Team Reference
Universidad de La Habana: UH++

ACM-ICPC Caribbean Finals 2015

Team Members: Marcelo José Fornet Fornés, José Carlos Gutiérrez Pérez, Abdel Rodríguez Solis
Coach: Alfredo Somoza Moreno

Contents

1. Data Structures

1.1. Union Find.

```
struct union_find
{
      vector<int> p;

      union_find(int n) : p(n, -1) {}

      bool join(int u, int v)
      {
            if ((u = root(u)) == (v = root(v)))
                  return false;
            if (p[u] > p[v])
                  swap(u, v);
            p[u] += p[v];
            p[v] = u;
            return true;
      }

      int root(int u)
      {
            return p[u] < 0 ? u : p[u] = root(p[u]);
      }
};
```

1.2. Randomized KD Tree.

```
typedef complex<double> point;

struct randomized_kd_tree
{
      struct node
      {
            point p;
            int d, s;
            node *l, *r;
            bool is_left_of(node *x)
            {
                  if (x->d)
                        return real(p) < real(x->p);
                  else
                        return imag(p) < imag(x->p);
            }
      }*root;

      randomized_kd_tree() : root(0) {}

      int size(node *t)
      {
            return t ? t->s : 0;
      }

      node *update(node *t)
      {
            t->s = 1 + size(t->l) + size(t->r);
            return t;
      }

      pair<node*, node*> split(node *t, node *x)
      {
            if (!t)
                  return {0, 0};
            if (t->d == x->d)
            {
                  if (t->is_left_of(x))
                  {
                        auto p = split(t->r, x);
                        t->r = p.first;
                        return {update(t), p.second};
                  }
                  else
                  {
                        auto p = split(t->l, x);
                        t->l = p.second;
                        return {p.first, update(t)};
                  }
            }
            else
```

```
        {
            auto l = split(t->l, x);
            auto r = split(t->r, x);
            if (t->is_left_of(x))
            {
                t->l = l.first;
                t->r = r.first;
                return {update(t), join(l.second, r.second, t->d)};
            }
            else
            {
                t->l = l.second;
                t->r = r.second;
                return {join(l.first, r.first, t->d), update(t)};
            }
        }
    }

    node *join(node *l, node *r, int d)
    {
        if (!l)
            return r;
        if (!r)
            return l;
        if (rand() % (size(l) + size(r)) < size(l))
        {
            if (l->d == d)
            {
                l->r = join(l->r, r, d);
                return update(l);
            }
            else
            {
                auto p = split(r, l);
                l->l = join(l->l, p.first, d);
                l->r = join(l->r, p.second, d);
                return update(l);
            }
        }
        else
        {
            if (r->d == d)
            {
                r->l = join(l, r->l, d);
                return update(r);
            }
            else
            {
                auto p = split(l, r);
                r->l = join(p.first, r->l, d);
                r->r = join(p.second, r->r, d);
                return update(r);
            }
        }
    }

    node *insert(node *t, node *x)
    {
        if (rand() % (size(t) + 1) == 0)
        {
            auto p = split(t, x);
            x->l = p.first;
            x->r = p.second;
            return update(x);
        }
        else
        {
            if (x->is_left_of(t))
                t->l = insert(t->l, x);
            else
                t->r = insert(t->r, x);
            return update(t);
        }
    }

    void insert(point p)
    {
        root = insert(root, new node({ p, rand() % 2 }));
    }

    node *remove(node *t, node *x)
    {
        if (!t)
            return t;
        if (t->p == x->p)
            return join(t->l, t->r, t->d);
        if (x->is_left_of(t))
            t->l = remove(t->l, x);
        else
            t->r = remove(t->r, x);
        return update(t);
    }

    void remove(point p)
```

```
	{
		node n = { p };
		root = remove(root, &n);
	}

	void closest(node *t, point p, pair<double, node*> &ub)
	{
		if (!t)
			return;
		double r = norm(t->p - p);
		if (r < ub.first)
			ub = {r, t};
		node *first = t->r, *second = t->l;
		double w = t->d ? real(p - t->p) : imag(p - t->p);
		if (w < 0)
			swap(first, second);
		closest(first, p, ub);
		if (ub.first > w * w)
			closest(second, p, ub);
	}

	point closest(point p)
	{
		pair<double, node*> ub(1.0 / 0.0, 0);
		closest(root, p, ub);
		return ub.second->p;
	}

	// verification
	int height(node *n)
	{
		return n ? 1 + max(height(n->l), height(n->r)) : 0;
	}
	int height()
	{
		return height(root);
	}

	int size_rec(node *n)
	{
		return n ? 1 + size_rec(n->l) + size_rec(n->r) : 0;
	}

	int size_rec()
	{
		return size_rec(root);
	}

	void display(node *n, int tab = 0)
	{
		if (!n)
			return;
		display(n->l, tab + 2);
		for (int i = 0; i < tab; ++i)
			cout << "␣";
		cout << n->p << "␣(" << n->d << ")" << endl;
		display(n->r, tab + 2);
	}

	void display()
	{
		display(root);
	}
};
```

1.3. Vantage Point Tree.

```
/*
	Vantage Point Tree (vp tree)

	Description:
	Vantage point tree is a metric tree.
	Each tree node has a point, radius, and two childs.
	The points of left descendants are contained in the ball B(p,r)
	and the points of right descendants are exluded from the ball.

	We can find k-nearest neighbors of a given point p efficiently
	by pruning search.

	Complexity:
	Construction: O(n log n)
	Search: O(log n)
*/

typedef complex<double> point;

namespace std
```

```
{
      bool operator <(point p, point q)
      {
            if (real(p) != real(q))
                  return real(p) < real(q);
            return imag(p) < imag(q);
      }
}

struct vantage_point_tree
{
      struct node
      {
            point p;
            double th;
            node *l, *r;
      }*root;

      vector<pair<double, point>> aux;

      vantage_point_tree(vector<point> ps)
      {
            for (int i = 0; i < ps.size(); ++i)
                  aux.push_back({ 0, ps[i] });
            root = build(0, ps.size());
      }

      node *build(int l, int r)
      {
            if (l == r)
                  return 0;
            swap(aux[l], aux[l + rand() % (r - l)]);
            point p = aux[l++].second;
            if (l == r)
                  return new node({ p });
            for (int i = l; i < r; ++i)
                  aux[i].first = norm(p - aux[i].second);
            int m = (l + r) / 2;
            nth_element(aux.begin() + l, aux.begin() + m, aux.begin() + r);
            return new node({ p, sqrt(aux[m].first), build(l, m), build(m, r) });
      }

      priority_queue<pair<double, node*>> que;

      void k_nn(node *t, point p, int k)
      {
            if (!t)
                  return;
            double d = abs(p - t->p);
            if (que.size() < k)
                  que.push({ d, t });
            else if (que.top().first > d)
            {
                  que.pop();
                  que.push({ d, t });
            }
            if (!t->l && !t->r)
                  return;
            if (d < t->th)
            {
                  k_nn(t->l, p, k);
                  if (t->th - d <= que.top().first)
                        k_nn(t->r, p, k);
            }
            else
            {
                  k_nn(t->r, p, k);
                  if (d - t->th <= que.top().first)
                        k_nn(t->l, p, k);
            }
      }

      vector<point> k_nn(point p, int k)
      {
            k_nn(root, p, k);
            vector<point> ans;
            for (; !que.empty(); que.pop())
                  ans.push_back(que.top().second->p);
            reverse(ans.begin(), ans.end());
            return ans;
      }
};
```

2. Dynamic Programming

2.1. Convex hull trick.

```
/*
	Tested: http://www.infoarena.ro/problema/euro
*/

typedef long long ll;

const ll oo = 0x3f3f3f3f3f3f3f3f;

// upper hull
struct convex_hull_trick
{
	convex_hull_trick(int sz = 50) : h(sz) {}

	void add(ll a, ll b)
	{
		hull cur = hull(a, b);
		int i;
		for (i = 1; !h[i].isEmpty(); ++i)
		{
			cur = merge(cur, h[i]);
			h[i].setEmpty();
		}
		h[i] = cur;
	}

	ll get_max(ll x)
	{
		ll ans = -oo;
		for (size_t i = 1; i < h.size(); ++i)
			if (!h[i].isEmpty())
				ans = max(ans, h[i].get(x));
		return ans;
	}

private:
	struct hull
	{
		vector<pair<ll, ll>> v;

		hull() { }

		hull(ll a, ll b)
		{
			v.clear();
			v.push_back(make_pair(a, b));
		}

		void add(ll a, ll b)
		{
			while (v.size() >= 1 && v.back().first == a
				&& v.back().second < b)
				v.pop_back();
			if (!v.empty() && v.back().first == a)
				return;
			while (v.size() >= 2)
			{
				ll a1, a2, b1, b2;
				a1 = v[(int) v.size() - 2].first;
				a2 = v[(int) v.size() - 1].first;
				b1 = v[(int) v.size() - 2].second;
				b2 = v[(int) v.size() - 1].second;
				if ((b1 - b2) * (a - a1) > (b1 - b) * (a2 - a1))
					v.pop_back();
				else
					break;
			}
			v.push_back(make_pair(a, b));
		}

		bool isEmpty()
		{
			return v.empty();
		}

		void setEmpty()
		{
			v.clear();
		}

		ll get(ll x)
		{
			int lo = 0, hi = (int) v.size() - 1;
			while (lo < hi)
			{
				int mid = (lo + hi) >> 1;
				if (f(v[mid].first, v[mid].second, x)
                        <= f(v[mid + 1].first, v[mid + 1].second, x))
                        lo = mid + 1;
                    else
                        hi = mid;
                }
                return f(v[lo].first, v[lo].second, x);
            }

            ll f(ll a, ll b, ll x)
            {
                return a * x + b;
            }
        };

        hull merge(const hull &a, const hull &b)
        {
            size_t i, j;
            i = j = 0;
            hull ans;
            while (i < a.v.size() && j < b.v.size())
            {
                if (a.v[i].first < b.v[j].first)
                    ans.add(a.v[i].first, a.v[i].second), ++i;
                else
                    ans.add(b.v[j].first, b.v[j].second), ++j;
            }
            while (i < a.v.size())
                ans.add(a.v[i].first, a.v[i].second), ++i;
            while (j < b.v.size())
                ans.add(b.v[j].first, b.v[j].second), ++j;
            return ans;
        }

        vector<hull> h;
    };
```

3. Geometry

3.1. Basics.

```
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
double 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); }

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();
      }
}
```

3.2. Polygon Area.

```
/*
      Tested: AIZU(judge.u-aizu.ac.jp) CGL.3A
      Complexity: O(n)
*/

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;
}
```

3.3. Polygon Centroid.

```
/*
      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;
}
```

3.4. Convex hull.

```
/*
	Tested: AIZU(judge.u-aizu.ac.jp) CGL.4A
	Complexity: O(n log n)
*/

polygon convex_hull(vector<point> &P)
{
	int n = P.size(), k = 0;
	vector<point> h(2 * n);
	sort(P.begin(), P.end());
	for (int i = 0; i < n; h[k++] = P[i++])
		while (k >= 2 && area2(h[k - 2], h[k - 1], P[i]) <= 0) --k;
	for (int i = n - 2, t = k + 1; i >= 0; h[k++] = P[i--])
		while (k >= t && area2(h[k - 2], h[k - 1], P[i]) <= 0) --k;
	return polygon(h.begin(), h.begin() + k - (k > 1));
}
```

