Team notebook

November 14, 2019

Contents

1 BlackBox

1.1 AnikDaHalfPlane

```
///code from Anikda

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

const double eps = 1e-6;

inline int dcmp (double x) { if (fabs(x) < eps) return 0; else
    return x < 0 ? -1 : 1; }

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

    Point operator + (const Point& u) { return Point(x + u.x, y
        + u.y); }
    Point operator - (const Point& u) { return Point(x - u.x, y
        - u.y); }
    Point operator * (const double u) { return Point(x * u, y *
        u); }
    Point operator / (const double u) { return Point(x / u, y /
        u); }
    double operator * (const Point& u) { return x*u.y - y*u.x; }
};
typedef Point Vector;

/// p, v describes the line
/// ang is the slope angle of the line
struct DirLine {
    Point p;
    Vector v;
    double ang;
    DirLine () {}
    DirLine (Point p, Vector v): p(p), v(v) { ang = atan2(v.y,
        v.x); }
    bool operator < (const DirLine& u) const { return ang <
        u.ang; }
};

double getCross (Vector a, Vector b) { return a.x * b.y - a.y *
    b.x; }
```

```
bool getIntersection (Point p, Vector v, Point q, Vector w,
    Point& o) {
    if (dcmp(getCross(v, w)) == 0) return false;
    Vector u = p - q;
    double k = getCross(w, u) / getCross(v, w);
    o = p + v * k;
    return true;
}

bool onLeft(DirLine l, Point p) { return dcmp(l.v * (p-l.p)) >=
    0; }

int halfPlaneIntersection(DirLine* li, int n, Point* poly) {
    sort(li, li + n);

    int first, last;
    Point* p = new Point[n];
    DirLine* q = new DirLine[n];
    q[first=last=0] = li[0];

    for (int i = 1; i < n; i++) {
        while (first < last && !onLeft(li[i], p[last-1])) last--;
        while (first < last && !onLeft(li[i], p[first])) first++;
        q[++last] = li[i];

        if (dcmp(q[last].v * q[last-1].v) == 0) {
            last--;
            if (onLeft(q[last], li[i].p)) q[last] = li[i];
        }

        if (first < last)
            getIntersection(q[last-1].p, q[last-1].v, q[last].p,
                q[last].v, p[last-1]);
    }
    while (first < last && !onLeft(q[first], p[last-1])) last--;
    if (last - first <= 1) { delete [] p; delete [] q; return 0;
        }
    getIntersection(q[last].p, q[last].v, q[first].p,
        q[first].v, p[last]);

    int m = 0;
    for (int i = first; i <= last; i++) poly[m++] = p[i];
    delete [] p; delete [] q;
    return m;
}
```

1.2 KDTree

```
//
    --------------------------------------------------------------------
// A straightforward, but probably sub-optimal KD-tree
    implmentation
// that's probably good enough for most things (current it's a
// 2D-tree)
//
//  - constructs from n points in O(n lg^2 n) time
//  - handles nearest-neighbor query in O(lg n) if points are
    well
//    distributed
//  - worst case for nearest-neighbor may be linear in
    pathological
//    case
//
// Sonny Chan, Stanford University, April 2009
//
    --------------------------------------------------------------------

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

```
using namespace std;

// number type for coordinates, and its maximum value
typedef long long ntype;
const ntype sentry = numeric_limits<ntype>::max();

// point structure for 2D-tree, can be extended to 3D
struct point {
    ntype x, y;
    point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}
};

bool operator==(const point &a, const point &b)
{
    return a.x == b.x && a.y == b.y;
}

// sorts points on x-coordinate
bool on_x(const point &a, const point &b)
{
    return a.x < b.x;
}

// sorts points on y-coordinate
bool on_y(const point &a, const point &b)
{
    return a.y < b.y;
}

// squared distance between points
ntype pdist2(const point &a, const point &b)
{
    ntype dx = a.x-b.x, dy = a.y-b.y;
    return dx*dx + dy*dy;
}
```

```
// bounding box for a set of points
struct bbox
{
    ntype x0, x1, y0, y1;

    bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry) {}

    // computes bounding box from a bunch of points
    void compute(const vector<point> &v) {
        for (int i = 0; i < v.size(); ++i) {
            x0 = min(x0, v[i].x); x1 = max(x1, v[i].x);
            y0 = min(y0, v[i].y); y1 = max(y1, v[i].y);
        }
    }

    // squared distance between a point and this bbox, 0 if
       inside
    ntype distance(const point &p) {
        if (p.x < x0) {
            if (p.y < y0)     return pdist2(point(x0, y0), p);
            else if (p.y > y1) return pdist2(point(x0, y1), p);
            else              return pdist2(point(x0, p.y), p);
        }
        else if (p.x > x1) {
            if (p.y < y0)     return pdist2(point(x1, y0), p);
            else if (p.y > y1) return pdist2(point(x1, y1), p);
            else              return pdist2(point(x1, p.y), p);
        }
        else {
            if (p.y < y0)     return pdist2(point(p.x, y0), p);
            else if (p.y > y1) return pdist2(point(p.x, y1), p);
            else              return 0;
        }
    }
```

```
};

// stores a single node of the kd-tree, either internal or leaf
struct kdnode
{
    bool leaf;     // true if this is a leaf node (has one point)
    point pt;      // the single point of this is a leaf
    bbox bound;    // bounding box for set of points in children

    kdnode *first, *second; // two children of this kd-node

    kdnode() : leaf(false), first(0), second(0) {}
    ~kdnode() { if (first) delete first; if (second) delete
        second; }

    // intersect a point with this node (returns squared
        distance)
    ntype intersect(const point &p) {
        return bound.distance(p);
    }

    // recursively builds a kd-tree from a given cloud of points
    void construct(vector<point> &vp)
    {
        // compute bounding box for points at this node
        bound.compute(vp);

        // if we're down to one point, then we're a leaf node
        if (vp.size() == 1) {
            leaf = true;
            pt = vp[0];
        }
        else {
            // split on x if the bbox is wider than high (not
                best heuristic...)
            if (bound.x1-bound.x0 >= bound.y1-bound.y0)
                sort(vp.begin(), vp.end(), on_x);
            // otherwise split on y-coordinate
            else
                sort(vp.begin(), vp.end(), on_y);

            // divide by taking half the array for each child
            // (not best performance if many duplicates in the
                middle)
            int half = vp.size()/2;
            vector<point> vl(vp.begin(), vp.begin()+half);
            vector<point> vr(vp.begin()+half, vp.end());
            first = new kdnode(); first->construct(vl);
            second = new kdnode(); second->construct(vr);
        }
    }
};

// simple kd-tree class to hold the tree and handle queries
struct kdtree
{
    kdnode *root;

    // constructs a kd-tree from a points (copied here, as it
        sorts them)
    kdtree(const vector<point> &vp) {
        vector<point> v(vp.begin(), vp.end());
        root = new kdnode();
        root->construct(v);
    }
    ~kdtree() { delete root; }

    // recursive search method returns squared distance to
        nearest point
    ntype search(kdnode *node, const point &p)
```

```
    {
        if (node->leaf) {
            // commented special case tells a point not to find
                itself
//          if (p == node->pt) return sentry;
//          else
              return pdist2(p, node->pt);
        }

        ntype bfirst = node->first->intersect(p);
        ntype bsecond = node->second->intersect(p);

        // choose the side with the closest bounding box to
            search first
        // (note that the other side is also searched if needed)
        if (bfirst < bsecond) {
            ntype best = search(node->first, p);
            if (bsecond < best)
                best = min(best, search(node->second, p));
            return best;
        }
        else {
            ntype best = search(node->second, p);
            if (bfirst < best)
                best = min(best, search(node->first, p));
            return best;
        }
    }

    // squared distance to the nearest
    ntype nearest(const point &p) {
        return search(root, p);
    }
};
```

```
int main()
{
    // generate some random points for a kd-tree
    vector<point> vp;
    for (int i = 0; i < 100000; ++i) {
        vp.push_back(point(rand()%100000, rand()%100000));
    }
    kdtree tree(vp);

    // query some points
    for (int i = 0; i < 10; ++i) {
        point q(rand()%100000, rand()%100000);
        cout << "Closest squared distance to (" << q.x << ", " <<
            q.y << ")"
             << " is " << tree.nearest(q) << endl;
    }

    return 0;
}
```

1.3 LiChao*parabolic*

```
/*
    This implementation works only if every pair of parabolas
    has at most one intersection. For example, y = (x-h)^2+c
*/

#include <bits/stdc++.h>
#define endl '\n'
using namespace std;
const int inf = (int)1e9 + 42;
const int MAXX = 1000;

template<class T>
```

```
struct LiChao_parabolic_min {
    struct Parabol {
        T a, b, c;
        Parabol() {a = 0; b = 0; c = inf;}
        Parabol(T a, T b, T c) : a(a), b(b), c(c) {}
        Parabol(const Parabol& rhs) {
            a = rhs.a;
            b = rhs.b;
            c = rhs.c;
        }
        T query(T x) {
            return a * x * x + b * x + c;
        }
    };
    struct Node {
        Parabol prb;
        Node *l, *r;
        Node() : l(0), r(0) {}
        T query(T x) {
            return prb.query(x);
        }
    };

    Node* rt;
    T offset;
    vector<Parabol> mem;

    LiChao_parabolic_min() : rt(0), offset(0) {}

    Node* upd(Node* p, int l, int r, int L, int R, Parabol prb) {
        int M = (L + R) >> 1;
        if (l > R || r < L) return p;
        if (!p) p = new Node();
        if (l <= L && r >= R) {
            if (prb.query(L) >= p->query(L) && prb.query(R) >=
                p->query(R)) {
                return p;
            }
            if (prb.query(L) <= p->query(L) && prb.query(R) <=
                p->query(R)) {
                p->prb = prb;
                return p;
            }
            if (prb.query(L) >= p->query(L) && prb.query(M) >=
                p->query(M)) {
                p->r = upd(p->r, l, r, M + 1, R, prb);
                return p;
            }
            if (prb.query(L) <= p->query(L) && prb.query(M) <=
                p->query(M)) {
                p->r = upd(p->r, l, r, M + 1, R, p->prb);
                p->prb = prb;
                return p;
            }
            if (prb.query(M + 1) >= p->query(M + 1) &&
                prb.query(R) >= p->query(R)) {
                p->l = upd(p->l, l, r, L, M, prb);
                return p;
            }
            if (prb.query(M + 1) <= p->query(M + 1) &&
                prb.query(R) <= p->query(R)) {
                p->l = upd(p->l, l, r, L, M, p->prb);
                p->prb = prb;
                return p;
            }
            return p;
        }
        else if (L < R) {
            p->l = upd(p->l, l, r, L, (L + R) >> 1, prb);
```

```
            p->r = upd(p->r, l, r, ((L + R) >> 1) + 1, R, prb);
        }
    }
    T query(Node* p, int i, int L, int R) {
        if (!p) return inf;
        if (i < L || i > R) return inf;
        T res = inf;
        res = min(res, p->query(i));
        if (L < R) {
            res = min(res, query(p->l, i, L, (L + R) >> 1));
            res = min(res, query(p->r, i, ((L + R) >> 1) + 1, R));
        }
        return res;
    }
    void upd(T a, T b, T c) {
        mem.push_back(Parabol(a, b, c));
        rt = upd(rt, 0, MAXX, 0, MAXX, Parabol(a, b, c));
    }
    T query(int i) {
        return query(rt, i, 0, MAXX);
    }

    void okay(Node* rt) {
        if (rt) {
            okay(rt->l);
            okay(rt->r);
            delete rt;
        }
    }

