Team Reference
Universidad de la Habana : UH++

ACM-ICPC Caribbean Finals 2016

Team Members: Marcelo José Fornet Fornés, Ariel Cruz Cruz, Eloy Pérez Torres
Developed by: José Carlos Gutiérrez Pérez

Contents

1. Data Structures

1.1. Heavy Light Decomposition.

```
/* Notes:
   Split a tree in several path in a way that there is
   at most log(n) path from any node u to the root.

   pos -> Position of node u in the list.
   ipos -> Reverse of pos (Node on position i)

   No need to initialize for several uses.
*/

typedef vector<vector<int>> graph;

struct heavy_light{
   int n, heavy[maxn], root[maxn], depth[maxn];
   int pos[maxn], ipos[maxn], parent[maxn];

   int dfs(int s, int f, graph &G){
      parent[s] = f, heavy[s] = -1;
      int size = 1, maxSubtree = 0;

      for (auto u : G[s]) if (u != f){
         depth[u] = depth[s] + 1;
         int subtree = dfs(u, s, G);

         if (subtree > maxSubtree)
            heavy[s] = u, maxSubtree = subtree;

         size += subtree;
      }

      return size;
   }

   void go(graph &G, int ROOT=0){
      n = (int)G.size();

      depth[ROOT] = 0;
      dfs(ROOT, -1, G);

      for (int i = 0, currentPos = 0; i < n; ++i)
         if (parent[i] == -1 || heavy[parent[i]] != i)
            for (int u = i; u != -1; u = heavy[u], currentPos++)
               root[u] = i, pos[u] = currentPos, ipos[currentPos] = u;
   }

   int lca(int u, int v, segment_tree &ST){
      int ans = oo;
      for (; root[u] != root[v]; v = parent[root[v]]) {
         if (depth[root[u]] > depth[root[v]]) swap(u, v);
         ans = min(ans, ST.operation(1, 0, n, pos[root[v]], pos[v] + 1));
      }
      if (depth[u] > depth[v]) swap(u, v);
      ans = min(ans, ST.operation(1, 0, n, pos[u], pos[v] + 1));

      // LCA at u
      return ans;
   }

   int go_up(int u, int k){
      // The kth node (0 indexed) in the path from (u to root)
      for (;pos[u] - pos[root[u]] < k; u = parent[root[u]])
         k -= pos[u] - pos[root[u]] + 1;
      return ipos[pos[u] - k];
   }
};
```

1.2. Order Statistic.

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

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

using namespace __gnu_pbds;

typedef tree<int, null_type, less<int>,
      rb_tree_tag, tree_order_statistics_node_update> ordered_set;
```

```
int main() {
   ordered_set X;
   for(int i = 1; i <= 16; i *= 2)
      X.insert(i);
   cout << *X.find_by_order(1) << endl; // 2
   cout << *X.find_by_order(2) << endl; // 4
   cout << *X.find_by_order(4) << endl; // 16
   cout << (X.end()==X.find_by_order(6)) << endl; // true

   cout<<X.order_of_key(-5)<<endl; // 0
   cout<<X.order_of_key(1)<<endl; // 0
   cout<<X.order_of_key(3)<<endl; // 2
   cout<<X.order_of_key(4)<<endl; // 2
   cout<<X.order_of_key(400)<<endl; // 5
}
```

1.3. **Persistent Array.**

```
struct node{
   node *l, *r;
   int value;
};

node* clone(node *u){
   node *ans = new node();
   ans->l = u->l, ans->r = u->r, ans->value = u->value;
   return ans;
}

node* build(int b, int e){
   node *ans = new node();
   if (b + 1 < e){
      int m = (b + e) >> 1;
      ans->l = build(b, m);
      ans->r = build(m, e);
   }
   return ans;
}

node* update(node *root, int b, int e, int x, int v){
   root = clone(root);

   if (b + 1 == e){
      root->value = v;
   }
   else{
      int m = (b + e) >> 1;
      if (x < m) root->l = update(root->l, b, m, x, v);
      else root->r = update(root->r, m, e, x, v);
   }

   return root;
}

int query(node *root, int b, int e, int x){
   if (b + 1 == e) return root->value;

   int m = (b + e) >> 1;
   if (x < m) return query(root->l, b, m, x);
   else return query(root->r, m, e, x);
}
```

1.4. **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)
      {
```

