Team Reference
Universidad de la Habana : UH_Class_Zero

ACM-ICPC Caribbean Finals 2017

Team Members: Denis Gómez Cruz, Ernesto Navarro Díaz, Kedir Enrique Lluch
Developed by: José Carlos Gutiérrez Pérez

Contents

1. Data Structures

1.1. Abi.

```
template<typename T>
struct ABI
{
    int n;
    vector<T> f;
    ABI(int n) : n(n), f(n + 1) {}

    T query(int b)
    {
        T ret = 0;
        for (; b; b -= b & -b)
            ret += f[b];
        return ret;
    }

    void update(int b, T c)
    {
        for (; b <= n; b += b & -b)
            f[b] += c;
    }
};
```

1.2. AVL.

```
template<typename T>
struct avl
{
        struct node
        {
                T key;
                int h, sz;
                node* ch[2];

                int bf(){ return ch[1]->h - ch[0]->h; };

                void update()
                {
                        h = 1 + max(ch[0]->h, ch[1]->h);
                        sz = 1 + ch[0]->sz + ch[1]->sz;
                }

        }*root, *null;

        node* new_node(const T &key)
        {
                node* u = new node();
                u->key = key;
                u->h = u->sz = 1;
                u->ch[0] = u->ch[1] = null;
                return u;
        }

        node* rotate(node* u, bool d)
        {
                if (u == null || u->ch[!d] == null)
                        return u;

                node* t = u->ch[!d];
                u->ch[!d] = t->ch[d];
                t->ch[d] = u;

                u->update();
                t->update();

                return t;
        }
```

```
node* balance(node* u)
{
	u->update();

	if (u->bf() > 1)
	{
		if (u->ch[1]->bf() <= 0) // Right on left child
			u->ch[1] = rotate(u->ch[1], 1);
		u = rotate(u, 0); // Left
	}
	else if (u->bf() < -1)
	{
		if (u->ch[0]->bf() >= 0) // Left on right child
			u->ch[0] = rotate(u->ch[0], 0);
		u = rotate(u, 1); // Right
	}

	u->update();
	return u;
}

node* insert(node* u, const T &key)
{
	if (u == null)
		return u = new_node(key);

	if (u->key == key)
		return u;

	bool d = (key < u->key);
	u->ch[!d] = insert(u->ch[!d], key);

	return u = balance(u);
}

node* erase(node* u, T key)
{
	if (u == null)
		return u;

	int tmp = u->key;
	if (u->key == key)
	{
		if (u->ch[0] == null || u->ch[1] == null)
			return u->ch[u->ch[0] == null];
		else
		{
			node* t = u->ch[0];
			while (t->ch[1] != null)
				t = t->ch[1];
			key = u->key = t->key;
		}
	}

	bool d = (key < tmp);
	u->ch[!d] = erase(u->ch[!d], key);

	return u = balance(u);
}

void insert(T key)
{
	root = insert(root, key);
}

void erase(T key)
{
	root = erase(root, key);
}

bool find(const T &key)
{
	node* u = root;
	while (true)
	{
```

```
                if (u == null)
                    return false;
                if (u->key == key)
                    return true;
                u = u->ch[!(key < u->key)];
            }
        }

        int order_of_key(const T &key)
        {
            int r = 0;
            node* u = root;
            while (true)
            {
                if (u == null)
                    return r;
                if (key <= u->key)
                    u = u->ch[0];
                else
                    r += u->sz - u->ch[1]->sz, u = u
                        ->ch[1];
            }
        }

        T kth(int k)
        {
            assert(k <= root->sz);
            node* u = root;
            while (true)
            {
                if (u->sz - u->ch[1]->sz == k)
                    return u->key;
                if (k <= u->ch[0]->sz)
                    u = u->ch[0];
                else
                    k -= u->sz - u->ch[1]->sz, u = u
                        ->ch[1];
            }
        }

        void in_order(node* u)
        {
            if (u == null)
                return;

            in_order(u->ch[0]);
            cout << u->key.first << "\n";
            in_order(u->ch[1]);
        }

        avl()
        {
            null = new node();
            null->h = null->sz = 0;
            null->ch[0] = null->ch[1] = 0;
            root = null;
        }
};
```

1.3. BooleanRange.

```
bool find(set<pair<int, int>> &S, int x, int y)
{
    auto it = S.lower_bound({ x, y });
    return (it != S.end() && it->first <= x && it->second >=
        y)
            || (it != S.begin() && (--it)->first <= x &&
                it->second >= y);
}

void insert(set<pair<int, int>> &S, int x, int y)
{
```

```
    if (find(S, x, y))
        return;

    auto it = S.lower_bound({ x, y });
    while (it != S.end() && it->first >= x && it->second <= y
        )
        it = S.erase(it);

    it = S.insert({ x, y }).first;

    if (++it != S.end() && y + 1 >= it->first)
    {
        int t = it->second;
        S.erase({ x, y });
        S.erase(it);
        it = S.insert({ x, y = t }).first;
    }
    else --it;

    if (it != S.begin() && (--it)->second + 1 >= x)
    {
        int t = it->first;
        S.erase({ x, y });
        S.erase(it);
        S.insert({ x = t, y });
    }
}

void erase(set<pair<int, int>> &S, int x, int y)
{
    auto it = S.lower_bound({ x, y });
    while (it != S.end() && it->first >= x && it->second <= y
        )
        it = S.erase(it);

    it = S.lower_bound({ x, y });

    if (it != S.end() && y >= it->first)
    {
        int t = it->second;
        it = S.erase(it);
        if (y < t) it = S.insert({ y + 1, t }).first;
    }

    if (it != S.begin() && (--it)->second >= x)
    {
        auto t = *it;
        S.erase(it);
        if (t.first < x) S.insert({ t.first, x - 1 });
        if (y < t.second) S.insert({y + 1, t.second});
    }
}
```

1.4. HashTable.

```
template<typename H, typename T>
struct hash_table
{
    static const int n = 5000010, mod = 5000007;

    H h[n];
    T val[n];
    int f, p, c, link[n], last[mod];

    hash_table() : f(0), p(0), c(0) { memset(last, -1, sizeof last); }

    T* find(H hash)
    {
        for (f = -1, p = last[hash % mod]; p != -1; f = p, p
            = link[p])
            if (h[p] == hash)
                return &val[p];
        return NULL;
```

```
	}

	void set(H hash, T num)
	{
		if (find(hash) != NULL)
			val[p] = num;
		else
		{
			h[c] = hash;
			val[c] = num;
			hash %= mod;
			link[c] = last[hash];
			last[hash] = c++;
		}
	}

	void erase(H hash)
	{
		if (find(hash) == NULL)
			return;

		if (f == -1)
			last[hash % mod] = link[p];
		else
			link[f] = link[p];
	}
};
```

1.5. MonotonicQueue.

```
struct monotonic_queue
{
	deque<pair<int, ll>> deq;

	void add(int k, ll v)
	{
		while (!deq.empty() && deq.back().second <= v)
			deq.pop_back();
		deq.push_back({k, v});
	}

	void remove(int k)
	{
		while (!deq.empty() && deq.front().first <= k)
			deq.pop_front();
	}

	ll max()
	{
		return deq.front().second;
	}
};
```

1.6. OrderStatistic.

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

#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
    tree_order_statistics_node_update

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
}
```

1.7. RandomizedKdTree.

```
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;
	}

	node* build(vector<point> &p, int lo, int hi, bool d)
	{
		if (lo >= hi)
			return NULL;

		int m = (lo + hi) / 2;
		nth_element(p.begin() + lo, p.begin() + m, p.begin()
		    + hi,
				[&](const point &a, const point &b)
					{ return d ? a.real() < b.real
					    () : a.imag() < b.imag();
					});

		node *t = new node({ p[m], d });
		t->l = build(p, lo, m, !d);
		t->r = build(p, m + 1, hi, !d);

		return update(t);
	}

	node* build2(vector<point> &p, int lo, int hi, bool d)
	{
		if (lo >= hi)
```

```
            return NULL;

        swap(p[lo], p[lo + rand() % (hi - lo)]);

        int m = partition(p.begin() + lo + 1, p.begin() + hi,
                    [&](const point &a)
                        { return d ? a.real() < p[lo].
                            real() : a.imag() < p[lo].
                            imag(); }) - p.begin();

        node *t = new node({ p[lo], d });
        t->l = build2(p, lo + 1, m, !d);
        t->r = build2(p, m, hi, !d);

        return update(t);
}

void build(vector<point> &p)
{
        root = build(p, 0, p.size(), true);
}

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

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

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

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

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

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

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

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

```
                closest(second, p, ub);
    }

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

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

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

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

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

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

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

1.8. Rmq.

```
template<typename T>min
struct RMQ
{
    int n, lg;
    vector<vector<T>> dp;

    RMQ(vector<T> &a) : n(a.size()), lg(log2(n)), dp(n + 1,
        vector<T>(lg + 1, 1<<30))
    {
        for (int j = 1; j <= n; j++)
            dp[j][0] = a[j - 1];
        for (int i = 1; i <= lg; i++)
            for (int j = 1; j <= n - (1 << i) + 1; j++)
                dp[j][i] = min(dp[j][i - 1], dp[j + (1
                    << (i - 1))][i - 1]);
    }

    T query(int a, int b)
    {
        int l = log2(b - a + 1);
        return min(dp[a][l], dp[b - (1 << l) + 1][l]);
    }
};
```

1.9. SegmentTree.

```
struct segment_tree
{
    int n;
    vector<ll> a;

    segment_tree(int n) : n(n), a(2 * n) {}

    void update(int p, ll v)
    {
        for (a[p += n] = v; p /= 2;)
            a[p] = __gcd(a[2 * p], a[2 * p + 1]);
    }

    ll query(int l, int r)
    {
        ll g = 0;

        for (l += n, r += n; l < r; l /= 2, r /= 2)
        {
            if (l & 1) g = __gcd(g, a[l++]);
            if (r & 1) g = __gcd(g, a[--r]);
        }

        return g;
    }
};
```

1.10. Treap.