    ~LiChao_parabolic_min() {
        okay(rt);
    }
};
```

```
void read()
{

}

void solve()
{

}

int main()
{
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);

        read();
        solve();
        return 0;
}
```

1.4 LinkCutTree

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

/**
 * Author: Simon Lindholm
 * Date: 2016-07-25
 * Source:
    https://github.com/ngthanhtrung23/ACM_Notebook_new/blob/master/Dat
 * Description: Represents a forest of unrooted trees. You can
    add and remove
```

```
 * edges (as long as the result is still a forest), and check
    whether
 * two nodes are in the same tree.
 * Time: All operations take amortized O(\log N).
 * Status: Fuzz-tested a bit for N <= 20
 * 1-indexed
 */

struct Node { // Splay tree. Root's pp contains tree's parent.
	Node *p = 0, *pp = 0, *c[2];
	bool flip = 0;
	Node() { c[0] = c[1] = 0; fix(); }
	void fix() {
		if (c[0]) c[0]->p = this;
		if (c[1]) c[1]->p = this;
		// (+ update sum of subtree elements etc. if
		   wanted)
	}
	void push_flip() {
		if (!flip) return;
		flip = 0; swap(c[0], c[1]);
		if (c[0]) c[0]->flip ^= 1;
		if (c[1]) c[1]->flip ^= 1;
	}
	int up() { return p ? p->c[1] == this : -1; }
	void rot(int i, int b) {
		int h = i ^ b;
		Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b
		   ? y : x;
		if ((y->p = p)) p->c[up()] = y;
		c[i] = z->c[i ^ 1];
		if (b < 2) {
			x->c[h] = y->c[h ^ 1];
			z->c[h ^ 1] = b ? x : this;
		}
		y->c[i ^ 1] = b ? this : x;
		fix(); x->fix(); y->fix();
		if (p) p->fix();
		swap(pp, y->pp);
	}
	void splay() { /// Splay this up to the root. Always
	   finishes without flip set.
		for (push_flip(); p; ) {
			if (p->p) p->p->push_flip();
			p->push_flip(); push_flip();
			int c1 = up(), c2 = p->up();
			if (c2 == -1) p->rot(c1, 2);
			else p->p->rot(c2, c1 != c2);
		}
	}
	Node* first() { /// Return the min element of the subtree
	   rooted at this, splayed to the top.
		push_flip();
		return c[0] ? c[0]->first() : (splay(), this);
	}
};

struct LinkCut {
	vector<Node> node;
	LinkCut(int N) : node(N) {}

	void link(int u, int v) { // add an edge (u, v)
		assert(!connected(u, v));
		make_root(&node[u]);
		node[u].pp = &node[v];
	}
	void cut(int u, int v) { // remove an edge (u, v)
		Node *x = &node[u], *top = &node[v];
		make_root(top); x->splay();
		assert(top == (x->pp ?: x->c[0]));
```

```
        if (x->pp) x->pp = 0;
        else {
                x->c[0] = top->p = 0;
                x->fix();
        }
}
bool connected(int u, int v) { // are u, v in the same
    tree?
        Node* nu = access(&node[u])->first();
        return nu == access(&node[v])->first();
}
void make_root(Node* u) { /// Move u to root of
    represented tree.
        access(u);
        u->splay();
        if(u->c[0]) {
                u->c[0]->p = 0;
                u->c[0]->flip ^= 1;
                u->c[0]->pp = u;
                u->c[0] = 0;
                u->fix();
        }
}
Node* access(Node* u) { /// Move u to root aux tree.
    Return the root of the root aux tree.
        u->splay();
        while (Node* pp = u->pp) {
                pp->splay(); u->pp = 0;
                if (pp->c[1]) {
                        pp->c[1]->p = 0; pp->c[1]->pp = pp;
                           }
                pp->c[1] = u; pp->fix(); u = pp;
        }
        return u;
}
```

```
};

///Solves SPOJ DYNACON1 - Dynamic Tree Connectivity
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    cin>>n>>m;
    LinkCut tree(n+1);

    while (m--) {
        string s;
        int u, v;
        cin>>s>>u>>v;

        if (s=="conn") cout<< ( tree.connected(u, v) ? "YES" :
            "NO")<<endl;
        else if (s=="add") tree.link(u, v);
        else               tree.cut(u, v);

    }
}
```

1.5 Polynomial

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

typedef long long LL;
const LL MOD = 1000000007;

inline LL Power(LL b, LL p) {
    LL ret = 1;
```

```
    while(p--) {
        ret *= b;
        ret %= MOD;
    }
    return ret;
}

inline vector <int> getSequence(LL n, LL c, LL k) {
    vector <int> sequence;
    LL ans=0, fib[n*c+3];
    fib[0] = 0;
    fib[1] = 1;
    for(LL i=2; i<=n*c; i++) {
        fib[i] = fib[i-1] + fib[i-2];
        if(fib[i] >= MOD) fib[i] -= MOD;
    }
    for(LL i=0; i<=n; i++) {
        ans += Power(fib[i*c], k);
        if(ans >= MOD) ans -= MOD;
        sequence.push_back((int)ans);
    }
    return sequence;
}

#define MAX 65536
#define MOD 1000000007 /// MOD must be prime and greater than 2
#define MOD_THRESHOLD 18 /// MOD * MOD * MOD_THRESHOLD < 2^64
#define clr(ar) memset(ar, 0, sizeof(ar))

namespace pol{ /// Polynomial namespace
    unsigned long long temp[128];
    int ptr = 0, A[MAX * 2], B[MAX * 2], buffer[MAX * 6];

    /// Use faster multiplication techniques if required
```

```
    void karatsuba(int n, int *a, int *b, int *res){ /// hash =
        829512
        int i, j, h;
        if (n < 17){
            for (i = 0; i < (n + n); i++) temp[i] = 0;
            for (i = 0; i < n; i++){
                if (a[i]){
                    for (j = 0; j < n; j++){
                        temp[i + j] += ((long long)a[i] * b[j]);
                    }
                }
            }
            for (i = 0; i < (n + n); i++) res[i] = temp[i] % MOD;
            return;
        }

        h = n >> 1;
        karatsuba(h, a, b, res);
        karatsuba(h, a + h, b + h, res + n);
        int *x = buffer + ptr, *y = buffer + ptr + h, *z = buffer
            + ptr + h + h;

        ptr += (h + h + n);
        for (i = 0; i < h; i++){
            x[i] = a[i] + a[i + h], y[i] = b[i] + b[i + h];
            if (x[i] >= MOD) x[i] -= MOD;
            if (y[i] >= MOD) y[i] -= MOD;
        }

        karatsuba(h, x, y, z);
        for (i = 0; i < n; i++) z[i] -= (res[i] + res[i + n]);
        for (i = 0; i < n; i++){
            res[i + h] = (res[i + h] + z[i]) % MOD;
            if (res[i + h] < 0) res[i + h] += MOD;
        }
```

```
    ptr -= (h + h + n);
}

int mul(int n, int *a, int m, int *b){ /// hash = 903808
    int i, r, c = (n < m ? n : m), d = (n > m ? n : m), *res
        = buffer + ptr;
    r = 1 << (32 - __builtin_clz(d) - (__builtin_popcount(d)
        == 1));
    for (i = d; i < r; i++) a[i] = b[i] = 0;
    for (i = c; i < d && n < m; i++) a[i] = 0;
    for (i = c; i < d && m < n; i++) b[i] = 0;

    ptr += (r << 1), karatsuba(r, a, b, res), ptr -= (r << 1);
    for (i = 0; i < (r << 1); i++) a[i] = res[i];
    return (n + m - 1);
}

inline void copy(int* to, const int* from, int n){
    memcpy(to, from, n * sizeof(int));
}

inline void add(int* res, const int* P, int pn, const int*
    Q, int qn){
    for (int i = 0; i < qn; i++){
        res[i] = P[i] + Q[i];
        if (res[i] >= MOD) res -= MOD;
    }
    copy(res + qn, P + qn, pn - qn);
}

inline void subtract(int* res, const int* P, int pn, const
    int* Q, int qn){
    for(int i = 0; i < qn; ++ i){
        res[i] = P[i] - Q[i];
        if (res[i] < 0) res[i] += MOD;
    }
    copy(res + qn, P + qn, pn - qn);
}

inline void shift(int* res, const int* P, int n, int k){
    for (int i = 0; i < n; i++) res[i] = ((long long)P[i] <<
        k) % MOD;
}

inline void reverse_poly(int* res, const int* P, int n){
    if (&res[0] == &P[0]) reverse(res, res + n);
    else{
        for (int i = 0; i < n; i++) res[n - i - 1] = P[i];
    }
}

struct polynomial{
    vector <int> coefficient;

    polynomial(){}
    polynomial(int c0) : coefficient(1, c0){}
    polynomial(int c0, int c1) : coefficient(2)
        {coefficient[0] = c0, coefficient[1] = c1;}

    inline void resize(int n){
        coefficient.resize(n);
    }

    inline int* data(){
        return coefficient.empty() ? 0 : &coefficient[0];
    }

    inline const int* data() const{
        return coefficient.empty() ? 0 : &coefficient[0];
    }
```

```
inline int size() const{
    return coefficient.size();
}

inline void set(int i, int x){
    if (size() <= i) resize(i + 1);
    coefficient[i] = x;
}

inline void normalize(){
    while (size() && coefficient.back() == 0)
        coefficient.pop_back();
}

static void multiply(polynomial &res, const polynomial&
    p, const polynomial& q){ /// hash = 569213
    if(&res == &p || &res == &q){
        polynomial temp;
        multiply(temp, p, q);
        res = temp;
        return;
    }

    res.coefficient.clear();
    if (p.size() != 0 && q.size() != 0){
        int i, j, l, n = 0, m = 0;
        for (i = 0; i < p.size(); i++) A[n++] =
            p.coefficient[i];
        for (i = 0; i < q.size(); i++) B[m++] =
            q.coefficient[i];

        l = mul(n, A, m, B);
        for (i = 0; i < l; i++)
            res.coefficient.push_back(A[i]);
    }
}

polynomial inverse(int n) const{
    polynomial res(n);
    res.resize(n);
    find_inverse(res.data(), n, data(), size());
    return res;
}

/// hash = 190805
static void divide(polynomial &quot, polynomial &rem,
    const polynomial &p, const polynomial &q, const
    polynomial &inv) {
    int pn = p.size(), qn = q.size();
    quot.resize(max(0, pn - qn + 1)), rem.resize(qn - 1);
    divide_remainder_inverse(quot.data(), rem.data(),
        p.data(), pn, q.data(), qn, inv.data());
    quot.normalize(), rem.normalize();
}

polynomial remainder(const polynomial &q, const
    polynomial &inv) const{
    polynomial quot, rem;
    divide(quot, rem, *this, q, inv);
    return rem;
}

polynomial power(const polynomial &q, long long k) const{
    /// hash = 556205
    int qn = q.size();
    if(qn == 1) return polynomial();
    if(k == 0) return polynomial(1);
    polynomial inv = q.inverse(max(size() - qn + 1, qn));
    polynomial p = this->remainder(q, inv);
```

```
        polynomial res = p;
        int l = 63 - __builtin_clzll(k);

        for(--l; l >= 0; --l){
            multiply(res, res, res);
            res = res.remainder(q, inv);
            if((k >> l) & 1){
                multiply(res, res, p);
                res = res.remainder(q, inv);
            }
        }
        return res;
    }

    static void multiply(int* res, const int* P, int pn,
        const int* Q, int qn);

    static void inverse_power_series(int* res, int res_n,
        const int* P, int pn);
    static void find_inverse(int* res, int res_n, const int*
        P, int pn);

    static void divide_inverse(int* res, int res_n, const
        int* revp, int pn, const int* inv);
    static void divide_remainder_inverse(int* quot, int* rem,
        const int* P, int pn, const int* Q, int qn, const
        int* inv);
};

void polynomial::multiply(int* res, const int* P, int pn,
    const int* Q, int qn){ /// hash = 25722
    polynomial P1, P2, P_res;
    for (int i = 0; i < pn; i++)
        P1.coefficient.push_back(P[i]);
    for (int i = 0; i < qn; i++)
        P2.coefficient.push_back(Q[i]);
    P_res.multiply(P_res, P1, P2);
    for (int i = 0; i < P_res.coefficient.size(); i++) res[i]
        = P_res.coefficient[i];
}

void polynomial::inverse_power_series(int* res, int res_n,
    const int* P, int pn){ /// hash = 553608
    if(res_n == 0) return;
    unique_ptr <int[]> ptr(new int[res_n * sizeof(int)]);
    int* u = ptr.get(), *v = u + res_n * 2, cur = 1, nxt = 1;

    clr(res);
    res[0] = P[0];
    while (cur < res_n){
        nxt = min(res_n, cur * 2);
        multiply(u, res, cur, res, cur);
        multiply(v, u, min(nxt, cur * 2 - 1), P, min(nxt,
            pn));
        shift(res, res, cur, 1);
        subtract(res, res, nxt, v, nxt);
        cur = nxt;
    }
}

void polynomial::find_inverse(int* res, int res_n, const
    int* P, int pn){
    unique_ptr <int[]> ptr(new int[pn]);
    int* tmp = ptr.get();
    reverse_poly(tmp, P, pn);
    inverse_power_series(res, res_n, tmp, pn);
}
```

```
    void polynomial::divide_inverse(int* res, int res_n, const
        int* revp, int pn, const int* inv){
        unique_ptr <int[]> ptr(new int[pn + res_n]);
        int* tmp = ptr.get();
        multiply(tmp, revp, pn, inv, res_n);
        reverse_poly(res, tmp, res_n);
    }

    /// hash = 203864
    void polynomial::divide_remainder_inverse(int* quot, int*
        rem, const int* P, int pn, const int* Q, int qn, const
        int* inv){
        if(pn < qn){
            copy(rem, P, pn);
            for (int i = 0; i < qn - pn - 1; i++) rem[i + pn] = 0;
            return;
        }

        if(qn == 1) return;
        int quot_n = pn - qn + 1;
        int rn = qn - 1, tn = min(quot_n, rn), un = tn + rn;
        unique_ptr <int[]> ptr(new int[pn + un + (quot != 0 ? 0 :
            quot_n)]);

        int* revp = ptr.get(), *qmul = revp + pn;
        if(quot == 0) quot = qmul + un;
        reverse_poly(revp, P, pn);
        divide_inverse(quot, quot_n, revp, pn, inv);
        multiply(qmul, Q, rn, quot, tn);
        subtract(rem, P, rn, qmul, rn);
    }
}

namespace bbl{ /// Black box linear algebra
    using namespace pol;
```

```
int mod_inverse(int x){
    int u = 1, v = 0, t = 0, a = x, b = MOD;
    while (b){
        t = a / b;
        a -= (t * b), u -= (t * v);
        swap(a, b), swap(u, v);
    }
    return (u + MOD) % MOD;
}

int convolution(const int* A, const int* B, int n){
    int i = 0, j = 0;
    unsigned long long res = 0;
    for (i = 0; (i + MOD_THRESHOLD) <= n; res %= MOD){
        for (j = 0; j < MOD_THRESHOLD; j++, i++) res +=
            (unsigned long long)A[i] * B[i];
    }

    for (j = 0; i < n; i++) res += (unsigned long long)A[i] *
        B[i];
    return res % MOD;
}

