else x--;
            ll d = __gcd(x > y ? x - y : y - x, n);
            if (d == n) break;
            if (d != 1) return d;
            if (i == k) { y = x, k <<= 1; }
```

```
            }
        }
        return 0;
    }
```

5.4. Chinese Remainder Theorem.

```
// return min x such that x % m[i] == a[i]
ll chinese_remainder_theorem(vector<ll> a, vector<ll> m){
    int n = a.size();
    ll s = 1, t, ans = 0, p, q;
    for (auto i : m) s *= i;
    for (int i = 0; i < n; i++){
        t = s / m[i];
        extended_euclid(t, m[i], p, q);
        ans = (ans + t * p * a[i]) % s;
    }
    if (ans < 0) ans += s;
    return ans;
}
// solve a * x = b (M)
ll linear_congruence(ll a, ll b, ll M){
    return chinese_remainder_theorem(vector<ll>
        (1, b * pow(a, euler_phi(M)-1, M) % M), vector<ll>(1, M));
}
// Solve x=ai(mod mi), for any i and j, (mi,mj)|ai-aj
// Return (x0,M) M=[m1..mn]. All solutions are x=x0+t*M
// Note: be carful with the overflow in the multiplication
pair<ll, ll> linear_congruences(const vector<ll> &a, const vector<ll> &m){
    int n = a.size();
    ll u = a[0], v = m[0], p, q;
    for (int i = 1; i < n; ++i){
        ll r = gcd(v, m[i], p, q), t = v;
        if ((a[i] - u) % r) return {-1, 0}; // no solution
        v = v / r * m[i];
        u = ((a[i] - u) / r * p * t + u) % v;
    }
    if (u < 0) u += v;
    return {u, v};
}
```

5.5. Discrete Logarithm.

```
ll dlog(ll a, ll b, ll M){
    map<ll, ll> _hash;
    ll n = euler_phi(M), k = sqrt(n)+10;
    for(ll i = 0, t = 1; i < k; ++i){
        _hash[t] = i;
        t = mul(t, a, M);
    }
    ll c = pow(a, n - k, M);
    for(ll i = 0; i * k < n; i++){
        if(_hash.find(b) != _hash.end()) return i * k + _hash[b];
        b = mul(b, c, M);
    }
    return -1;
}
```

5.6. Discrete Roots.

```
vector<ll> discrete_root(ll k, ll a, ll n){
    if (a == 0) return {0};
    ll g = primitive_root(n);
    ll sq = (ll) sqrt(n + .0) + 1;
    vector<pair<ll, ll>> dec(sq);
    for (ll i = 1; i <= sq; ++i)
        dec[i - 1] = {pow(g, ll(i * sq * 1ll * k % (n - 1)), n), i};
    sort(dec.begin(), dec.end());
    ll any_ans = -1;
    for (int i = 0; i < sq; ++i){
        ll my = ll(pow(g, ll(i * 1ll * k % (n - 1)), n) * 1ll * a % n);
        auto it = lower_bound(dec.begin(), dec.end(), make_pair(my, 0ll));
        if (it != dec.end() && it->first == my){
            any_ans = it->second * sq - i;
            break;
        }
    }
    if (any_ans == -1) return {};
    ll delta = (n - 1) / __gcd(k, n - 1);
    vector<ll> ans;
    for (ll cur = any_ans % delta; cur < n - 1; cur += delta)
        ans.push_back(pow(g, cur, n));
    sort(ans.begin(), ans.end());
    return ans;
}
```

5.7. Modular Arithmetics.

```
typedef long long ll;
typedef vector<ll> vec;
typedef vector<vec> mat;
// inverse of 1, 2, ..., n mod P in O(n) (P must be prime)
vector<ll> inverses(int n, int P){
    vector<ll> inv(n + 1, 1);
    for (int i = 2; i <= n; ++i)
        inv[i] = inv[P % i] * (P - P / i) % P;
    return inv;
}
```