3.5. Circles.

```
/*
	Circles

	Tested: AIZU
*/

// 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;
}

/*
	Area of the intersection of a circle with a polygon
	Circle's center lies in (0, 0)
	Polygon must be given counterclockwise

	Tested: LightOJ 1358
	Complexity: O(n)
*/

#define x(_t) (xa + (_t) * a)
#define y(_t) (ya + (_t) * b)

double radian(double xa, double ya, double xb, double yb)
{
	return atan2(xa * yb - xb * ya, xa * xb + ya * yb);
}

double part(double xa, double ya, double xb, double yb, double r)
{
	double l = sqrt((xa - xb) * (xa - xb) + (ya - yb) * (ya - yb));
	double a = (xb - xa) / l, b = (yb - ya) / l, c = a * xa + b * ya;
	double d = 4.0 * (c * c - xa * xa - ya * ya + r * r);
	if (d < eps)
		return radian(xa, ya, xb, yb) * r * r * 0.5;
	else
	{
		d = sqrt(d) * 0.5;
		double s = -c - d, t = -c + d;
		if (s < 0.0) s = 0.0;
		else if (s > l) s = l;
		if (t < 0.0) t = 0.0;
		else if (t > l) t = l;
		return (x(s) * y(t) - x(t) * y(s)
				+ (radian(xa, ya, x(s), y(s))
				+ radian(x(t), y(t), xb, yb)) * r * r) * 0.5;
	}
}

double intersection_circle_polygon(const polygon &P, double r)
{
	double s = 0.0;
	int n = P.size();;
	for (int i = 0; i < n; i++)
		s += part(P[i].real(), P[i].imag(),
			P[NEXT(i)].real(), P[NEXT(i)].imag(), r);
	return fabs(s);
}
```

3.6. Closest Pair.

```
/*
	Compute distance between closest points.

	Tested: AIZU(judge.u-aizu.ac.jp) CGL.5A
	Complexity: O(n log n)
*/

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;
}
```

3.7. Point in Polygon.

```
/*
	Determine the position of a point relative
	to a polygon.

	Tested: AIZU(judge.u-aizu.ac.jp) CGL.3C
	Complexity: O(n)
*/

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;
}
```

3.8. Convex cut.

```
/*
	Cut a convex polygon by a line and
	return the part to the left of the line

	Tested: AIZU(judge.u-aizu.ac.jp) CGL.4C
	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;
}
```

3.9. Lines and Segments.

```
/*
	Line and segments predicates

	Tested: AIZU(judge.u-aizu.ac.jp) CGL
*/

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);
	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);
}
```

3.10. Pick Theorem.

```
/*
	Pick's theorem
	A = I + B/2 - 1:
	A = Area of the polygon
	I = Number of integer coordinates points inside
	B = Number of integer coordinates points on the boundary

	Polygon's vertex must have integer coordinates

	Tested: LightOJ 1418
	Complexity: O(n)
*/

typedef long long ll;
typedef complex<ll> point;
struct segment { point p, q; };

ll points_on_segment(const segment &s)
{
	point p = s.p - s.q;
	return __gcd(abs(p.real()), abs(p.imag()));
}
```

```
// <Lattice points (not in boundary), Lattice points on boundary>
pair<ll, ll> pick_theorem(polygon &P)
{
      ll A = area2(P), B = 0, I = 0;
      for (int i = 0, n = P.size(); i < n; ++i)
            B += points_on_segment({P[i], P[NEXT(i)]});
      A = abs(A);
      I = (A - B) / 2 + 1;
      return {I, B};
}
```

3.11. Rectangle Union.

```
/*
      Tested: MIT 2008 Team Contest 1 (Rectangles)
      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;
}
```

3.12. Rectilinear MST.

```
/*
	Tested: USACO OPEN08 (Cow Neighborhoods)
	Complexity: O(n log n)
*/

typedef long long ll;
typedef complex<ll> point;

ll rectilinear_mst(vector<point> ps)
{
	vector<int> id(ps.size());
	iota(id.begin(), id.end(), 0);

	struct edge
	{
		int src, dst;
		ll weight;
	};

	vector<edge> edges;
	for (int s = 0; s < 2; ++s)
	{
		for (int t = 0; t < 2; ++t)
		{
			sort(id.begin(), id.end(), [&](int i, int j)
			{
				return real(ps[i] - ps[j]) < imag(ps[j] - ps[i]);
			});

			map<ll, int> sweep;

			for (int i : id)
			{
				for (auto it = sweep.lower_bound(-imag(ps[i]));
						it != sweep.end(); sweep.erase(it++))
				{
					int j = it->second;
					if (imag(ps[j] - ps[i]) < real(ps[j] - ps[i]))
						break;
					ll d = abs(real(ps[i] - ps[j]))
							+ abs(imag(ps[i] - ps[j]));
					edges.push_back({ i, j, d });
				}
				sweep[-imag(ps[i])] = i;
			}

			for (auto &p : ps)
				p = point(imag(p), real(p));
		}

		for (auto &p : ps)
			p = point(-real(p), imag(p));
	}

	ll cost = 0;
	sort(edges.begin(), edges.end(), [](edge a, edge b)
	{
		return a.weight < b.weight;
	});

	union_find uf(ps.size());
	for (edge e : edges)
		if (uf.join(e.src, e.dst))
			cost += e.weight;

	return cost;
}
```

3.13. Antipodal Points.

```
/*
	Antipodal points

	Tested: AIZU(judge.u-aizu.ac.jp) CGL.4B
	Complexity: O(n)
*/

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;
}
```

3.14. Polygon width.

```
/*
	Compute the width of a convex polygon

	Tested: LiveArchive 5138
	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;
}
```

3.15. Point 3D.

```
const double pi = acos(-1.0);

// Construct a point on a sphere with center on the origin and radius R
// TESTED [COJ-1436]
struct point3d
{
      double x, y, z;

      point3d(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}

      double operator*(const point3d &p) const
      {
            return x * p.x + y * p.y + z * p.z;
      }

      point3d operator-(const point3d &p) const
      {
            return point3d(x - p.x, y - p.y, z - p.z);
      }
};

double abs(point3d p)
{
      return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
}

point3d from_polar(double lat, double lon, double R)
{
      lat = lat / 180.0 * pi;
      lon = lon / 180.0 * pi;
      return point3d(R * cos(lat) * sin(lon), R * cos(lat) * cos(lon), R * sin(lat));
}

struct plane
{
      double A, B, C, D;
};

double euclideanDistance(point3d p, point3d q)
{
      return abs(p - q);
}

/*
 Geodisic distance between points in a sphere
 R is the radius of the sphere
 */
double geodesic_distance(point3d p, point3d q, double r)
{
      return r * acos(p * q / r / r);
}

const double eps = 1e-9;

// Find the rect of intersection of two planes on the space
// The rect is given parametrical
// TESTED [TIMUS 1239]
void planePlaneIntersection(plane p, plane q)
{
      if (abs(p.C * q.B - q.C * p.B) < eps)
            return; // Planes are parallel

      double mz = (q.A * p.B - p.A * q.B) / (p.C * q.B - q.C * p.B);
      double nz = (q.D * p.B - p.D * q.B) / (p.C * q.B - q.C * p.B);

      double my = (q.A * p.C - p.A * q.C) / (p.B * q.C - p.C * q.B);
      double ny = (q.D * p.C - p.D * q.C) / (p.B * q.C - p.C * q.B);

      // parametric rect: (x, my * x + ny, mz * x * nz)
}
```

4. Graph

4.1. Chu-Liu/Edmonds $O(E \log V)$.

```
/*
	Minimum Arborescence
	Tested: UVA 11183
	Complexity: O(m log n)
*/

template<typename T>
struct edge
{
	int src, dst;
	T weight;
};

template<typename T>
struct skew_heap
{
	struct node
	{
		node *ch[2];
		edge<T> key;
		T delta;
	}*root;

	skew_heap() : root(nullptr) {}

	void propagate(node *a)
	{
		a->key.weight += a->delta;
		if (a->ch[0])
			a->ch[0]->delta += a->delta;
		if (a->ch[1])
			a->ch[1]->delta += a->delta;
		a->delta = 0;
	}

	node *merge(node *a, node *b)
	{
		if (!a || !b)
			return a ? a : b;
		propagate(a);
		propagate(b);
		if (a->key.weight > b->key.weight)
			swap(a, b);
		a->ch[1] = merge(b, a->ch[1]);
		swap(a->ch[0], a->ch[1]);
		return a;
	}

	void push(edge<T> key)
	{
		node *n = new node();
		n->ch[0] = n->ch[1] = 0;
		n->key = key;
		n->delta = 0;
		root = merge(root, n);
	}

	void pop()
	{
		propagate(root);
		root = merge(root->ch[0], root->ch[1]);
	}

	edge<T> top()
	{
		propagate(root);
		return root->key;
	}

	bool empty()
	{
		return !root;
	}

	void add(T delta)
	{
		root->delta += delta;
	}

	void merge(skew_heap x)
	{
		root = merge(root, x.root);
	}
};

template<typename T>
```

```
struct minimum_aborescense
{
      vector<edge<T>> edges;

      void add_edge(int src, int dst, T weight)
      {
            edges.push_back({ src, dst, weight });
      }

      T solve(int r)
      {
            int n = 0;
            for (auto e : edges)
                  n = max(n, max(e.src, e.dst) + 1);

            union_find uf(n);
            vector<skew_heap<T>> heap(n);
            for (auto e : edges)
                  heap[e.dst].push(e);

            T score = 0;
            vector<int> seen(n, -1);
            seen[r] = r;
            for (int s = 0; s < n; ++s)
            {
                  vector<int> path;
                  for (int u = s; seen[u] < 0;)
                  {
                        path.push_back(u);
                        seen[u] = s;
                        if (heap[u].empty())
                              return numeric_limits<T>::max();

                        edge<T> min_e = heap[u].top();
                        score += min_e.weight;
                        heap[u].add(-min_e.weight);
                        heap[u].pop();

                        int v = uf.root(min_e.src);
                        if (seen[v] == s)
                        {
                              skew_heap<T> new_heap;
                              while (1)
                              {
                                    int w = path.back();
                                    path.pop_back();
                                    new_heap.merge(heap[w]);
                                    if (!uf.join(v, w))
                                          break;
                              }
                              heap[uf.root(v)] = new_heap;
                              seen[uf.root(v)] = -1;
                        }
                        u = uf.root(v);
                  }
            }

            return score;
      }
};
```

4.2. **Chu-Liu/Edmonds $O(EV)$.**

```
/*
      Minimum Arborescence (Chu-Liu/Edmonds)
      Tested: UVA 11183
      Complexity: O(mn)
*/

template<typename T>
struct minimum_aborescense
{
      struct edge
      {
            int src, dst;
            T weight;
      };

      vector<edge> edges;

      void add_edge(int u, int v, T w)
      {
            edges.push_back({ u, v, w });
      }

      T solve(int r)
      {
            int n = 0;
            for (auto e : edges)
            n = max(n, max(e.src, e.dst) + 1);

        int N = n;

        for (T res = 0;;)
        {
            vector<edge> in(N, { -1, -1, numeric_limits<T>::max() });
            vector<int> C(N, -1);
            for (auto e : edges) // cheapest comming edges
                if (in[e.dst].weight > e.weight)
                    in[e.dst] = e;

            in[r] = {r, r, 0};
            for (int u = 0; u < N; ++u)
            { // no comming edge ==> no aborescense
                if (in[u].src < 0)
                    return numeric_limits<T>::max();
                res += in[u].weight;
            }

            vector<int> mark(N, -1); // contract cycles
            int index = 0;
            for (int i = 0; i < N; ++i)
            {
                if (mark[i] != -1)
                    continue;
                int u = i;
                while (mark[u] == -1)
                {
                    mark[u] = i;
                    u = in[u].src;
                }
                if (mark[u] != i || u == r)
                    continue;
                for (int v = in[u].src; u != v; v = in[v].src)
                    C[v] = index;
                C[u] = index++;
            }

            if (index == 0)
                return res; // found arborescence
            for (int i = 0; i < N; ++i) // contract
                if (C[i] == -1)
                    C[i] = index++;

            vector<edge> next;
            for (auto &e : edges)
                if (C[e.src] != C[e.dst] && C[e.dst] != C[r])
                    next.push_back({ C[e.src], C[e.dst],
                        e.weight - in[e.dst].weight });
            edges.swap(next);
            N = index;
            r = C[r];
        }
    }
};
```