```
struct Treap
{
        struct node
        {
                node *ch[2];
                int key, prio, sz, acum[26], lazy;
                bool rev;

                node(int key) : key(key)
                {
                        ch[0] = ch[1] = NULL;
                        memset(acum, 0, sizeof acum);
                        lazy = -1;
                        rev = false;
                        acum[key] = sz = 1;
                        prio = rand();
                }
        }*root;

        node* new_node(int key)
        {
                return new node(key);
        }

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

        int getC(node *u, int c)
        {
                return u ? u->acum[c] : 0;
        }

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

                if (u->rev)
```

```
        {
            swap(u->ch[0], u->ch[1]);
            for (int i = 0; i < 2; ++i)
                if (u->ch[i])
                    u->ch[i]->rev ^= true;
        }

        if (u->lazy != -1)
            for (int i = 0; i < 2; ++i)
                if (u->ch[i])
                {
                    memset(u->ch[i]->acum, 0,
                        sizeof u->ch[i]->acum);
                    u->ch[i]->acum[u->lazy] =
                        u->ch[i]->sz;
                    u->ch[i]->lazy = u->ch[i
                        ]->key = u->lazy;
                }

        u->lazy = -1;
        u->rev = false;
    }

    node* update(node *u)
    {
        if (u)
        {
            u->sz = size(u->ch[0]) + size(u->ch[1]) +
                1;

            for (int i = 0; i < 26; ++i)
                u->acum[i] = getC(u->ch[0], i) +
                    getC(u->ch[1], i);

            ++u->acum[u->key];
        }
        return u;
    }
```

```
    pair<node*, node*> split(node* u, int k)
    {// split for the kth first elements
        push(u);

        if (!u)
            return { u, u };

        if (size(u->ch[0]) >= k)
        {
            auto s = split(u->ch[0], k);
            u->ch[0] = s.second;
            return { s.first, update(u) };
        }

        auto s = split(u->ch[1], k - size(u->ch[0]) - 1);
        u->ch[1] = s.first;
        return { update(u), s.second };
    }

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

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

        if (u->prio > v->prio)
        {
            u->ch[1] = merge(u->ch[1], v);
            return update(u);
        }

        v->ch[0] = merge(u, v->ch[0]);
        return update(v);
    }

    Treap() : root(NULL) {}
};
```

1.11. UnionFind.

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

	union_find(int n) : p(n, −1) {}

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

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

1.12. VantagePointTree.

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

1.13. WaveletTree.

```
template<typename T>
struct wavelet_tree
{
    struct node
    {
        T lo, hi;
        vector<int> a;
        node *l, *r;
        node(T x, T y, int sz) :
```

```
            lo(x), hi(y), a(1, 0), l(NULL), r(NULL
                ) { a.reserve(sz); }
}*root;

wavelet_tree(vector<T> &a, T MAX) // 1-based
{
    root = build(a, 1, a.size(), 0, MAX);
}

node* build(vector<T> &a, int l, int r, T lo, T hi)
{
    node *cur = new node(lo, hi, r - l + 1);
    if (lo == hi || l >= r) return cur;

    T md = (lo + hi) / 2;
    for (int i = l; i < r; ++i)
        cur->a.push_back(cur->a.back() + (a[i] <=
            md));

    auto p = stable_partition(a.begin() + l, a.begin() + r,
        [md](int lo){ return lo <= md; });
    cur->l = build(a, l, p - a.begin(), lo, md);
    cur->r = build(a, p - a.begin(), r, md + 1, hi);
    return cur;
}

T kth(node *cur, int l, int r, int k)
{
    if (l > r) return 0;
    if (cur->lo == cur->hi) return cur->lo;
    int al = cur->a[l - 1], ar = cur->a[r];
    if (k <= ar - al) return kth(cur->l, al + 1, ar, k);
    return kth(cur->r, l - al, r - ar, k - ar + al);
}

int less_equal(node *cur, int l, int r, T k)
{
    if (l > r || k < cur->lo) return 0;
    if (cur->hi <= k) return r - l + 1;
    int al = cur->a[l - 1], ar = cur->a[r];
    return less_equal(cur->l, al + 1, ar, k) + less_equal(
        cur->r, l - al, r - ar, k);
}

int equal(node *cur, int l, int r, T k)
{
    if (l > r || k < cur->lo || k > cur->hi) return 0;
    if (cur->lo == cur->hi) return r - l + 1;
    int al = cur->a[l - 1], ar = cur->a[r], md = (cur->
        lo + cur->hi) / 2;
    if (k <= md) return equal(cur->l, al + 1, ar, k);
    return equal(cur->r, l - al, r - ar, k);
}
};
```

2. Dynamic Programming

2.1. Convex Hull Trick.

```
struct convex_hull_trick // upper_hull
{
	typedef long long ll;
	struct point { ll x, y; };
	vector<point> hull;
	int last;

	convex_hull_trick()
	{
		last = 0;
		hull.clear();
		hull.push_back({ 0, 0 });
	}

	void add(ll x, ll y)
	{
		int p = hull.size();
		while (--p && (y - hull[p - 1].y) * (hull[p].x - x) +
			y * (x - hull[p - 1].x)
				>= hull[p].y * (x -
					hull[p - 1].x))
					hull.pop_back();
		last = min(last, (int) hull.size());
		hull.push_back({ x, y });
	}

	inline ll f(int i, ll x)
	{
		return hull[i].x * x + hull[i].y;
	}

	ll query(ll x)
	{
		while (last + 1 < (int) hull.size() && f(last + 1, x)
			>= f(last, x)) ++last;
		return f(last, x);
	}
};
```

2.2. Divide And Conquer Opt.

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
ll n, k, x[1005], w[1005], dp[1005][1005], cost[1005][1005];

void solve(int lo, int hi, int optL, int optR, int k)
{
	if (lo > hi)
		return;
	int m = (lo + hi) / 2, opt;
	for (int i = optL; i <= min(optR, m); ++i)
		if (dp[k - 1][i] + cost[i + 1][m] < dp[k][m])
		{
			dp[k][m] = dp[k - 1][i] + cost[i + 1][m];
			opt = i;
		}

	solve(lo, m - 1, optL, opt, k);
	solve(m + 1, hi, opt, optR, k);
}
```

```
int main()
{
	ios_base::sync_with_stdio(0), cin.tie(0);

	while (cin >> n >> k)
	{
		for (int i = 1; i <= n; ++i)
			cin >> x[i] >> w[i];

		memset(cost, 0, sizeof cost);
		memset(dp, 63, sizeof dp);

		for (int i = 1; i <= n; ++i)
		{
			for (int j = i - 1; j; --j)
				cost[j][i] = cost[j + 1][i] + (x[i] - x[j])
					* w[j];
			dp[1][i] = cost[1][i];
		}

		for (int i = 2; i <= k; ++i)
			solve(1, n, 1, n, i);

		cout << dp[k][n] << "\n";
	}

	return 0;
}
```

3. Geometry

3.1. AntipodalPoints.

```
/*
	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. BasicsComplex.

```
typedef complex<double> point;
typedef vector<point> polygon;

#define NEXT(i) (((i) + 1) % n)

struct circle { point p; double r; };
struct line { point p, q; };
using segment = line;
```

```
const double eps = 1e-9;

// fix comparations on doubles with this two functions
int sign(double x) { return x < -eps ? -1 : x > eps; }

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

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

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

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

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();
        }
}

double angle(point a, point b, point c) // returns the angle abc (cos(x)
     = Va * Vb / |Va| * |Vb|)
{
    a -= b, c -= b;
    return acos((a.X * c.X + a.Y * c.Y) / (sqrt((double) sqr(a.X) +
         sqr(a.Y)) * sqrt((double) sqr(c.X) + sqr(c.Y))));
}

// contrary clock side direction
pair<double, double> rotacion(double x, double y, double ang)
{
        ang = (acos(-1.0) * ang) / 180.0;
        return { x * cos(ang) - y * sin(ang), x * sin(ang) + y * cos(
             ang) };
}

// contrary clock side direction
point rotacion(point x, ld ang)
{
        ang = (acos(-1.0) * ang) / 180.0;
        return x * polar(1.0, ang);//ang en radianes...
}
```

3.3. Basics.

```
double INF = 1e100;
double EPS = 1e-12;

struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y) {}
  PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }
  PT operator * (double c) const { return PT(x*c, y*c ); }
  PT operator / (double c) const { return PT(x/c, y/c ); }
};
```

```
double dot(PT p, PT q) { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q) { return dot(p-q,p-q); }
double cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
PT RotateCW90(PT p) { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) >
        0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}
```

```
// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+
       RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
      c = !c;
  }
  return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS
        )
      return true;
    return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R)
    {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}

// This code computes the area or centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion. Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
```

```
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}

// tests whether or not a given polygon (in CW or CCW order) is
    simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}
```

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 > l) s = l;
        if (t < 0.0) t = 0.0;
        else if (t > l) t = l;
        return (x(s) * y(t) - x(t) * y(s)
                + (radian(xa, ya, x(s), y(s))
                + radian(x(t), y(t), xb, yb)) * r * r) *
                    0.5;
    }
}

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

3.6. ClosestPairPoints.

```
/*
    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. ConvexCut.

```
/*
    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;
}
```

3.8. ConvexHull.

```
vector<point> convex_hull(vector<point> v)
{
    int n = v.size(), k = 0;
    vector<point> ch(2 * n);

    sort(v.begin(), v.end(), cmp);
  for (ll i = k = 0; i < n; ch[k++] = v[i++])
    while (k > 1 && cross(ch[k - 2], ch[k - 1], v[i]) <= 0) --k;
  for (ll i = n - 2, t = k; i >= 0; ch[k++] = v[i--])
    while (k > t && cross(ch[k - 2], ch[k - 1], v[i]) <= 0) --k;
  ch.resize(k - (k > 1));

    return ch;
}
```

3.9. LineSegmentIntersections.

```
/*
        Line and segments predicates

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

bool intersectLL(const line &l, const line &m)
{
        return abs(cross(l.q - l.p, m.q - m.p)) > eps || // non-
            parallel
                        abs(cross(l.q - l.p, m.p - l.p)) < eps; // same
                            line
}

bool intersectLS(const line &l, const segment &s)
{
        return cross(l.q - l.p, s.p - l.p) * // s[0] is left of l
                        cross(l.q - l.p, s.q - l.p) < eps; // s[1] is right
                            of l
}

bool intersectLP(const line &l, const point &p)
{
        return abs(cross(l.q - p, l.p - p)) < eps;
}

bool intersectSS(const segment &s, const segment &t)
{
        return ccw(s.p, s.q, t.p) * ccw(s.p, s.q, t.q) <= 0
                        && ccw(t.p, t.q, s.p) * ccw(t.p, t.q, s.q) <= 0;
}

bool intersectSP(const segment &s, const point &p)
{
        return abs(s.p - p) + abs(s.q - p) - abs(s.q - s.p) < eps;
        // triangle inequality
        return min(real(s.p), real(s.q)) <= real(p)
                        && real(p) <= max(real(s.p), real(s.q))
                        && min(imag(s.p), imag(s.q)) <= imag(p)
                        && imag(p) <= max(imag(s.p), imag(s.q))
                        && cross(s.p - p, s.q - p) == 0;
}

point projection(const line &l, const point &p)
{
        double t = dot(p - l.p, l.p - l.q) / norm(l.p - l.q);
        return l.p + t * (l.p - l.q);
}

point reflection(const line &l, const point &p)
{
        return p + 2.0 * (projection(l, p) - p);
}

double distanceLP(const line &l, const point &p)
{
        return abs(p - projection(l, p));
}

double distanceLL(const line &l, const line &m)
{
        return intersectLL(l, m) ? 0 : distanceLP(l, m.p);
}

double distanceLS(const line &l, const line &s)
{
        if (intersectLS(l, s)) return 0;
        return min(distanceLP(l, s.p), distanceLP(l, s.q));
}

double distanceSP(const segment &s, const point &p)
{
        const point r = projection(s, p);
        if (intersectSP(s, r)) return abs(r - p);
```

```
	return min(abs(s.p - p), abs(s.q - p));
}

double distanceSS(const segment &s, const segment &t)
{
	if (intersectSS(s, t)) return 0;
	return min(min(distanceSP(s, t.p), distanceSP(s, t.q)),
			min(distanceSP(t, s.p), distanceSP(t, s.q)));
}

point crosspoint(const line &l, const line &m)
{
	double A = cross(l.q - l.p, m.q - m.p);
	double B = cross(l.q - l.p, l.q - m.p);
	if (abs(A) < eps && abs(B) < eps)
		return m.p; // same line
	if (abs(A) < eps)
		assert(false); // !!!PRECONDITION NOT
			SATISFIED!!!
	return m.p + B / A * (m.q - m.p);
}
```

3.10. 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;
}
```

3.11. PickTheorem.