```cpp
template <typename T, typename U>
T pow_mod(T a, U b, int mod){
	T r = 1;
	for (; b > 0; b >>= 1){
		if (b & 1) r = (ll)r * a % mod;
		a = (ll)a * a % mod;
	}
	return r;
}
ll inv(ll b, ll M){
	ll u = 1, x = 0, s = b, t = M;
	while (s){
		ll q = t / s;
		swap(x -= u * q, u);
		swap(t -= s * q, s);
	}
	return (x %= M) >= 0 ? x : (x + M);
}
// solve a x == b (mod M) (sol iff (a, m) | b same as (a, m) | (b, m))
ll div(ll a, ll b, ll M){
	ll u = 1, x = 0, s = a, t = M;
	while (s){
		ll q = t / s;
		swap(x -= u * q, u);
		swap(t -= s * q, s);
	}
	if (b % t) return -1; // infeasible
	return (x < 0 ? (x + M) : x) * (b / t) % M;
}
// assume: M is prime (singular ==>
mat inv(mat A, ll M){
	int n = A.size();
	mat B(n, vec(n));
	for (int i = 0; i < n; ++i) B[i][i] = 1;
	for (int i = 0; i < n; ++i){
		int j = i;
		while (j < n && A[j][i] == 0) ++j;
		if (j == n) return {};
		swap(A[i], A[j]);
		swap(B[i], B[j]);
		ll inv = div(1, A[i][i], M);
		for (int k = i; k < n; ++k) A[i][k] = A[i][k] * inv % M;
		for (int k = 0; k < n; ++k) B[i][k] = B[i][k] * inv % M;
		for (int j = 0; j < n; ++j) {
			if (i == j || A[j][i] == 0) continue;
			ll cor = A[j][i];
			for (int k = i; k < n; ++k)
				A[j][k] = (A[j][k] - cor * A[i][k] % M + M) % M;
			for (int k = 0; k < n; ++k)
				B[j][k] = (B[j][k] - cor * B[i][k] % M + M) % M;
		}
	}
	return B;
}
```

5.8. Primitive Root.

```cpp
// only prime and p>2, O(sqrt(p))
ll primitive_root(ll p){
	auto v = prime_divisors(p - 1);
	for (ll g = 1;; g++){
		bool ok = 1;
		for (auto d : v)
			if (pow(g, (p - 1) / d, p) == 1){
				ok = 0;
				break;
			}
		if (ok) return g;
	}
}
// Note: Only 2, 4, p^n, 2p^n have primitive roots
ll primitive_root(ll m){
	if (m == 1) return 0;
	if (m == 2) return 1;
	if (m == 4) return 3;
	auto pr = primes(0, sqrt(m) + 1); // fix upper bound
	ll t = m;
	if (!(t & 1)) t >>= 1;
	for (ll p : pr){
		if (p > t) break;
		if (t % p) continue;
		do
			t /= p;
		while (t % p == 0);
		if (t > 1 || p == 2) return 0;
	}
	ll x = euler_phi(m), y = x, n = 0;
	vector<ll> f(32);
	for (ll p : pr) {
		if (p > y) break;
		if (y % p) continue;
		do
			y /= p;
		while (y % p == 0);
		f[n++] = p;
	}
	if (y > 1) f[n++] = y;
	for (ll i = 1; i < m; ++i) {
		if (__gcd(i, m) > 1) continue;
		bool flag = 1;
		for (ll j = 0; j < n; ++j) {
			if (pow(i, x / f[j], m) == 1){
				flag = 0;
				break;
			}
		}
		if (flag) return i;
	}
	return 0;
}
```

6. String

6.1. Suffix Array.

```
// Notes: The suffix starting in |S| is always the lowest
// and have lcp 0 with the next suffix.
//   lcp[i] is the longest common prefix between
//   the suffix in sa[i] and sa[i-1]
struct suffix_array{
	int n;
	vector<int> sa, lcp, rank;
	template<typename RAIter>
	suffix_array(const RAIter &bg, const RAIter &nd, int alp = 256)
		: n(nd - bg + 1), sa(n), lcp(n), rank(n){
		vector<int> ws(max(n, alp));
		auto &x = lcp, &y = rank;
		copy(bg, nd, x.begin());
		iota(sa.begin(), sa.end(), 0);
		for (int j = 0, p = 0; p < n; j = max(1, j * 2), alp = p){
			p = j, iota(y.begin(), y.end(), n - j);
			fill(ws.begin(), ws.end(), 0);
			for (int i = 0; i < n; ws[x[i++]]++)
				if (sa[i] >= j) y[p++] = sa[i] - j;
			partial_sum(ws.begin(), ws.end(), ws.begin());
			for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
			x.swap(y), p = 1, x[sa[0]] = 0;
			for (int i = 1, a, b; i < n; ++i)
				a = sa[i - 1], b = sa[i], x[b] =
				(y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
		}
		for (int i = 0; i < n; ++i) rank[sa[i]] = i;
		for (int i = 0, j, k = lcp[0] = 0; i < n - 1; lcp[rank[i++]] = k)
			for (k && k--, j = sa[rank[i] - 1];
					bg[i + k] == bg[j + k]; k++);
	}
};
```

6.2. Aho-Corasick.