/// Berlekamp Massey algorithm in O(n ^ 2)
/// Finds the shortest linear recurrence that will generate
    the sequence S and returns it in C
/// S[i] * C[l - 1] + S[i + 1] * C[l - 2] + ... + S[j] *
    C[0] = 0

int berlekamp_massey(vector <int> S, vector <int>& C){ ///
    hash = 768118
    assert((S.size() % 2) == 0);

    int n = S.size();
```

```
    C.assign(n + 1, 0);
    vector <int> T, B(n + 1, 0);
    reverse(S.begin(), S.end());

    C[0] = 1, B[0] = 1;
    int i, j, k, d, x, l = 0, m = 1, b = 1, deg = 0;

    for (i = 0; i < n; i++){
        d = S[n - i - 1];
        if (l > 0) d = (d + convolution(&C[1], &S[n - i], l))
            % MOD;
        if (d == 0) m++;
        else{
            if (l * 2 <= i) T.assign(C.begin(), C.begin() + l
                + 1);
            x = (((long long)mod_inverse(b) * (MOD - d)) % MOD
                + MOD) % MOD;

            for (j = 0; j <= deg; j++){
                C[m + j] = (C[m + j] + (unsigned long long)x *
                    B[j]) % MOD;
            }
            if (l * 2 <= i){
                B.swap(T);
                deg = B.size() - 1;
                b = d, m = 1, l = i - l + 1;
            }
            else m++;
        }
    }

    C.resize(l + 1);
    return l;
}
```

```
vector <int> recurrence_coefficients(const vector <int>&
    recurrence, long long k){ /// hash = 713914
    polynomial p, res;
    int n = recurrence.size();

    p.resize(n + 1), p.set(n, 1);
    for (int i = 0; i < n; i++) p.set(i, recurrence[i]);

    vector <int> v;
    res = polynomial(0, 1).power(p, k);
    for (int i = 0; i < n && i < res.size(); i++)
        v.push_back(res.coefficient[i]);
    return v;
}

vector <int> interpolate(const vector<int> &base_sequence,
    const vector<int> &polynomial, long long k, int n){ ///
    hash = 347802
    int i, j, len = polynomial.size() - 1;
    vector <int> recurrence(len);
    for (i = 0; i < len; i++) recurrence[i] = polynomial[i];
    vector <int> coefficient =
        recurrence_coefficients(recurrence, k);

    vector <int> res;
    len = min(len, (int)coefficient.size());
    len = min(len, (int)base_sequence.size());
    for (j = 0; j < n; j++){
        long long r = 0;
        for (i = 0; i < len; i++){
            assert((i + j) < base_sequence.size());
            r = (r + (long long)coefficient[i] *
                base_sequence[i + j]) % MOD;
        }
        res.push_back(r);
```

```
        }
        return res;
    }
}

/// Returns consecutive n terms of the recurrence starting from
    the k'th term
/// MOD must be prime and greater than 2
/// 0.5s for n <= 4000, 4.25s for n <= 16000

vector <int> interpolate(vector <int>& recurrence, long long k,
    int n){
    using namespace bbl;
    assert(recurrence.size() && (recurrence.size() % 2) == 0);

    vector <int> polynomial;
    int l = berlekamp_massey(recurrence, polynomial);
        reverse(polynomial.begin(), polynomial.begin() + l + 1);
    return interpolate(recurrence, polynomial, k, n);
}

/// Returns the k'th term of the recurrence

int interpolate(vector <int>& recurrence, long long k){
    vector <int> res = interpolate(recurrence, k, 1);
    return res[0];
}

const int SEQ_LIM = 69;
inline LL solve(LL n, LL c, LL k) {
    auto sequence = getSequence(SEQ_LIM, c, k);
    if(n <= SEQ_LIM) return sequence[n];
    return interpolate(sequence, n);
}
```

```
int main() {
//    assert(freopen("input.txt", "r", stdin));
//    assert(freopen("alter_out.txt", "w", stdout));

    clock_t st = clock();

    int test, kase=1;
    LL n, c, k;

    assert(scanf("%d", &test) == 1);
    while(test--) {
        assert(scanf("%lld %lld %lld", &n, &c, &k) == 3);
        printf("Case %d: %lld\n", kase++, solve(n, c, k));
    }

    clock_t en = clock();
    fprintf(stderr, "%0.3f sec\n",
        (double)(en-st)/(double)CLOCKS_PER_SEC);

    return 0;
}
```

2 Geometry

2.1 ClosestPairOfPoints

```
///Gives squared Distance
/// O(n log n)
long long ClosestPair(vector<pair<int, int>> pts) {
    int n = pts.size();
    sort(pts.begin(), pts.end());
    set<pair<int, int>> s;
```

```
    long long best_dist = 1e18;
    int j = 0;
    for (int i = 0; i < n; ++i) {
        int d = ceil(sqrt(best_dist));
        while (pts[i].first - pts[j].first >= best_dist) {
            s.erase({pts[j].second, pts[j].first});
            j += 1;
        }

        auto it1 = s.lower_bound({pts[i].second - d,
            pts[i].first});
        auto it2 = s.upper_bound({pts[i].second + d,
            pts[i].first});

        for (auto it = it1; it != it2; ++it) {
            int dx = pts[i].first - it->second;
            int dy = pts[i].second - it->first;
            best_dist = min(best_dist, 1LL * dx * dx + 1LL * dy *
                dy);
        }
        s.insert({pts[i].second, pts[i].first});
    }
    return best_dist;
}
```

2.2 LinePolygonIntersection

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

typedef long long LL;
struct PT {
  LL x, y;
  PT() {}
  PT(LL x, LL 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 * (LL c)   const { return PT(x*c, y*c ); }
  PT operator / (LL c)   const { return PT(x/c, y/c ); }
};

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

///Distance from O to OX intersection AB, divided by length(OX)
double distance(PT O, PT X, PT A, PT B) {
    B = B-A; A = A-O; X=X-O;
    assert(cross(X, B));
    return 1.0*cross(A, B)/cross(X, B);
}

int sign(LL a) {
    return a==0 ? 0 : ( a>0 ? 1 : -1);
}

///Special Points are represented by pair<double, int> (a, b)
///Let O be a special point with OX and OY the incident edges.
///Then a = AO/AB, b is an integer denoting type.
///Cases: OX and OY are on the same side: ignore
///Cases: OX and OY are on the different side: b = 0
///Cases: OX collinear, OY on some side     b = sign(cross(A, B,
    Y))
```

```
double LinePolygonIntersection(PT A, PT B, const vector<PT> &p) {
    int n = p.size();
    vector<pair<double, int>> special;
    for (int i=0; i<n; i++) {
        PT X = p[i], Y = p[(i+1)%n], W = p[(i-1+n)%n];
        LL crx = cross(A, B, X), cry = cross(A, B, Y), crw =
            cross(A, B, W);

        if (crx == 0) {
            double f;
            if (B.x != A.x) f = 1.0*(X.x-A.x)/(B.x-A.x);
            else            f = 1.0*(X.y-A.y)/(B.y-A.y);

            if (sign(crw) && sign(cry)) {
                if (sign(crw) != sign(cry))  special.push_back({f,
                    0});
            }
            else if (sign(cry))          special.push_back({f,
                sign(cry)});
            else if (sign(crw))          special.push_back({f,
                sign(crw)});
        }
        else if (sign(crx) == -sign(cry)) {
            double f = distance(A, B, X, Y);
            special.push_back({f, 0});
        }
    }

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

    bool active = false;
    int sgn = 0;
    double prv = 0, ans = 0;

    for (auto &pr: special) {
        double d = pr.first;
        int tp = pr.second;

        if (sgn) {
            assert(sgn && tp);
            if (sgn != tp) active = !active;
            ans += d - prv;
            sgn = 0;
        }
        else {
            if (active)       ans += d - prv;
            if (tp == 0)      active = !active;
            else              sgn = tp;
        }
        prv = d;
    }
    return ans*sqrt(dot(B-A, B-A));
}

///Solves CF 598F - Cut Length
const int PR = 100;
PT inputPoint() {
    double x, y;
    cin>>x>>y;
    return PT(round(PR*x), round(PR*y));
}


int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    cin>>n>>m;
```

```
    vector<PT> p(n);
    for (int i=0; i<n; i++) p[i] = inputPoint();

    while (m--) {
        PT A, B;
        A = inputPoint();
        B = inputPoint();
        cout<<setprecision(14)<<fixed<<LinePolygonIntersection(A,
            B, p)/PR<<endl;
    }
}
```

2.3 circle-cover

```
///Circle Cover

#include<bits/stdc++.h>
using namespace std;
double EPS = 1e-10;

int dcmp (double a, double b) {
    if (abs(a-b) < EPS)   return 0;
    return a>b ? 1 : -1;
}

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 ); }

  bool operator < (const PT &p) const {
    if (dcmp(x, p.x) == 0)   return dcmp(y, p.y) < 0;
    return dcmp(x, p.x) < 0;
  }
};

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; }
double cross(PT a, PT b, PT c) {return cross(b-a, c-a);}
ostream &operator<<(ostream &os, const PT &p) {
  return os << "(" << p.x << "," << p.y << ")";
}
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }

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;
}

const double PI = acos(-1);

bool CoverCircle(PT O, double R, vector<PT> &cen, vector<double>
    &rad, PT OO, double RR) {
```

```
cen.insert(cen.begin(), OO);
rad.insert(rad.begin(), RR);
assert(cen.size() == rad.size());
int n = cen.size();

vector<pair<double, double>> arcs;

for (int i=0; i<n; i++) {
    PT P = cen[i];
    double r = rad[i];

    if (i!=0 && R + sqrt(dist2(O, P)) < r)  return true;
    if (i==0 && r + sqrt(dist2(O, P)) < R)  return true;

    vector<PT> inter = CircleCircleIntersection(O, P, R, r);
    if (inter.size() <= 1) continue;

    PT X = inter[0], Y = inter[1];
    if (cross(O, X, Y) < 0) swap(X, Y);
    if (!(cross(O, X, P) >= 0 && cross(O, Y, P) <= 0))
        swap(X, Y);
    if (i==0)      swap(X, Y);

    X = X-O; Y=Y-O;
    double ll = atan2(X.y, X.x);
    double rr = atan2(Y.y, Y.x);
    if (rr < ll) rr += 2*PI;
    arcs.emplace_back(ll, rr);
}

if (arcs.empty()) return false;
sort(arcs.begin(), arcs.end());

double st = arcs[0].first, en = arcs[0].second, ans = 0;
for (int i=1; i<arcs.size(); i++) {
        if (arcs[i].first <= en + EPS) en = max(en,
            arcs[i].second);
        else st = arcs[i].first, en = arcs[i].second;
        ans = max(ans, en-st);
    }
    return ans >= 2*PI;
}

bool check(PT O, double R, vector<PT> cen, double mid) {
    int n = cen.size();
    for (int i=0; i<n; i++) {
        vector<PT> cc = cen;
        vector<double> rr(n, mid);
        cc.erase(cc.begin()+i);
        rr.erase(rr.begin()+i);
        if (!CoverCircle(cen[i], mid, cc, rr, O, R)) return false;
    }
    return true;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, R;
    cin>>n>>R;

    PT O(0, 0);
    vector<pair<int, int>> v(n);
    for (int i=0; i<n; i++) cin>>v[i].first>>v[i].second;
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());

    n = v.size();
    vector<PT> p;
```

```
    for (int i=0; i<n; i++)  p.emplace_back(v[i].first,
        v[i].second);

    double lo = 0, hi = 1e4;
    while (hi - lo > 5e-6){
        double mid = (lo + hi)/2;
        if (check(O, R, p, mid))          hi = mid;
        else                             lo = mid;
    }
    cout<<setprecision(7)<<fixed<<lo<<endl;
}
```