```
/*
	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};
}
```

3.12. Points3D.

```
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)
}
```

3.13. PointsinPolygon.

```
/*
	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;
}
```

3.14. PolygonArea.

```
/*
	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;
}
```

3.15. PolygonWidth.

```
/*
	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;
}
```

3.16. RectangleUnion.

```
/*
	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;
}
```

3.17. RectilinearMst.

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

typedef long long ll;

typedef complex<ll> point;

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

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

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

		map<ll, int> sweep;

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

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

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

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

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

return cost;
}
```

3.18. Triangles.

```
double area_heron(double const &a, double const &b, double const &c
  )
{
	double s=(a+b+c)/2;
	return sqrt(s*(s-a)*(s-b)*(s-c));
}
double circumradius(const double &a, const double &b, const double
  &c)
```

```
{
        return a*b*c/4/AreaHeron(a,b,c);
}
double inradius(const double &a, const double &b, const double &c)
{
        return 2*AreaHeron(a,b,c)/(a+b+c);
}
/*
        Center of the circumference of a triangle
        [Tested COJ 1572 - Joining the Centers]
 */
point circunference_center(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;
}
bool circunference_center(point &a,point &b,point &c,point &r)
{
        double d=(a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y))*2.0;
        if(fabs(d)<EPS) return false;

        r.x=((a.x*a.x+a.y*a.y)*(b.y-c.y)+(b.x*b.x+b.y*b.y)*(c.y-a.y
            )
                +(c.x*c.x+c.y*c.y)*(a.y-b.y))/d;
        r.y=-((a.x*a.x+a.y*a.y)*(b.x-c.x)+(b.x*b.x+b.y*b.y)*(c.x-a.
            x)
                +(c.x*c.x+c.y*c.y)*(a.x-b.x))/d;
        return true;
}
double incenter(vect &a,vect &b,vect &c,vect &r)
{
        double u=(b-c).length(),v=(c-a).length(),w=(a-b).length(),s
            =u+v+w;
        if(s<EPS) {r=a;return 0.0;}
        r.x=(a.x*u+b.x*v+c.x*w)/s;
        r.y=(a.y*u+b.y*v+c.y*w)/s;
        return sqrt((v+w-u)*(w+u-v)*(u+v-w)/s)*0.5;
}
bool orthocenter(vect &a,vect &b,vect &c,vect &r)
{
        double d=a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
        if(fabs(d)<EPS) return false;
        r.x=((c.x*b.x+c.y*b.y)*(c.y-b.y)+(a.x*c.x+a.y*c.y)*(a.y-c.y)
                +(b.x*a.x+b.y*a.y)*(b.y-a.y))/d;
        r.y=-((c.x*b.x+c.y*b.y)*(c.x-b.x)+(a.x*c.x+a.y*c.y)*(a.x-c.
            x)
                +(b.x*a.x+b.y*a.y)*(b.x-a.x))/d;
        return true;
}
double signed_area(const point &p1, const point &p2, const point &
    p3)
{
        return cross(p2-p1,p3-p1);
}
double triangle_area(const point &a, const point &b,const point &c)
{
        return 0.5* abs( cross(b-a,c-a) );
}
```

4. Graph

4.1. Aborescense.

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

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

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

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

			int index = 0, sz = 0;
			vector<int> C(n, -1), mark(n, -1);
			for (int i = 0, u = 0; i < n; u = ++i) //
				contract cycles
			{
				if (mark[i] != -1) continue;

				while (mark[u] == -1)
					mark[u] = i, u = in[u].src;

				if (mark[u] != i || u == r) continue;

				for (int v = in[u].src; u != v; C[v] =
					index, v = in[v].src) ;
				C[u] = index++;
			}

			if (index == 0) return res; // found
				arborescence

			for (int i = 0; i < n; ++i) // contract
				if (C[i] == -1) C[i] = index++;

			for (auto &e : edges)
				if (C[e.src] != C[e.dst] && C[e.dst] !=
					C[r])
					edges[sz++] = { C[e.src], C[e.
						dst], e.weight - in[e.dst].
						weight };

			edges.resize(sz), n = index, r = C[r];
		}
	}
};
```

4.2. Aborescense2.

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

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

	skew_heap() : root(NULL) {}

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

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

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

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

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

	bool empty()
	{
		return !root;
	}

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

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

template<typename T>
struct minimum_aborescense
{
        vector<edge<T>> edges;

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

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

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

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

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

                return score;
        }
};
```

4.3. **ArticulationPoints.**

```
struct articulation_points
{
        int V, dt;
        vector<bool> ap;
```

```
    vector<int> dfsnum, low;
    vector<vector<int>> G;

    articulation_points(int n) :
            V(n), dt(0), ap(n, 0), dfsnum(n, 0), low(n, 0), G(n, vector
                <int>(0)) {}

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

    void dfs(int u)
    {
        dfsnum[u] = low[u] = ++dt;
        for (int v : G[u])
            if (!dfsnum[v])
            {
                dfs(v), low[u] = min(low[u], low[v]);
                if ((dfsnum[u] == 1 && dfsnum[v] > 2) || (dfsnum[u]
                    != 1 && low[v] >= dfsnum[u])) ap[u] = true;
            }
            else low[u] = min(low[u], dfsnum[v]);
    }

    vector<bool> solve()
    {
        for (int i = 0; i < V; ++i)
            if (!dfsnum[i]) dt = 0, dfs(i);
        return ap;
    }
};
```

4.4. BellmanFord.

```
struct bellman_ford
{
        int V, E;
        vector<ll> d;
        vector<int> parent, cycle;
        struct edge { int u, v; ll w; };
        vector<edge> G;

        bellman_ford(int n) :
                        V(n), E(0), d(n, oo), parent(n, -1), cycle(0),
                            G(0) {}

        void add_edge(int u, int v, ll w)
        {
                G.push_back({ u, v, w });
        }

        void negative_cycle(int u)
        {
                for (int i = 0; i < V; ++i)
                        u = parent[u];

                cycle.push_back(u);
                for (int v = parent[u]; v != u; v = parent[v])
                        cycle.push_back(v);
        }

        bool solve(int source = 0)
        {
                d[source] = 0;
                E = G.size();
                bool r = true;

                for (int i = 1; i <= V && r; ++i)
                {
                        r = false;
                        for (auto e : G)
```

```
                if (d[e.u] != oo && d[e.u] + e.w < d[e
                    .v])
                {
                    r = true;
                    parent[e.v] = e.u;
                    d[e.v] = d[e.u] + e.w;

                    if (i == V)
                    {
                        negative_cycle(e.v);
                        return true;
                    }
                }
            }

            return false;
        }
    };
```

4.5. BiconnectedComponentBlocks.

```
int num[N], low[N], timer;
bool artP[N];
vector<int> G[N],Stack;


void dfs(int u, vector<vector<int> > &cmps)
{
    Stack.push_back(u);
    num[u]=low[u]=timer++;

    for(int i=0;i<G[u].size();++i)
    {
        int v=G[u][i];
        if(!num[v])
        {
            dfs(v, cmps);
            low[u]=min(low[v], low[u]);
            if(low[v]>=num[u])
            {
                if(u!=0||num[v]>2)
                    artP[u]=1;

                cmps.push_back(vector<int>(1,u));
                do
                {
                    cmps.back().push_back(Stack.back());
                    Stack.pop_back();
                } while (cmps.back().back()!=v);

            }
        }
        else
            low[u]=min(low[u], num[v]);
    }
}
```

4.6. BiconnectedComponentEdges.

```
vector<int> num(n), low(n), stk;
vector<vector<int>> comps;
int timer = 0;

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

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

                if (num[u] == low[u])
                {
                        comps.push_back({});

                        do
                        {
                                comps.back().push_back(stk.back());
                                stk.pop_back();
                        }
                        while (comps.back().back() != u);
                }
        };
```

4.7. BiconnectedComponent.

```
typedef pair<int, int> ii;

struct biconnected_component
{
        int V, E, dt;
        vector<ii> S;
        vector<bool> usd;
        vector<int> dfsnum, low;
        vector<vector<ii>> G, BCC;

        biconnected_component(int n) :
                        V(n), E(0), dt(0), S(0), usd(0), dfsnum(n, 0),
                             low(n, 0), G(n, vector<ii>(0)), BCC(0)
                             {}

        void add_edge(int u, int v)
        {
                G[u].push_back(ii(v, E)), G[v].push_back(ii(u, E++))
                     ;
        }

        void dfs(int u)
        {
                dfsnum[u] = low[u] = ++dt;

                for (auto i : G[u])
                {
                        int v = i.first, e = i.second;

                        if (!usd[e])
                                S.push_back(ii(u, v)), usd[e] = true;

                        if (!dfsnum[v])
                        {
                                dfs(v);
                                if (low[v] >= dfsnum[u])
                                {
                                        BCC.push_back(vector<ii
                                             >(0));
                                        do
                                        {
                                                BCC.back().
                                                     push_back(S.
                                                     back()), S.
                                                     pop_back();
```

```
                    } while (BCC.back().back()
                        != ii(u, v));
                }
                else low[u] = min(low[u], low[v]);
            }
            else low[u] = min(low[u], dfsnum[v]);
        }
    }

    vector<vector<ii>> solve()
    {
        usd.resize(E, false);
        for (int i = 0; i < V; ++i)
            if (!dfsnum[i]) dfs(i);
        return BCC;
    }
};
```

4.8. BipartiteMatching.

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

4.9. Bridges.

```
struct bridge
{
    int V, dt;
    vector<int> low, dfsnum, parent;
    vector<pair<int, int>> bridges;
    vector<vector<int>> G;

    bridge(int n) :
            V(n), dt(0), low(n, 0), dfsnum(n, 0), parent(n), bridges(0),
                G(n, vector<int>(0)) {}

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

    void dfs(int u)
    {
        dfsnum[u] = low[u] = ++dt;
        for (int v : G[u])
            if (!dfsnum[v])
            {
                parent[v] = u, dfs(v), low[u] = min(low[u], low[v]);
                if (dfsnum[u] < low[v]) bridges.push_back(make_pair(
                    u, v));
            }
            else if (v != parent[u])
                low[u] = min(low[u], dfsnum[v]);
    }

    vector<pair<int, int>> solve()
    {
        for (int i = 0; i < V; ++i)
            if (!dfsnum[i]) dfs(i);
        return bridges;
    }
};
```