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

1.5. Treap.

```
/*
   Treap implementation.
   jcg solution of Robotic Sort
*/

const int oo = 0x3f3f3f3f;

struct node
{
   pair<int, int> val, mn;
   int prio, size, rev;
   node *l, *r;

   node(pair<int, int> val) : val(val), mn(val), prio(rand()),
                        size(1), rev(0), l(0), r(0) {}
};

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

pair<int, int> mn(node *u)
{
   return u ? u->mn : make_pair(oo, oo);
}

node *update(node *u)
{
   if (u)
   {
      // Change this
      u->mn = min({u->val, mn(u->l), mn(u->r)});
      u->size = 1 + size(u->l) + size(u->r);
   }

   return u;
}

void push(node *u)
{
   if (!u) return;

   // Change this

   if (u->rev)
   {
      swap(u->l, u->r);
      if (u->l) u->l->rev = !u->l->rev;
      if (u->r) u->r->rev = !u->r->rev;
      u->rev = 0;
   }
}

node *merge(node *u, node *v)
{
   push(u); push(v);

   if (!u || !v) return u ? u : v;

   if (u->prio > v->prio)
   {
      u->r = merge(u->r, v);
      return update(u);
   }
   else
   {
      v->l = merge(u, v->l);
      return update(v);
   }
}

pair<node*, node*> split(node *u, int k)
{
   push(u);

   if (!u) return make_pair(nullptr, nullptr);

   if (size(u->l) >= k)
   {
```

```
      auto p = split(u->l, k);
      u->l = p.second;
      return make_pair(p.first, update(u));
   }
   else
   {
      auto p = split(u->r, k - size(u->l) - 1);
      u->r = p.first;
      return make_pair(update(u), p.second);
   }
}

int find_min(node *u)
{
   push(u);

   if (u->mn == u->val)
      return size(u->l);
   if (u->mn == mn(u->l))
      return find_min(u->l);
   return 1 + size(u->l) + find_min(u->r);
}


void dfs(node *u){
   // Debug

   if (u){
      push(u);
      if (u->l) dfs(u->l);
      cout << u->val << endl;
      if (u->r) dfs(u->r);
   }
}

int main()
{
   node *root = nullptr;

   for (int i = 1; i <= N; ++i)
   {
      int P;
      cin >> P;

      node *u = new node(make_pair(P, i));

      root = merge(root, update(u));
   }

   for (int i = 1; i <= N; ++i)
   {
      int k = find_min(root);

      cout << i + k << " \n"[i == N];

      pair<node*, node*> a = split(root, k);
      pair<node*, node*> b = split(a.second, 1);

      if (a.first) a.first->rev = !a.first->rev;
      root = merge(a.first, b.second);

      delete b.first;
   }
}
```

1.6. 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 excluded 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.

```
/*
	Dynamic hull for max dot queries

	Complexity:
	- Add: O(log n)
	- Query: O(log^2 n) but very fast in practice

	Tested: http://codeforces.com/gym/100377/problem/L
*/

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

ll cross(point a, point b) { return imag(conj(a) * b); }

ll dot(point a, point b) { return real(conj(a) * b); }

ll area2(point a, point b, point c) { return cross(b - a, c - a); }

namespace std
{
	bool operator<(const point &a, const point &b)
	{
		return real(a) < real(b) || (real(a) == real(b) && imag(a) < imag(b));
	}
}

const ll oo = 0x3f3f3f3f3f3f3f3f;

struct dynamic_hull
{
	dynamic_hull() : hulls() {}

	void add_point(point p)
	{
		hull h;
		h.add_point(p);

		for (hull &_h : hulls)
			if (_h.empty())
			{
				h.swap(_h);
				break;
			}
			else h = merge(h, _h), _h.clear();

		if (!h.empty()) hulls.emplace_back(h);
	}

	ll max_dot(point p)
	{
		ll best = -oo;

		for (hull &h : hulls)
			if (!h.empty()) best = max(best, h.max_dot(p));

		return best;
	}

private:
	struct hull : vector<point>
	{
		void add_point(point p)
		{
			for (int s = size(); s > 1; --s)
				if (area2(at(s - 2), at(s - 1), p) < 0) break;
				else pop_back();
			push_back(p);
		}

		ll max_dot(point p)
		{
			int lo = 0, hi = (int) size() - 1, mid;

			while (lo < hi)
			{
				mid = (lo + hi) / 2;

				if (dot(at(mid), p) <= dot(at(mid + 1), p))
					lo = mid + 1;
				else hi = mid;
			}

			return dot(at(lo), p);
		}
	};
```

```
static hull merge(const hull &a, const hull &b)
{
      hull h;
      size_t i = 0, j = 0;

      while (i < a.size() && j < b.size())
            if (a[i] < b[j]) h.add_point(a[i++]);
            else h.add_point(b[j++]);
            while (i < a.size()) h.add_point(a[i++]);

            while (j < b.size()) h.add_point(b[j++]);

            return h;
      }

      vector<hull> hulls;
};
```

3. Geometry

3.1. 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;
}
```

3.2. 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. 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
int sign(double x) { return x < -eps ? -1 : x > eps; }

int dblcmp(double x, double y) { return sign(x - y); }

double dot(point a, point b) { return real(conj(a) * b); }

double cross(point a, point b) { return imag(conj(a) * b); }

double area2(point a, point b, point c) { return cross(b - a, c - a); }

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();
	}
}
```

3.4. 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;
}
```

3.5. Circle.

```
/*
	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;
}

/*
   Get the center of the circles that pass through p0 and p1
   and has ratio r.