```
struct aho_corasick{
	static const int alpha = 26;
	vector<array<int, alpha>> go;
	vector<int> fail, endpos;
	aho_corasick() { add_node(); }
	int add_string(const string &str){
		int e = 0;
		for (char c : str){
			if (!go[e][c-'a']){
				int nn = add_node();
				go[e][c-'a'] = nn;
			}
			e = go[e][c-'a'];
		}
		++endpos[e];
		return e;
	}
	void build(){
		queue<int> que;
		for (int c = 0; c < alpha; ++c)
			if (go[0][c]) que.push(go[0][c]);
		for (; !que.empty(); que.pop()){
			int e = que.front();
			int f = fail[e];
			for (int c = 0; c < alpha; ++c)
				if (!go[e][c]) go[e][c] = go[f][c];
				else {
					fail[go[e][c]] = go[f][c];
					endpos[go[e][c]] += endpos[go[f][c]];
					que.push(go[e][c]);
				}
		}
	}
private:
	int add_node(){
		int pos = go.size();
		go.emplace_back(array<int, alpha>());
		fail.emplace_back(0);
		endpos.emplace_back(0);
		return pos;
	}
};
```

6.3. Manacher.

```
// longest palindrome centered in positon p
// with odd and even length -> rad[2*p], rad[2*p+1]
vector<int> manacher(const string &s){
	int n = 2 * s.length();
	vector<int> rad(n);
	for (int i = 0, j = 0, k; i < n; i += k, j = max(j - k, 0)){
		for (; i >= j && i + j + 1 < n
			&& s[(i - j) / 2] == s[(i + j + 1) / 2]; ++j);
		rad[i] = j;
		for (k = 1; i >= k &&
			rad[i] >= k && rad[i - k] != rad[i] - k; ++k)
			rad[i + k] = min(rad[i - k], rad[i] - k);
	}
	return rad;
}
```

6.4. Z Algorithm.

```
// z[i] = length of the longest common prefix of s and s[i..n]
vector<int> zfunction(const string &s){
	int n = s.length();
	vector<int> z(n, n);
	for (int i = 1, g = 0, f; i < n; ++i)
		if (i < g && z[i - f] != g - i) z[i] = min(z[i - f], g - i);
		else {
			for (g = max(g, i), f = i; g < n && s[g] == s[g - f]; ++g);
			z[i] = g - f;
		}
	return z;
}
```

6.5. Suffix Tree.

```
template <typename charT, typename Container>
struct suffix_tree{
	vector<charT> s;
	vector<Container> next;
	vector<int> spos, len, link;
	int node, pos, last;
	suffix_tree() { make_node(0), node = pos = 0; }
	int make_node(int p, int l = 2e9){
		spos.push_back(p);
		len.push_back(l);
		link.push_back(0);
		next.emplace_back();
		return spos.size() - 1;
	}
	void extend(charT c){
		for (s.push_back(c), ++pos, last = 0; pos > 0;){
			int n = s.size();
			while (pos > len[next[node][s[n - pos]]])
				node = next[node][s[n - pos]], pos -= len[node];
			charT edge = s[n - pos];
			int v = next[node][edge];
			charT t = s[spos[v] + pos - 1];
			if (v == 0){
				v = make_node(n - pos);
				link[last] = node;
				last = 0;
			}
			else if (t == c){ link[last] = node; return; }
			else{
				int u = make_node(spos[v], pos - 1);
				next[u][c] = spos.size(), make_node(n - 1);
				next[u][t] = v;
				spos[v] += pos - 1;
				len[v] -= pos - 1;
				v = last = link[last] = u;
			}
			next[node][edge] = v;
			if (node == 0) --pos;
			else node = link[node];
		}
	}
	int get_len(int p) { return p == 0 ? 0 :
				min(len[p], (int)s.size() - spos[p]); }
};
```

6.6. Maximal Suffix.