4.10. CentroidDescomposition.

```
struct centroid_descomposition
{
    int n;
        vector<bool> del;
    vector<vector<int>> adj;
        vector<int> size, parent;
    vector<vector<pair<int, int>>> anc;

    centroid_descomposition(int n) : n(n), del(n), adj(n), size(n),
        parent(n), anc(n) {};

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

    int centroid(int u)
    {
        vector<int> seen = {u};
        parent[u] = -1;

        for (size_t i = 0; i < seen.size(); ++i)
        {
            u = seen[i];
            for (int v : adj[u])
                if (!del[v] && v != parent[u])
                    parent[v] = u, seen.push_back(v);
```

```
        }

        for (int sz = seen.size(), i = sz - 1, mx = 0; i >= 0; --i, mx
            = 0)
        {
            size[u = seen[i]] = 1;
            for (int v : adj[u])
                if (!del[v] && v != parent[u])
                    size[u] += size[v], mx = max(mx, size[v]);
            if (max(sz - size[u], mx) <= sz / 2)
                return u;
        }

        return -1;
    }

    void dfs(int u, int p, int d, int c)
    {
        anc[u].push_back({c, d});
        for (int v : adj[u])
            if (!del[v] && v != p)
                dfs(v, u, d + 1, c);
    }

    void rootify(int r)
    {
        int c = centroid(r);
        assert(c != -1);
        del[c] = true;
        dfs(c, -1, 0, c);

        for (int v : adj[c])
            if (!del[v]) rootify(v);
    }

    void update(int u)
    {
        for(auto p : anc[u]) ;
    }

    void query(int u)
    {
        for(auto p : anc[u]) ;
    }
};
```

4.11. Dinic.

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

    int n;
    vector<vector<edge>> adj;
    vector<int> level, que, iter;

    dinic(int n) : n(n), adj(n), level(n), que(n), iter(n) {}

    void add_edge(int u, int v, T cuv, T cvu = 0)
    {
        adj[u].push_back({u, v, 0, cuv, (int) adj[v].size()});
        if (u == v) ++adj[u].back().rev;
        adj[v].push_back({v, u, 0, cvu, (int) adj[u].size() - 1});
    }

    bool bfs(int source, int sink)
```

```
    {
        level.assign(n, -1);
        que[0] = sink;
        level[sink] = 0;

        for (int qf = 0, qb = 1; qf < qb; ++qf)
        {
            sink = que[qf];
            for (edge &e : adj[sink])
            {
                edge &erev = adj[e.dst][e.rev];
                if (level[erev.src] == -1 && erev.flow < erev.cap)
                    level[que[qb++] = erev.src] = 1 + level[sink];
            }
        }

        return level[source] >= 0;
    }

    T dfs(int source, int sink, T flow)
    {
        if (source == sink) return flow;

        for (; iter[source] < (int) adj[source].size(); ++iter[source])
        {
            edge &e = adj[source][iter[source]];

            if (e.flow < e.cap && level[e.dst] + 1 == level[source])
            {
                T delta = dfs(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;
    }

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

    T max_flow(int source, int sink)
    {
        T flow = 0;

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

        while (bfs(source, sink))
        {
            iter.assign(n, 0);

            for (T cur; (cur = dfs(source, sink, oo)) > 0;)
                flow += cur;
        }

        return flow;
    }
};
```

4.12. DominatorTree.

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

    Tested: SPOJ EN

    Complexity: O(m log n)

    in control flow graphs, a node d dominates a node n
    if every path from the entry node to n must go through d
```

```
*/

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;
        }
    };
```

4.13. **Euleriangraph.**

```
typedef vector<vector<int>> vvi;

/*
    Euler path undirected (path to use once all edges)
    the degree of all nodes must be even (euler cycle)
    or only exists two odd nodes (euler path)
*/
void visit(const vvi &G, vvi &adj, int s, vector<int> &path)
{
    for (auto v : G[s])
        if (adj[s][v])
        {
            --adj[s][v], --adj[v][s];
            visit(G, adj, v, path);
        }
    path.push_back(s);
}

bool euler_path(const vvi &G, int s, vector<int> &path)
{
    int n = G.size(), odd = 0, m = 0;
    for (int i = 0; i < n; ++i)
    {
        odd += G[i].size() & 1;
        m += G[i].size();
    }
    if (odd == 0 || (odd == 2 && G[s].size() % 2 == 0))
    {
        vvi adj(n, vector<int>(n));
        for (int u = 0; u < n; ++u)
            for (auto v : G[u])
                ++adj[u][v];

        visit(G, adj, s, path);
        reverse(path.begin(), path.end());
        return path.size() == m / 2 + 1;
    }
    return false;
}

/*
    Euler path directed (path to use once all edges)
    the in-degree - out-degree == 0 for all nodes (euler cycle)
    or only exists two nodes with |in-degree - out-degree| == 1
        (euler path)
*/
void visit(vvi &G, int u, vector<int>& path)
{
    while (!G[u].empty())
    {
        int v = G[u].back();
        G[u].pop_back();
        visit(G, v, path);
```

```
        }
        path.push_back(u);
}

bool euler_path(vvi G, int s, vector<int> &path)
{
        int n = G.size(), m = 0;
        vector<int> deg(n);
        for (int u = 0; u < n; ++u)
        {
                m += G[u].size();
                for (auto v : G[u])
                        --deg[v];   // in-deg
                deg[u] += G[u].size(); // out-deg
        }

        int k = n - count(deg.begin(), deg.end(), 0);
        if (k == 0 || (k == 2 && deg[s] == 1))
        {
                visit(G, s, path);
                reverse(path.begin(), path.end());
                return path.size() == m + 1;
        }
        return false;
}
```

4.14. FloydWarshall.

```
for (int k = 0; k < V; ++k)
        for (int i = 0; i < V; ++i)
                if (c[i][k] < oo)
                        for (int j = 0, w; j < V; ++j)
                                if ((w = c[i][k] + c[k][j]) < c[i][j])
                                        c[i][j] = w;
```

4.15. GabowEdmonds.

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

4.16. GomoryHuTree.

```
/*
        Gomory−Hu tree

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

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

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

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

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

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

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

```
            }
        }
        return level[t];
    }

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

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

    vector<edge> tree;

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

4.17. HeavyLightDescomposition.

```
struct heavy_light_descomposition
{
    int n;
    vector<vector<int>> G;
    vector<int> parent, depth, size, head, position;
    ABI<int> abi;
    heavy_light_descomposition(int n) :n(n), G(n), parent(n, -1),
        depth(n), size(n), head(n), position(n), abi(n) {}

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

    void rotify(int r = 0)
    {
        vector<int> heavy(n, -1), Q(1, r);

        for (int i = 0, u; i < n; ++i)
        {
            size[u = Q[i]] = 1;
            for (auto v : G[u])
                if (parent[u] != v)
                    Q.push_back(v), parent[v] = u, depth[v] =
                        depth[u] + 1;
        }

        for (int i = n - 1, u; i >= 0; i--)
        {
```

```
            u = Q[i];
            for (auto v : G[u])
                if (parent[u] != v)
                {
                    size[u] += size[v];
                    if (heavy[u] == -1 || size[v] > size[heavy[u]])
                        heavy[u] = v;
                }
        }

        for (int u = 0, pos = 0; u < n; u++)
            if (u == r || heavy[parent[u]] != u)
                for (int v = u; v != -1; v = heavy[v])
                    head[v] = u, position[v] = ++pos;
    }

    int LCA(int u, int v)
    {
        while (head[u] != head[v])
        {
            if (depth[head[u]] < depth[head[v]])
                swap(u, v);
            u = parent[head[u]];
        }
        return (depth[u] < depth[v] ? u : v);
    }

    int query_up(int u, int v)
    {
        int ans = 0;
        while (head[u] != head[v])
        {
            ans += abi.query(position[u]) - abi.query(position[
                head[u]] - 1);
            u = parent[head[u]];
        }
        return (ans + abi.query(position[u]) - abi.query(position[v
            ] - 1));
    }

    int query(int u, int v)
    {
        int L = LCA(u, v);
        return query_up(u, L) + query_up(v, L) - query_up(L,
            L);
    }

    void update(int u, int c)
    {
        abi.update(position[u], c - (abi.query(position[u]) - abi.
            query(position[u] - 1)));
    }
};
```

4.18. **HopcroftKarpMcbm.**

```
struct hopcroft_karp
{
    int N, M;
    vector<vector<int> > G;
    vector<int> match, d, q;

    hopcroft_karp(int n, int m) :
        N(n), M(m), G(n, vector<int>(0)), match(n
            + m, -1) {}

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

```
	bool bfs()
	{
		d.assign(N + M, 0), q.clear(), q.reserve(N);
		for (int i = 0; i < N; ++i)
			if (match[i] == -1) q.push_back(i);

		bool f = false;
		for (size_t i = 0; i < q.size(); ++i)
			for (auto v : G[q[i]])
				if (!d[N + v])
				{
					d[N + v] = d[q[i]] + 1;
					if (match[N + v] != -1)
						d[match[N + v]] = d[
						    N + v] + 1, q.
						    push_back(
						    match[N + v]);
					else
						f = true;
				}

		return f;
	}

	bool dfs(int u)
	{
		for (auto v : G[u])
			if (d[N + v] == d[u] + 1)
			{
				d[N + v] = 0;
				if (match[N + v] == -1 || dfs(match[
				    N + v]))
				{
					match[u] = v, match[N + v]
					    = u;
					return true;
				}
			}

		return false;
	}

	int solve()
	{
		int flow = 0;
		while (bfs())
			for (int i = 0; i < N; ++i)
				flow += (match[i] == -1 && dfs(i));
		return flow;
	}
};
```

4.19. IsGraphic.

```
/*
	Grahpic sequence recognition

	Tested: UVA 11414, 10720
*/
// Receives a sorted degree sequence (non ascending)
bool is_graphic(vector<int> d)
{
	int n = d.size();
	sort(d.rbegin(), d.rend());
	vector<int> s(n + 1);
	for (int i = 0; i < n; ++i)
		s[i + 1] = s[i] + d[i];
	if (s[n] % 2)
		return false;
	for (int k = 1; k <= n; ++k)
	{
```

```
            int p = lower_bound(d.begin() + k, d.end(), k,
                greater<int>())
                        - d.begin();
            if (s[k] > k * (p - 1) + s[n] - s[p])
                return false;
        }
        return true;
    }
```

4.20. **LcaEulerTour.**

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

4.21. Lca.