4.3. Articulation Points.

```
/*
    Articulation points / Biconnected components
    Tested: SPOJ SUBMERGE
    Complexity: O(n + m)
*/

struct graph
{
    int n;
    vector<vector<int>> adj;

    graph(int n) : n(n), adj(n) {}

    void add_edge(int src, int dst)
    {
        adj[src].push_back(dst);
        adj[dst].push_back(src);
    }
};

void biconnected_components(const graph &g)
{
    vector<int> arts(g.n), num(g.n), low(g.n), S;
    vector<vector<int>> comps;
    function<void(int, int, int&)> dfs = [&](int u, int p, int &t)
    {
        num[u] = low[u] = ++t;
        S.push_back(u);
        for (int v : g.adj[u])
        {
```

```
                  if (v == p) continue;
                  if (num[v] == 0)
                  {
                        dfs(v, u, t);
                        low[u] = min(low[u], low[v]);
                        if (num[u] <= low[v])
                        {
                              if (num[u] != 1 || low[v] > 2) arts[u] = true;
                              for (comps.push_back({u});
                                    comps.back().back() != v; S.pop_back())
                              comps.back().push_back(S.back());
                        }
                  }
                  else low[u] = min(low[u], num[v]);
            }
      };
      for (int u = 0, t; u < g.n; ++u)
            if (num[u] == 0)
                  dfs(u, -1, t = 0);
}
```

4.4. Bridges.

```
/*
      Tested: AIZU(judge.u-aizu.ac.jp) GRL_3_B
      Complexity: O(n + m)
*/

struct graph
{
      int n;
      vector<vector<int>> adj;

      graph(int n) : n(n), adj(n) {}

      void add_edge(int src, int dst)
      {
            adj[src].push_back(dst);
            adj[dst].push_back(src);
      }
};

void bridge_components(const graph &g)
{
      vector<int> num(g.n), low(g.n);
      vector<pair<int, int>> S, bridges;
      vector<vector<pair<int, int>>> comps;

      function<void(int, int, int&)> dfs = [&](int u, int p, int &t)
      {
            num[u] = low[u] = ++t;
            for (int v : g.adj[u])
            {
                  if (v == p) continue;
                  S.push_back({u, v});
                  if (num[v] == 0)
                  {
                        dfs(v, u, t);
                        low[u] = min(low[u], low[v]);
                        if (num[u] < low[v])
                        {
                              pair<int, int> e = {u, v};
                              for (comps.push_back({});
                                    S.back() != e; S.pop_back())
                              comps.back().push_back(S.back());
                              bridges.push_back(e);
                              S.pop_back();
                        }
                  }
                  else low[u] = min(low[u], num[v]);
            }
      };

      for (int u = 0, t; u < g.n; ++u)
            if (num[u] == 0)
            {
                  dfs(u, -1, t = 0);
                  if (!S.empty())
                  {
                        for (comps.push_back({}); !S.empty(); S.pop_back())
                              comps.back().push_back(S.back());
                  }
            }
}
```

4.5. Bipartite Matching (Kuhn).

```
/*
	Tested: AIZU(judge.u-aizu.ac.jp) GRL_7_A
	Complexity: O(nm)
*/

struct graph
{
	int L, R;
	vector<vector<int>> adj;

	graph(int L, int R) : L(L), R(R), adj(L + R) {}

	void add_edge(int u, int v)
	{
		adj[u].push_back(v + L);
		adj[v + L].push_back(u);
	}

	int maximum_matching()
	{
		vector<int> visited(L), mate(L + R, -1);
		function<bool(int)> augment = [&](int u)
		{
			if (visited[u]) return false;
			visited[u] = true;
			for (int w : adj[u])
			{
				int v = mate[w];
				if (v < 0 || augment(v))
				{
					mate[u] = w;
					mate[w] = u;
					return true;
				}
			}
			return false;
		};
		int match = 0;
		for (int u = 0; u < L; ++u)
		{
			fill(visited.begin(), visited.end(), 0);
			if (augment(u))
				++match;
		}
		return match;
	}
};
```

4.6. Dinic.

```
/*
	Tested: SPOJ MAXFLOW
	Complexity: (n^2*m)
*/

template<typename T>
struct dinic
{
	struct edge
	{
		int src, dst;
		T cap, flow;
		int rev;
	};

	int n;
	vector<vector<edge>> adj;

	dinic(int n) : n(n), adj(n) {}

	void add_edge(int src, int dst, T cap)
	{
		adj[src].push_back({ src, dst, cap, 0, (int) adj[dst].size() });
		if (src == dst)
			adj[src].back().rev++;
		adj[dst].push_back({ dst, src, 0, 0, (int) adj[src].size() - 1 });
	}

	vector<int> level, iter;

	T augment(int u, int t, T cur)
	{
		if (u == t)
			return cur;
		for (int &i = iter[u]; i < (int) adj[u].size(); ++i)
		{
			edge &e = adj[u][i];
			if (e.cap - e.flow > 0 && level[u] > level[e.dst])
			{
				T f = augment(e.dst, t, min(cur, e.cap - e.flow));
				if (f > 0)
				{
					e.flow += f;
					adj[e.dst][e.rev].flow -= f;
					return f;
				}
			}
		}
		return 0;
	}

	int bfs(int s, int t)
	{
		level.assign(n, n);
		level[t] = 0;
		queue<int> Q;
		for (Q.push(t); !Q.empty(); Q.pop())
		{
			int u = Q.front();
			if (u == s)
				break;
			for (edge &e : adj[u])
			{
				edge &erev = adj[e.dst][e.rev];
				if (erev.cap - erev.flow > 0
					&& level[e.dst] > level[u] + 1)
				{
					Q.push(e.dst);
					level[e.dst] = level[u] + 1;
				}
			}
		}
		return level[s];
	}

	const T oo = numeric_limits<T>::max();

	T max_flow(int s, int t)
	{
		for (int u = 0; u < n; ++u) // initialize
			for (auto &e : adj[u])
				e.flow = 0;

		T flow = 0;
		while (bfs(s, t) < n)
		{
			iter.assign(n, 0);
			for (T f; (f = augment(s, t, oo)) > 0;)
				flow += f;
		} // level[u] == n ==> s-side
		return flow;
	}
};
```

4.7. Dominator Tree.

```
/*
	Dominator Tree (Lengauer-Tarjan)

	Tested: SPOJ EN
	Complexity: O(m log n)
*/

struct graph
{
	int n;
	vector<vector<int>> adj, radj;

	graph(int n) : n(n), adj(n), radj(n) {}

	void add_edge(int src, int dst)
	{
		adj[src].push_back(dst);
		radj[dst].push_back(src);
	}

	vector<int> rank, semi, low, anc;

	int eval(int v)
	{
		if (anc[v] < n && anc[anc[v]] < n)
		{
			int x = eval(anc[v]);
			if (rank[semi[low[v]]] > rank[semi[x]])
				low[v] = x;
			anc[v] = anc[anc[v]];
		}
		return low[v];
	}

	vector<int> prev, ord;

	void dfs(int u)
	{
		rank[u] = ord.size();
		ord.push_back(u);
		for (auto v : adj[u])
		{
			if (rank[v] < n)
				continue;
			dfs(v);
			prev[v] = u;
		}
	}

	vector<int> idom; // idom[u] is an immediate dominator of u

	void dominator_tree(int r)
	{
		idom.assign(n, n);
		prev = rank = anc = idom;
		semi.resize(n);
		iota(semi.begin(), semi.end(), 0);
		low = semi;
		ord.clear();
		dfs(r);
		vector<vector<int>> dom(n);
		for (int i = (int) ord.size() - 1; i >= 1; --i)
		{
			int w = ord[i];
			for (auto v : radj[w])
			{
				int u = eval(v);
				if (rank[semi[w]] > rank[semi[u]])
					semi[w] = semi[u];
			}
			dom[semi[w]].push_back(w);
			anc[w] = prev[w];
			for (int v : dom[prev[w]])
			{
				int u = eval(v);
				idom[v] = (rank[prev[w]] > rank[semi[u]]
					? u : prev[w]);
			}
			dom[prev[w]].clear();
		}

		for (int i = 1; i < (int) ord.size(); ++i)
		{
			int w = ord[i];
			if (idom[w] != semi[w])
				idom[w] = idom[idom[w]];
		}
	}

	vector<int> dominators(int u)
	{
		vector<int> S;
		for (; u < n; u = idom[u])
			S.push_back(u);
		return S;
	}
};
```

4.8. Flow with lower bound.

```
/*
	Flow with lower bound

	Tested: ZOJ 3229
	Complexity: O(n^2 m)
*/

template<typename T>
struct dinic
{
	struct edge
	{
		int src, dst;
		T low, cap, flow;
		int rev;
	};

	int n;
	vector<vector<edge>> adj;

	dinic(int n) : n(n), adj(n + 2) {}

	void add_edge(int src, int dst, T low, T cap)
	{
		adj[src].push_back({ src, dst, low, cap, 0, (int) adj[dst].size() });
		if (src == dst)
			adj[src].back().rev++;
		adj[dst].push_back({ dst, src, 0, 0, 0, (int) adj[src].size() - 1 });
	}

	vector<int> level, iter;

	T augment(int u, int t, T cur)
	{
		if (u == t)
			return cur;
		for (int &i = iter[u]; i < (int) adj[u].size(); ++i)
		{
			edge &e = adj[u][i];
			if (e.cap - e.flow > 0 && level[u] > level[e.dst])
			{
				T f = augment(e.dst, t, min(cur, e.cap - e.flow));
				if (f > 0)
				{
					e.flow += f;
					adj[e.dst][e.rev].flow -= f;
					return f;
				}
			}
		}
		return 0;
	}

	int bfs(int s, int t)
	{
		level.assign(n + 2, n + 2);
		level[t] = 0;
		queue<int> Q;
		for (Q.push(t); !Q.empty(); Q.pop())
		{
			int u = Q.front();
			if (u == s)
				break;
			for (edge &e : adj[u])
			{
				edge &erev = adj[e.dst][e.rev];
				if (erev.cap - erev.flow > 0
					&& level[e.dst] > level[u] + 1)
				{
					Q.push(e.dst);
					level[e.dst] = level[u] + 1;
				}
			}
		}
		return level[s];
	}

	const T oo = numeric_limits<T>::max();

	T max_flow(int source, int sink)
	{
		vector<T> delta(n + 2);

		for (int u = 0; u < n; ++u) // initialize
			for (auto &e : adj[u])
			{
				delta[e.src] -= e.low;
				delta[e.dst] += e.low;
				e.cap -= e.low;
				e.flow = 0;
			}

		T sum = 0;
		int s = n, t = n + 1;

		for (int u = 0; u < n; ++u)
		{
			if (delta[u] > 0)
			{
				add_edge(s, u, 0, delta[u]);
				sum += delta[u];
			}
			else if (delta[u] < 0)
				add_edge(u, t, 0, -delta[u]);
		}
```

```
        add_edge(sink, source, 0, oo);
        T flow = 0;

        while (bfs(s, t) < n + 2)
        {
            iter.assign(n + 2, 0);
            for (T f; (f = augment(s, t, oo)) > 0;)
                flow += f;
        }

        if (flow != sum)
            return -1; // no solution

        for (int u = 0; u < n; ++u)
            for (auto &e : adj[u])
            {
                e.cap += e.low;
                e.flow += e.low;
                edge &erev = adj[e.dst][e.rev];
                erev.cap -= e.low;
                erev.flow -= e.low;
            }

        adj[sink].pop_back();
        adj[source].pop_back();

        while (bfs(source, sink) < n + 2)
        {
            iter.assign(n + 2, 0);
            for (T f; (f = augment(source, sink, oo)) > 0;)
                flow += f;
        } // level[u] == n + 2 ==> s-side

        return flow;
    }
};
```