2.4 visibility-polygon

```
///Visibility Polygon

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

typedef long long LL;
struct PT {
  LL x, y;
  PT() {}
  PT(LL x, LL 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 * (LL c)   const { return PT(x*c, y*c ); }
  PT operator / (LL c)   const { return PT(x/c, y/c ); }
};

LL dot(PT p, PT q)   { return p.x*q.x+p.y*q.y; }
LL dist2(PT p, PT q) { return dot(p-q,p-q); }
LL cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
LL cross(PT a, PT b, PT c) { return cross(b-a, c-a);}

ostream &operator<<(ostream &os, const PT &p) {
  return os << "(" << p.x << "," << p.y << ")";
}

bool half(PT p) {
    assert(p.x || p.y);
    return p.y > 0 || (p.y == 0 && p.x > 0);
}

int compare(PT a, PT b) {
    auto l = make_tuple(half(b), 0);
    auto r = make_tuple(half(a), cross(a, b));
    return l==r ? 0 : ( l<r ? -1 : 1) ;
}

double distance(PT X, PT A, PT B) {
    B = B-A;
    assert(cross(X, B));
    return sqrt(dot(X, X))*cross(A, B)/cross(X, B);
}

bool compareDis(PT X, PT A, PT B, PT AA, PT BB) {
    B = B-A;
    BB = BB-AA;
    assert(cross(X, B) && cross(X, BB));
    return 1.0*cross(A, B)/cross(X, B) < 1.0*cross(AA,
        BB)/cross(X, BB);
}

pair<double, double> shoot(PT X, double len) {
    double rat = len/sqrt(dot(X, X));
    return make_pair(X.x*rat, X.y*rat);
}
```

```
vector<pair<double, double>> getVisibilityPolygon(PT Z,
    vector<PT> &p) {
    for (PT &X: p) X = X-Z;
    int n = p.size();
    PT O(0, 0);

    auto comp = [](PT a, PT b) {return compare(a, b) < 0;};
    map<PT, vector<int>, decltype(comp) > events(comp);

    for (int i=1; i<=n; i++) {
        PT X = p[i-1], Y = p[i%n];
        if (cross(O, X, Y) < 0) swap(X, Y);
        if (compare(X, Y) == 0) continue;
        events[X].push_back(i);
        events[Y].push_back(-i);
    }

    PT dir, last = events.rbegin() -> first;
    auto comp2 = [&dir, &p, &n](int i, int j) {return
        compareDis(dir, p[i-1], p[i%n], p[j-1], p[j%n]); };
    multiset<int, decltype(comp2)> st(comp2);

    vector<bool> open(n+1);
    for (auto pr: events) {
        for (int v: pr.second)
            if (v > 0) open[v] = 1;
        for (int v: pr.second)
            if (v < 0) open[-v] = 0;
    }

    vector<int> pending;
    vector<pair<double, double>> poly;
    for (int i=1; i<=n; i++)
        if (open[i]) pending.push_back(i);

     for (auto pr: events) {
        PT nw = pr.first;
        dir = nw+last;

        for (int i: pending) st.insert(i);
        pending.clear();

        int i = *st.begin();
        poly.push_back(shoot(last, distance(last, p[i-1],
            p[i%n])));
        poly.push_back(shoot(nw, distance(nw, p[i-1], p[i%n])));

        for (int i: pr.second) {
            if (i < 0) st.erase(-i);
            else       pending.push_back(i);
        }
        last = nw;
    }
    return poly;

}


///Solves Timus 1464. Light
const int PR = 100000;
PT inputPoint() {
    double x, y;
    cin>>x>>y;
    return PT( round(x*PR), round(y*PR) );
}

int main() {
```

```
    ios::sync_with_stdio(0);
    cin.tie(0);
    PT Z = inputPoint();

    int n;
    cin>>n;

    vector<PT> p;
    for (int i=1; i<=n; i++) {
        p.push_back(inputPoint());
    }

    double ans = 0;
    auto poly = getVisibilityPolygon(Z, p);

    for (int i=0; i<poly.size(); i++) {
        int n = (i+1)%poly.size();
        ans += poly[i].first*poly[n].second -
            poly[i].second*poly[n].first;
    }
    cout<<setprecision(5)<<fixed<<abs(ans/2.0/PR/PR)<<"\n";
}
```

3 Temp

3.1 3dGeometry

```
/**
Almost everything is untested
Use at own risk :P - Anachor
*/

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

double EPS = 1e-10;

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

double dot(PT p, PT q)  { return p.x*q.x+p.y*q.y+p.z*q.z; }
double dist2(PT p, PT q) { return dot(p-q,p-q); }
PT cross(PT p, PT q)    { return PT(p.y*q.z-p.z*q.y,
    p.z*q.x-p.x*q.z, p.x*q.y-p.y*q.x); }
double len2(PT p)       { return dot(p, p);}
double len(PT p)        { return sqrt(dot(p, p));}
double triple(PT a, PT b, PT c) {return dot(a, cross(b, c));}

bool isCoplanar(PT a, PT b, PT c, PT d) {
    return abs(triple(b-a, c-a, d-a)) < EPS;
}

ostream &operator<<(ostream &os, const PT &p) {
    return os << "(" << p.x << "," << p.y << "," << p.z << ")";
}
```

```
istream &operator>>(istream &is, PT &p) {
    return is >> p.x >> p.y >> p.z;
}

/// compute center of circle given three non collinear points
PT ComputeCircleCenter(PT a, PT b, PT c) {
    PT x = cross(b-a, c-a);
    assert(len2(x) > EPS);

    PT num1 = cross(x, b-a) * len2(c-a);
    PT num2 = cross(c-a, x) * len2(b-a);
    return a + (num1 + num2)/(len2(x)*2);
}

/// distance of Line ab from o
double PointLineDistance(PT a, PT b, PT o) {
    PT ab = b-a;
    PT oa = a-o;
    return len(cross(ab, oa))/len(ab);
}

/// distance from Line ab to Line cd
/// Unstable for nearly parallel lines
double LineLineDistance(PT a, PT b, PT c, PT d) {
    PT ab = b-a;
    PT cd = d-c;
    PT dir = cross(ab, cd);

    double sz = len2(dir);
    if (sz < EPS) {
        return PointLineDistance(a,b,c);
    }

    dir = dir/len(dir);
    return abs(dot(dir, a-c));
}

struct Plane {
    PT normal;
    double d;     ///Ax + By + Cz = D
    Plane(double a, double b, double c, double d) : normal(a, b,
        c), d(d) {}
    Plane(PT normal, double d) : normal(normal), d(d) {}
};

///Get plane given by three Non-Collinear Points
Plane getPlane(PT a, PT b, PT c) {
    PT normal = cross(b-a, c-a);
    assert(len2(normal) > EPS);
    double d = dot(normal, a);
    return Plane(normal, d);
}

ostream &operator<<(ostream &os, const Plane &p) {
    return os << p.normal.x << "x + " << p.normal.y << "y + " <<
        p.normal.z << "z = "<<p.d;
}

/// distance from point a to plane p
double PointPlaneDistance(PT a, Plane p) {
    return abs(dot(p.normal, a) - p.d) / len(p.normal);
}


int main()
{
    PT a, b, c;
    cin>>a>>b>>c;

    Plane p = getPlane(a, b, c);
```

```
    cout<<p<<endl;
}
```

3.2 Articulation Point

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

const int N = 1e6+7;
int n; // number of nodes
vector<int> adj[N]; // adjacency list of graph
bool visited[N];
int tin[N], fup[N];
int isAP[N];
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = fup[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            fup[v] = min(fup[v], tin[to]);
        } else {
            dfs(to, v);
            fup[v] = min(fup[v], fup[to]);
            if (fup[to] >= tin[v] && p!=-1)
                isAP[v] = 1;
            ++children;
        }
    }
    if(p == -1 && children > 1)
        isAP[v] = 1;
}

void find_cutpoints() {
    timer = 0;

    memset(visited, 0, sizeof visited);
    memset(tin, -1, sizeof tin);
    memset(fup, -1, sizeof fup);
    memset(isAP, 0, sizeof isAP);

    for (int i = 1; i <= n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}

int main()
{
    int m;
    while (true)
    {
        cin>>n>>m;
        if (!n && !m)  break;

        for (int i=1; i<=n; i++)
            adj[i].clear();

        while (m--)
        {
            int x, y;
            cin>>x>>y;
            adj[x].push_back(y);
            adj[y].push_back(x);
        }
```

```
        find_cutpoints();

        int ans = 0;
        for (int i=1; i<=n; i++)
            if (isAP[i])
                ans++;
        cout<<ans<<endl;
    }
}
```

3.3 BigInt

```
/*
  ############################################################################
  ######################## THE  BIG   INT
     ##########################
*/
const int base = 1000000000;
const int base_digits = 9;
struct bigint {
    vector<int> a;
    int sign;
    /*<arpa>*/
    int size(){
        if(a.empty())return 0;
        int ans=(a.size()-1)*base_digits;
        int ca=a.back();
        while(ca)
            ans++,ca/=10;
        return ans;
    }
    bigint operator ^(const bigint &v){
        bigint ans=1,a=*this,b=v;
        while(!b.isZero()){
            if(b%2)
                ans*=a;
            a*=a,b/=2;
        }
        return ans;
    }
    string to_string(){
        stringstream ss;
        ss << *this;
        string s;
        ss >> s;
        return s;
    }
    int sumof(){
        string s = to_string();
        int ans = 0;
        for(auto c : s) ans += c - '0';
        return ans;
    }
    /*</arpa>*/
    bigint() :
        sign(1) {
    }

    bigint(long long v) {
        *this = v;
    }

    bigint(const string &s) {
        read(s);
    }

    void operator=(const bigint &v) {
        sign = v.sign;
        a = v.a;
```

```
}

void operator=(long long v) {
    sign = 1;
    a.clear();
    if (v < 0)
        sign = -1, v = -v;
    for (; v > 0; v = v / base)
        a.push_back(v % base);
}

bigint operator+(const bigint &v) const {
    if (sign == v.sign) {
        bigint res = v;

        for (int i = 0, carry = 0; i < (int) max(a.size(),
            v.a.size()) || carry; ++i) {
            if (i == (int) res.a.size())
                res.a.push_back(0);
            res.a[i] += carry + (i < (int) a.size() ? a[i] :
                0);
            carry = res.a[i] >= base;
            if (carry)
                res.a[i] -= base;
        }
        return res;
    }
    return *this - (-v);
}

bigint operator-(const bigint &v) const {
    if (sign == v.sign) {
        if (abs() >= v.abs()) {
            bigint res = *this;
            for (int i = 0, carry = 0; i < (int) v.a.size() ||
                carry; ++i) {
                res.a[i] -= carry + (i < (int) v.a.size() ?
                    v.a[i] : 0);
                carry = res.a[i] < 0;
                if (carry)
                    res.a[i] += base;
            }
            res.trim();
            return res;
        }
        return -(v - *this);
    }
    return *this + (-v);
}

void operator*=(int v) {
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < (int) a.size() || carry;
        ++i) {
        if (i == (int) a.size())
            a.push_back(0);
        long long cur = a[i] * (long long) v + carry;
        carry = (int) (cur / base);
        a[i] = (int) (cur % base);
        //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) :
            "A"(cur), "c"(base));
    }
    trim();
}

bigint operator*(int v) const {
    bigint res = *this;
    res *= v;
```

```
    return res;
}

void operator*=(long long v) {
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < (int) a.size() || carry;
        ++i) {
        if (i == (int) a.size())
            a.push_back(0);
        long long cur = a[i] * (long long) v + carry;
        carry = (int) (cur / base);
        a[i] = (int) (cur % base);
        //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) :
            "A"(cur), "c"(base));
    }
    trim();
}

bigint operator*(long long v) const {
    bigint res = *this;
    res *= v;
    return res;
}

friend pair<bigint, bigint> divmod(const bigint &a1, const
    bigint &b1) {
    int norm = base / (b1.a.back() + 1);
    bigint a = a1.abs() * norm;
    bigint b = b1.abs() * norm;
    bigint q, r;
    q.a.resize(a.a.size());

    for (int i = a.a.size() - 1; i >= 0; i--) {
        r *= base;
        r += a.a[i];
        int s1 = r.a.size() <= b.a.size() ? 0 :
            r.a[b.a.size()];
        int s2 = r.a.size() <= b.a.size() - 1 ? 0 :
            r.a[b.a.size() - 1];
        int d = ((long long) base * s1 + s2) / b.a.back();
        r -= b * d;
        while (r < 0)
            r += b, --d;
        q.a[i] = d;
    }

    q.sign = a1.sign * b1.sign;
    r.sign = a1.sign;
    q.trim();
    r.trim();
    return make_pair(q, r / norm);
}

bigint operator/(const bigint &v) const {
    return divmod(*this, v).first;
}

bigint operator%(const bigint &v) const {
    return divmod(*this, v).second;
}

void operator/=(int v) {
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
        long long cur = a[i] + rem * (long long) base;
        a[i] = (int) (cur / v);
        rem = (int) (cur % v);
    }
```