```
// position of maximun lexicographical suffix
int maximal_suffix(const string &s){
	int n = s.length(), i = 0, j = 1;
	for (int k = 0; j < n - 1; k = 0){
		while (j + k < n - 1 && s[i + k] == s[j + k]) ++k;
		if (s[i + k] < s[j + k]){
			i += (k / (j - i) + 1) * (j - i);
			j = i +1;
		}
		else j += k + 1;
	}
	return i;
}
```

6.7. Minimum Rotation.

```
// minimun lexicographical rotation
int minimum_rotation(const string &s){
	int n = s.length(), i = 0, j = 1, k = 0;
	while (i + k < 2 * n && j + k < 2 * n){
		char a = i + k < n ? s[i + k] : s[i + k - n];
		char b = j + k < n ? s[j + k] : s[j + k - n];
		if (a > b){
			i += k + 1;
			k = 0;
			if (i <= j) i = j + 1;
		}
		else if (a < b){ j += k + 1; k = 0; if (j <= i) j = i + 1; }
		else ++k;
	}
	return min(i, j);
}
```

7. Useful

7.1. Random.

```
mt19937 rng(chrono::high_resolution_clock::now().time_since_epoch().count());
template<typename T>
static T randint(T lo, T hi) { return uniform_int_distribution<T>(lo, hi)(rng); }
```

7.2. Launch Json.

```
{
   "version": "0.2.0",
   "configurations": [
      {
         "name": "(gdb)␣Launch",
         "type": "cppdbg",
         "request": "launch",
         "program": "${fileDirname}/sol",
         "args": ["<", "${fileDirname}/test.in"],
         "stopAtEntry": false,
         "cwd": "${workspaceFolder}",
         "environment": [],
         "console": "externalTerminal",
         "MIMode": "gdb",
         "setupCommands": [
            {
               "description": "Enable␣pretty-printing␣for␣gdb",
               "text": "-enable-pretty-printing",
               "ignoreFailures": true
            }
         ],
         "preLaunchTask": "Build␣active␣file"
      }
   ]
}
```

7.3. Tasks Json.