4.9. **Gabow Edmonds.**

```
/*
    Tested: Timus 1099
    Complexity: O(n^3)
*/

struct graph
{
    int n;
    vector<vector<int>> adj;

    graph(int n) : n(n), adj(n) {}

    void add_edge(int u, int v)
    {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    queue<int> q;
    vector<int> label, mate, cycle;

    void rematch(int x, int y)
    {
        int m = mate[x];
        mate[x] = y;
        if (mate[m] == x)
        {
            if (label[x] < n)
                rematch(mate[m] = label[x], m);
            else
            {
                int s = (label[x] - n) / n, t = (label[x] - n) % n;
                rematch(s, t);
                rematch(t, s);
            }
        }
    }

    void traverse(int x)
    {
        vector<int> save = mate;
        rematch(x, x);
        for (int u = 0; u < n; ++u)
            if (mate[u] != save[u])
                cycle[u] ^= 1;
        save.swap(mate);
    }

    void relabel(int x, int y)
    {
```

```
		cycle = vector<int>(n, 0);
		traverse(x);
		traverse(y);
		for (int u = 0; u < n; ++u)
		{
			if (!cycle[u] || label[u] >= 0)
				continue;
			label[u] = n + x + y * n;
			q.push(u);
		}
	}

	int augment(int r)
	{
		label.assign(n, -2);
		label[r] = -1;
		q = queue<int>();
		for (q.push(r); !q.empty(); q.pop())
		{
			int x = q.front();
			for (int y : adj[x])
			{
				if (mate[y] < 0 && r != y)
				{
					rematch(mate[y] = x, y);
					return 1;
				}
				else if (label[y] >= -1)
					relabel(x, y);
				else if (label[mate[y]] < -1)
				{
					label[mate[y]] = x;
					q.push(mate[y]);
				}
			}
		}
		return 0;
	}

	int maximum_matching()
	{
		mate.assign(n, -2);
		int matching = 0;
		for (int u = 0; u < n; ++u)
			if (mate[u] < 0)
				matching += augment(u);
		return matching;
	}
};
```

4.10. Gomory-Hu Tree.

```
/*
	Gomory-Hu tree

	Tested: SPOj MCQUERY
	Complexity: O(n-1) max-flow call
*/

template<typename T>
struct graph
{
	struct edge
	{
		int src, dst;
		T cap, flow;
		int rev;
	};

	int n;
	vector<vector<edge>> adj;

	graph(int n) : n(n), adj(n) {}

	void add_edge(int src, int dst, T cap)
	{
		adj[src].push_back({ src, dst, cap, 0, (int) adj[dst].size() });
		if (src == dst)
			adj[src].back().rev++;
		adj[dst].push_back({ dst, src, cap, 0, (int) adj[src].size() - 1 });
	}

	vector<int> level, iter;
	T augment(int u, int t, T cur)
	{
		if (u == t)
			return cur;
		for (int &i = iter[u]; i < (int) adj[u].size(); ++i)
		{
			edge &e = adj[u][i];
			if (e.cap - e.flow > 0 && level[u] < level[e.dst])
			{
				T f = augment(e.dst, t, min(cur, e.cap - e.flow));
				if (f > 0)
				{
					e.flow += f;
					adj[e.dst][e.rev].flow -= f;
					return f;
				}
			}
		}
		return 0;
	}

	int bfs(int s, int t)
	{
		level.assign(n, -1);
		level[s] = 0;
		queue<int> Q;
		for (Q.push(s); !Q.empty(); Q.pop())
		{
			int u = Q.front();
			if (u == t)
				break;
			for (auto &e : adj[u])
			{
				if (e.cap - e.flow > 0 && level[e.dst] < 0)
				{
					Q.push(e.dst);
					level[e.dst] = level[u] + 1;
				}
			}
		}
		return level[t];
	}

	const T oo = numeric_limits<T>::max();

	T max_flow(int s, int t)
	{
		for (int u = 0; u < n; ++u) // initialize
			for (auto &e : adj[u])
				e.flow = 0;
		T flow = 0;
		while (bfs(s, t) >= 0)
		{
			iter.assign(n, 0);
			for (T f; (f = augment(s, t, oo)) > 0;)
				flow += f;
		} // level[u] == -1 ==> t-side
		return flow;
	}

	vector<edge> tree;

	void gomory_hu()
	{
		tree.clear();
		vector<int> parent(n);
		for (int u = 1; u < n; ++u)
		{
			tree.push_back({ u, parent[u], max_flow(u, parent[u]) });
			for (int v = u + 1; v < n; ++v)
				if (level[v] >= 0 && parent[v] == parent[u])
					parent[v] = u;
		}
	}
};
```

4.11. Bipartite Matching (Hopcroft-Karp).

```
/*
	Tested: SPOJ MATCHING
	Complexity: O(m n^1.5)
*/

struct graph
{
	int L, R;
	vector<vector<int>> adj;

	graph(int L, int R) : L(L), R(R), adj(L + R) {}

	void add_edge(int u, int v)
	{
		adj[u].push_back(v + L);
		adj[v + L].push_back(u);
```

```
    }

    int maximum_matching()
    {
        vector<int> level(L), mate(L + R, -1);

        function<bool(void)> levelize = [&]()
        {
            queue<int> Q;
            for (int u = 0; u < L; ++u)
            {
                level[u] = -1;
                if (mate[u] < 0)
                {
                    level[u] = 0;
                    Q.push(u);
                }
            }
            while (!Q.empty())
            {
                int u = Q.front(); Q.pop();
                for (int w : adj[u])
                {
                    int v = mate[w];
                    if (v < 0) return true;
                    if (level[v] < 0)
                    {
                        level[v] = level[u] + 1;
                        Q.push(v);
                    }
                }
            }
            return false;
        };

        function<bool(int)> augment = [&](int u)
        {
            for (int w : adj[u])
            {
                int v = mate[w];
                if (v < 0 || (level[v] > level[u] && augment(v)))
                {
                    mate[u] = w;
                    mate[w] = u;
                    return true;
                }
            }
            return false;
        };
        int match = 0;
        while (levelize())
            for (int u = 0; u < L; ++u)
                if (mate[u] < 0 && augment(u))
                    ++match;
        return match;
    }
};
```

4.12. Hungarian.

```
/*
    Tested: TIMUS 1833
    Complexity: O(n^3)
*/

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

4.13. Min-cost Flow (Dijkstra).

```
/*
	Tested: ZOJ 3885
*/

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;
	vector<edge*> prev;
	vector<T> curflow;

	bool dijkstra(int s, int t)
	{
		dist.assign(n, oo);
		prev.assign(n, nullptr);
		curflow.assign(n, 0);
		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)
				{
					dist[e.dst] = dist[u] + e.cost;
					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)
	{
		T flow = 0;
		C cost = 0;

		while (dijkstra(s, t))
		{
			T delta = curflow[t];
			flow += delta;
			cost += delta * dist[t];
			for (edge *e = prev[t]; e != nullptr; e = prev[e->src])
			{
				e->flow += delta;
				adj[e->dst][e->rev].flow -= delta;
			}
		}

		return {flow, cost};
	}
};
```

4.14. Min-cost Flow.

```
/*
	Maximum flow of minimum cost with potentials
	Tested: ZOJ 3885
	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)
    {
        T flow = 0;
        C cost = 0;

        // can be safely commented if
        // edges costs are non-negative
        bellman_ford(s, t);
        curflow.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};
    }
};
```

4.15. **Off-line LCA (Tarjan).**

```
/*
    Tested: SPOJ LCA
    Complexity: O((n + m) A(n))
*/

struct query
{
    int u, v, a;
};

struct tree
{
    int n;
    vector<vector<int>> adj;

    tree(int n) : n(n), adj(n) {}

    void add_edge(int src, int dst)
    {
        adj[src].push_back(dst);
        adj[dst].push_back(src);
    }

    void lca(vector<query> &queries)
    {
        vector<vector<query*>> Q(n);
        for (auto &q : queries)
        {
            Q[q.u].push_back(&q);
            Q[q.v].push_back(&q);
        }

        union_find uf(n);
        vector<int> anc(n), color(n);
        iota(anc.begin(), anc.end(), 0);

        function<void(int, int)> rec = [&](int u, int p)
        {
            for (int v : adj[u])
                if (v != p)
                {
                    rec(v, u);
                    uf.join(u, v);
                    anc[uf.root(u)] = u;
                }
            color[u] = 1;
            for (auto it : Q[u])
            {
                if (it->u != u) swap(it->u, it->v);
                if (color[it->v] == 1) it->a = anc[uf.root(it->v)];
            }
        };

        rec(0, -1);
    }
};
```

4.16. LCA (Euler-tour + RMQ).

```
struct tree
{
    int n;
    vector<vector<int>> adj;

    tree(int n) : n(n), adj(n) {}

    void add_edge(int s, int t)
    {
        adj[s].push_back(t);
        adj[t].push_back(s);
    }

    vector<int> pos, tour, depth;
    vector<vector<int>> table;

    int argmin(int i, int j)
    {
        return depth[i] < depth[j] ? i : j;
    }

    void rootify(int r)
    {
        pos.resize(n);
        function<void(int, int, int)> dfs = [&](int u, int p, int d)
        {
            pos[u] = depth.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(r, r, 0);
        int logn = __lg(tour.size()); // log2
        table.resize(logn + 1, vector<int>(tour.size()));
        iota(table[0].begin(), table[0].end(), 0);
        for (int h = 0; h < logn; ++h)
            for (int i = 0; i + (1 << h) < (int) tour.size(); ++i)
                table[h + 1][i] = argmin(table[h][i],
                                         table[h][i + (1 << h)]);
    }

    int lca(int u, int v)
    {
        int i = pos[u], j = pos[v];
        if (i > j) swap(i, j);
        int h = __lg(j - i); // = log2
        return i == j ? u : tour[argmin(table[h][i],
                                         table[h][j - (1 << h)])];
    }
};
```

4.17. Stoer Wagner.

```
/*
      Tested: ZOJ 2753
      Complexity: O(n^3)
*/

template<typename T>
pair<T, vector<int>> stoer_wagner(vector<vector<T>> &weights)
{
      int n = weights.size();
      vector<int> used(n), cut, best_cut;
      T best_weight = -1;

      for (int phase = n - 1; phase >= 0; --phase)
      {
            vector<T> w = weights[0];
            vector<int> added = used;
            int prev, last = 0;

            for (int i = 0; i < phase; ++i)
            {
                  prev = last;
                  last = -1;
                  for (int j = 1; j < n; ++j)
                        if (!added[j] && (last == -1 || w[j] > w[last]))
                              last = j;

                  if (i == phase - 1)
                  {
                        for (int j = 0; j < n; ++j)
                              weights[prev][j] += weights[last][j];
                        for (int j = 0; j < n; ++j)
                              weights[j][prev] = weights[prev][j];

                        used[last] = true;
                        cut.push_back(last);

                        if (best_weight == -1 || w[last] < best_weight)
                        {
                              best_cut = cut;
                              best_weight = w[last];
                        }
                  }
                  else
                  {
                        for (int j = 0; j < n; j++)
                              w[j] += weights[last][j];
                        added[last] = true;
                  }
            }
      }

      return make_pair(best_weight, best_cut);
}
```

4.18. Tree Isomorphism.