   Be careful with epsilon.
*/
vector<point> two_point_ratio_circle(point p0, point p1, double r){
   if (abs(p1 - p0) > 2 * r + eps) // Points are too far.
      return {};

   point pm = (p1 + p0) / 2.0l;
   point pv = p1 - p0;
   pv = point(-pv.imag(), pv.real());

   double x1 = p1.real(), y1 = p1.imag();
   double xm = pm.real(), ym = pm.imag();
   double xv = pv.real(), yv = pv.imag();

   double A = (sqr(xv) + sqr(yv));
   double C = sqr(xm - x1) + sqr(ym - y1) - sqr(r);
   double D = sqrt( - 4 * A * C );
   double t = D / 2.0 / A;

   if (abs(t) <= eps)
      return {pm};

   return {c1, c2};
}

/*
	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 > 1) s = 1;
            if (t < 0.0) t = 0.0;
            else if (t > 1) t = 1;
            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 Points.

```
/*
      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. Points 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. 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));
}
```

3.10. Line Segment Intersections.

```
/*
	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);
}
```

3.11. Minkowski.

```
/*
   Minkowski sum of two convex polygons. O(n + m)

   Note: Polygons MUST be counterclockwise
*/

polygon minkowski(polygon &A, polygon &B){
      int na = (int)A.size(), nb = (int)B.size();

      if (A.empty() || B.empty()) return polygon();

      rotate(A.begin(), min_element(A.begin(), A.end()), A.end());
      rotate(B.begin(), min_element(B.begin(), B.end()), B.end());

      int pa = 0, pb = 0;

      polygon M;

      while (pa < na && pb < nb){
            M.push_back(A[pa] + B[pb]);
            double x = cross(A[(pa + 1) % na] - A[pa],
                              B[(pb + 1) % nb] - B[pb]);
            if (x <= eps) pb++;
            if (-eps <= x) pa++;
      }

      while (pa < na) M.push_back(A[pa++] + B[0]);
      while (pb < nb) M.push_back(B[pb++] + A[0]);

      return M;
}
```

3.12. 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};
}
```

3.13. Points 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)
}
```

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. 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;
}
```

3.16. 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;
}
```

4. Graph

4.1. Articulation Points.

```
/*
	Articulation points / Biconnected components

	Description:
	- Let G = (V, E). If G-v is disconnected, v in V is said to
	be an articulation point. If G has no articulation points,
	it is said to be biconnected.
	- A biconnected component is a maximal biconnected subgraph.
	The algorithm finds all articulation points and biconnected
	components.

	Complexity: O(n + m)

	Tested:
	- http://www.spoj.com/problems/SUBMERGE/
	- http://codeforces.com/problemset/problem/487/E
*/

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);
	}

	int add_node()
	{
		adj.push_back({});
		return n++;
	}

	vector<int>& operator[](int u) { return adj[u]; }
};

vector<vector<int>> biconnected_components(graph &adj)
{
	int n = adj.n;

	vector<int> num(n), low(n), art(n), stk;
	vector<vector<int>> comps;

	function<void(int, int, int&)> dfs = [&](int u, int p, int &t)
	{
		num[u] = low[u] = ++t;
		stk.push_back(u);

		for (int v : adj[u]) if (v != p)
		{
			if (!num[v])
			{
				dfs(v, u, t);
				low[u] = min(low[u], low[v]);

				if (low[v] >= num[u])
				{
					art[u] = (num[u] > 1 || num[v] > 2);

					comps.push_back({u});
					while (comps.back().back() != v)
						comps.back().push_back(stk.back()),
						stk.pop_back();
				}
			}
			else low[u] = min(low[u], num[v]);
		}
	};

	for (int u = 0, t; u < n; ++u)
		if (!num[u]) dfs(u, -1, t = 0);

	// build the block cut tree
	function<graph()> build_tree = [&]()
	{
		graph tree(0);
		vector<int> id(n);

		for (int u = 0; u < n; ++u)
			if (art[u]) id[u] = tree.add_node();

		for (auto &comp : comps)
```

```
        {
            int node = tree.add_node();
            for (int u : comp)
                if (!art[u]) id[u] = node;
                else tree.add_edge(node, id[u]);
        }
        return tree;
    };

    return comps;
}
```