```
{
   "version": "2.0.0",
   "tasks": [{
      "type": "shell",
      "label": "Build␣active␣file",
      "command": "/usr/bin/g++",
      "args": [
         "-fdiagnostics-color=always",
         "-std=c++17",
         "-g3",
         "-Wall",
         "-D_GLIBCXX_DEBUG",
         "-D_GLIBCXX_DEBUG_PEDANTIC",
         "${file}",
         "-o",
         "${fileDirname}/sol"
      ]
   }]
}
```

8. Tips

Mobius Inversion: $g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$, $\sum_{d|n} \mu(d) = [n == 1]$, $\sum_{d|n} phi(n) = n$

Vertex Cover: Sea el grafo bipartito G(L, R). Sea M un matching máximo de G. Busco un corte mínimo. Un cubrimiento mínimo son los extremos de las aristas cortadas que no son s ni t. Sea $U = \{x \in L : \textit{ x no pertenece al matching } M\}$. Sea $Z = \{x : \textit{ x es alcanzable desde U siguiendo algun camino alternante}\}$. Un vertex cover de G es $VC = (L - Z)U(R \cap Z)$. $|VC| = |M|$.

Maximum Independent Set:$MIS = VC^c$. $|MIS| = n - |M|$.

Edge Cover: Sea el grafo bipartito G(L, R). Sea M un matching máximo de G. $EC = M+$ (1 arista incidente en cada vértice que no cubra el matching máximo). $|EC| = n - |M|$.

Mínima descomposición en cadenas: Duplicar los nodos y colocar las aristas con su correspondiente. Obtener el matching máximo. Reconstruir usando las aristas del matching máximo.

Máxima anticadena: Duplicar los nodos y colocar las aristas con su correspondiente. Obtenemos un cubrimiento por nodos del grafo. Tomamos los nodos que no tienen ninguna copia en el cubrimiento.

Theoretical Computer Science Cheat Sheet

Definitions	
$f(n) = O(g(n))$	iff $\exists$ positive c, n_0 such that $0 \le f(n) \le cg(n)\ \forall n \ge n_0$.
$f(n) = \Omega(g(n))$	iff $\exists$ positive c, n_0 such that $f(n) \ge cg(n) \ge 0\ \forall n \ge n_0$.
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
$f(n) = o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim\limits_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.
$\liminf\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\limsup\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.
$\left[{n \atop k}\right]$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.
$\left\{{n \atop k}\right\}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.
$\left\langle{n \atop k}\right\rangle$	1st order Eulerian numbers: Permutations $\pi_1\pi_2\dots\pi_n$ on $\{1, 2, \dots, n\}$ with k ascents.
$\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle$	2nd order Eulerian numbers.
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.

Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k n^{m+1-k}.$$

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \quad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n}\frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \quad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\binom{n}{k} = \binom{n}{n-k}$,

4. $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1$,

12. $\left\{{n \atop 2}\right\} = 2^{n-1} - 1$, **13.** $\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$,

14. $\left[{n \atop 1}\right] = (n-1)!$, **15.** $\left[{n \atop 2}\right] = (n-1)!H_{n-1}$, **16.** $\left[{n \atop n}\right] = 1$, **17.** $\left[{n \atop k}\right] \ge \left\{{n \atop k}\right\}$,

18. $\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$, **19.** $\left\{{n \atop n-1}\right\} = \left[{n \atop n-1}\right] = \binom{n}{2}$, **20.** $\sum_{k=0}^{n}\left[{n \atop k}\right] = n!$, **21.** $C_n = \frac{1}{n+1}\binom{2n}{n}$,

22. $\left\langle{n \atop 0}\right\rangle = \left\langle{n \atop n-1}\right\rangle = 1$, **23.** $\left\langle{n \atop k}\right\rangle = \left\langle{n \atop n-1-k}\right\rangle$, **24.** $\left\langle{n \atop k}\right\rangle = (k+1)\left\langle{n-1 \atop k}\right\rangle + (n-k)\left\langle{n-1 \atop k-1}\right\rangle$,

25. $\left\langle{0 \atop k}\right\rangle = \begin{cases}1 & \text{if } k = 0,\\ 0 & \text{otherwise}\end{cases}$ **26.** $\left\langle{n \atop 1}\right\rangle = 2^n - n - 1$, **27.** $\left\langle{n \atop 2}\right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2}$,

28. $x^n = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{x+k}{n}$, **29.** $\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $m!\left\{{n \atop m}\right\} = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{k}{n-m}$,

31. $\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{n}\left\{{n \atop k}\right\}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\left\langle\!\left\langle{n \atop 0}\right\rangle\!\right\rangle = 1$, **33.** $\left\langle\!\left\langle{n \atop n}\right\rangle\!\right\rangle = 0$ for $n \ne 0$,

34. $\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = (k+1)\left\langle\!\left\langle{n-1 \atop k}\right\rangle\!\right\rangle + (2n-1-k)\left\langle\!\left\langle{n-1 \atop k-1}\right\rangle\!\right\rangle$, **35.** $\sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