```
/*
      Tested: SPOJ TREEISO
      Complexity: O(n log n)
*/

#define all(c) (c).begin(), (c).end()

struct tree
{
      int n;
      vector<vector<int>> adj;

      tree(int n) : n(n), adj(n) {}

      void add_edge(int src, int dst)
      {
            adj[src].push_back(dst);
            adj[dst].push_back(src);
      }

      vector<int> centers()
      {
            vector<int> prev;
            int u = 0;
            for (int k = 0; k < 2; ++k)
            {
```

```
            queue<int> q;
            prev.assign(n, -1);
            for (q.push(prev[u] = u); !q.empty(); q.pop())
            {
                  u = q.front();
                  for (auto v : adj[u])
                  {
                        if (prev[v] >= 0)
                              continue;
                        q.push(v);
                        prev[v] = u;
                  }
            }
      }

      vector<int> path = { u };
      while (u != prev[u])
            path.push_back(u = prev[u]);

      int m = path.size();
      if (m % 2 == 0)
            return {path[m/2-1], path[m/2]};
      else
            return {path[m/2]};
}

vector<vector<int>> layer;
vector<int> prev;

int levelize(int r)
{
      prev.assign(n, -1);
      prev[r] = n;
      layer = {{r}};
      while (1)
      {
            vector<int> next;
            for (int u : layer.back())
                  for (int v : adj[u])
                  {
                        if (prev[v] >= 0)
                              continue;
                        prev[v] = u;
                        next.push_back(v);
                  }

            if (next.empty())
                              break;
                  layer.push_back(next);
            }
            return layer.size();
      }
};

bool isomorphic(tree S, int s, tree T, int t)
{
      if (S.n != T.n)
            return false;
      if (S.levelize(s) != T.levelize(t))
            return false;

      vector<vector<int>> longcodeS(S.n + 1), longcodeT(T.n + 1);
      vector<int> codeS(S.n), codeT(T.n);
      for (int h = (int) S.layer.size() - 1; h >= 0; --h)
      {
            map<vector<int>, int> bucket;
            for (int u : S.layer[h])
            {
                  sort(all(longcodeS[u]));
                  bucket[longcodeS[u]] = 0;
            }
            for (int u : T.layer[h])
            {
                  sort(all(longcodeT[u]));
                  bucket[longcodeT[u]] = 0;
            }

            int id = 0;
            for (auto &p : bucket)
                  p.second = id++;
            for (int u : S.layer[h])
            {
                  codeS[u] = bucket[longcodeS[u]];
                  longcodeS[S.prev[u]].push_back(codeS[u]);
            }
            for (int u : T.layer[h])
            {
                  codeT[u] = bucket[longcodeT[u]];
                  longcodeT[T.prev[u]].push_back(codeT[u]);
            }
      }

      return codeS[s] == codeT[t];
}
```

```
bool isomorphic(tree S, tree T)
{
	auto x = S.centers(), y = T.centers();
	if (x.size() != y.size())
		return false;
	if (isomorphic(S, x[0], T, y[0]))
		return true;
	return x.size() > 1 && isomorphic(S, x[1], T, y[0]);
}
```

4.19. **Erdos-Gallai Theorem.**

```
/*
	Grahpic sequence recognition

	Tested: UVA 11414, 10720
*/

bool is_graphic(vector<int> d)
{
	int n = d.size();
	sort(d.rbegin(), d.rend());
	vector<int> s(n + 1);
	for (int i = 0; i < n; ++i)
		s[i + 1] = s[i] + d[i];
	if (s[n] % 2)
		return false;
	for (int k = 1; k <= n; ++k)
	{
		int p = lower_bound(d.begin() + k, d.end(), k, greater<int>())
				- d.begin();
		if (s[k] > k * (p - 1) + s[n] - s[p])
			return false;
	}
	return true;
}
```

4.20. **Gabow SCC.**

```
/*
	Gabow's strongly connected component

	Complexity: O(n + m)
*/

struct graph
{
	int n;
	vector<vector<int>> adj;

	graph(int n) : n(n), adj(n) {}

	void add_edge(int src, int dst)
	{
		adj[src].push_back(dst);
	}

	vector<vector<int>> strongly_connected_components()
	{
		vector<vector<int>> scc;
		vector<int> S, B, I(n);
		function<void(int)> dfs = [&](int u)
		{
			B.push_back(I[u] = S.size());
			S.push_back(u);
			for (int v : adj[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();
				for (; I[u] < S.size(); S.pop_back())
				{
					scc.back().push_back(S.back());
					I[S.back()] = n + scc.size();
				}
			}
		};
		for (int u = 0; u < n; ++u)
			if (!I[u]) dfs(u);
```

```
        return scc; // I[u] - n is the index of u
}
```

```
};
```

5. Math

5.1. Minimum assignment (Jonker-Volgenant).

```
/*
      Minimum assignment (simplified Jonker-Volgenant)

      Description:
      We are given a cost table of size n times m with n <= m.
      It finds a minimum cost assignment, i.e.,
            min sum_{ij} c(i,j) x(i,j)
            where sum_{i in [n]} x(i,j) = 1,
            sum_{j in [m]} x(i,j) <= 1.

      Tested: SPOJ SCITIES
      Complexity: O(n^3)

      Note:
      - It finds minimum cost maximal matching.
      - To find the minimum cost non-maximal matching,
            we add n dummy vertices to the right side.
*/

template<typename T>
T min_assignment(const vector<vector<T>> &c)
{
      const int n = c.size(), m = c[0].size(); // assert(n <= m);
      vector<T> v(m), dist(m); // v: potential
      vector<int> matchL(n, -1), matchR(m, -1); // matching pairs
      vector<int> index(m), prev(m);
      iota(index.begin(), index.end(), 0);

      auto residue = [&](int i, int j)
      {
            return c[i][j] - v[j];
      };

      for (int f = 0; f < n; ++f)
      {
            for (int j = 0; j < m; ++j)
            {
                  dist[j] = residue(f, j);
                  prev[j] = f;
            }
            T w;
            int j, l;
            for (int s = 0, t = 0;;)
            {
                  if (s == t)
                  {
                        l = s;
                        w = dist[index[t++]];
                        for (int k = t; k < m; ++k)
                        {
                              j = index[k];
                              T h = dist[j];
                              if (h <= w)
                              {
                                    if (h < w)
                                    {
                                          t = s;
                                          w = h;
                                    }
                                    index[k] = index[t];
                                    index[t++] = j;
                              }
                        }
                        for (int k = s; k < t; ++k)
                        {
                              j = index[k];
                              if (matchR[j] < 0)
                                    goto aug;
                        }
                  }
                  int q = index[s++], i = matchR[q];
                  for (int k = t; k < m; ++k)
                  {
                        j = index[k];
                        T h = residue(i, j) - residue(i, q) + w;
                        if (h < dist[j])
                        {
                              dist[j] = h;
                              prev[j] = i;
                              if (h == w)
                              {
                                    if (matchR[j] < 0)
                                          goto aug;
                                    index[k] = index[t];
                                    index[t++] = j;
                              }
```

```
                    }
                }
            }
            aug: for (int k = 0; k < l; ++k)
                v[index[k]] += dist[index[k]] - w;
            int i;
            do
            {
                matchR[j] = i = prev[j];
                    swap(j, matchL[i]);
                } while (i != f);
            }
            T opt = 0;
            for (int i = 0; i < n; ++i)
                opt += c[i][matchL[i]]; // (i, matchL[i]) is a solution
            return opt;
        }
```

5.2. Fast Fourier Transform.

```
/*
    Tested: SPOJ VFMUL
    Complexity: O(n log n)
*/

typedef complex<double> point;

void fft(vector<point> &a, int sign = 1)
{
    int n = a.size(); // n should be a power of two
    double theta = 8 * sign * atan(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)
    {
        int irev = 0;
        for (int i = 0; i < n; i += m)
        {
            point w = polar(1.0, 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;
            }
        }
    }
    if (sign == -1)
        for (auto &p : a)
            p /= n;
}

// for parse integers as polynomials

typedef long long ll;

const int WIDTH = 5;
const ll RADIX = 100000; // = 10^WIDTH

vector<point> parse(const char s[])
{
    int n = strlen(s);
    int m = (n + WIDTH - 1) / WIDTH;
    vector<point> v(m);
    for (int i = 0; i < m; ++i)
    {
        int b = n - WIDTH * i, x = 0;
        for (int a = max(0, b - WIDTH); a < b; ++a)
            x = x * 10 + s[a] - '0';
        v[i] = x;
    }
    return v;
}

void print(const vector<point> &v)
{
    int i, N = v.size();
    vector<ll> digits(N + 1, 0);
    long double err = 0;

    for (i = 0; i < N; i++)
        digits[i] = (ll) (v[i].real() + 0.5);
    ll c = 0;
```

```
	for (i = 0; i < N; ++i)
	{
		c += digits[i];
		digits[i] = c % RADIX;
		c /= RADIX;
	}
	for (i = N - 1; i > 0 && digits[i] == 0; --i);
	printf("%lld", digits[i]);
	for (--i; i >= 0; --i)
		printf("%.*lld", WIDTH, digits[i]);
	printf("\n");
}
```

5.3. Fast Modulo Transform.

```
/*
	Fast Modulo Transform and
	Fast Convolution in any Modulo

	Note:
	- We assume n is a power of 2 and n < 2^23 (>= 8*10^6)

	Tested: SPOJ VFMUL
	Complexity: O(n log n)
*/

typedef long long ll;

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;
}

ll pow(ll a, ll b, ll M)
{
	ll x = 1;
	for (; b > 0; b >>= 1)
	{
		if (b & 1)
			x = (a * x) % M;
		a = (a * a) % M;
	}
	return x;
}

// fast modulo transform
// (1) n = 2^k < 2^23
// (2) only predetermined mod can be used
void fmt(vector<ll> &x, ll mod, int sign = +1)
{
	int n = x.size();
	ll h = pow(3, (mod - 1) / n, mod);
	if (sign < 0) h = inv(h, mod);
	for (int i = 0, j = 1; j < n - 1; ++j)
	{
		for (int k = n >> 1; k > (i ^= k); k >>= 1);
		if (j < i) swap(x[i], x[j]);
	}
	for (int m = 1; m < n; m *= 2)
	{
		ll w = 1, wk = pow(h, n / (2 * m), mod);
		for (int i = 0; i < m; ++i)
		{
			for (int j = i; j < n; j += 2 * m)
			{
				ll u = x[j], d = x[j + m] * w % mod;
				if ((x[j] = u + d) >= mod)
					x[j] -= mod;
				if ((x[j + m] = u - d) < 0)
					x[j + m] += mod;
			}
			w = w * wk % mod;
		}
	}
	if (sign < 0)
	{
		ll n_inv = inv(n, mod);
		for (auto &a : x)
			a = (a * n_inv) % mod;
	}
}
```

```
// convolution via fast modulo transform
vector<ll> conv(vector<ll> x, vector<ll> y, ll mod)
{
      fmt(x, mod, +1);
      fmt(y, mod, +1);
      for (int i = 0; i < x.size(); ++i)
            x[i] = (x[i] * y[i]) % mod;
      fmt(x, mod, -1);
      return x;
}