```
template<typename T>
struct LCA
{
    int n;
    vector<vector<pair<int, T>>> adj;
    vector<vector<int>> lca;
    vector<vector<T>> dist;
    vector<int> p, lvl;

    LCA(int n) : n(n), adj(n + 1), lca(__lg(n) + 1, vector<int>(
        n + 1))
                                , dist(__lg(n) + 1, vector<T
                                    >(n + 1)), p(n + 1), lvl(
                                    n + 1) {}

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

    void buildlca(int r = 1)
    {
        queue<int> q;
        q.push(r);
        p[r] = -1;
        lvl[r] = 0;

        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            for (auto i : adj[u])
            {
```

```
                int v = i.first, w = i.second;
                if (v == p[u]) continue;

                q.push(v);
                p[v] = u;
                lca[0][v] = u;
                dist[0][v] = w;
                lvl[v] = lvl[u] + 1;

                for (int i = 1, lg = __lg(lvl[v]); i <=
                     lg; i++)
                {
                    lca[i][v] = lca[i - 1][lca[i - 1][
                        v]];
                    dist[i][v] = dist[i - 1][lca[i -
                        1][v]] + dist[i - 1][v];
                }
            }
        }
    }

    T query(int u, int v)
    {
        if (lvl[v] > lvl[u])
            swap(u, v);

        T D = 0;
        for (int i = __lg(lvl[u]); i >= 0; i--)
            if (lvl[u] - (1 << i) >= lvl[v])
            {
                D += dist[i][u];
                u = lca[i][u];
            }

        if (u == v)
            return D; //u;

        for (int i = __lg(lvl[u]); i >= 0; i--)
            if ((1 << i) <= lvl[u] && lca[i][u] != lca[i][v])
            {
                D += dist[i][v] + dist[i][u];
                u = lca[i][u];
                v = lca[i][v];
            }

        D += dist[0][u] + dist[0][v];
        return D; //p[u];
    }
};
```

4.22. MinCostFlow(BellmanFord).

```
/*
    Tested: ZOJ 3885
*/

template<typename T, typename C = T>
struct min_cost_flow
{
    struct edge
    {
        int src, dst;
        T cap, flow;
        C cost;
        int rev;
    };

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

    min_cost_flow(int n) : n(n), adj(n) {}
```

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

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

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

	bool bellman_ford(int s, int t)
	{
		dist.assign(n, oo);
		prev.assign(n, nullptr);
		curflow.assign(n, 0);
		dist[s] = 0;
		curflow[s] = numeric_limits<T>::max();

		for (int it = 0, change = true; it < n && change; ++it)
		{
			change = false;
			for (int u = 0; u < n; ++u)
				if (dist[u] != oo)
				{
					for (auto &e : adj[u])
						if (e.flow < e.cap && dist[e.dst] > dist[u] + e.cost)
						{
							dist[e.dst] = dist[u] + e.cost;
							prev[e.dst] = &e;
							curflow[e.dst] = min(curflow[u], e.cap - e.flow);
							change = true;
						}
				}
		}

		return dist[t] < oo;
	}

	pair<T, C> max_flow(int s, int t)
	{
		T flow = 0;
		C cost = 0;

		while (bellman_ford(s, t))
		{
			T delta = curflow[t];
			flow += delta;
			cost += delta * dist[t];

			for (edge *e = prev[t]; e != nullptr; e = prev[e->src])
			{
				e->flow += delta;
				adj[e->dst][e->rev].flow -= delta;
			}
		}

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

4.23. Min-costFlow(Dijkstra).

```
/*
	Tested: ZOJ 3885
*/

template<typename T, typename C = T>
struct min_cost_flow
{
	struct edge
	{
		int src, dst;
		T cap, flow;
		C cost;
		int rev;
	};

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

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

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

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

	vector<C> dist, pot;
	vector<edge*> prev;

	bool dijkstra(int s, int t)
	{
		dist.assign(n, oo);
		prev.assign(n, nullptr);
		dist[s] = 0;

		using pci = pair<C, int>;
		priority_queue<pci, vector<pci>, greater<pci>> pq;
		pq.push({ 0, s });

		while (!pq.empty())
		{
			C d; int u;
			tie(d, u) = pq.top();
			pq.pop();

			if (d != dist[u])
				continue;

			for (auto &e : adj[u])
				if (e.flow < e.cap
						&& dist[e.dst] > dist[
						u] + e.cost + pot
						[u] - pot[e.dst])
				{
					dist[e.dst] = dist[u] + e.cost
						+ pot[u] - pot[e.dst];
					prev[e.dst] = &e;
					pq.push({ dist[e.dst], e.dst });
				}
		}

		return dist[t] < oo;
	}

	pair<T, C> max_flow(int s, int t)
	{
		T flow = 0;
		C cost = 0;
```

```
        pot.assign(n, 0);
        while (dijkstra(s, t))
        {
            for (int u = 0; u < n; ++u)
                if (dist[u] < oo)
                    pot[u] += dist[u];

            T delta = numeric_limits<T>::max();
            for (edge *e = prev[t]; e != nullptr; e = prev[
                e->src])
                delta = min(delta, e->cap - e->flow
                    );
            flow += delta;
            for (edge *e = prev[t]; e != nullptr; e = prev[
                e->src])
            {
                e->flow += delta;
                adj[e->dst][e->rev].flow -= delta;
                cost += delta * e->cost;
            }
        }

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

4.24. MinCostFlow3.

```
/*
    Maximum flow of minimum cost with potentials
    Tested: ZOJ 3885
    Complexity: O(min(m^2 n log n, m log n flow))
*/

template<typename T, typename C = T>
struct min_cost_flow
{
    struct edge
    {
        int src, dst;
        T cap, flow;
        C cost;
        int rev;
    };

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

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

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

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

    vector<C> dist, pot;
    vector<edge*> prev;
    vector<T> curflow;

    void bellman_ford(int s, int t)
    {
        pot.assign(n, oo);
        pot[s] = 0;
```

```
		for (int it = 0, change = true; it < n && change; ++
			it)
		{
			change = false;
			for (int u = 0; u < n; ++u)
				if (pot[u] != oo)
				{
					for (auto &e : adj[u])
						if (e.flow < e.cap
							&& pot[e.dst]
								> pot[u]
								+ e.cost
							)
						{
							pot[e.dst] =
								pot[u] +
								e.cost;
							change =
								true;
						}
				}
		}
	}

	bool dijkstra(int s, int t)
	{
		dist.assign(n, oo);
		prev.assign(n, nullptr);
		dist[s] = 0;
		curflow[s] = numeric_limits<T>::max();

		using pci = pair<C, int>;
		priority_queue<pci, vector<pci>, greater<pci>> pq;
		pq.push({ 0, s });

		while (!pq.empty())
		{
			C d; int u;
			tie(d, u) = pq.top();
			pq.pop();

			if (d != dist[u])
				continue;

			for (auto &e : adj[u])
				if (e.flow < e.cap
				&& dist[e.dst] > dist[u] + e.cost +
					pot[u] - pot[e.dst])
				{
					dist[e.dst] = dist[u] + e.cost
							+ pot[u] -
								pot[e.dst
								];
					prev[e.dst] = &e;
					curflow[e.dst] = min(curflow[u
						], e.cap - e.flow);
					pq.push({ dist[e.dst], e.dst });
				}
		}

		return dist[t] < oo;
	}

	pair<T, C> max_flow(int s, int t)
	{
		T flow = 0;
		C cost = 0;

		// can be safely commented if
		// edges costs are non-negative
		bellman_ford(s, t);
		curflow.assign(n, 0);

		while (dijkstra(s, t))
		{
			for (int u = 0; u < n; ++u)
				if (dist[u] < oo)
```

```
					pot[u] += dist[u];

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

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

4.25. PushRelabel.

```
template<typename T>
struct push_relabel
{
	struct edge { int v, next; T w, f; };

	int n;
	queue<int> q;
	vector<bool> m;
	vector<T> e;
	vector<int> h, p, c;
	vector<edge> G;

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

	void add_edge(int u, int v, T w)
	{
		G.push_back({ v, p[u], w, 0 });
		p[u] = G.size() - 1;
		G.push_back({ u, p[v], 0, 0 });
		p[v] = G.size() - 1;
	}

	inline void enqueue(int u)
	{
		if (!m[u] && e[u])
		{
			m[u] = true;
			q.push(u);
		}
	}

	void push(int u, int i)
	{
		T mf = min(e[u], G[i].w - G[i].f);
		if (mf && h[u] > h[G[i].v])
		{
			G[i].f += mf;
			e[G[i].v] += mf;
			G[i ^ 1].f -= mf;
			e[u] -= mf;
			enqueue(G[i].v);
		}
	}

	void relabel(int u)
	{
		--c[h[u]];
		h[u] = 2 * n;
		for (int i = p[u]; i != -1; i = G[i].next)
			if (G[i].w - G[i].f && h[G[i].v] < h[u])
				h[u] = h[G[i].v];
		++c[++h[u]];
		enqueue(u);
```

```
    }

    T max_flow(int s, int t)
    {
        e.assign(n, 0);
        h.assign(n, 0);
        c.assign(2 * n, 0);
        m.assign(n, false);
        for (auto &e : G)
            e.f = 0;

        c[0] = n - 1;
        c[n] = 1;
        h[s] = n;
        m[s] = m[t] = true;
        for (int i = p[s]; i != -1; i = G[i].next)
        {
            e[s] += G[i].w;
            push(s, i);
        }

        for (int u; !q.empty(); q.pop())
        {
            u = q.front();
            m[u] = false;

            for (int i = p[u]; e[u] && i != -1; i = G[i].
                next)
                push(u, i);

            if (e[u])
            {
                if (c[h[u]] == 1)
                {
                    for (int i = 0; i < n; ++i)
                        if (h[i] >= h[u])
                        {
                            --c[h[i]];
                            h[i] = max(h[
                                i], n + 1)
                                ;
                            ++c[h[i]];
                            enqueue(i);
                        }
                }
                else
                    relabel(u);
            }
        }

        T flow = 0;
        for (int i = p[s]; i != -1; i = G[i].next)
            flow += G[i].f;
        return flow;
    }
};
```

4.26. StoerWagner.

```
/*
    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);
}
```

4.27. StronglyConnectedComponent.