36. $\left\{{x \atop x-n}\right\} = \sum_{k=0}^{n}\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\left\{{n+1 \atop m+1}\right\} = \sum_{k}\binom{n}{k}\left\{{k \atop m}\right\} = \sum_{k=0}^{n}\left\{{k \atop m}\right\}(m+1)^{n-k}$,

Theoretical Computer Science Cheat Sheet

Identities Cont.

38. $\left[{n+1 \atop m+1}\right] = \sum_k \left[{n \atop k}\right]\binom{k}{m} = \sum_{k=0}^{n} \left[{k \atop m}\right] n^{\underline{n-k}} = n! \sum_{k=0}^{n} \frac{1}{k!}\left[{k \atop m}\right]$,

39. $\left[{x \atop x-n}\right] = \sum_{k=0}^{n} \left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle \binom{x+k}{2n}$,

40. $\left\{{n \atop m}\right\} = \sum_k \binom{n}{k}\left\{{k+1 \atop m+1}\right\}(-1)^{n-k}$,

41. $\left[{n \atop m}\right] = \sum_k \left[{n+1 \atop k+1}\right]\binom{k}{m}(-1)^{m-k}$,

42. $\left\{{m+n+1 \atop m}\right\} = \sum_{k=0}^{m} k \left\{{n+k \atop k}\right\}$,

43. $\left[{m+n+1 \atop m}\right] = \sum_{k=0}^{m} k(n+k)\left[{n+k \atop k}\right]$,

44. $\binom{n}{m} = \sum_k \left\{{n+1 \atop k+1}\right\}\left[{k \atop m}\right](-1)^{m-k}$,

45. $(n-m)!\binom{n}{m} = \sum_k \left[{n+1 \atop k+1}\right]\left\{{k \atop m}\right\}(-1)^{m-k}$, for $n \geq m$,

46. $\left\{{n \atop n-m}\right\} = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left[{m+k \atop k}\right]$,

47. $\left[{n \atop n-m}\right] = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left\{{m+k \atop k}\right\}$,

48. $\left\{{n \atop \ell+m}\right\}\binom{\ell+m}{\ell} = \sum_k \left\{{k \atop \ell}\right\}\left\{{n-k \atop m}\right\}\binom{n}{k}$,

49. $\left[{n \atop \ell+m}\right]\binom{\ell+m}{\ell} = \sum_k \left[{k \atop \ell}\right]\left[{n-k \atop m}\right]\binom{n}{k}$.

Trees

Every tree with n vertices has $n-1$ edges.

Kraft inequality: If the depths of the leaves of a binary tree are $d_1, \ldots, d_n$:
$$\sum_{i=1}^{n} 2^{-d_i} \leq 1,$$
and equality holds only if every internal node has 2 sons.

Recurrences

Master method:
$$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1$$

If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then
$$T(n) = \Theta(n^{\log_b a}).$$

If $f(n) = \Theta(n^{\log_b a})$ then
$$T(n) = \Theta(n^{\log_b a} \log_2 n).$$

If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \leq cf(n)$ for large n, then
$$T(n) = \Theta(f(n)).$$

Substitution (example): Consider the following recurrence
$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have
$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by 2^{i+1} we get
$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find
$$u_{i+1} = \tfrac{1}{2} + u_i, \qquad u_1 = \tfrac{1}{2},$$

which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$.

Summing factors (example): Consider the following recurrence
$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving T are on the left side
$$T(n) - 3T(n/2) = n.$$

Now expand the recurrence, and choose a factor which makes the left side "telescope"

$$\begin{aligned}
1\big(T(n) - 3T(n/2) &= n\big) \\
3\big(T(n/2) - 3T(n/4) &= n/2\big) \\
\vdots \quad \vdots \quad &\vdots \\
3^{\log_2 n - 1}\big(T(2) - 3T(1) &= 2\big)
\end{aligned}$$

Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get
$$\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\tfrac{3}{2}\right)^i.$$

Let $c = \frac{3}{2}$. Then we have
$$\begin{aligned}
n \sum_{i=0}^{m-1} c^i &= n\left(\frac{c^m - 1}{c - 1}\right) \\
&= 2n(c^{\log_2 n} - 1) \\
&= 2n(c^{(k-1)\log_c n} - 1) \\
&= 2n^k - 2n,
\end{aligned}$$

and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider
$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that
$$T_{i+1} = 1 + \sum_{j=0}^{i} T_j.$$

Subtracting we find
$$\begin{aligned}
T_{i+1} - T_i &= 1 + \sum_{j=0}^{i} T_j - 1 - \sum_{j=0}^{i-1} T_j \\
&= T_i.
\end{aligned}$$

And so $T_{i+1} = 2T_i = 2^{i+1}$.

Generating functions:

1. Multiply both sides of the equation by x^i.
2. Sum both sides over all i for which the equation is valid.
3. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$.
3. Rewrite the equation in terms of the generating function $G(x)$.
4. Solve for $G(x)$.
5. The coefficient of x^i in $G(x)$ is g_i.

Example:
$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$

Multiply and sum:
$$\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i.$$

We choose $G(x) = \sum_{i \geq 0} x^i g_i$. Rewrite in terms of $G(x)$:
$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$$

Simplify:
$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$

Solve for $G(x)$:
$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

Expand this using partial fractions:
$$\begin{aligned}
G(x) &= x\left(\frac{2}{1-2x} - \frac{1}{1-x}\right) \\
&= x\left(2\sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i\right) \\
&= \sum_{i \geq 0} (2^{i+1} - 1) x^{i+1}.
\end{aligned}$$

So $g_i = 2^i - 1$.