```
    trim();
}

bigint operator/(int v) const {
    bigint res = *this;
    res /= v;
    return res;
}

int operator%(int v) const {
    if (v < 0)
        v = -v;
    int m = 0;
    for (int i = a.size() - 1; i >= 0; --i)
        m = (a[i] + m * (long long) base) % v;
    return m * sign;
}

void operator+=(const bigint &v) {
    *this = *this + v;
}
void operator-=(const bigint &v) {
    *this = *this - v;
}
void operator*=(const bigint &v) {
    *this = *this * v;
}
void operator/=(const bigint &v) {
    *this = *this / v;
}

bool operator<(const bigint &v) const {
    if (sign != v.sign)
        return sign < v.sign;
    if (a.size() != v.a.size())
        return a.size() * sign < v.a.size() * v.sign;
    for (int i = a.size() - 1; i >= 0; i--)
        if (a[i] != v.a[i])
            return a[i] * sign < v.a[i] * sign;
    return false;
}

bool operator>(const bigint &v) const {
    return v < *this;
}
bool operator<=(const bigint &v) const {
    return !(v < *this);
}
bool operator>=(const bigint &v) const {
    return !(*this < v);
}
bool operator==(const bigint &v) const {
    return !(*this < v) && !(v < *this);
}
bool operator!=(const bigint &v) const {
    return *this < v || v < *this;
}

void trim() {
    while (!a.empty() && !a.back())
        a.pop_back();
    if (a.empty())
        sign = 1;
}

bool isZero() const {
    return a.empty() || (a.size() == 1 && !a[0]);
}

bigint operator-() const {
```

```
    bigint res = *this;
    res.sign = -sign;
    return res;
}

bigint abs() const {
    bigint res = *this;
    res.sign *= res.sign;
    return res;
}

long long longValue() const {
    long long res = 0;
    for (int i = a.size() - 1; i >= 0; i--)
        res = res * base + a[i];
    return res * sign;
}

friend bigint gcd(const bigint &a, const bigint &b) {
    return b.isZero() ? a : gcd(b, a % b);
}
friend bigint lcm(const bigint &a, const bigint &b) {
    return a / gcd(a, b) * b;
}

void read(const string &s) {
    sign = 1;
    a.clear();
    int pos = 0;
    while (pos < (int) s.size() && (s[pos] == '-' || s[pos]
        == '+')) {
        if (s[pos] == '-')
            sign = -sign;
        ++pos;
    }
    for (int i = s.size() - 1; i >= pos; i -= base_digits) {
        int x = 0;
        for (int j = max(pos, i - base_digits + 1); j <= i;
            j++)
            x = x * 10 + s[j] - '0';
        a.push_back(x);
    }
    trim();
}

friend istream& operator>>(istream &stream, bigint &v) {
    string s;
    stream >> s;
    v.read(s);
    return stream;
}

friend ostream& operator<<(ostream &stream, const bigint &v)
    {
    if (v.sign == -1)
        stream << '-';
    stream << (v.a.empty() ? 0 : v.a.back());
    for (int i = (int) v.a.size() - 2; i >= 0; --i)
        stream << setw(base_digits) << setfill('0') << v.a[i];
    return stream;
}

static vector<int> convert_base(const vector<int> &a, int
    old_digits, int new_digits) {
    vector<long long> p(max(old_digits, new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < (int) p.size(); i++)
        p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
```

```
    int cur_digits = 0;
    for (int i = 0; i < (int) a.size(); i++) {
        cur += a[i] * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits) {
            res.push_back(int(cur % p[new_digits]));
            cur /= p[new_digits];
            cur_digits -= new_digits;
        }
    }
    res.push_back((int) cur);
    while (!res.empty() && !res.back())
        res.pop_back();
    return res;
}

typedef vector<long long> vll;

static vll karatsubaMultiply(const vll &a, const vll &b) {
    int n = a.size();
    vll res(n + n);
    if (n <= 32) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                res[i + j] += a[i] * b[j];
        return res;
    }

    int k = n >> 1;
    vll a1(a.begin(), a.begin() + k);
    vll a2(a.begin() + k, a.end());
    vll b1(b.begin(), b.begin() + k);
    vll b2(b.begin() + k, b.end());

    vll a1b1 = karatsubaMultiply(a1, b1);
    vll a2b2 = karatsubaMultiply(a2, b2);

    for (int i = 0; i < k; i++)
        a2[i] += a1[i];
    for (int i = 0; i < k; i++)
        b2[i] += b1[i];

    vll r = karatsubaMultiply(a2, b2);
    for (int i = 0; i < (int) a1b1.size(); i++)
        r[i] -= a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        r[i] -= a2b2[i];

    for (int i = 0; i < (int) r.size(); i++)
        res[i + k] += r[i];
    for (int i = 0; i < (int) a1b1.size(); i++)
        res[i] += a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        res[i + n] += a2b2[i];
    return res;
}

bigint operator*(const bigint &v) const {
    vector<int> a6 = convert_base(this->a, base_digits, 6);
    vector<int> b6 = convert_base(v.a, base_digits, 6);
    vll a(a6.begin(), a6.end());
    vll b(b6.begin(), b6.end());
    while (a.size() < b.size())
        a.push_back(0);
    while (b.size() < a.size())
        b.push_back(0);
    while (a.size() & (a.size() - 1))
        a.push_back(0), b.push_back(0);
    vll c = karatsubaMultiply(a, b);
    bigint res;
```

```
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < (int) c.size(); i++) {
            long long cur = c[i] + carry;
            res.a.push_back((int) (cur % 1000000));
            carry = (int) (cur / 1000000);
        }
        res.a = convert_base(res.a, 6, base_digits);
        res.trim();
        return res;
    }
};
/*
  ######################## THE  BIG   INT
     ##########################
  ##########################################################################
*/
```

3.4 Bitset

```
/**
    Simple Bitset Class with ability to perform range bitwise
        operations.
    Written by - Anachor
**/


#include <bits/stdc++.h>
typedef unsigned long long ULL;
using namespace std;

struct Bitset {
    const static int B = 6, K = 64, X = 63;

    ///returns mask with bits l to r set, and others reset
    static inline ULL getmask(int l, int r) {
        if (r==X) return -(1ULL<<l);
        return (1ULL<<(r+1)) - (1ULL<<l);
    }

    vector<ULL> bs;
    int N;

    Bitset(int n) {
        N = n/K+1;
        bs.resize(N);
    }

    void assign(ULL x) {
        fill(bs.begin()+1, bs.end(), 0);
        bs[0] = x;
    }

    bool get(int i) {
        return bs[i>>B] & (1ULL<<(i&X));
    }

    void set(int i) {
        bs[i>>B] |= (1ULL<<(i&X));
    }

    void reset(int i) {
        bs[i>>B] &= ~(1ULL<<(i&X));
    }

    void flip(int i) {
        bs[i>>B] ^= (1ULL<<(i&X));
    }

    void set(int l, int r) {
```

```
    int idl = l>>B;
    int idr = r>>B;
    int posl = l&X;
    int posr = r&X;

    if (idl == idr) {
        bs[idl] |= getmask(posl, posr);
        return;
    }

    bs[idl] |= getmask(posl, X);
    bs[idr] |= getmask(0, posr);

    for (int id = idl+1; id < idr; id++)
        bs[id] = -1;
}

void reset(int l, int r) {
    int idl = l>>B;
    int idr = r>>B;
    int posl = l&X;
    int posr = r&X;

    if (idl == idr) {
        bs[idl] &= ~getmask(posl, posr);
        return;
    }

    bs[idl] &= ~getmask(posl, X);
    bs[idr] &= ~getmask(0, posr);

    for (int id = idl+1; id < idr; id++)
        bs[id] = 0;
}
```

```
    void flip(int l, int r) {
        int idl = l>>B;
        int idr = r>>B;
        int posl = l&X;
        int posr = r&X;

        if (idl == idr) {
            bs[idl] ^= getmask(posl, posr);
            return;
        }

        bs[idl] ^= getmask(posl, X);
        bs[idr] ^= getmask(0, posr);

        for (int id = idl+1; id < idr; id++)
            bs[id] = ~bs[id];
    }
};


///Solves CF 1146E - Hot is Cold
const int N = 1e5+7;
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, q;
    cin>>n>>q;
    vector<int> a(n);
    for (int i=0; i<n; i++) cin>>a[i];

    Bitset bs(2*N+1);
    bs.set(0+N, N+N);
```

```
    while (q--) {
        char c;
        int x;
        cin>>c>>x;

        if (c=='<') {
            if (x<=0)   bs.set(-N+N, x-1+N), bs.set(-x+1+N, N+N);
            else        bs.set(-N+N, -x+N), bs.flip(-x+1+N,
                x-1+N), bs.set(x+N, N+N);
        }
        else {
            if (x>=0)   bs.reset(-N+N, -x-1+N), bs.reset(x+1+N,
                N+N);
            else        bs.reset(-N+N, x+N), bs.flip(x+1+N,
                -x-1+N), bs.reset(-x+N, N+N);
        }
    }

    for (int i=0; i<n; i++) {
        bool b = bs.get(a[i]+N);
        if (b) a[i] = abs(a[i]);
        else   a[i] = -abs(a[i]);
        cout<<a[i]<<" ";
    }
}
```

3.5 Bridge

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

const int N = 1e6+7;
int n; // number of nodes
vector<int> adj[N]; // adjacency list of graph

bool visited[N];
int tin[N], fup[N];
int timer;

set<pair<int, int> > bridges;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = fup[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            fup[v] = min(fup[v], tin[to]);
        } else {
            dfs(to, v);
            fup[v] = min(fup[v], fup[to]);
            if (fup[to] > tin[v])
                bridges.insert(make_pair(v, to));
        }
    }
}

void find_bridges() {
    timer = 0;
    memset(visited, 0, sizeof visited);
    memset(tin, -1, sizeof tin);
    memset(fup, -1, sizeof fup);

    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

3.6 CHTLinear

```
/**
Linear Convex Hull Trick
Requirement:
    Minimum:
        M increasing, x decreasing, useless(s-1, s-2, s-3)
        M decreasing, x increasing, useless(s-3, s-2, s-1)
    Maximum:
        M increasing, x increasing, useless(s-3, s-2, s-1)
        M decreasing, x decreasing, useless(s-1, s-2, s-3)
**/


#include<bits/stdc++.h>
using namespace std;
typedef long long LL;

struct CHT {
    vector<LL> M;
    vector<LL> C;
    int ptr = 0;

    ///Use double comp if M,C is LL range
    bool useless(int l1, int l2, int l3) {
        return (C[l3]-C[l1])*(M[l1]-M[l2]) <=
            (C[l2]-C[l1])*(M[l1]-M[l3]);
    }

    LL f(int id, LL x) {
        return M[id]*x+C[id];
    }

    void add(LL m, LL c) {
        M.push_back(m);
        C.push_back(c);
        int s = M.size();

        while (s >= 3 && useless(s-3, s-2, s-1)) {
            M.erase(M.end()-2);
            C.erase(C.end()-2);
            s--;
        }
    }

    LL query(LL x) {
        if (ptr >= M.size()) ptr = M.size()-1;
        while (ptr < M.size()-1 && f(ptr, x) > f(ptr+1, x))
            ptr++; /// change > to < for maximum
        return f(ptr, x);
    }
};


///Solves SPOJ Acquire
const int N = 1e5+7;
LL dp[N];

typedef pair<LL, LL> PLL;
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n;
    cin>>n;

    vector<PLL> v(n), t;

    for (int i=0; i<n; i++) cin>>v[i].first>>v[i].second;
    sort(v.begin(), v.end());
```

```
    for (int i=0; i<n; i++) {
        while (t.size() && t.back().second <= v[i].second)
            t.pop_back();
        t.push_back(v[i]);
    }

    n = t.size();
    dp[0] = 0;

    CHT cht;
    for (int i=1; i<=n; i++) {
        cht.add(t[i-1].second, dp[i-1]);
        dp[i] = cht.query(t[i-1].first);
    }

    cout<<dp[n]<<endl;
}
```

3.7 Congruence

```
/// This is a collection of useful code for solving problems that
/// involve modular linear equations. Note that all of the
/// algorithms described here work on NON-NEGATIVE INTEGERS.
/// Source: Stanford Notebook (modified)

#include <bits/stdc++.h>
#define LL long long
using namespace std;
typedef pair<LL, LL> PLL;


/// Computes gcd(a,b)
/// Range: LL
LL gcd(LL u, LL v) {
    if (u == 0) return v;
    if (v == 0) return u;
    int shift = __builtin_ctzll(u | v);
    u >>= __builtin_ctzll(u);
    do {
        v >>= __builtin_ctzll(v);
        if (u > v) swap(u, v);
        v = v - u;
    } while (v);
    return u << shift;
}

/// computes lcm(a,b)
/// Range: int
LL lcm(LL a, LL b) {
        return (a/gcd(a, b))*b;
}

/// (a^b) mod m via successive squaring
/// Range: int
LL power(LL a, LL b, LL m) {
    a = (a%m+m)%m;
        LL ans = 1;
        while (b) {
                if (b & 1) ans = (ans*a)%m;
                a = (a*a)%m;
                b >>= 1;
        }
        return ans;
}

/// returns g = gcd(a, b); finds x, y such that d = ax + by
/// Range: int (tested on CF 982E :( )
```

```
LL egcd(LL a, LL b, LL &x, LL &y) {
	LL xx = y = 0;
	LL yy = x = 1;
	while (b) {
		LL q = a/b;
		LL t = b; b = a%b; a = t;
		t = xx; xx = x-q*xx; x = t;
		t = yy; yy = y-q*yy; y = t;
	}
	return a;
}