```
struct strongly_connected_component
{
    int V, dt;
    vector<bool> del;
    vector<int> dfsnum, low, S;
    vector<vector<int> > G, SCC;

    strongly_connected_component(int n) :
            V(n), dt(0), del(n, 0), dfsnum(n, 0), low(n, 0),
                S(0), G(n, vector<int>(0)), SCC(0) {}

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

    void dfs(int u)
    {
        S.push_back(u);
        dfsnum[u] = low[u] = ++dt;

        for (auto v : G[u])
            low[u] = min(low[u], !dfsnum[v] ? dfs(v), low[v
                ] : !del[v] ? dfsnum[v] : low[u]);

        if (low[u] == dfsnum[u])
```

```
        {
            SCC.push_back(vector<int>(0));
            while (!del[u])
            {
                SCC.back().push_back(S.back());
                del[S.back()] = true;
                S.pop_back();
            }
        }
    }

    vector<vector<int> > solve()
    {
        for (int i = 0; i < V; ++i)
            if (!dfsnum[i]) dfs(i);
        return SCC;
    }
};
```

4.28. **TreeIsomorphism.**

```
/*
    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. Math

5.1. 2-SAT.

```
struct two_satisfability // 0-based
{
	int V, dt, k;
	vector<bool> del, usd;
	vector<int> dfsnum, low, scc, ord, S;
	vector<vector<int>> G, iG;

	two_satisfability(int n) :
		V(2 * n), dt(0), k(0), del(V, 0), usd(V, 0), dfsnum(V,
			0), low(V, 0), scc(V), ord(0), S(0), G(V, vector<
			int>(0)), iG(V, vector<int>(0)) {}

	void add_or(int u, int v)
	{
		G[u ^ 1].push_back(v), G[v ^ 1].push_back(u);
		iG[u].push_back(v ^ 1), iG[v].push_back(u ^ 1);
	}

	void top_sort(int u)
	{
		usd[u] = true;
		for (int i = 0, sz = G[u].size(); i < sz; i++)
			if (!usd[G[u][i]]) top_sort(G[u][i]);
		ord.push_back(u);
	}

	void dfs(int u)
	{
		S.push_back(u), dfsnum[u] = low[u] = ++dt;

		for (int v : iG[u])
			low[u] = min(low[u], !dfsnum[v] ? dfs(v), low[v
				] : !del[v] ? dfsnum[v] : low[u]);

		if (low[u] == dfsnum[u] && ++k)
			while (!del[u]) scc[S.back()] = k, del[S.back()]
				= true, S.pop_back();
	}

	bool solve(vector<bool> &out)
	{
		for (int i = 0; i < V; ++i)
			if (!usd[i]) top_sort(i);

		for (int i = V - 1; i >= 0; --i)
			if (!dfsnum[ord[i]]) dfs(ord[i]);

		for (int i = 0; i < V; i += 2)
		{
			if (scc[i] == scc[i ^ 1]) return false;
			out.push_back(scc[i] > scc[i ^ 1]);
		}

		return true;
	}
};
```

5.2. FastFourierTransform.

```
typedef complex<double> point;

void fft(point a[], size_t n, int sign)
{
	for (size_t i = 1, j = 0; i + 1 < n; ++i)
	{
		for (size_t k = n >> 1; (j ^= k) < k; k >>= 1);
		if (i < j) swap(a[i], a[j]);
```

```
    }
    const double theta = 2 * acos(−1) * sign;
    for (size_t m, mh = 1; (m = mh << 1) <= n; mh = m)
    {
        point wm = polar(1.0, theta / m);
        for (size_t i = 0; i < n; i += m)
        {
            point w(1.0);
            for (size_t j = i; j < i + mh; ++j)
            {
                point u = a[j], v = a[j + mh] * w;
                a[j] = u + v;
                a[j + mh] = u − v;
                w *= wm;
            }
        }
    }
    if (sign == −1)
        for (size_t i = 0; i < n; ++i) a[i] /= n;
}

vector<ll> convolve(const vector<ll> &a, const vector<ll> &b)
{
        int n = a.size() + b.size() − 1, size = 1;
        while (size < n) size *= 2;
        vector<point> pa(size), pb(size);
        for (int i = 0; i < a.size(); ++i) pa[i] = a[i];
        for (int i = 0; i < b.size(); ++i) pb[i] = b[i];
        fft(pa.data(), pa.size(), +1);
        fft(pb.data(), pb.size(), +1);
        for (int i = 0; i < size; ++i) pa[i] *= pb[i];
                fft(pa.data(), pa.size(), −1);
        vector<ll> ans(n);
        for (int i = 0; i < n; ++i) ans[i] = round(real(pa[i]));
        return ans;
}
```

5.3. FastModuloTransform.

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

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

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

typedef long long ll;

ll inv(ll b, ll M)
{
        ll u = 1, x = 0, s = b, t = M;
        while (s)
        {
                ll q = t / s;
                swap(x −= u * q, u);
                swap(t −= s * q, s);
        }
        return (x %= M) >= 0 ? x : x + M;
}

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

```
	return x;
}

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

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

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

const int WIDTH = 5;
```

```
const ll RADIX = 100000; // = 10^WIDTH

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

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

5.4. Gauss.

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

const int oo = 0x3f3f3f3f;
const double eps = 1e-9;

int gauss(vector<vector<double>> a, vector<double> &ans)
{
	int n = (int) a.size();
	int m = (int) a[0].size() - 1;

	vector<int> where(m, -1);
	for (int col = 0, row = 0; col < m && row < n; ++col)
	{
		int sel = row;
		for (int i = row; i < n; ++i)
			if (abs(a[i][col]) > abs(a[sel][col]))
				sel = i;
		if (abs(a[sel][col]) < eps)
			continue;
		for (int i = col; i <= m; ++i)
			swap(a[sel][i], a[row][i]);
		where[col] = row;

		for (int i = 0; i < n; ++i)
			if (i != row)
			{
				double c = a[i][col] / a[row][col];
				for (int j = col; j <= m; ++j)
					a[i][j] -= a[row][j] * c;
```

```
                }

            ++row;
        }

        ans.assign(m, 0);

        for (int i = 0; i < m; ++i)
            if (where[i] != -1)
                ans[i] = a[where[i]][m] / a[where[i]][i];

        for (int i = 0; i < n; ++i)
        {
            double sum = 0;
            for (int j = 0; j < m; ++j)
                sum += ans[j] * a[i][j];
            if (abs(sum - a[i][m]) > eps)
                return 0;
        }

        for (int i = 0; i < m; ++i)
            if (where[i] == -1)
                return oo;
        return 1;
    }
```

5.5. GoldsectionSearch.

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

    Tested: COJ 2890 :(
*/

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

5.6. Hungarian.

```
// max weight matching

template<typename T>
```

```
T hungarian(const vector<vector<T>> &cost)
{
	int n = cost.size(), p, q;
	vector<T> fx(n, numeric_limits<T>::min()), fy(n, 0);
	vector<int> x(n, -1), y(n, -1);
	for (int i = 0; i < n; ++i)
		for (int j = 0; j < n; ++j)
			fx[i] = max(fx[i], cost[i][j]);
	for (int i = 0; i < n;)
	{
		vector<int> t(n, -1), s(n + 1, i);
		for (p = q = 0; p <= q && x[i] < 0; ++p)
			for (int k = s[p], j = 0; j < n && x[i] < 0; ++
				j)
				if (fx[k] + fy[j] == cost[k][j] && t[j] <
					0)
				{
					s[++q] = y[j], t[j] = k;
					if (s[q] < 0)
						for (p = j; p >= 0; j
							= p)
							y[j] = k = t[j
								], p = x[
								k], x[k]
								= j;
				}
		if (x[i] < 0)
		{
			T d = numeric_limits<T>::max();
			for (int k = 0; k <= q; ++k)
				for (int j = 0; j < n; ++j)
					if (t[j] < 0)
						d = min(d, fx[s[k]] +
							fy[j] - cost[s[k]][j
							]);
			for (int j = 0; j < n; ++j)
				fy[j] += (t[j] < 0 ? 0 : d);
			for (int k = 0; k <= q; ++k)
				fx[s[k]] -= d;
		}
		else
			++i;
	}
	T ret = 0;
	for (int i = 0; i < n; ++i)
		ret += cost[i][x[i]];
	return ret;
}
```

5.7. Integrate.

```
//
// Numerical Integration (Adaptive Gauss--Lobatto formula)
//
// Description:
// Gauss--Lobatto formula is a numerical integrator
// that is exact for polynomials of degree <= 2n+1.
// Adaptive Gauss--Lobatto recursively decomposes the
// domain and computes integral by using G-L formula.
//
// Algorithm:
// Above.
//
// Complexity:
// O(#pieces) for a piecewise polynomials.
// In general, it converges in O(1/n^6) for smooth functions.
// For (possibly) non-smooth functions, this is the best integrator.
//
// Verified:
// AOJ 2034
//
```

```
// References:
// W. Gander and W. Gautschi (2000):
// Adaptive quadrature - revisited.
// BIT Numerical Mathematics, vol.40, no.1, pp.84--101.
//

template<class F>
double integrate(F f, double lo, double hi, double eps = 1e-9)
{
    const double th = eps / 1e-14; // (= eps / machine_epsilon)
    function<double(double, double, double, double, int)> rec =
    [&](double x0, double x6, double y0, double y6, int d)
    {
        const double a = sqrt(2.0/3.0), b = 1.0 / sqrt(5.0);
        double x3 = (x0 + x6)/2, y3 = f(x3), h = (x6 - x0)
            /2;
        double x1 = x3-a*h, x2 = x3-b*h, x4 = x3+b*h, x5
            = x3+a*h;
        double y1 = f(x1), y2 = f(x2), y4 = f(x4), y5 = f(x5);
        double I1 = (y0+y6 + 5*(y2+y4)) * (h/6);
        double I2 = (77*(y0+y6) + 432*(y1+y5) + 625*(y2+
            y4) + 672*y3) * (h/1470);
        if (x3 + h == x3 || d > 50) return 0.0;
        if (d > 4 && th + (I1-I2) == th) return I2; // avoid
            degeneracy
        return (double)(rec(x0, x1, y0, y1, d+1) + rec(x1, x2,
            y1, y2, d+1)
                    + rec(x2, x3, y2, y3, d+1) + rec(x3,
                        x4, y3, y4, d+1)
                    + rec(x4, x5, y4, y5, d+1) + rec(x5,
                        x6, y5, y6, d+1));
    };
    return rec(lo, hi, f(lo), f(hi), 0);
}
```

5.8. LinearRecursion.

```
/*
    Linear Recurrence Solver

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

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

typedef long long ll;

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

```
        }
        return c;
    };
    vector<ll> c = rec(k);
    ll ans = 0;
    for (int i = 0; i < x.size(); ++i)
        ans += c[i] * x[i];
    return ans;
}
```

5.9. MatrixComputationAlgorithms.

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

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

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

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

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

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

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

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

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

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

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

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

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

// TODO: verify
mat solve(mat A, mat B)
{
	// A^{-1} B
```

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

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

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

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

5.10. Minimumassignment(Jonker-Volgenant).

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

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

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

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

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

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

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

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

5.11. RootsNewton.

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

5.12. Simplex.

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

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

const double eps = 1e-9, oo = numeric_limits<double>::infinity();