4.2. Bipartite Matching.

```
/*
    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;
    }
};
```

4.3. Bridges.

```
/*
    Bridges / Bridges connected components

    Description:
    - Let G = (V, E). If G-(u,v) is disconnected, (u,v) in V is said to
    be a bridge.
    - Bridge-blocks or bridge-connected components are the components of
    G formed by deleting all the bridges. The bridge-blocks partition V
    in equivalences classes such that two vertices are in the same class
    if and only if there is a (not necessarily simple) cycle of G
    containing both of them.
    The algorithm finds all briges and bridge-blocks.

    Complexity: O(n + m)

    Tested: http://codeforces.com/gym/100114/problem/J
*/
```

```cpp
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);
	}

	vector<int>& operator[](int u) { return adj[u]; }
};

vector<vector<int>> bridge_blocks(graph &adj)
{
	int n = adj.n;

	vector<int> num(n), low(n), stk;
	vector<vector<int>> comps;
	vector<pair<int, int>> bridges;

	function<void(int, int, int&)> dfs = [&](int u, int p, int &t)
	{
		num[u] = low[u] = ++t;
		stk.push_back(u);

		// remove if there isn't parallel edges
		sort(adj[u].begin(), adj[u].end());

		for (int i = 0, sz = adj[u].size(); i < sz; ++i)
		{
			int v = adj[u][i];

			if (v == p)
			{
				if (i + 1 < sz && adj[u][i + 1] == v)
					low[u] = min(low[u], num[v]);
				continue;
			}
			if (!num[v])
			{
				dfs(v, u, t);
				low[u] = min(low[u], low[v]);

				if (low[v] == num[v])
					bridges.push_back({u, v});
			}
			else low[u] = min(low[u], num[v]);
		}

		if (num[u] == low[u])
		{
			comps.push_back({});
			for (int v = -1; v != u; stk.pop_back())
				comps.back().push_back(v = stk.back());
		}
	};

	for (int u = 0, t; u < n; ++u)
		if (!num[u]) dfs(u, -1, t = 0);

	// this is for build the bridge-block tree
	function<graph()> build_tree = [&]()
	{
		vector<int> id(n);

		for (int i = 0; i < (int) comps.size(); ++i)
			for (int u : comps[i]) id[u] = i;

		graph tree(comps.size());

		for (auto &e : bridges)
			tree.add_edge(id[e.first], id[e.second]);

		return tree;
	};

	return comps;
}
```

4.4. Centroid Decomposition.

```cpp
/*
	Centroid decomposition of a tree.
```

```
   Find the centroid of the subtree that contains node c.

   Nodes availables are those which aren't marked, i.e mk[u] == False
*/

vi adj[maxn];
bool mk[maxn];
int q[maxn], p[maxn], sz[maxn], mc[maxn];

int centroid(int c){
      int b = 0, e = 0;
      q[e++] = c, p[c] = -1, sz[c] = 1, mc[c] = 0;

      while (b < e){
            int u = q[b++];
            for (auto v : adj[u]) if (v != p[u] && !mk[v])
                  p[v] = u, sz[v] = 1, mc[v] = 0, q[e++] = v;
      }

      for (int i = e - 1; ~i; --i){
            int u = q[i];
            int bc = max(e - sz[u], mc[u]);
            if (2 * bc <= e) return u;
            sz[p[u]] += sz[u], mc[p[u]] = max(mc[p[u]], sz[u]);
      }

      assert(false);
      return -1;
}
```

4.5. 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;
	}
};
```

4.6. 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;
    }
};
```

4.7. 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;
        }
};
```

4.8. Gomory Hu Tree.

```
/*
    Gomory-Hu tree

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

template<typename flow_type>
struct edge
{
    int src, dst;
    flow_type cap;
};

template<typename flow_type>
vector<edge<flow_type>> gomory_hu(dinic<flow_type> &adj)
{
    int n = adj.n;

    vector<edge<flow_type>> tree;
    vector<int> parent(n);

    for (int u = 1; u < n; ++u)
    {
        tree.push_back({ u, parent[u], adj.max_flow(u, parent[u]) });
        for (int v = u + 1; v < n; ++v)
            if (adj.level[v] == -1 && parent[v] == parent[u])
                parent[v] = u;
    }

    return tree;
}
```

4.9. 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;
  }
};
```

4.10. Hungarian.

```
/*
    Maximum assignment (Kuhn-Munkres)

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

    Complexity: O(n^3)

    Tested: http://www.spoj.com/problems/SCITIES/
*/

template<typename T>
T max_assignment(const vector<vector<T>> &a)
{
    int n = a.size(), m = a[0].size();
    assert(n <= m);
    vector<int> x(n, -1), y(m, -1);
    vector<T> px(n, numeric_limits<T>::min()), py(m, 0);

    for (int u = 0; u < n; ++u)
        for (int v = 0; v < m; ++v) px[u] = max(px[u], a[u][v]);

    for (int u = 0, p, q; u < n;)
    {
        vector<int> s(n + 1, u), t(m, -1);

        for (p = q = 0; p <= q && x[u] < 0; ++p)
            for (int k = s[p], v = 0; v < m && x[u] < 0; ++v)
                if (px[k] + py[v] == a[k][v] && t[v] < 0)
                {
                    s[++q] = y[v], t[v] = k;
                    if (s[q] < 0)
                        for (p = v; p >= 0; v = p)
                            y[v] = k = t[v], p = x[k], x[k] = v;
                }
```

```
        if (x[u] < 0)
        {
            T delta = numeric_limits<T>::max();

            for (int i = 0; i <= q; ++i)
                for (int v = 0; v < m; ++v) if (t[v] < 0)
                    delta = min(delta, px[s[i]] + py[v] - a[s[i]][v]);

            for (int i = 0; i <= q; ++i)
                px[s[i]] -= delta;

            for (int v = 0; v < m; ++v)
                py[v] += (t[v] < 0 ? 0 : delta);
        }
        else ++u;
    }

    T cost = 0;

    for (int u = 0; u < n; ++u)
        cost += a[u][x[u]];

    return cost;
}
```

4.11. 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)])];
    }
};
```

4.12. Max Flow Dinic.