// general convolution by using fmts with chinese remainder thm.
vector<ll> convolution(vector<ll> x, vector<ll> y, ll mod)
{
      for (auto &a : x) a %= mod;
      for (auto &b : y) b %= mod;
      int n = x.size() + y.size() - 1, size = n - 1;
      for (int s : { 1, 2, 4, 8, 16 })
            size |= (size >> s);
      size += 1;
      x.resize(size);
      y.resize(size);
      ll A = 167772161, B = 469762049, C = 1224736769, D = (A * B % mod);
      vector<ll> z(n), a = conv(x, y, A), b = conv(x, y, B), c = conv(x, y, C);
      for (int i = 0; i < n; ++i)
      {
            z[i] = A * (104391568 * (b[i] - a[i]) % B);
            z[i] += D * (721017874 * (c[i] - (a[i] + z[i]) % C) % C);
            if ((z[i] = (z[i] + a[i]) % mod) < 0)
                  z[i] += mod;
      }
      return z;
}

const int WIDTH = 5;
const ll RADIX = 100000; // = 10^WIDTH

vector<ll> parse(const char s[])
{
      int n = strlen(s);
      int m = (n + WIDTH - 1) / WIDTH;
      vector<ll> v(m);
      for (int i = 0; i < m; ++i)
      {
            int b = n - WIDTH * i, x = 0;
            for (int a = max(0, b - WIDTH); a < b; ++a)
                  x = x * 10 + s[a] - '0';
            v[i] = x;
      }
      v.push_back(0);
      return v;
}

void print(const vector<ll> &v)
{
      int i, N = v.size();
      vector<ll> digits(N + 1, 0);
      for (i = 0; i < N; ++i)
            digits[i] = v[i];
      ll c = 0;
      for (i = 0; i < N; ++i)
      {
            c += digits[i];
            digits[i] = c % RADIX;
            c /= RADIX;
      }
      for (i = N - 1; i > 0 && digits[i] == 0; --i);
      printf("%lld", digits[i]);
      for (--i; i >= 0; --i)
            printf("%.*lld", WIDTH, digits[i]);
      printf("\n");
}
```

5.4. Gauss Jordan.

```
/*
      Tested: SPOJ GS
      Complexity: O(n^3)
*/

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;
}
```

5.5. Goldsection Search.

```
/*
    Minimum of unimodal function (goldsection search)

    Tested: COJ 2890 :(
*/

template<class F>
double find_min(F f, double a, double d, double eps = 1e-9)
{
    const int iter = 150;
    const double r = 2 / (3 + sqrt(5.));
    double b = a + r * (d - a), c = d - r * (d - a), fb = f(b), fc = f(c);
    for (int it = 0; it < iter && d - a > eps; ++it)
    {
        // '<': maximum, '>': minimum
        if (fb > fc)
        {
            a = b;
            b = c;
            c = d - r * (d - a);
            fb = fc;
            fc = f(c);
        }
        else
        {
            d = c;
            c = b;
            b = a + r * (d - a);
            fc = fb;
            fb = f(b);
        }
    }
    return c;
}
```

5.6. Linear Recursion.

```
/*
      Linear Recurrence Solver

      Description: Consider
      x[i+n] = a[0] x[i] + a[1] x[i+1] + ... + a[n-1] x[i+n-1]
      with initial solution x[0], x[1], ..., x[n-1]
      We compute k-th term of x in O(n^2 log k) time.

      Tested: SPOJ REC
      Complexity: O(n^2 log k) time, O(n log k) space
*/

typedef long long ll;

ll linear_recurrence(vector<ll> a, vector<ll> x, ll k)
{
      int n = a.size();
      vector<ll> t(2 * n + 1);
      function<vector<ll>(ll)> rec = [&](ll k)
      {
            vector<ll> c(n);
            if (k < n) c[k] = 1;
            else
            {
                  vector<ll> b = rec(k / 2);
                  fill(t.begin(), t.end(), 0);
                  for (int i = 0; i < n; ++i)
                        for (int j = 0; j < n; ++j)
                              t[i+j+(k&1)] += b[i]*b[j];
                  for (int i = 2*n-1; i >= n; --i)
                        for (int j = 0; j < n; ++j)
                              t[i-n+j] += a[j]*t[i];
                  for (int i = 0; i < n; ++i)
                        c[i] = t[i];
            }
            return c;
      };
      vector<ll> c = rec(k);
      ll ans = 0;
      for (int i = 0; i < x.size(); ++i)
            ans += c[i] * x[i];
      return ans;
}
```

5.7. Matrix Computation Algorithms.

```
/*
      Matrix Computation Algorithms (double)
*/

typedef vector<double> vec;
typedef vector<vec> mat;

int sign(double x)
{
      return x < 0 ? -1 : 1;
}

mat eye(int n)
{
      mat I(n, vec(n));
      for (int i = 0; i < n; ++i)
            I[i][i] = 1;
      return I;
}

mat add(mat A, const mat &B)
{
      for (int i = 0; i < A.size(); ++i)
            for (int j = 0; j < A[0].size(); ++j)
                  A[i][j] += B[i][j];
      return A;
}

mat mul(mat A, const mat &B)
{
      for (int i = 0; i < A.size(); ++i)
      {
            vec x(A[0].size());
            for (int k = 0; k < B.size(); ++k)
                  for (int j = 0; j < B[0].size(); ++j)
                        x[j] += A[i][k] * B[k][j];
            A[i].swap(x);
      }
      return A;
}

mat pow(mat A, int k)
{
      mat X = eye(A.size());
      for (; k > 0; k /= 2)
      {
            if (k & 1)
                  X = mul(X, A);
            A = mul(A, A);
      }
      return X;
}

double diff(vec a, vec b)
{
      double S = 0;
      for (int i = 0; i < a.size(); ++i)
            S += (a[i] - b[i]) * (a[i] - b[i]);
      return sqrt(S);
}

double diff(mat A, mat B)
{
      double S = 0;
      for (int i = 0; i < A.size(); ++i)
            for (int j = 0; j < A[0].size(); ++j)
                  S += (A[i][j] - B[i][j]) * (A[i][j] - B[i][j]);
      return sqrt(S);
}

vec mul(mat A, vec b)
{
      vec x(A.size());
      for (int i = 0; i < A.size(); ++i)
            for (int j = 0; j < A[0].size(); ++j)
                  x[i] += A[i][j] * b[j];
      return x;
}

mat transpose(mat A)
{
      for (int i = 0; i < A.size(); ++i)
            for (int j = 0; j < i; ++j)
                  swap(A[i][j], A[j][i]);
      return A;
}

double det(mat A)
{
      double D = 1;
      for (int i = 0; i < A.size(); ++i)
      {
            int p = i;
            for (int j = i + 1; j < A.size(); ++j)
                  if (fabs(A[p][i]) < fabs(A[j][i]))
                        p = j;
            swap(A[p], A[i]);
            for (int j = i + 1; j < A.size(); ++j)
                  for (int k = i + 1; k < A.size(); ++k)
                        A[j][k] -= A[i][k] * A[j][i] / A[i][i];
            D *= A[i][i];
            if (p != i)
                  D = -D;
      }
      return D;
}

// assume: A is non-singular
vec solve(mat A, vec b)
{
      for (int i = 0; i < A.size(); ++i)
      {
            int p = i;
            for (int j = i + 1; j < A.size(); ++j)
                  if (fabs(A[p][i]) < fabs(A[j][i]))
                        p = j;
            swap(A[p], A[i]);
            swap(b[p], b[i]);
            for (int j = i + 1; j < A.size(); ++j)
            {
                  for (int k = i + 1; k < A.size(); ++k)
                        A[j][k] -= A[i][k] * A[j][i] / A[i][i];
                  b[j] -= b[i] * A[j][i] / A[i][i];
            }
      }
      for (int i = A.size() - 1; i >= 0; --i)
      {
            for (int j = i + 1; j < A.size(); ++j)
                  b[i] -= A[i][j] * b[j];
            b[i] /= A[i][i];
```

```
    }
    return b;
}

// TODO: verify
mat solve(mat A, mat B)
{
    // A^{-1} B
    for (int i = 0; i < A.size(); ++i)
    {
        // forward elimination
        int p = i;
        for (int j = i + 1; j < A.size(); ++j)
            if (fabs(A[p][i]) < fabs(A[j][i]))
                p = j;
        swap(A[p], A[i]);
        swap(B[p], B[i]);
        for (int j = i + 1; j < A.size(); ++j)
        {
            double coef = A[j][i] / A[i][i];
            for (int k = i; k < A.size(); ++k)
                A[j][k] -= A[i][k] * coef;
            for (int k = 0; k < B[0].size(); ++k)
                B[j][k] -= B[i][k] * coef;
        }
    }
    for (int i = A.size() - 1; i >= 0; --i)
    {
        // backward substitution
        for (int j = i + 1; j < A.size(); ++j)
            for (int k = 0; k < 0; ++k)
                B[i][k] -= A[i][j] * B[j][k];
        for (int k = 0; k < B[0].size(); ++k)
            B[i][k] /= A[i][i];
    }
    return B;
}

// LU factorization
struct lu_data
{
    mat A;
    vector<int> pi;
};

lu_data lu(mat A)
{
    vector<int> pi;
    for (int i = 0; i < A.size(); ++i)
    {
        int p = i;
        for (int j = i + 1; j < A.size(); ++j)
            if (fabs(A[p][i]) < fabs(A[j][i]))
                p = j;
        pi.push_back(p);
        swap(A[p], A[i]);
        for (int j = i + 1; j < A.size(); ++j)
        {
            for (int k = i + 1; k < A.size(); ++k)
                A[j][k] -= A[i][k] * A[j][i] / A[i][i];
            A[j][i] /= A[i][i];
        }
    }
    return {A, pi};
}

vec solve(lu_data LU, vec b)
{
    mat &A = LU.A;
    vector<int> &pi = LU.pi;
    for (int i = 0; i < pi.size(); ++i)
        swap(b[i], b[pi[i]]);
    for (int i = 0; i < A.size(); ++i)
        for (int j = 0; j < i; ++j)
            b[i] -= A[i][j] * b[j];
    for (int i = A.size() - 1; i >= 0; --i)
    {
        for (int j = i + 1; j < A.size(); ++j)
            b[i] -= A[i][j] * b[j];
        b[i] /= A[i][i];
    }
    return b;
}
```

5.8. Roots Newton.

```
template<class F, class G>
```

```
double find_root(F f, G df, double x)
{
      for (int iter = 0; iter < 100; ++iter)
      {
            double fx = f(x), dfx = df(x);
            x -= fx / dfx;
            if (fabs(fx) < 1e-12)
                  break;
      }
      return x;
}
```

5.9. **2-SAT.**

```
/*
      Tested: CODEFORCES
      Complexity: O(n + m)
*/

struct sat2
{
      int n;
      vector<vector<int>> adj, radj;

      sat2(int n) : n(n), adj(2*n), radj(2*n) {}

      void add_edge(int u, int v)
      {
            adj[u].push_back(v);
            radj[v].push_back(u);
      }

      int neg(int u)
      {
            return 2*n-u-1;
      }

      void implication(int u, int v)
      {
            add_edge(u, v);
            add_edge(neg(v), neg(u));
      }

      vector<bool> value;

      bool solve()
      {
            vector<int> seen(2 * n);
            value.assign(2 * n, false);

            function<void(int, vector<vector<int>>&, vector<int>&)> visit =
                        [&](int u, vector<vector<int>> &adj, vector<int> &vec)
            {
                  seen[u] = true;
                  for (int v : adj[u])
                        if (!seen[v])
                              visit(v, adj, vec);
                  vec.push_back(u);
            };

            vector<int> order;

            for (int u = 0; u < n; ++u)
            {
                  if (!seen[u]) visit(u, adj, order);
                  if (!seen[neg(u)]) visit(neg(u), adj, order);
            }

            seen.assign(2 * n, false);
            reverse(order.begin(), order.end());

            vector<int> comp, id(2 * n);
            int cc = 0;

            for (int u : order)
                  if (!seen[u])
                  {
                        comp.clear(); ++cc;
                        visit(u, radj, comp);
                        for (int v : comp)
                              id[v] = cc;
                  }

            for (int u = 0; u < n; ++u)
                  if (id[u] == id[neg(u)]) return false;
                  else value[u] = id[u] > id[neg(u)];

            return true;
      }
};
```

5.10. Parametric Self-Dual Simplex.