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

double simplexMethodPD(mat &A, vec &b, vec &c)
{
```

```
int n = c.size(), m = b.size();
mat T(m + 1, vec(n + m + 1));
vector<int> base(n + m), row(m);
for(int j = 0; j < m; ++j)
{
        for (int i = 0; i < n; ++i)
                T[j][i] = A[j][i];
        T[j][n + j] = 1;
        base[row[j] = n + j] = 1;
        T[j][n + m] = b[j];
}
for (int i = 0; i < n; ++i)
        T[m][i] = c[i];

while (1)
{
        int p = 0, q = 0;
        for (int i = 0; i < n + m; ++i)
                if (T[m][i] <= T[m][p])
                        p = i;
        for (int j = 0; j < m; ++j)
                if (T[j][n + m] <= T[q][n + m])
                        q = j;
        double t = min(T[m][p], T[q][n + m]);

        if (t >= -eps)
        {
                vec x(n);
                for (int i = 0; i < m; ++i)
                        if (row[i] < n) x[row[i]] = T[i][n + m];
                // x is the solution
                return -T[m][n + m]; // optimal
        }

        if (t < T[q][n + m])
        {
                // tight on c -> primal update
                for (int j = 0; j < m; ++j)
                        if (T[j][p] >= eps)
                                if (T[j][p] * (T[q][n + m] - t)
                                                >= T[q][p] *
                                                   (T[j][n +
                                                    m] - t))
                                        q = j;
                if (T[q][p] <= eps)
                        return oo; // primal infeasible
        }
        else
        {
                // tight on b -> dual update
                for (int i = 0; i < n + m + 1; ++i)
                        T[q][i] = -T[q][i];
                for (int i = 0; i < n + m; ++i)
                        if (T[q][i] >= eps)
                                if (T[q][i] * (T[m][p] - t)
                                        >= T[q][p] * (T[m][i]
                                           - t))
                                        p = i;
                if (T[q][p] <= eps)
                        return -oo; // dual infeasible
        }
        for (int i = 0; i < m + n + 1; ++i)
                if (i != p)
                        T[q][i] /= T[q][p];
        T[q][p] = 1; // pivot(q, p)
        base[p] = 1;
        base[row[q]] = 0;
        row[q] = p;
        for (int j = 0; j < m + 1; ++j)
                if (j != q)
                {
                        double alpha = T[j][p];
                        for (int i = 0; i < n + m + 1; ++i)
                                T[j][i] -= T[q][i] * alpha;
                }
}
return oo;
}
```

```
/*
        Solve the linear programming problem (in integers)
        max: C * x
        sa: A * x <= B

        Answer is stored in best
        IMPORTANT: best must be initialized on INFINITY

        [UNTESTED YET]
*/
bool SolveInteger(mat &A, vec &B, vec &C, int64 &best, vector<int64
    > &solution)
{

    vec x;
    double v = simplexMethodPD(A, B, C, x);

    // Infeasible
    if (v == oo || v == -oo) return false;

    if ((int64)ceil(v) >= best) return true;

    for (int i = 0; i < (int)x.size(); ++i)
    {
        double a = floor(x[i]);
        double b = ceil(x[i]);

        if (min(x[i] - a, b - x[i]) >= eps)
        {
                mat NA = A;
                vec NB = B;
            int vars = C.size();
                vec nv(vars);
                nv[i] = -1;

                NA.push_back(nv);
                NB.push_back(-b);

                bool ok = SolveInteger(NA, NB, C, best, solution);

                NA.pop_back();
                NB.pop_back();

                nv[i] = 1;
                NA.push_back(nv);
                NB.push_back(a);
                ok |= SolveInteger(NA, NB, C, best, solution);

                return ok;
            }
        }

        // Solution is stored in x.
        // You may safely assume that it will be integer.

        int64 cur_value = (int64)round(v);

        if (cur_value < best){
            best = cur_value;
            solution = vector<int64>(x.begin(), x.end());
        }

        return true;
    }
```

5.13. **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;
}
```

5.14. StableMarriageProblem.

```
// the list of woman preference in descending order and the
    attractiveness for each man
vector<int> stable_marriage_problem(vector<vector<int>> &man,
    vector<vector<int>> &wom)
{
	const int n = man.size();
	vector<int> husb(wom.size(), -1), wife(n, -1), p(n, 0);

	for (int i = 0, m = 0, w, old; i < n; m = ++i)
		for (; m != -1; old = husb[w], wife[husb[w] = m] = w,
			m = old)
			do
			{
				w = man[m][p[m]++];
			} while (husb[w] != -1 && wom[w][m] < wom
				[w][husb[w]]);

	return wife;
}
```

6. Number Theory

6.1. BigInteger.

```
#define iszero(t) (t.len==1&&t.s[0]==0)
#define setlen(l,t) t.len=l;while(t.len>1&&t.s[t.len-1]==0) t.len--;
const int maxlen=100;
struct bigint
{
	int len, s[maxlen];
	bigint() { *this = 0; }
	bigint(int a) { *this = a; }
	bigint(const char *a) { *this = a; }
	bigint operator=(int);
	bigint operator=(const char*);
	bigint operator=(const bigint&); //Optional
	friend ostream& operator<<(ostream&, const bigint&);
	bigint operator+(const bigint&);
	bigint operator-(const bigint&);
	bigint operator*(const bigint&);
	bigint operator/(const bigint&); //Require - cmp
	bigint operator%(const bigint&); //Require - cmp
	static int cmp(const bigint&, const bigint&);
	static bigint sqrt(const bigint&); //Require - * cmp
};

bigint bigint::operator=(int a)
{
	len = 0;
	do
	{
		s[len++] = a % 10;
		a /= 10;
	} while (a > 0);
	return *this;
}

bigint bigint::operator=(const char *a)
{
	len = strlen(a);
	for (int i = 0; i < len; i++)
		s[i] = a[len - i - 1] - '0';
	return *this;
}

bigint bigint::operator=(const bigint &a)
{
	len = a.len;
	memcpy(s, a.s, sizeof(*s) * len);
	return *this;
}

ostream& operator<<(ostream &os, const bigint &a)
{
	for (int i = a.len - 1; i >= 0; i--)
		os << a.s[i];
	return os;
}

bigint bigint::operator+(const bigint &a)
{
	bigint b;
	b.s[b.len = max(len, a.len)] = 0;
	for (int i = 0; i < b.len; i++)
		b.s[i] = (i < len ? s[i] : 0) + (i < a.len ? a.s[i] : 0);
	for (int i = 0; i < b.len; i++)
		if (b.s[i] >= 10)
		{
			b.s[i] -= 10;
			b.s[i + 1]++;
		}
	if (b.s[b.len])
		b.len++;
	return b;
```

```
}

//Make sure *this>=a
bigint bigint::operator-(const bigint &a)
{
	bigint b;
	for (int i = 0; i < len; i++)
		b.s[i] = s[i] - (i < a.len ? a.s[i] : 0);
	for (int i = 0; i < len; i++)
		if (b.s[i] < 0)
		{
			b.s[i] += 10;
			b.s[i + 1]--;
		}
	setlen(len, b);
	return b;
}

bigint bigint::operator*(const bigint &a)
{
	bigint b;
	memset(b.s, 0, sizeof(*s) * (len + a.len + 1));
	for (int i = 0; i < len; i++)
		for (int j = 0; j < a.len; j++)
			b.s[i + j] += s[i] * a.s[j];
	for (int i = 0; i < len + a.len; i++)
	{
		b.s[i + 1] += b.s[i] / 10;
		b.s[i] %= 10;
	}
	setlen(len + a.len + 1, b);
	return b;
}

bigint bigint::operator/(const bigint &a)
{
	bigint b, c;
	for (int i = len - 1; i >= 0; i--)
	{
		if (!iszero(b))
		{
			for (int j = b.len; j > 0; j--)
				b.s[j] = b.s[j - 1];
			b.len++;
		}
		b.s[0] = s[i];
		c.s[i] = 0;
		while (cmp(b, a) >= 0)
		{
			b = b - a;
			c.s[i]++;
		}
	}
	setlen(len, c);
	return c;
}

bigint bigint::operator%(const bigint &a)
{
	bigint b;
	for (int i = len - 1; i >= 0; i--)
	{
		if (!iszero(b))
		{
			for (int j = b.len; j > 0; j--)
				b.s[j] = b.s[j - 1];
			b.len++;
		}
		b.s[0] = s[i];
		while (cmp(b, a) >= 0)
			b = b - a;
	}
	return b;
}

int bigint::cmp(const bigint &a, const bigint &b)
{
	if (a.len < b.len)
```

```
		return -1;
	else if (a.len > b.len)
		return 1;
	for (int i = a.len - 1; i >= 0; i--)
		if (a.s[i] != b.s[i])
			return a.s[i] - b.s[i];
	return 0;
}

bigint bigint::sqrt(const bigint &a)
{
	int n = (a.len - 1) / 2, p;
	bigint b, d;
	b.len = n + 1;
	for (int i = n; i >= 0; i--)
	{
		if (!iszero(d))
		{
			for (int j = d.len + 1; j > 1; j--)
				d.s[j] = d.s[j - 2];
			d.s[0] = a.s[i * 2];
			d.s[1] = a.s[i * 2 + 1];
			d.len += 2;
		}
		else
			d = a.s[i * 2] + (i * 2 + 1 < a.len ? a.s[i * 2 +
			    1] * 10 : 0);
		bigint c;
		c.s[1] = 0;
		for (int j = 1; j <= n - i; j++)
		{
			c.s[j] += b.s[i + j] << 1;
			if (c.s[j] >= 10)
			{
				c.s[j + 1] = 1;
				c.s[j] -= 10;
			}
			else
				c.s[j + 1] = 0;
		}
		c.len = n - i + 1 + c.s[n - i + 1];
		for (p = 1;; p++)
		{
			c.s[0] = p;
			if (cmp(d, c * p) < 0)
				break;
		}
		b.s[i] = c.s[0] = p - 1;
		d = d - c * (p - 1);
	}
	return b;
}
```

6.2. CarmichaelLambda.

```
/*
	Carmichael Lambda (Universal Totient Function)

	Description:
	lambda(n) is a smallest number that satisfies
	a^lambda(n) = 1 (mod n) for all integer a that is coprime
	    with n.
	This is also known as an universal totien function psi(n).
*/

typedef long long ll;

ll gcd(ll a, ll b)
{
	while (a) swap(a, b %= a);
	return b;
```

```
}

ll lcm(ll a, ll b)
{
    return a * (b / gcd(a, b));
}

ll carmichael_lambda(ll n)
{
    ll lambda = 1;
    if (n % 8 == 0)
        n /= 2;
    for (ll d = 2; d * d <= n; ++d)
        if (n % d == 0)
        {
            n /= d;
            ll y = d - 1;
            while (n % d == 0)
            {
                n /= d;
                y *= d;
            }
            lambda = lcm(lambda, y);
        }
    if (n > 1)
        lambda = lcm(lambda, n - 1);
    return lambda;
}

// lambda(n) for all n in [lo, hi)
vector<ll> carmichael_lambda(ll lo, ll hi)
{
    vector<ll> ps = primes(sqrt(hi) + 1);
    vector<ll> res(hi - lo), lambda(hi - lo, 1);
    iota(res.begin(), res.end(), lo);
    for (ll k = ((lo + 7) / 8) * 8; k < hi; k += 8)
        res[k - lo] /= 2;
    for (ll p : ps)
        for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p)
        {
            if (res[k - lo] < p)
                continue;
            ll t = p - 1;
            res[k - lo] /= p;
            while (res[k - lo] > 1 && res[k - lo] % p ==
                0)
            {
                t *= p;
                res[k - lo] /= p;
            }
            lambda[k - lo] = lcm(lambda[k - lo], t);
        }
    for (ll k = lo; k < hi; ++k)
        if (res[k - lo] > 1)
            lambda[k - lo] = lcm(lambda[k - lo], res[k -
                lo] - 1);
    return lambda; // lambda[k-lo] = lambda(k)
}
```