```
/*
	Maximum Flow (Dinitz)

	Complexity: O(n^2 m) but very fast in practice

	Tested: http://www.spoj.com/problems/FASTFLOW/
*/

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

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

	dinic(int n) : n(n), adj(n), level(n), q(n), it(n) {}

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

	vector<int> level, q, it;

	bool bfs(int source, int sink)
	{
		fill(level.begin(), level.end(), -1);
		for (int qf = level[q[0] = sink] = 0, qb = 1; qf < qb; ++qf)
		{
			sink = q[qf];
			for (edge &e : adj[sink])
			{
				edge &r = adj[e.dst][e.rev];
				if (r.flow < r.cap && level[e.dst] == -1)
					level[q[qb++] = e.dst] = 1 + level[sink];
			}
		}
		return level[source] != -1;
	}

	flow_type augment(int source, int sink, flow_type flow)
	{
		if (source == sink) return flow;
		for (; it[source] != adj[source].size(); ++it[source])
		{
			edge &e = adj[source][it[source]];
			if (e.flow < e.cap && level[e.dst] + 1 == level[source])
			{
				flow_type delta = augment(e.dst, sink,
						min(flow, e.cap - e.flow));
				if (delta > 0)
				{
					e.flow += delta;
					adj[e.dst][e.rev].flow -= delta;
					return delta;
				}
			}
		}
		return 0;
	}

	flow_type max_flow(int source, int sink)
	{
		for (int u = 0; u < n; ++u)
			for (edge &e : adj[u]) e.flow = 0;
		flow_type flow = 0;
		flow_type oo = numeric_limits<flow_type>::max();

		while (bfs(source, sink))
		{
			fill(it.begin(), it.end(), 0);
			for (flow_type f; (f = augment(source, sink, oo)) > 0;)
				flow += f;

		} // level[u] = -1 => source side of min cut
		return flow;
	}
};
```

4.13. Max Flow Push Relabel.

```
/*
	Maximum Flow (Goldberg-Tarjan)

	Complexity: O(n^3) faster than Dinic in most cases

	Tested: http://www.spoj.com/problems/FASTFLOW/
*/

template<typename flow_type>
struct goldberg_tarjan
{
	struct edge
	{
		size_t src, dst, rev;
		flow_type flow, cap;
	};

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

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

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

	flow_type max_flow(int source, int sink)
	{
		vector<flow_type> excess(n);
		vector<int> dist(n), active(n), count(2 * n);
		queue<int> q;
		auto enqueue = [&](int v)
		{
			if (!active[v] && excess[v] > 0)
			{
				active[v] = true;
				q.push(v);
			}
		};
		auto push = [&](edge &e)
		{
			flow_type f = min(excess[e.src], e.cap - e.flow);
			if (dist[e.src] <= dist[e.dst] || f == 0) return;
			e.flow += f;
			adj[e.dst][e.rev].flow -= f;
			excess[e.dst] += f;
			excess[e.src] -= f;
			enqueue(e.dst);
		};
		dist[source] = n;
		active[source] = active[sink] = true;
		count[0] = n - 1;
		count[n] = 1;
		for (int u = 0; u < n; ++u)
			for (edge &e : adj[u]) e.flow = 0;
		for (edge &e : adj[source])
		{
			excess[source] += e.cap;
			push(e);
		}
		for (int u; !q.empty(); q.pop())
		{
			active[u = q.front()] = false;
			for (auto &e : adj[u]) push(e);
			if (excess[u] > 0)
			{
				if (count[dist[u]] == 1)
				{
					int k = dist[u]; // Gap Heuristics
					for (int v = 0; v < n; v++)
					{
						if (dist[v] < k)
							continue;
						count[dist[v]]--;
						dist[v] = max(dist[v], n + 1);
						count[dist[v]]++;
						enqueue(v);
					}
				}
				else
				{
					count[dist[u]]--; // Relabel
					dist[u] = 2 * n;
					for (edge &e : adj[u])
						if (e.cap > e.flow)
							dist[u] = min(dist[u], dist[e.dst] + 1);
					count[dist[u]]++;
```

```
                    enqueue(u);
                }
            }
        }
        flow_type flow = 0;
        for (edge e : adj[source])
            flow += e.flow;
        return flow;
    }
};
```

4.14. Min Cost Max Flow.