/// finds all solutions to ax = b (mod m)
/// Range: int (not tested)
vector<LL> SolveCongruence(LL a, LL b, LL m) {
	LL x, y;
	vector<LL> ans;
	LL g = egcd(a, m, x, y);
	if (b%g == 0) {
		x = (x*(b/g))%m;
		if (x<0)   x+=m;
		for (LL i=0; i<g; i++) {
			ans.push_back(x);
			x = (x+m/g)%m;
		}
	}
	return ans;
}

/// Computes b such that ab = 1 (mod m), returns -1 on failure
/// Range: int
LL inverse(LL a, LL m) {
	LL x, y;
	LL g = egcd(a, m, x, y);
	if (g > 1) return -1;
	return (x%m+m)%m;
}

/// Chinese remainder theorem (special case):
/// find z such that z % m1 = r1, z % m2 = r2.
/// Here, z is unique modulo M = lcm(m1, m2).
/// Return (z, M). On failure, M = -1.
/// Range: int (tested on CF 982E :( )
PLL CRT(LL m1, LL r1, LL m2, LL r2) {
	LL s, t;
	LL g = egcd(m1, m2, s, t);
	if (r1%g != r2%g) return PLL(0, -1);
	LL M = m1*m2;
	LL ss = ((s*r2)%m2)*m1;
	LL tt = ((t*r1)%m1)*m2;
	LL ans = ((ss+tt)%M+M)%M;
	return PLL(ans/g, M/g);
}


/// Chinese remainder theorem:
/// find z such that z % m[i] = r[i] for all i.
/// The solution is unique modulo M = lcm(m[i]).
/// Return (z, M). On failure, M = -1.
/// Note that we do not require the mod values to be co-prime.
/// Range: int (if LCM fits in LL)
PLL CRT(const vector<LL> &m, const vector<LL> &r) {
	PLL ans = PLL(r[0], m[0]);
	for (LL i = 1; i < m.size(); i++) {
		ans = CRT(ans.second, ans.first, m[i], r[i]);
		if (ans.second == -1) break;
	}
	return ans;
}
```

```
/// computes x and y such that ax + by = c
/// returns whether the solution exists
/// Range: int
bool LinearDiophantine(LL a, LL b, LL c, LL &x, LL &y) {
	if (!a && !b) {
		if (c) return false;
		x = y = 0;
		return true;
	}
	if (!a) {
		if (c%b) return false;
		x = 0; y = c/b;
		return true;
	}
	if (!b) {
		if (c%a) return false;
		x = c/a; y = 0;
		return true;
	}
	LL g = gcd(a, b);
	if (c%g) return false;
	x = c/g * inverse(a/g, b/g);
	y = (c-a*x)/b;
	return true;
}

int main() {
	// expected: 2
	cout << gcd(14, 30) << endl;

	// expected: 2 -2 1
	LL x, y;
	LL g = egcd(14, 30, x, y);
	cout << g << " " << x << " " << y << endl;

	// expected: 95 45
    vector<long long> sols = SolveCongruence(14, 30, 100);
	for (LL i = 0; i < sols.size(); i++) cout << sols[i] << "
	    ";
	cout << endl;

	// expected: 8
	cout << inverse(8, 9) << endl;

	// expected: 23 105
	//           11 12
	PLL ans = CRT({3,5,7}, {2,3,2});
	cout << ans.first << " " << ans.second << endl;
	ans = CRT({4,6}, {3,5});
	cout << ans.first << " " << ans.second << endl;

	// expected: 5 -15
	if (!LinearDiophantine(7, 2, 5, x, y)) cout << "ERROR" <<
	    endl;
	else cout << x << " " << y << endl;
	return 0;
}
```

3.8 DivideConquerOptimization

```
/*
CodeChef - CHEFAOR
You are given an array A of integers and an integer K. Your goal
is to divide the array into K consecutive disjoint non-empty
groups, so that any array element belongs to exactly one group.
The cost of such a group equals to the value of bitwise OR of
all elements in the group. The cost of array for some particular
group division equals to the sum of costs for all the groups.
You have to find the maximal achievable cost of the given array.
```

```
O(NKlogN)
*/

#include<bits/stdc++.h>
using namespace std;
#define ll long long

ll dp[5001][5001];
ll cost[5001][5001];

ll a[5001];


void calculate(int level, int L, int R, int idL, int idR)
{
    int md = (L+R)/2;

    int j, k = idL;
    dp[level][md] = 0;

    for (j = idL; j <= idR && j < md; j++) {
        if (dp[level-1][j]+cost[j+1][md]>dp[level][md]) {
            dp[level][md] = dp[level-1][j]+cost[j+1][md];
            k = j;
        }
    }

    if (L <= md-1) calculate(level, L, md-1, idL, k);
    if (md+1 <= R) calculate(level, md+1, R, k, idR);
}


int main(void)
{
    int t;

    scanf("%d", &t);

    while (t--) {
        int n, k;

        scanf("%d %d", &n, &k);

        int i, j;
        for (i = 1; i <= n; i++) {
            scanf("%lld", &a[i]);
            dp[1][i] = dp[1][i-1] | a[i];
        }

        for (i = 1; i <= n; i++) {
            cost[i][i] = a[i];
            for (j = i+1; j <= n; j++) {
                cost[i][j] = cost[i][j-1] | a[j];
            }
        }

        for (i = 2; i <= k; i++) calculate(i, i, n, i-1, n-1);

        printf("%lld\n", dp[k][n]);
    }



    return 0;
}
```

3.9 DominatorTree

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

```
typedef vector<int> VI;
typedef vector<VI> VVI;

struct ChudirBhai
{
    int n;
    VVI g, tree, rg, bucket;
    VI sdom, par, dom, dsu, label;
    VI arr, rev;
    int T;

    ChudirBhai(int n): n(n), g(n+1), tree(n+1), rg(n+1),
        bucket(n+1),
                    sdom(n+1), par(n+1), dom(n+1), dsu(n+1),
                        label(n+1),
                    arr(n+1), rev(n+1), T(0)
    {
        for(int i = 1; i <= n; i++) sdom[i] = dom[i] = dsu[i] =
            label[i] = i;
    }

    void addEdge(int u, int v) { g[u].push_back(v); }

    void dfs0(int u)
    {
        T++; arr[u] = T, rev[T] = u;
        label[T] = T, sdom[T] = T, dsu[T] = T;

        for(int i = 0; i < g[u].size(); i++)
        {
            int w = g[u][i];

            if(!arr[w]) dfs0(w), par[arr[w]] = arr[u];
            rg[arr[w]].push_back(arr[u]);
        }
    }

    int Find(int u, int x = 0)
    {
        if(u == dsu[u]) return x? -1: u;

        int v = Find(dsu[u], x+1);
        if(v < 0) return u;

        if(sdom[label[dsu[u]]] < sdom[label[u]]) label[u] =
            label[dsu[u]];
        dsu[u] = v;

        return x? v: label[u];
    }

    void Union(int u, int v) { dsu[v] = u; }

    VVI buildAndGetTree(int s)
    {
        dfs0(s);

        for(int i = n; i >= 1; i--)
        {
            for(int j = 0; j < rg[i].size(); j++)
                sdom[i] = min(sdom[i], sdom[Find(rg[i][j])]);

            if(i > 1) bucket[sdom[i]].push_back(i);

            for(int j = 0; j < bucket[i].size(); j++)
            {
                int w = bucket[i][j], v = Find(w);

                if(sdom[v] == sdom[w]) dom[w] = sdom[w];
                else dom[w] = v;
```

```
            }

            if(i > 1) Union(par[i], i);
        }

        for(int i = 2; i <= n; i++)
        {
            if(dom[i] != sdom[i]) dom[i] = dom[dom[i]];

            tree[rev[i]].push_back(rev[dom[i]]);
            tree[rev[dom[i]]].push_back(rev[i]);
        }

        return tree;
    }

};

int main()
{
    int n, m;

    scanf("%d %d", &n, &m);

    ChudirBhai bhai(n);

    for(int i = 1; i <= m; i++)
    {
        int u, v;
        scanf("%d %d", &u, &v);

        bhai.addEdge(u, v);
    }

    VVI tree = bhai.buildAndGetTree(1);

    for(int i = 1; i <= n; i++)
    {
        printf("%d: ", i);
        for(int j = 0; j < tree[i].size(); j++) printf("%d ",
            tree[i][j]);
        printf("\n");
    }

    return 0;
}
```

3.10 EulerPath

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

struct Edge {
    int u, v;
    int other(int x) {return u^v^x;}
};

struct Eulerpath {
    int n;
    vector<Edge> edges;
    vector<bool> vis;
    vector<vector<int>> adj;

    Eulerpath(int nn) : n(nn), adj(nn) {}
    void addEdge(int u, int v) {
        edges.push_back({u, v});
        vis.push_back(0);
        int id = edges.size()-1;
        adj[u].push_back(id);
```

```
    adj[v].push_back(id);
}

bool getTour(vector<int> &path) {
    int u = 0, cnt=0;
    for (int i=0; i<n; i++)
        if(adj[i].size()%2) {
            u = i;
            ++cnt;
        }

    if (cnt !=0 && cnt != 2) return false;

    stack<int> st;
    path.clear();

    while (true) {
        while (adj[u].size() && vis[adj[u].back()] == 1)
            adj[u].pop_back();

        if (adj[u].empty()) {
            path.push_back(u);
            if (st.empty()) break;
            u = st.top();
            st.pop();
        }
        else {
            st.push(u);
            int id = adj[u].back();
            vis[id] = 1;
            u = edges[id].other(u);
        }
    }

    for (int i=0; i<edges.size(); i++)
        if (!vis[i])
            return false;
    return true;
    }
};


int main() {
    int n, m;
    cin>>n>>m;

    Eulerpath solver(n);
    while (m--) {
        int u, v;
        cin>>u>>v;
        solver.addEdge(u, v);
    }

    vector<int> v;
    bool b = solver.getTour(v);
    if (!b) cout<<"None"<<endl;
    else for (int x: v) cout<<x<<" ";
}
```

3.11 FWHT

```
/**
Iterative implementation of Fast WalshHadamard transform
Complexity: O(N log N)

XOR convolution:
    Given two arrays A, B; Find C where
    C[k] = sum(a[i]*b[j]); i^j=k
AND convolution:
```

```
    Given two arrays A, B; Find C where
    C[k] = sum(a[i]*b[j]); i&j=k
OR convolution:
    Given two arrays A, B; Find C where
    C[k] = sum(a[i]*b[j]); i|j=k
**/


#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
#define bitwiseXOR 1
///#define bitwiseAND 2
///#define bitwiseOR 3

void FWHT(vector< LL >&p, bool inverse)
{
    int n = p.size();
    assert((n&(n-1))==0);

    for (int len = 1; 2*len <= n; len <<= 1) {
        for (int i = 0; i < n; i += len+len) {
            for (int j = 0; j < len; j++) {
                LL u = p[i+j];
                LL v = p[i+len+j];

                #ifdef bitwiseXOR
                p[i+j] = u+v;
                p[i+len+j] = u-v;
                #endif // bitwiseXOR

                #ifdef bitwiseAND
                if (!inverse) {
                    p[i+j] = v;
                    p[i+len+j] = u+v;
                } else {
                    p[i+j] = -u+v;
                    p[i+len+j] = u;
                }
                #endif // bitwiseAND

                #ifdef bitwiseOR
                if (!inverse) {
                    p[i+j] = u+v;
                    p[i+len+j] = u;
                } else {
                    p[i+j] = v;
                    p[i+len+j] = u-v;
                }
                #endif // bitwiseOR
            }
        }
    }

    #ifdef bitwiseXOR
    if (inverse) {
        for (int i = 0; i < n; i++) {
            assert(p[i]%n==0);
            p[i] /= n;
        }
    }
    #endif // bitwiseXOR
}


int main()
{
    vector< LL >p(4);
    p[0] = 2;  p[1] = 4;  p[2] = 6;  p[3] = 8;
    vector< LL >q(4);
```

```
    q[0] = 3;  q[1] = 5;  q[2] = 7; q[3] = 9;

    FWHT(p, false);
    FWHT(q, false);

    vector< LL >r(4);
    for (int i = 0; i < 4; i++) r[i] = p[i]*q[i];

    FWHT(r, true);
    for (int i = 0; i < 4; i++) {
        cout << "r[" << bitset<2>(i) << "] = " << r[i] << endl;
    }

    return 0;
}
```