```
/*
      Description:
            Solve a canonical LP:
                  min. c x
                  s.t. A x <= b
                      x >= 0

      Tested: http://codeforces.com/contest/375/problem/E
      Complexity: O(n+m) iterations on average
*/

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;
}
```

5.11. Simpson.

```
template<class F>
double simpson(F f, double a, double b, int n = 2000)
{
      double h = (b - a) / (2 * n), fa = f(a), nfa, res = 0;
      for (int i = 0; i < n; ++i, fa = nfa)
      {
            nfa = f(a + 2 * h);
            res += (fa + 4 * f(a + h) + nfa);
            a += 2 * h;
      }
      res = res * h / 3;
      return res;
}
```

6. Miscelaneus

6.1. Cube.

```
template<class T>
struct cube
{
      T F, U, D, L, R, B;

      void rotX()
      {
            swap(D, B);
            swap(B, U);
            swap(U, F);
      } // FUBD -> DFUB

      void rotY()
      {
            swap(D, R);
            swap(R, U);
            swap(U, L);
      } // LURD -> DLUR

      void rotZ()
      {
            swap(B, R);
            swap(R, F);
            swap(F, L);
      } // LFRB -> BLFR
};
```

6.2. Josephus.

```
/*
      Tested: ??????
*/

// n-cantidad de personas, m es la longitud del salto.
// comienza en la k-esima persona.
ll josephus(ll n, ll m, ll k)
{
      ll x = -1;
      for (ll i = n - k + 1; i <= n; ++i)
            x = (x + m) % i;
      return x;
}

ll josephus_inv(ll n, ll m, ll x)
{
      for (ll i = n;; i--)
      {
            if (x == i)
                  return n - i;
            x = (x - m % i + i) % i;
      }
      return -1;
}
```

6.3. Partition $O(n\sqrt{n})$.

```
typedef long long ll;

ll partition(ll n)
{
      vector<ll> dp(n + 1);
      dp[0] = 1;
      for (int i = 1; i <= n; i++)
            for (int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0; j++, r *= -1)
            {
                  dp[i] += dp[i - (3 * j * j - j) / 2] * r;
                  if (i - (3 * j * j + j) / 2 >= 0)
                        dp[i] += dp[i - (3 * j * j + j) / 2] * r;
            }
      return dp[n];
}
```

7. Number Theory

7.1. Modular Arithmetics.

```
/*
      Modular arithmetics (long long)

      Note:
            int < 2^31 < 10^9
            long long < 2^63 < 10^18
      feasible for M < 2^62 (10^18 < 2^62 < 10^19)

      Tested: SPOJ
*/

typedef long long ll;
typedef vector<ll> vec;
typedef vector<vec> mat;

ll add(ll a, ll b, ll M)
{
      a += b;
      if (a >= M) a -= M;
      return a;
}

ll sub(ll a, ll b, ll M)
{
      if (a < b) a += M;
      return a - b;
}

ll mul(ll a, ll b, ll M)
{
      ll q = (long double) a * (long double) b / (long double) M;
      ll r = a * b - q * M;
      return (r + 5 * M) % M;
}

ll pow(ll a, ll b, ll M)
{
      ll x = 1;
      for (; b > 0; b >>= 1)
      {
            if (b & 1) x = mul(x, a, M);
            a = mul(a, a, M);
      }
      return x;
}

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 (M)
ll div(ll a, 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);
      }
      if (a % t) return -1; // infeasible
      return mul(x < 0 ? x + M : x, a / t, M);
}

// Modular Matrix
mat eye(int n)
{
      mat I(n, vec(n));
      for (int i = 0; i < n; ++i)
            I[i][i] = 1;
      return I;
}

mat zeros(int n)
{
      return mat(n, vec(n));
}
```

```
mat mul(mat A, mat B, ll M)
{
      int l = A.size(), m = B.size(), n = B[0].size();
      mat C(l, vec(n));
      for (int i = 0; i < l; ++l)
            for (int k = 0; k < m; ++k)
                  for (int j = 0; j < n; ++j)
                        C[i][j] = add(C[i][j], mul(A[i][k], B[k][j], M), M);
      return C;
}

mat pow(mat A, ll b, ll M)
{
      mat X = eye(A.size());
      for (; b > 0; b >>= 1)
      {
            if (b & 1) X = mul(X, A, M);
            A = mul(A, A, M);
      }
      return X;
}

// assume: M is prime (singular ==>
// verify: SPOJ9832
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] = mul(A[i][k], inv, M);
            for (int k = 0; k < n; ++k)
                  B[i][k] = mul(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] = sub(A[j][k], mul(cor, A[i][k], M), M);
                  for (int k = 0; k < n; ++k)
                        B[j][k] = sub(B[j][k], mul(cor, B[i][k], M), M);
            }
      }

      return B;
}
```

7.2. Sieve.

```
/*
      Tested: SPOJ PRIME1, ETFS
      Complexity: O(n log log n)
*/

typedef long long ll;

// primes in [lo, hi)
vector<ll> primes(ll lo, ll hi)
{
      const ll M = 1 << 14, SQR = 1 << 16;
      vector<bool> composite(M), small_composite(SQR);
      vector<pair<ll, ll>> sieve;
      for (ll i = 3; i < SQR; i += 2)
            if (!small_composite[i])
            {
                  ll k = i * i + 2 * i * max(0.0, ceil((lo - i*i)/(2.0*i)));
                  sieve.push_back({ 2 * i, k });
                  for (ll j = i * i; j < SQR; j += 2 * i)
                        small_composite[j] = 1;
            }
      vector<ll> ps;
      if (lo <= 2)
      {
            ps.push_back(2);
            lo = 3;
      }
      for (ll k = lo | 1, low = lo; low < hi; low += M)
```

```
	{
		ll high = min(low + M, hi);
		fill(composite.begin(), composite.end(), 0);
		for (auto &z : sieve)
			for (; z.second < high; z.second += z.first)
				composite[z.second - low] = 1;
		for (; k < high; k += 2)
			if (!composite[k - low])
				ps.push_back(k);
	}
	return ps;
}

vector<ll> primes(ll hi)
{
	return primes(0, hi);
}
```

7.3. Divisor Sigma.

```
typedef long long ll;

ll divisor_sigma(ll n)
{
	ll sigma = 0, d = 1;
	for (; d * d < n; ++d)
		if (n % d == 0)
			sigma += d + n / d;
	if (d * d == n)
		sigma += d;
	return sigma;
}

// sigma(n) for all n in [lo, hi)
vector<ll> divisor_sigma(ll lo, ll hi)
{
	vector<ll> ps = primes(sqrt(hi) + 1);
	vector<ll> res(hi - lo), sigma(hi - lo, 1);
	iota(res.begin(), res.end(), lo);
	for (ll p : ps)
		for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p)
		{
			ll b = 1;
			while (res[k - lo] > 1 && res[k - lo] % p == 0)
			{
				res[k - lo] /= p;
				b = 1 + b * p;
			}
			sigma[k - lo] *= b;
		}
	for (ll k = lo; k < hi; ++k)
		if (res[k - lo] > 1)
			sigma[k - lo] *= (1 + res[k - lo]);
	return sigma; // sigma[k-lo] = sigma(k)
}
```

7.4. Euler Phi.

```
/*
	Euler Phi (Totient Function)

	Tested: SPOJ ETFS, AIZU NTL_1_D
*/

typedef long long ll;

ll euler_phi(ll n)
{
	if (n == 0)
		return 0;
	ll ans = n;
	for (ll x = 2; x * x <= n; ++x)
		if (n % x == 0)
		{
			ans -= ans / x;
			while (n % x == 0)
				n /= x;
		}
	if (n > 1)
		ans -= ans / n;
	return ans;
}
```

```
// phi(n) for all n in [lo, hi)
vector<ll> euler_phi(ll lo, ll hi)
{
      vector<ll> ps = primes(sqrt(hi) + 1);
      vector<ll> res(hi - lo), phi(hi - lo, 1);
      iota(res.begin(), res.end(), lo);
      for (ll p : ps)
            for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p)
            {
                  if (res[k - lo] < p)
                        continue;
                  phi[k - lo] *= (p - 1);
                  res[k - lo] /= p;
                  while (res[k - lo] > 1 && res[k - lo] % p == 0)
                  {
                        phi[k - lo] *= p;
                        res[k - lo] /= p;
                  }
            }
      for (ll k = lo; k < hi; ++k)
            if (res[k - lo] > 1)
                  phi[k - lo] *= (res[k - lo] - 1);
      return phi; // phi[k-lo] = phi(k)
}
```

7.5. Mobius Mu.

```
typedef long long ll;

ll mobius_mu(ll n)
{
      if (n == 0)
            return 0;
      ll mu = 1;
      for (ll x = 2; x * x <= n; ++x)
            if (n % x == 0)
            {
                  mu = -mu;
                  n /= x;
                  if (n % x == 0)
                        return 0;
            }
      return n > 1 ? -mu : mu;
}

// phi(n) for all n in [lo, hi)
vector<ll> mobius_mu(ll lo, ll hi)
{
      vector<ll> ps = primes(sqrt(hi) + 1);
      vector<ll> res(hi - lo), mu(hi - lo, 1);
      iota(res.begin(), res.end(), lo);
      for (ll p : ps)
            for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p)
            {
                  mu[k - lo] = -mu[k - lo];
                  if (res[k - lo] % p == 0)
                  {
                        res[k - lo] /= p;
                        if (res[k - lo] % p == 0)
                        {
                              mu[k - lo] = 0;
                              res[k - lo] = 1;
                        }
                  }
            }
      for (ll k = lo; k < hi; ++k)
            if (res[k - lo] > 1)
                  mu[k - lo] = -mu[k - lo];
      return mu; // mu[k-lo] = mu(k)
}
```

7.6. Extended Gcd.

```
/*
      Solve ax+by=(a,b)

      Tested: Benelux 2014 I, AIZU NTL_1_E
*/

ll gcd(ll a, ll b, ll &x, ll &y)
{
	if(b == 0)
		return x = 1, y = 0, a;
	ll r = gcd(b, a % b, y, x);
	y -= a / b * x;
	return r;
}
```

7.7. Farey Sequence.

```
/*
	Tested: UVA 10408
*/

vector<pair<ll, ll>> farey_sequence(int n)
{
	ll a = 0, b = 1, c = 1, d = n;
	vector<pair<ll, ll>> s;
	s.push_back({ 0, 1 });
	while (c < n)
	{
		ll k = (n + b) / d;
		ll e = k * c - a;
		ll f = k * d - b;
		a = c; b = d;
		c = e, d = f;
		s.push_back({ a, b });
	}
	return s;
}
```

7.8. $C(n,m) \mod p$.