```
/*
    Minimum Cost Flow (Tomizawa, Edmonds-Karp)

    Complexity: O(F m log n), where F is the amount of maximum flow

    Tested: Codeforces [http://codeforces.com/problemset/problem/717/G]
*/

template<typename flow_type, typename cost_type>
struct min_cost_max_flow
{
    struct edge
    {
        size_t src, dst, rev;
        flow_type flow, cap;
        cost_type cost;
    };

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

    min_cost_max_flow(int n) : n(n), adj(n), potential(n), dist(n), back(n) {}

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

    vector<cost_type> potential;

    inline cost_type rcost(const edge &e)
    {
        return e.cost + potential[e.src] - potential[e.dst];
    }

    void bellman_ford(int source)
    {
        for (int k = 0; k < n; ++k)
            for (int u = 0; u < n; ++u)
                for (edge &e : adj[u])
                    if (e.cap > 0 && rcost(e) < 0)
                        potential[e.dst] += rcost(e);
    }

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

    vector<cost_type> dist;
    vector<edge*> back;

    cost_type dijkstra(int source, int sink)
    {
        fill(dist.begin(), dist.end(), oo);

        typedef pair<cost_type, int> node;
        priority_queue<node, vector<node>, greater<node>> pq;

        for (pq.push({dist[source] = 0, source}); !pq.empty();)
        {
            node p = pq.top(); pq.pop();

            if (dist[p.second] < p.first) continue;
            if (p.second == sink) break;

            for (edge &e : adj[p.second])
                if (e.flow < e.cap &&
                    dist[e.dst] > dist[e.src] + rcost(e))
                {
                    back[e.dst] = &e;
                    pq.push({dist[e.dst] = dist[e.src] + rcost(e),
                                e.dst});
                }
        }
```

```
        return dist[sink];
    }

    pair<flow_type, cost_type> max_flow(int source, int sink)
    {
        flow_type flow = 0;
        cost_type cost = 0;

        for (int u = 0; u < n; ++u)
            for (edge &e : adj[u]) e.flow = 0;

        potential.assign(n, 0);
        dist.assign(n, 0);
        back.assign(n, nullptr);

        bellman_ford(source); // remove negative costs

        while (dijkstra(source, sink) < oo)
        {
            for (int u = 0; u < n; ++u)
                if (dist[u] < dist[sink])
                    potential[u] += dist[u] - dist[sink];

            flow_type f = numeric_limits<flow_type>::max();

            for (edge *e = back[sink]; e; e = back[e->src])
                f = min(f, e->cap - e->flow);
            for (edge *e = back[sink]; e; e = back[e->src])
                e->flow += f, adj[e->dst][e->rev].flow -= f;

            flow += f;
            cost += f * (potential[sink] - potential[source]);

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

4.15. Satisfiability Twosat.

```
/*
    Two-Sat

    Complexity: O(n)

    Tested: POI (Gates)
*/

struct satisfiability_twosat
{
    int n;
    vector<vector<int>> imp;

    satisfiability_twosat(int n) : n(n), imp(2 * n) {}

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

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

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

    vector<bool> solve()
    {
        int size = 2 * n;
        vector<int> S, B, I(size);

        function<void(int)> dfs = [&](int u)
        {
            B.push_back(I[u] = S.size());
            S.push_back(u);

            for (int v : imp[u])
                if (!I[v]) dfs(v);
                else while (I[v] < B.back()) B.pop_back();

            if (I[u] == B.back())
                for (B.pop_back(), ++size; I[u] < S.size(); S.pop_back())
                    I[S.back()] = size;
        };

        for (int u = 0; u < 2 * n; ++u)
```

```
                if (!I[u]) dfs(u);

            vector<bool> values(n);

            for (int u = 0; u < n; ++u)
                if (I[u] == I[neg(u)]) return {};
                else values[u] = I[u] < I[neg(u)];

            return values;
        }
};
```

4.16. Gabow SCC.

```
/*
    Gabow's strongly connected component

    Complexity: O(n + m)

    Tested: http://www.spoj.com/problems/CAPCITY/
*/

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);
    }

    vector<int>& operator[](int u) { return adj[u]; }
};

vector<vector<int>> scc_gabow(graph &adj)
{
    int n = adj.n;

    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({});
            for (B.pop_back(); 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; // in reverse topological order
}
```

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]);
}
```

5. Helpers

5.1. Vectors.