3.12 HLD

```
/*
    Heavy-Light Decomposition
    1. flat[] (0-indexed) has the flattened array of the tree
       according
       to the decomposition into chains
    2. flatIdx[] is the reverse map of flat[]
    3. There will be O(logN) segments of chains between node u &
       v.
    4. getChainSegments(u, v) calculates these in O(logN) time
    5. dfs(u, p) & HLD(u, p) are essential. The rest of the
       functions
       are auxiliary
*/

#include<bits/stdc++.h>
using namespace std;
typedef pair<int,int>PII;
const int MAXN = 500007;
const int LOGN = 20;
vector<int>edg[MAXN];

int sbtr[MAXN], lvl[MAXN], pr[MAXN][LOGN];
int chainIdx[MAXN], chainHead[MAXN], flatIdx[MAXN], flat[MAXN];
int chainCnt, flatCnt;

void dfs(int u, int p)
{
    lvl[u] = lvl[p] + 1;
    pr[u][0] = p;
    for (int k = 1; k < LOGN; k++) {
        pr[u][k] = pr[pr[u][k-1]][k-1];
    }

    sbtr[u] = 1;
    for (int v : edg[u]) {
        if (v==p) continue;
        dfs(v, u);
        sbtr[u] += sbtr[v];
    }
}

/// auxiliary function
int getLCA(int u, int v)
{
    if (lvl[u] < lvl[v]) swap(u, v);
    for (int k = LOGN-1; k >= 0; k--) {
        if (lvl[u]-(1<<k) >= lvl[v]) {
            u = pr[u][k];
        }
    }
```

```
    if (u==v) return u;
    for (int k = LOGN-1; k >= 0; k--) {
        if (pr[u][k] != pr[v][k]) {
            u = pr[u][k];
            v = pr[v][k];
        }
    }
    return pr[u][0];
}

void HLD(int u, int p)
{
    chainIdx[u] = chainCnt;
    flatIdx[u] = flatCnt;
    flat[flatCnt] = u;
    flatCnt++;

    int biggie = -1, mx = 0;
    for (int v : edg[u]) {
        if (v==p) continue;
        if (mx < sbtr[v]) {
            mx = sbtr[v];
            biggie = v;
        }
    }
    if (biggie==-1) return;

    HLD(biggie, u);
    for (int v : edg[u]) {
        if (v==p||v==biggie) continue;
        chainCnt++;
        chainHead[chainCnt] = v;
        HLD(v, u);
    }
}
```

```
/// upSegments(l, u, vp) add segments for (l, u] to vp vector
/// provided l is an ancestor of u
void upSegments(int l, int u, vector<PII>&vp)
{
    while (chainIdx[l] != chainIdx[u]) {
        int uhead = chainHead[chainIdx[u]];
        vp.push_back(PII(flatIdx[uhead], flatIdx[u]));
        u = pr[uhead][0];
    }
    if (l!=u) {
        vp.push_back(PII(flatIdx[l]+1, flatIdx[u]));
    }
}

vector<PII>getChainSegments(int u, int v)
{
    int l = getLCA(u, v);
    vector<PII>rt;
    rt.push_back(PII(flatIdx[l], flatIdx[l]));
    if (u==v) return rt;
    upSegments(l, u, rt);
    upSegments(l, v, rt);
    return rt;
}

PII getSubtreeSegment(int u) {
    return PII(flatIdx[u], flatIdx[u]+sbtr[u]-1);
}

void performHLD(int root)
{
    dfs(root, 0);
    chainCnt = 0;
    flatCnt = 0;
```

```
    chainHead[0] = root;
    HLD(root, 0);
}

int main()
{
    int n;
    cin >> n;

    for (int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        edg[u].push_back(v);
        edg[v].push_back(u);
    }

    performHLD(1);

    for (int i = 0; i < n; i++) cout << flat[i] << ' ';


    return 0;
}
```

3.13 HalfPlaneIntersection

```
#include<bits/stdc++.h>
using namespace std;
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 len(PT p)        { return sqrt(dot(p, p));}
double cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  return os << "(" << p.x << "," << p.y << ")";
}

PT RotateCCW90(PT p)  { return PT(-p.y,p.x); }
PT RotateCW90(PT p)   { return PT(p.y,-p.x); }

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

PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  return a + b*cross(c, d)/cross(b, d);
}

/// Set of points C such that A,B,C are ccw.
struct HalfPlane {
    PT A, B;

    bool onHalfPlane (PT p) {
        return cross(A-B, B-p) >= -EPS;
    }
};
```

```
///Get Halfplane for ax+by+c >= 0
HalfPlane getHalfPlane(double a, double b, double c) {
    PT A, B, C;
    if (abs(a) < EPS) {
        A = PT(0, -c/b);
        B = PT(1, -c/b);
    }
    else {
        A = PT(-c/a, 0);
        B = PT(-(b+c)/a, 1);
    }
    C = A + RotateCCW90(B-A);
    if (a*C.x+b*C.y+c < 0) swap(A, B);
    return {A, B};
}

double INF = 1e100;

bool halfPlaneIntersection(vector<HalfPlane> planes) {
    int n = planes.size();
    shuffle(planes.begin(), planes.end(), mt19937(time(NULL)));

    PT best(INF, INF);
    for (int i=0; i<n; i++) {
        HalfPlane &hp = planes[i];
        if (hp.onHalfPlane(best)) continue;

        PT dir = hp.B - hp.A;
        dir = dir/len(dir);
        PT X = hp.A+dir*INF, Y = hp.A-dir*INF;

        for (int j=0; j<i; j++) {
            HalfPlane &cp = planes[j];
            if (LinesParallel(hp.A, hp.B, cp.A, cp.B)) {
                if (!cp.onHalfPlane(hp.A)) return false;
            }
            else {
                PT O = ComputeLineIntersection(hp.A, hp.B, cp.A,
                    cp.B);
                bool bX = cp.onHalfPlane(X);
                bool bY = cp.onHalfPlane(Y);

                if (bX && bY)  continue;
                else if (bX)     Y = O;
                else if (bY)     X = O;
                else return false;
            }
        }

        if (X.x+X.y < Y.x + Y.y)  best = Y;
        else                      best = X;
    }
    return true;
}

///Solves DMOJ - Arrow (https://dmoj.ca/problem/ccoprep3p3)
///Set EPS to 1e-20 and use long double

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n;
    cin>>n;

    vector<HalfPlane> planes;
    planes.push_back(getHalfPlane(-1, 0, 0));
    planes.push_back(getHalfPlane(0, 1, 0));
    for (int i=0; i<n; i++) {
```

```
        double x, y1, y2;
        cin>>x>>y1>>y2;
        planes.push_back(getHalfPlane(x*x, x, -y1));
        planes.push_back(getHalfPlane(-x*x, -x, y2));
    }

    int lo = 0, hi = n;
    while (lo < hi) {
        int m = (lo + hi+1)/2;
        vector<HalfPlane> cur(planes.begin(), planes.begin() +
            2*m+2);
        if (halfPlaneIntersection(cur)) lo = m;
        else                            hi = m-1;
    }
    cout<<lo<<endl;
}
```

3.14 HopcroftKarp

```
/**
    Hopcroft Karp Weighted Bipartite Matching Algorithm
    Complexity: E sqrt(V)
    Source: Foreverbell ICPC cheat sheet
*/

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

const int maxN = 50000+5, maxM = 50000+5;
struct HopcroftKarp {
  int n;
  int vis[maxN], level[maxN], ml[maxN], mr[maxM];
  vector<int> edge[maxN]; // constructing edges for left part
      only

HopcroftKarp(int n) : n(n) {   // n = nodes in left part
  for (int i = 1; i <= n; ++i) edge[i].clear();
}

void add(int u, int v) {
  edge[u].push_back(v);
}

bool dfs(int u) {
  vis[u] = true;
  for (vector<int>::iterator it = edge[u].begin(); it !=
      edge[u].end(); ++it) {
    int v = mr[*it];
    if (v == -1 || (!vis[v] && level[u] < level[v] && dfs(v))) {
      ml[u] = *it;
      mr[*it] = u;
      return true;
    }
  }
  return false;
}

int matching() { // n for left
  memset(vis, 0, sizeof vis);
  memset(level, 0, sizeof level);
  memset(ml, -1, sizeof ml);
  memset(mr, -1, sizeof mr);

  for (int match = 0;;) {
    queue<int> que;
    for (int i = 1; i <= n; ++i) {
      if (ml[i] == -1) {
        level[i] = 0;
        que.push(i);
```

```
      } else level[i] = -1;
    }
    while (!que.empty()) {
      int u = que.front();
      que.pop();
      for (vector<int>::iterator it = edge[u].begin(); it !=
          edge[u].end(); ++it) {
        int v = mr[*it];
        if (v != -1 && level[v] < 0) {
          level[v] = level[u] + 1;
          que.push(v);
        }
      }
    }
    for (int i = 1; i <= n; ++i) vis[i] = false;
    int d = 0;
    for (int i = 1; i <= n; ++i) if (ml[i] == -1 && dfs(i)) ++d;
    if (d == 0) return match;
    match += d;
  }
 }
};


/// The following code solves SPOJ MATCHING - Fast Maximum
    Matching

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m, k;
    cin>>n>>m>>k;
    HopcroftKarp solver(n);

    while(k--) {
        int a, b;
        cin>>a>>b;
        solver.add(a, b);
    }

    cout<<solver.matching()<<endl;
}
```

3.15 IntervalContainer

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

/// Maintains an array of size n, initially all values are same.
/// Color(l, r, c): Set a[i] = c for l <= i <= r.
/// get(x): return a[x];
struct IntervalContainer {
    int n;
    map<int, int> a;

    int get(int x) {
        auto it = a.upper_bound(x);
        return (--it)->second;
    }

    /// Container size n with initial value c
    IntervalContainer(int n, int c) {
        this->n = n;
        a[1] = c;
        a[n+1] = c;
    }
```

```
    void color(int l, int r, int c) {
        int rc = get(r);
        a[l] = c;
        auto it = a.upper_bound(l);
        while (it->first <= r) it = a.erase(it);
        if (it->first > r+1) a[r+1] = rc;
    }
};



///Solves CF 1208E
const int N = 1e6+7;
typedef long long LL;
typedef pair<int, int> PII;
LL ans[N];

int main() {
    int n, w;
    cin>>n>>w;

    for (int i=1; i<=n; i++) {
        int m;
        cin>>m;

        vector<PII> row(m);
        for (int j=0; j<m; j++) {
            cin>>row[j].first;
            row[j].second = j+1;
        }

        sort(row.begin(), row.end());
        IntervalContainer ic(w, -1e9-7);

        bool nopos = true;

        for (PII pr: row) {
            int v = pr.first;
            int pos = pr.second;
            if (v > 0 && nopos) {
                nopos = false;
                if (m<w) ic.color(1, w-m, 0);
                if (m<w) ic.color(m+1, w, 0);
            }
            int mn = pos, mx = w-m+pos;
            ic.color(mn, mx, v);
        }

        if (nopos) {
            nopos = false;
            if (m<w) ic.color(1, w-m, 0);
            if (m<w) ic.color(m+1, w, 0);
        }

        vector<PII> v;
        for (auto pr: ic.a) v.push_back(pr);

        for (int i=1; i<v.size(); i++) {
            int l = v[i-1].first;
            int r = v[i].first;
            int c = v[i-1].second;
            ans[l] += c;
            ans[r] -= c;
//            cout<<l<<" "<<r<<" "<<c<<endl;
        }
    }

    for (int i=1; i<=w; i++) {
        ans[i] += ans[i-1];
```

```
        cout<<ans[i]<<" ";
    }

}
```

3.16 JavaIO

```
import java.io.OutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.StringTokenizer;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Main {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
        InputReader in = new InputReader(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
        Task solver = new Task();
        solver.solve(1, in, out);
        out.close();
    }

    static class InputReader {
        public BufferedReader reader;
        public StringTokenizer tokenizer;

        public InputReader(InputStream stream) {
            reader = new BufferedReader(new
                InputStreamReader(stream), 32768);
            tokenizer = null;
        }

        public String next() {
            while (tokenizer == null ||
                !tokenizer.hasMoreTokens()) {
                try {
                    tokenizer = new
                        StringTokenizer(reader.readLine());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
            return tokenizer.nextToken();
        }

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

    // Here
    static class Task {
        public void solve(int testNumber, InputReader in,
            PrintWriter out) {
            int n = in.nextInt();
            BigInteger sum = new BigInteger(in.next());

            data[] bids = new data[n];

            for (int i = 0; i < n; i++) {
                bids[i] = new data();
                bids[i].name = in.next();
```

```
            bids[i].bid = new BigInteger(in.next());
        }

        Arrays.sort(bids);
        ArrayList<String> ans = new ArrayList<>();
        for (int i = n - 1; i >= 0; i--) {
            BigInteger bid = bids[i].bid;
            if (sum.compareTo(bid) < 0) continue;
            sum = sum.subtract(bid);
            ans.add(bids[i].name);
        }
        if (sum.compareTo(BigInteger.ZERO) != 0)
            out.println(0);
        else {
            out.println(ans.size());
            for (String name : ans)
                out.println(name);
        }
    }
    public class data implements Comparable{
        BigInteger bid;
        String name;

        public data() {
        }

        public data(BigInteger bid, String name) {
            this.bid = bid;
            this.name = name;
        }

        @Override
        public int compareTo(Object o) {
            return bid.compareTo(((data) o).bid);
        }
```