6.3. **ChineseRemainderTheorem.**

```
// return min x such that x % m[i] == a[i]
int chinese_remainder_theorem(vector<int> a, vector<int> m)
{
    int n = a.size(), s = 1, t, ans = 0, p, q;
    for (auto i : m) s *= i;
    for (int i = 0; i < n; i++)
    {
        t = s / m[i];
        extended_euclid(t, m[i], p, q);
        ans = (ans + t * p * a[i]) % s;
    }
    if (ans < 0) ans += s;
```

```
	return ans;
}

/*
	Solve 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};
}
```

6.4. DiscreteLogarithm.

```
/*
	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;
}
```

6.5. DiscreteRoots.

```
/*
	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;
}
```

6.6. DivisorSigma.

```
typedef long long ll;

// count the number of divisors of n
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)
}
```

6.7. EulerPhi.

```
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)
}
```

6.8. ExtendedEuclidAndDiophantineEquation.

```
// returns (d, x, y) such that d = gcd(a, b) = ax + by
int extended_euclid(int a, int b, int &x, int &y)
{
    if (b == 0) { x = 1, y = 0; return a; }
    int r = extended_euclid(b, a % b, y, x);
    y -= a / b * x;
    return r;
}

// returns (x, y) such that c = ax + by
pair<int, int> diophantine_equation(int a, int b, int c)
{
    int g, x, y;
    g = extended_euclid(a, b, x, y);
    int k = 0; // k e Z
    return { x * c / g + (k * b / g), y * c / g - (k * a / g) };
}
```

6.9. MillerRabin.

```
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;
}

// return n is possible prime
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;
}
```

6.10. MobiusMu.

```
/*
For any positive integer n, define (n) as the sum of the primitive n-th
    roots of unity
    (n) = 1 if n is a square-free positive integer with an even number
        of prime factors.
    (n) = 1 if n is a square-free positive integer with an odd number
        of prime factors.
    (n) = 0 if n has a squared prime factor.
*/

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)
}
```

6.11. ModFact.

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

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

6.12. ModularArithmetics.

```
/*
	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;
}
```

6.13. PollardRho.

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

```
            if (i == k)
            {
                y = x;
                k <<= 1;
            }
        }
    }
    return 0;
}
```

6.14. **PrimitiveRoot.**

```
/*
    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;
}
```

6.15. Sieve.

```
for (int i = 3; i * i <= n; i += 2)
    if (!p[i])
        for (int j = i * i; j <= n; j += 2 * i)
            p[j] = i;

vector<int> line_of_assambly(int N) // generate primes numbers O(
    N log N), sqrt(N) in memory
{
    vector<int> p(1, 2);
    set<pair<int, int>> S;
    S.insert( { 4, 2 });

    for (int i = 3; i < N; ++i)
    {
        auto it = S.lower_bound({ i, 0 });
        if (it->first > i)
        {
            p.push_back(i);
            if (i * i < N)
                S.insert( { i * i, 2LL * i });
        }
        else
            do
            {
                S.insert({ it->first + it->second, it->second });
                S.erase(it);
                it = S.lower_bound( { i, 0 });
            } while (it->first == i);
    }

    return p;
}
```

```
// primes in [lo, hi)
vector<ll> primes(ll lo, ll hi)
{
        const ll M = 1 << 14, SQR = 1 << 16;
        vector<bool> composite(M), small_composite(SQR);
        vector<pair<ll, ll>> sieve;
        for (ll i = 3; i < SQR; i += 2)
                if (!small_composite[i])
                {
                        ll k = i * i + 2 * i * max(0.0, ceil((lo - i*i)
                            /(2.0*i)));
                        sieve.push_back({ 2 * i, k });
                        for (ll j = i * i; j < SQR; j += 2 * i)
                                small_composite[j] = 1;
                }
        vector<ll> ps;
        if (lo <= 2)
        {
                ps.push_back(2);
                lo = 3;
        }
        for (ll k = lo | 1, low = lo; low < hi; low += M)
        {
                ll high = min(low + M, hi);
                fill(composite.begin(), composite.end(), 0);
                for (auto &z : sieve)
                        for (; z.second < high; z.second += z.first)
                                composite[z.second - low] = 1;
                for (; k < high; k += 2)
                        if (!composite[k - low])
                                ps.push_back(k);
        }
        return ps;
```

```
}

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

7. String

7.1. AhoCorasik.

```
struct aho_corasick
{
	static const int alpha = 26;

	vector<array<int, alpha>> go;
	vector<int> fail;
	vector<int> endpos;

	aho_corasick() { add_node(); }

	int add_string(const string &str)
	{
		int e = 0;
		for (char c : str)
		{
			if (!go[e][c-'a'])
			{
				int nn = add_node();
				go[e][c-'a'] = nn;
			}
			e = go[e][c-'a'];
		}
		++endpos[e];
		return e;
	}

	void build()
	{
		queue<int> que;
		for (int c = 0; c < alpha; ++c)
			if (go[0][c]) que.push(go[0][c]);
		for (; !que.empty(); que.pop())
		{
			int e = que.front();
			int f = fail[e];
			for (int c = 0; c < alpha; ++c)
				if (!go[e][c]) go[e][c] = go[f][c];
				else
				{
					fail[go[e][c]] = go[f][c];
					endpos[go[e][c]] += endpos[go
						[f][c]];
					que.push(go[e][c]);
				}
		}
	}

private:
	int add_node()
	{
		int pos = go.size();
		go.emplace_back(array<int, alpha>());
		fail.emplace_back(0);
		endpos.emplace_back(0);
		return pos;
	}
};
```

7.2. Hash.

```
struct Hash
{
	typedef unsigned long long ull;

	int n;
	vector<ull> h, p;
```

```
    Hash(const string &s, ull base = 31) : n(s.length()), h(n + 1),
        p(n + 1, 1)
    {
        for (int i = 1; i <= n; ++i)
            h[i] = h[i - 1] + (s[i - 1] - 'a' + 1) * (p[i] =
                p[i - 1] * base);
    }

    bool equal(int a, int b, int k)
    {
        return (h[a + k] - h[a]) * p[b - a] == h[b + k] - h[b
            ];
    }
};
```

7.3. Manacher.

```
// longest palindrome centered in positon p with odd and even length
    -> rad[2*p], rad[2*p+1]
vector<int> manacher(const string &s)
{
    int n = 2 * s.length();
    vector<int> rad(n);

    for (int i = 0, j = 0, k; i < n; i += k, j = max(j - k, 0))
    {
        for (; i >= j && i + j + 1 < n
            && s[(i - j) / 2] == s[(i + j + 1) / 2]; ++j);
        rad[i] = j;
        for (k = 1; i >= k &&
                rad[i] >= k && rad[i - k] != rad[i] - k; ++k)
            rad[i + k] = min(rad[i - k], rad[i] - k);
    }

    return rad;
}

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;
}
```

7.4. MaximalSuffix.

```
// position of maximun lexicographical suffix
int maximal_suffix(const string &s)
{
    int n = s.length(), i = 0, j = 1;

    for (int k = 0; j < n - 1; k = 0)
    {
        while (j + k < n - 1 && s[i + k] == s[j + k]) ++k;

        if (s[i + k] < s[j + k])
        {
            i += (k / (j - i) + 1) * (j - i);
            j = i +1;
        }
        else j += k + 1;
    }

    return i;
}
```

7.5. MinimumRotation.

```
// minimun lexicographical rotation
int minimum_rotation(const string &s)
{
	int n = s.length(), i = 0, j = 1, k = 0;

	while (i + k < 2 * n && j + k < 2 * n)
	{
		char a = i + k < n ? s[i + k] : s[i + k - n];
		char b = j + k < n ? s[j + k] : s[j + k - n];

		if (a > b)
		{
			i += k + 1;
			k = 0;
			if (i <= j)
				i = j + 1;
		}
		else if (a < b)
		{
			j += k + 1;
			k = 0;
			if (j <= i)
				j = i + 1;
		}
		else ++k;
	}

	return min(i, j);
}
```

7.6. PalindromicTree.

```
/*
	at each step when add a character:
	suf->len is the length of longest suffix palindrome
	suf->suf is the next node with longest suffix palindrome

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

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

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

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

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

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

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

7.7. PiFunction.

```
// longest length that the preffix ending in i is equal to suffix start in
	0
vector<int> pi_function(const string &s)
{
	int n = s.length();
	vector<int> pi(n);

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

	return pi;
}

// matchs of pattern in text
vector<int> kmp(const string &text, const string &patt)
{
	int n = text.length(), m = patt.length();
	vector<int> ans, pi = pi_function(patt);

	for (int i = 0, k = 0; i < n; ++i)
	{
		while (k && text[i] != patt[k])
			k = pi[k - 1];
		k += text[i] == patt[k];
		if (k == m)
		{
			ans.push_back(i - k + 1);
			k = pi[k - 1];
		}
	}

	return ans;
}
```

```
// minimum length l that s can be represented as a concatenation of
    copies l
int compression_line(const string &s)
{
        int l = pi_function(s).last();
        return s.length() % l == 0 ? l : s.length();
}

// counting the number of occurrences of each prefix
vector<int> prefix_occurrences(const string &s)
{
        int n = s.length();
        vector<int> ans(n + 1), pi = pi_function(s);

        for (int p : pi)
                ++ans[p];
        for (int i = n - 1; i; --i)
                ans[pi[i - 1]] += ans[i];

        ans.erase(ans.begin());
        return ans;
}
```

7.8. **SuffixArray.**

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

        suffix_array(const string &s) : n(s.length() + 1), sa(n), lcp(n),
                rank(n)
        {
                vector<int> top(max(256, n));
                for (int i = 0; i < n; ++i)
                        top[rank[i] = s[i] & 0xff]++;

                partial_sum(top.begin(), top.end(), top.begin());
                for (int i = 0; i < n; ++i)
                        sa[--top[rank[i]]] = i;

                vector<int> tmp(n);
                for (int len = 1, j; len < n; len <<= 1)
                {
                        for (int i = 0; i < n; ++i)
                        {
                                j = sa[i] - len;
                                if (j < 0) j += n;
                                tmp[top[rank[j]]++] = j;
                        }

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

                        copy(sa.begin(), sa.end(), rank.begin());
                        copy(tmp.begin(), tmp.end(), sa.begin());
                        if (j >= n - 1) break;
                }

                int i, j, k;
                for (j = rank[lcp[i = k = 0] = 0]; i < n - 1; ++i, ++k
                        )
                {
```

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

7.9. SuffixAutomaton.

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

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

    vector<state*> states;
    state *last;

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

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

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

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

        last = nlast;
    }

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

7.10. ZAlgorithm.

```
// z[i] = length of the longest common prefix of s and s[i..n]
vector<int> zfunction(const string &s)
{
	int n = s.length();
	vector<int> z(n, n);

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

	return z;
}
```