```
/*
	Returns C(n, m) (mod p)

	Note: p can be any number

	Tested: XV OpenCup GP of Tatarstan,
	http://codeforces.com/gym/100633/problem/J
*/

ll c1(ll n, ll p, ll pk)
{
	if (n == 0)
		return 1;
	ll i, k, ans = 1;
	for (i = 2; i <= pk; i++)
		if (i % p)
			ans = ans * i % pk;
	ans = pow(ans, n / pk, pk);
	for (k = n % pk, i = 2; i <= k; i++)
		if (i % p)
			ans = ans * i % pk;
	return ans * c1(n / p, p, pk) % pk;
}

ll cal(ll n, ll m, ll p, ll pi, ll pk)
{
	ll i, k = 0, a, b, c, ans;
	a = c1(n, pi, pk), b = c1(m, pi, pk), c = c1(n - m, pi, pk);
	for (i = n; i; i /= pi)
		k += i / pi;
	for (i = m; i; i /= pi)
		k -= i / pi;
	for (i = n - m; i; i /= pi)
		k -= i / pi;
	ans = a * inv(b, pk) % pk * inv(c, pk) % pk * pow(p, k, pk) % pk;
	return ans * (p / pk) % p * inv(p / pk, pk) % p;
}

ll comb(ll n, ll m, ll p)
{
	ll ans = 0, x, i, k;
	for (x = p, i = 2; x > 1; i++)
		if (x % i == 0)
		{
			for (k = 1; x % i == 0; x /= i)
				k *= i;
			ans = (ans + cal(n, m, p, i, k)) % p;
		}
	return ans;
}
```

7.9. Discrete Logarithm.

```
/*
	Solve a^x=b (mod M)
	Tested: LIGHTOJ 1325
*/

ll dlog(ll a, ll b, ll M)
{
	map<ll, ll> _hash;
	ll n = euler_phi(M), k = sqrt(n);
	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;
}
```

7.10. Discrete Roots.

```
/*
	Solve x^k=a (mod n)
*/

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;
}
```

7.11. Linear Congruences.

```
/*
	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
	Tested: LIGHTOJ 1319
*/

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);
		ll 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};
}
```

7.12. Miller-Rabin.

```
/*
	Tested: SPOJ PON, FACT0
	Note: be carful with overflow
*/

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;
}
```

7.13. ModFact.

```
/*
	Return a (mod p) where n!=a*p^k
	Complexity: O(p log n)
*/

ll mod_fact(ll n, ll p)
{
	ll res = 1;
	while (n > 0)
	{
		for (ll i = 1, m = n % p; i <= m; ++i)
			res = res * i % p;
		if ((n /= p) % 2 > 0)
			res = p - res;
	}
	return res;
}
```

7.14. Pollard-Rho.

```
/*
      Return a proper divisor of n

      Note: n shouldn't be prime
      Tested: SPOJ FACT1
*/

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;
}
```

7.15. Primitive Root.

```
/*
      Find a primitive root of m

      Note: Only 2, 4, p^n, 2p^n have primitive roots
      Tested: http://codeforces.com/contest/488/problem/E
*/

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;
}
```

8. String

8.1. KMP.

```
/*
	Tested: SPOJ NHAY
	Complexity: O(n + m)
*/

vector<int> prefix_function(const string &s)
{
	int n = s.length();
	vector<int> pi(n + 1);

	for (int i = 0, j = pi[0] = -1; i < n; pi[++i] = ++j)
		while (j >= 0 && s[i] != s[j]) j = pi[j];

	return pi;
}

vector<int> kmp(const string &s, const string &p)
{
	auto pi = prefix_function(p);
	int n = s.length(), m = p.length();

	vector<int> ans;

	for (int i = 0, j = 0; i < n; ++i)
	{
		while (j >= 0 && s[i] != p[j])
			j = pi[j];

		if (++j == m)
			ans.push_back(i - m + 1);
	}

	return ans;
}
```

8.2. Manacher.

```
/*
	Tested: SPOJ LPS
	Complexity: O(n)
*/

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;
}

bool is_pal(const vector<int> &rad, int b, int e)
{
	int n = rad.size() / 2;
	return b >= 0 && e < n && rad[b + e] >= e - b + 1;
}
```

8.3. Maximal Suffix.

```
/*
	Complexity: O(n)
*/

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;
}
```

8.4. Minimum Rotation.

```
/*
	Complexity: O(n)
*/

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);
}
```

8.5. Palindromic Tree.

```
/*
	Tested: SPOJ LPS, APIO14_A
	Complexity: O(n)
*/

template<typename T>
struct palindromic_tree
{
	struct node
	{
		int len;
		map<T, node*> next;
		node *suf;
	};

	vector<T> s;
	vector<node*> nodes;
	node *neg, *zero, *suf;

	node* new_node()
	{
		nodes.push_back(new node());
		return nodes.back();
	}

	palindromic_tree()
	{
		(neg = new_node())->len = -1;
		suf = zero = new_node();
		neg->suf = zero->suf = neg;
	}

	void add(T c)
	{
		int i = s.size();
		s.push_back(c);
		node *p = suf;
		for (; i - 1 - p->len < 0 || s[i - 1 - p->len] != c; p = p->suf);
		if (p->next.count(c))
		{
			suf = p->next[c];
			return;
		}
		suf = new_node();
		suf->len = p->len + 2;
		p->next[c] = suf;
		if (suf->len == 1)
			suf->suf = zero;
		else
		{
			p = p->suf;
			for (; i - 1 - p->len < 0 ||
				s[i - 1 - p->len] != c; p = p->suf);
			suf->suf = p->next[c];
		}
	}

	~palindromic_tree()
	{
		for (auto p : nodes)
			delete p;
	}
};
```

8.6. Suffix Array $O(n \log n)$.

```
/*
	Tested: SARRAY
	Complexity: O(n log n)
*/

// !!!! c++11
struct suffix_array
{
	int n;
	vector<int> sa, lcp, rank;

	suffix_array(const string &s) : n(s.length() + 1), sa(n), lcp(n), rank(n)
	{
		vector<int> top(max(256, n));
		for (int i = 0; i < n; ++i)
			top[rank[i] = s[i] & 0xff]++;
		partial_sum(top.begin(), top.end(), top.begin());
		for (int i = 0; i < n; ++i)
			sa[--top[rank[i]]] = i;
		vector<int> tmp(n);
		for (int len = 1, j; len < n; len <<= 1)
		{
			for (int i = 0; i < n; ++i)
			{
				j = sa[i] - len;
				if (j < 0)
					j += n;
				tmp[top[rank[j]]++] = j;
			}
			sa[tmp[top[0] = 0]] = j = 0;
			for (int i = 1, j = 0; i < n; ++i)
			{
				if (rank[tmp[i]] != rank[tmp[i - 1]]
				   || rank[tmp[i] + len] != rank[tmp[i - 1] + len])
					top[++j] = i;
				sa[tmp[i]] = j;
			}
			copy(sa.begin(), sa.end(), rank.begin());
			copy(tmp.begin(), tmp.end(), sa.begin());
			if (j >= n - 1)
				break;
		}
		int i, j, k;
		for (j = rank[lcp[i = k = 0] = 0]; i < n - 1; ++i, ++k)
		{
			while (k >= 0 && s[i] != s[sa[j - 1] + k])
				lcp[j] = k--, j = rank[sa[j] + 1];
		}
	}
};
```

8.7. Suffix Array $O(n \log^2 n)$.

```
/*
	Tested: SPOJ SARRAY
	Complexity: O(n (log n)^2)
*/

struct suffix_array
{
	int n;
	vector<int> sa, rank, lcp;

	suffix_array(const string &s) : n(s.length()), sa(n), rank(n), lcp(n)
	{
		vector<int> t(n);
		for (int i = 0; i < n; ++i)
			rank[sa[i] = i] = s[i];
		for (int h = 1; t[n - 1] != n - 1; h *= 2)
		{
			auto cmp = [&](int i, int j)
			{
				if (rank[i] != rank[j]) return rank[i] < rank[j];
				return i + h < n && j + h < n
					? rank[i + h] < rank[j + h] : i > j;
			};
			sort(sa.begin(), sa.end(), cmp);
			for (int i = 0; i + 1 < n; ++i)
				t[i + 1] = t[i] + cmp(sa[i], sa[i + 1]);
			for (int i = 0; i < n; ++i)
				rank[sa[i]] = t[i];
		}
		for (int i = 0, k = 0; i < n; i++, k ? --k : 0)
		{
			if (rank[i] == n - 1)
			{
				k = 0;
				continue;
			}
			int j = sa[rank[i] + 1];
			while (i + k < n && j + k < n && s[i + k] == s[j + k])
				k++;
			lcp[rank[i]] = k;
		}
	}
};
```

8.8. Suffix Automaton.

```
/*
	Tested: SPOJ LCS
	COmplexity: O(n)
*/

template<typename T>
struct suffix_automata
{
	struct state
	{
		int len;
		state *link;
		map<T, state*> next;
	};

	vector<state*> states;
	state *last;

	suffix_automata()
	{
		states.push_back(new state{ 0, nullptr });
		last = states.front();
	}

	void extend(T c)
	{
		state *nlast = new state{ last->len + 1 }, *p;
		states.push_back(nlast);

		for (p = last; p != nullptr && !p->next.count(c); p = p->link)
			p->next[c] = nlast;

		if (p == nullptr)
			nlast->link = states.front();
		else
		{
```

```
                state *q = p->next[c];
                if (p->len + 1 == q->len)
                    nlast->link = q;
                else
                {
                    state *clone = new state{ p->len + 1,
                        q->link, q->next };
                    states.push_back(clone);
                    for (; p != nullptr && p->next[c] == q; p = p->link)
                        p->next[c] = clone;
                    q->link = nlast->link = clone;
                }
            }

            last = nlast;
        }

        ~suffix_automata()
        {
            for (state *e : states)
                delete e;
        }
};
```

8.9. Z-function.

```
/*
    Tested: Everywhere
*/

// 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;
}

// suff[i] = length of the longest common suffix of s and s[1..i]
vector<int> suffixes(const string &s)
{
    int n = s.length();
    int g = n - 1, f;

    vector<int> suff(n);
    suff[g] = n;

    for (int i = n - 2; i >= 0; --i)
    {
        if (i > g && suff[i + n - 1 - f] != i - g)
            suff[i] = min(suff[i + n - 1 - f], i - g);
        else
        {
            for (g = min(g, f = i); g >= 0
                && s[g] == s[g + n - 1 - f]; --g);
            suff[i] = f - g;
        }
    }

    return suff;
}
```

9. Java Stuff

9.1. Template.

```
import java.io.*;
import java.math.*;
import java.util.*;

public class Main {
      InputReader in;
      PrintWriter out;

      public void solve() throws IOException {
      }

      public void run() {
            try {
                  in = new InputReader(System.in);
                  out = new PrintWriter(System.out);
                  solve();
                  out.close();
            } catch (IOException e) {
                  e.printStackTrace();
            }
      }

      class InputReader {
            BufferedReader br;
            StringTokenizer st;

            InputReader(File f) {
                  try {
                        br = new BufferedReader(new FileReader(f));
                  } catch (FileNotFoundException e) {
                        e.printStackTrace();
                  }
            }

            public InputReader(InputStream f) {
                  br = new BufferedReader(new InputStreamReader(f));
            }

            String next() {
                  while (st == null || !st.hasMoreTokens()) {
                        try {
                              st = new StringTokenizer(br.readLine());
                        } catch (IOException e) {
                              e.printStackTrace();
                        }
                  }
                  return st.nextToken();
            }

            int nextInt() {
                  return Integer.parseInt(next());
            }
      }

      public static void main(String[] arg) {
            new Main().run();
      }
}
```