```
template<class T>
ostream &operator<<(ostream &os, const vector<T> &v)
{
	os << "[";
	for (int i = 0; i < v.size(); os << v[i++])
		if (i > 0) os << " ";
	os << "]";
	return os;
}
```

```
template<class T>
ostream &operator<<(ostream &os, const vector<vector<T>> &v)
{
	os << "[";
	for (int i = 0; i < v.size(); os << v[i++])
		if (i > 0) os << endl << " ";
	os << "]";
	return os;
}
```

6. Java

6.1. Template.

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

public class Main {
	InputReader in;
	PrintWriter out;

	public void solve() throws IOException {
		// Code here...
	}

	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();
	}
}
```

7. Math

7.1. Fast Fourier Transform.

```
/*
	Fast Fourier Transform

	Complexity: O(n log n)

	Tested: http://codeforces.com/gym/100285/problem/G
*/

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

point operator+(const point &a, const point &b) { return {a.x + b.x, a.y + b.y}; }
point operator-(const point &a, const point &b) { return {a.x - b.x, a.y - b.y}; }
point operator*(const point &a, const point &b) {
	return {a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x};
}
point operator/(const point &a, double d) { return {a.x / d, a.y / d}; }

void fft(vector<point> &a, int sign = +1)
{
	int n = a.size();

	for (int i = 1, j = 0; i < n - 1; ++i)
	{
		for (int k = n >> 1; (j ^= k) < k; k >>= 1);

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

	double theta = 2 * atan2(0, -1) * sign;

	for (int m, mh = 1; (m = mh << 1) <= n; mh = m)
	{
		point wm(cos(theta / m), sin(theta / m)), w(1, 0);

		for (int i = 0; i < n; i += m, w = point(1, 0))
			for (int j = i, k = j + mh; j < i + mh; ++j, ++k, w = w * wm)
			{
				point x = a[j], y = a[k] * w;
				a[j] = x + y;
				a[k] = x - y;
			}
	}

	if (sign == -1)
		for (point &p : a) p = p / n;
}
```

7.2. 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");
}
```

7.3. Gauss.

```
/*
        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;
}
```

7.4. 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;
}
```

7.5. 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;
}
```

7.6. 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;
}
```

7.7. 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;
}
```

7.8. Simplex.

```
/*
      Parametric Self-Dual Simplex method

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

      Complexity: O(n+m) iterations on average

      Tested: http://codeforces.com/contest/375/problem/E
*/

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;
}
```

7.9. 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;
}
```

8. Misc

8.1. 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
};
```

8.2. 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;
}
```

8.3. 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];
}
```

8.4. Useful.

```
// TIME
for (int a = 0; ;++a){
   if (clock()>=2.5*CLOCKS_PER_SEC) break;
   // It will stop when 2.5 seconds have passed
}
```

```
// LAMBDA
function<bool(int, int)> add_edge = [&](int u, int v)
{
   // code here...
   return true;
};
```

9. Number Theory

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

```
/*
	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;
}
```

9.2. 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;
}
```

9.3. 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;
}
```

9.4. 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)
}
```

9.5. 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)
}
```

9.6. 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;
}
```

9.7. 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};
}
```

9.8. 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;
}
```

9.9. 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)
}
```

9.10. 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;
}
```

9.11. Mod Fact.

```
/*
	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;
}
```

9.12. 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;
}
```

9.13. 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;
}
```

9.14. 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);
}
```

10. String

10.1. KMP.

```
/*
	Prefix function and Knuth-Morris-Pratt string matching

	Complexity: O(n + m)

	Tested: http://www.spoj.com/problems/NHAY/
*/

vector<int> prefix_function(const string &p)
{
	int n = p.length();

	vector<int> pref(n + 1);

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

	return pref;
}

vector<int> knuth_morris_pratt(const string &s, const string &p)
{
	int n = s.length(), m = p.length();

	vector<int> pref = prefix_function(p), matches;

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

		if (++j == m)
			matches.push_back(i - m + 1), j = pref[j];
	}

	return matches;
}
```

10.2. 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;
}
```

10.3. 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;
}
```

10.4. 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);
}
```

10.5. Palindromic Tree.

```
/*
	Palindromic Tree

	Complexity: O(n)

	Tested: ??
*/

template<size_t maxlen, size_t alpha>
struct PalindromicTree
{
	int go[maxlen + 2][alpha], slink[maxlen + 2], length[maxlen + 2];
	int s[maxlen], slength, size, last;

	int new_node()
	{
		memset(go[size], 0, sizeof go[size]);
		slink[size] = length[size] = 0;
		return size++;
	}

	PalindromicTree() { reset(); }

	void reset()
	{
		size = slength = 0;
		length[new_node()] = -1;
		last = new_node();
```

```
	}

	int get_link(int p)
	{
		for (int i = slength - 1;
			 i - 1 - length[p] < 0 || s[i - 1 - length[p]] != s[i];)
			p = slink[p];
		return p;
	}

	int _extend(int c)
	{
		s[slength++] = c;
		int p = get_link(last), np;
		if (go[p][c]) return go[p][c];
		length[np = new_node()] = 2 + length[p];
		go[p][c] = np;
		if (length[np] == 1) return slink[np] = 1, np;
		p = slink[p];
		slink[np] = go[get_link(p)][c];
		return np;
	}

	void extend(int c) { last = _extend(c); }
};
```

10.6. Suffix Array.

```
/*
	Suffix array + lcp

	Complexity: O(n log n)

	Tested:
	- http://www.spoj.com/problems/SARRAY/
	- http://acm.timus.ru/problem.aspx?space=1&num=1393
	- http://wcipeg.com/problem/coci092p6
	- http://www.spoj.com/problems/LCS/

	Note: lcp[i] = lcp(s[sa[i-1]...], s[sa[i]...])
*/

template<typename charT>
struct SuffixArray
{
	int n;
	vector<int> sa, rank, lcp;

	SuffixArray(const basic_string<charT> &s) :
		n(s.length() + 1), sa(n), rank(n), lcp(n)
	{
		vector<int> _sa(n), bucket(n);

		iota(sa.rbegin(), sa.rend(), 0);
		sort(next(sa.begin()), sa.end(),
			[&](int i, int j) { return s[i] < s[j]; });

		for (int i = 1, j = 0; i < n; ++i)
		{
			rank[sa[i]] = rank[sa[i - 1]] +
					(i == 1 || s[sa[i - 1]] < s[sa[i]]);
			if (rank[sa[i]] != rank[sa[i - 1]])
				bucket[++j] = i;
		}

		for (int len = 1; len <= n; len += len)
		{
			for (int i = 0, j; i < n; ++i)
			{
				if ((j = sa[i] - len) < 0) j += n;
				_sa[bucket[rank[j]]++] = j;
			}

			sa[_sa[bucket[0] = 0]] = 0;

			for (int i = 1, j = 0; i < n; ++i)
			{
				if (rank[_sa[i]] != rank[_sa[i - 1]] ||
					rank[_sa[i] + len] != rank[_sa[i - 1] + len])
					bucket[++j] = i;

				sa[_sa[i]] = j;
			}

			copy(sa.begin(), sa.end(), rank.begin());
			sa.swap(_sa);

			if (rank[sa[n - 1]] == n - 1) break;
```

```
        }

        for (int i = 0, j = rank[lcp[0] = 0], k = 0; i < n - 1; ++i, ++k)
            while (k >= 0 && s[i] != s[sa[j - 1] + k])
                lcp[j] = k--, j = rank[sa[j] + 1];
    }
};
```

10.7. Suffix Automaton.

```
/*
    Generalized Suffix Automaton

    Complexity: O(n)

    Tested:
    - http://codeforces.com/contest/616/problem/F
    - http://codeforces.com/contest/452/problem/E
    - http://codeforces.com/contest/204/problem/E
*/


template<size_t maxlen, size_t alpha>
struct SuffixAutomaton
{
    int go[2 * maxlen][alpha], slink[2 * maxlen], length[2 * maxlen];
    int size, last;

    int new_node()
    {
        memset(go[size], 0, sizeof go[size]);
        slink[size] = length[size] = 0;
        return size++;
    }

    SuffixAutomaton() { reset(); }

    void reset()
    {
        size = last = 0;
        new_node();
        slink[0] = -1;
    }

    int _extend(int c)
    {
        int p, q, np, nq;
        if (q = go[last][c])
        {
            if (length[q] == 1 + length[last]) return q;
            int nq = new_node();
            length[nq] = 1 + length[last];
            memcpy(go[nq], go[q], sizeof go[q]);
            slink[nq] = slink[q];
            slink[q] = nq;
            for (p = last; p != -1 && go[p][c] == q; p = slink[p])
                go[p][c] = nq;
            return nq;
        }
        np = new_node();
        length[np] = 1 + length[last];
        for (p = last; p != -1 && !go[p][c]; p = slink[p])
            go[p][c] = np;
        if (p == -1) return slink[np] = 0, np;
        if (length[q = go[p][c]] == 1 + length[p]) return slink[np] = q, np;
        nq = new_node();
        length[nq] = 1 + length[p];
        memcpy(go[nq], go[q], sizeof go[q]);
        slink[nq] = slink[q];
        slink[q] = slink[np] = nq;
        for (; p != -1 && go[p][c] == q; p = slink[p])
            go[p][c] = nq;
        return np;
    }

    void extend(int c) { last = _extend(c); }

    int bucket[maxlen + 1], order[2 * maxlen];

    void top_sort()
    {
        int maxl = 0;
        for (int e = 0; e < size; ++e)
            maxl = max(maxl, length[e]);
        for (int l = 0; l <= maxl; ++l)
            bucket[l] = 0;
        for (int e = 0; e < size; ++e)
            ++bucket[length[e]];
```

```
		for (int l = 1; l <= maxl; ++l)
			bucket[l] += bucket[l - 1];
		for (int e = 0; e < size; ++e)
			order[--bucket[length[e]]] = e;
	}
};
```

10.8. Z-function.

```
/*
	zfunction and suffix function

	Complexity: O(n)

	Tested:
	- http://www.spoj.com/problems/QUERYSTR/
	- http://codeforces.com/contest/126/problem/B
*/

// 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, 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[0..i]
vector<int> suffixes(const string &s)
{
	int n = s.length();

	vector<int> suff(n, n);

	for (int i = n - 2, g = n - 1, f; 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;
}
```