```
        }
    }
}
```

3.17 Knuth Optimization

```
/**
You have to cut a wood stick into pieces. The most affordable
    company, The Analog Cutting Machinery,
Inc. (ACM), charges money according to the length of the stick
    being cut. Their procedure of work
requires that they only make one cut at a time.
It is easy to notice that different selections in the order of
    cutting can led to different prices. For
example, consider a stick of length 10 meters that has to be cut
    at 2, 4 and 7 meters from one end.
There are several choices. One can be cutting first at 2, then
    at 4, then at 7. This leads to a price of 10
+ 8 + 6 = 24 because the first stick was of 10 meters, the
    resulting of 8 and the last one of 6. Another
choice could be cutting at 4, then at 2, then at 7. This would
    lead to a price of 10 + 4 + 6 = 20, which
is a better price.
Your boss trusts your computer abilities to find out the minimum
    cost for cutting a given stick.
*//

#include <bits/stdc++.h>
#define LL long long
using namespace std;

const int N=1000+7;
LL a[N];
LL dp[N][N];
```

```
int opt[N][N];

const long long INF=1e15;


//solves ZOJ 2860

int main ()
{
    ios::sync_with_stdio(false);
    int l, n;

    while (cin>>l>>n)
    {
        a[0] = 0;
        a[++n] = l;

        for (int i=1; i<n; i++)
            cin>>a[i];

        for (int i=1; i<=n; i++)
            opt[i-1][i] = i-1;

        for (int len = 2; len<=n; len++)
            for (int l=0; l+len<=n; l++)
            {
                int r=l+len;
                int optl = opt[l][r-1];
                int optr = opt[l+1][r];
                dp[l][r] = INF;

                for (int i=optl; i<=optr; i++)
                {
                    LL cost = dp[l][i] + dp[i][r] + (a[r] - a[l]);
                    if (cost < dp[l][r])
                        dp[l][r] = cost,
                        opt[l][r] = i;
                }
            }

        cout<<dp[0][n]<<endl;
    }
}
```

3.18 LinkCutTree

```
/*
 Petar 'PetarV' Velickovic
 Data Structure: Link/cut Tree
 Source:
    https://github.com/PetarV-/Algorithms/blob/master/Data%20Structure
*/

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

/*
 The link/cut tree data structure enables us to efficiently
    handle a dynamic forest of trees.
 It does so by storing a decomposition of the forest into
    "preferred paths", where a path is
 preferred to another when it has been more recently accessed.
 Each preferred path is stored in a splay tree which is keyed by
    depth.

 The tree supports the following operations:
    - make_tree(v): create a singleton tree containing the node
       v
    - find_root(v): find the root of the tree containing v
```

```
    - link(v, w):  connect v to w
                   (precondition: v is root of its own tree,
                   and v and w are not in the same tree!)
    - cut(v):      cut v off from its parent
    - path(v):     access the path from the root of v's tree to v
                   (in order to e.g. perform an aggregate query
                      on that path)

 More complex operations and queries are possible that require
    the data structure
 to be augmented with additional data. Here I will demonstrate
    the LCA(p, q)
 (lowest common ancestor of p and q) operation.

 Complexity:   O(1) for make_tree
               O(log n) amortized for all other operations
*/



const int MAXN = 100007;

struct LinkCutTree {
    struct Node {
        int L, R, P, lazyFlip;
        int PP;
    };

    Node LCT[MAXN];

    void normalize(int u) {
        assert(u != -1);
        if (LCT[u].L != -1) LCT[LCT[u].L].P = u;
        if (LCT[u].R != -1) LCT[LCT[u].R].P = u;
        // (+ update sum of subtree elements etc. if wanted
    }

    // set v as p's left child
    void setLeftChild(int p, int v) {
        LCT[p].L = v;
        normalize(p);
    }
    // set v as p's right child
    void setRightChild(int p, int v) {
        LCT[p].R = v;
        normalize(p);
    }

    void pushLazy(int u) {
        if (!LCT[u].lazyFlip) return;
        swap(LCT[u].L, LCT[u].R);
        LCT[u].lazyFlip = 0;
        if (LCT[u].L != -1) LCT[LCT[u].L].lazyFlip ^= 1;
        if (LCT[u].R != -1) LCT[LCT[u].R].lazyFlip ^= 1;
    }

    void make_tree(int v) {
        LCT[v].L = LCT[v].R = LCT[v].P = LCT[v].PP = -1;
        LCT[v].lazyFlip = 0;
    }

    void rotate(int v) {
        if (LCT[v].P == -1) return;
        int p = LCT[v].P;
        int g = LCT[p].P;
        if (LCT[p].L == v) {
            setLeftChild(p, LCT[v].R);
//          LCT[p].L = LCT[v].R;
//          if (LCT[v].R != -1) {
//              LCT[LCT[v].R].P = p;
```

```
//          }
            setRightChild(v, p);
//          LCT[v].R = p;
//          LCT[p].P = v;
        } else {
            setRightChild(p, LCT[v].L);
//          LCT[p].R = LCT[v].L;
//          if (LCT[v].L != -1) {
//              LCT[LCT[v].L].P = p;
//          }
            setLeftChild(v, p);
//          LCT[v].L = p;
//          LCT[p].P = v;
        }

        LCT[v].P = g;
        if (g != -1) {
            if (LCT[g].L == p) {
                setLeftChild(g, v);
//              LCT[g].L = v;
            } else {
                setRightChild(g, v);
//              LCT[g].R = v;
            }
        }

        // must preserve path-pointer!
        // (this only has an effect when g is -1)
        LCT[v].PP = LCT[p].PP;
        LCT[p].PP = -1;
    }

    void pushEmAll(int v) {
        if (LCT[v].P != -1) pushEmAll(LCT[v].P);
        pushLazy(v);
    }

    void splay(int v) {
//      cout << "splay " << v << endl;
        pushEmAll(v);
        while (LCT[v].P != -1) {
            int p = LCT[v].P;
            int g = LCT[p].P;
            if (g == -1) { // zig
                rotate(v);
            } else if ((LCT[p].L == v) == (LCT[g].L == p)) { //
                zig-zig
                rotate(p);
                rotate(v);
            } else { // zig-zag
                rotate(v);
                rotate(v);
            }
        }
    }

    /// returns v if v is in the root auxiliary tree
    /// otherwise returns the topmost unpreferred edge's parent
    int access(int v) {
        splay(v); // now v is root of its aux. tree
        if (LCT[v].R != -1) {
            LCT[LCT[v].R].PP = v;
            LCT[LCT[v].R].P = -1;
            setRightChild(v, -1);
//          LCT[v].R = -1;
        }

        int ret = v;
        while (LCT[v].PP != -1) {
            int w = LCT[v].PP;
```

```
            splay(w);
            if (LCT[w].PP == -1) ret = w;
            if (LCT[w].R != -1) {
                LCT[LCT[w].R].PP = w;
                LCT[LCT[w].R].P = -1;
            }
            LCT[v].PP = -1; /// ** missed ** Do we really need
                this?
            setRightChild(w, v);
//            LCT[w].R = v;
//            LCT[v].P = w;
            splay(v);
        }
        return ret;
    }

    int find_root(int v) {
        access(v);
        int ret = v;
        while (LCT[ret].L != -1) {
            ret = LCT[ret].L;
            pushLazy(ret);
        }
        access(ret);
        return ret;
    }

    /// make w, parent of v where v is a root
    void link(int v, int w) {// attach v's root to w
        access(w);
        // the root can only have right children in
        // its splay tree, so no need to check
        setLeftChild(v, w);
//        LCT[v].L = w;
//        LCT[w].P = v;
        LCT[w].PP = -1;
    }

    void cut(int v) {
        access(v);
        if (LCT[v].L != -1) {
            LCT[LCT[v].L].P = -1;
            LCT[LCT[v].L].PP = -1;
            setLeftChild(v, -1);
//            LCT[v].L = -1;
        }
    }

    void make_root(int v) {
        access(v);
        int l = LCT[v].L;
        if (l != -1) {
            setLeftChild(v, -1);
            LCT[l].P = -1;
            LCT[l].PP = v;
            LCT[l].lazyFlip ^= 1;
        }
    }

    bool isConnected(int p, int q) {
        return find_root(p)==find_root(q);
    }

    /// assuming p and q is in the same tree
    int LCA(int p, int q) {
        access(p);
        return access(q);
    }
};
```

```
int main()
{
    // This is the code I used for the problem Dynamic LCA
       (DYNALCA)
    // on Sphere Online Judge (SPOJ)

    ios::sync_with_stdio(false);
    cin.tie(0);

    int n, m;
    cin >> n >> m;

    LinkCutTree lct;
    for (int i = 1; i <= n; i++) lct.make_tree(i);

    while (m--) {
        string cmd;
        cin >> cmd;
        if (cmd == "link") {
            int p, q;
            cin >> p >> q;
            lct.link(p, q);
        } else if (cmd == "cut") {
            int p;
            cin >> p;
            lct.cut(p);
        } else if (cmd == "lca") {
            int p, q;
            cin >> p >> q;
            cout << lct.LCA(p, q) << "\n";
        }
    }

    return 0;
}
```

3.19 MCMF

```
/**
Min Cost Max Flow
Complexity: Jani na (Should be fast enough)
Source: Repon da
*/

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

const int maxn = 105;
typedef long long T;
struct Edge {
    int u , v ;
    T cap ,flow , cost;
};

struct MCMF {
    int n,m,s,t;     // Total Number of Node Including S,T
    vector<int> G[maxn];   //Graph
    vector<Edge> E;       // EdgeList
    T d[maxn];           // Distance Array of BeTmanFord
    bool inq[maxn];         // Is node in queue
    int p[maxn];
    T a[maxn];
    const T INF = 1e18+7;

    MCMF(int n) : n(n) {}

    void addEdge( int u , int v , T cap , T cost ){
        E.push_back({u, v , cap , 0 , cost});   // Positive Cost
```

```
    E.push_back({v,u , 0, 0 , -cost } );    // Negative Cost
    m = (int) E.size();
    G[u].push_back(m - 2);
    G[v].push_back(m - 1);
}

bool BelmanFord(T &flow , T &cost ) {
    for(int i = 0; i < n ;i ++) d[i] = INF ;
    memset(inq, 0, sizeof(inq));
    d[s] = 0; inq[s] = 1; p[s] = 0; a[s] = INF;

    queue<int> Q;
    Q.push(s);
    while(!Q.empty() ) {
        int u = Q.front(); Q.pop();
        inq[u] = 0;

        for( int i = 0 ; i < G[u].size() ; i ++ ) {
            Edge &e = E[G[u][i]];
            if( e.cap > e.flow && d[e.v] > d[u] + e.cost ) {
                d[e.v] = d[u] + e.cost;
                p[e.v] = G[u][i];
                a[e.v] = min( a[u] , e.cap - e.flow );

                if( inq[e.v ] == 0 ) {
                    Q.push(e.v);
                    inq[e.v] = 1;
                }
            }
        }
    }

    if( d[t] == INF ) return false; // No augmenting Path
    flow = a[t];
    cost = d[t] ;            // Unit cost
        int u = t ;
        while( u != s ) {
            E[p[u]].flow += a[t];
            E[p[u]^1].flow -= a[t];
            u = E[p[u]].u;
        }
        return true;
    }

    pair<T,T> Mincost (int s, int t) {
        this->s=s,this->t=t;
        T Mcost = 0;
        T Flow = 0;
        T f = 0 ;   // For Each CaT , The flow
        T d = 0;     // Shortest Distance / Cost Per Flow

        while(BelmanFord(f,d)) {
            Flow += f;
            Mcost += f *d ;
        }
        return make_pair(Flow, Mcost);
    }
};


///Solves Uva Data flow :|
const int M = 5000+5;
int u[M], v[M], c[M];

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    while (cin>>n>>m) {
```

```
        MCMF solver(n+1);
        for (int i=0; i<m; i++)
            cin>>u[i] >> v[i] >> c[i];

        int k, d;
        cin>>d>>k;

        for (int i=0; i<m; i++)
            solver.addEdge(u[i], v[i], k, c[i]),
            solver.addEdge(v[i], u[i], k, c[i]);

        solver.addEdge(0, 1, d, 0);
        pair<T, T> ans = solver.Mincost(0, n);
        if (ans.first != d) cout<<"Impossible.\n";
        else                cout<<ans.second<<"\n";
    }
}
```

3.20 NTT

```
/**
Iterative Implementation of Number Theoretic Transform
Complexity: O(N log N)
Slower than regular fft

Possible Optimizations:
1. Remove leading zeroes
2. You may use ints everywhere, but be careful with overflow

Suggested mods (mod, root, inv, pw) :
7340033, 5, 4404020, 1<<20
13631489, 11799463,6244495, 1<<20
23068673, 177147,17187657, 1<<21
463470593, 428228038, 182429, 1<<21
415236097, 73362476, 247718523, 1<<22
918552577, 86995699, 324602258, 1<<22
998244353, 15311432, 469870224, 1<<23
167772161, 243, 114609789, 1<<25
469762049, 2187, 410692747, 1<<26

If required mod is not above, use nttdata function OFFLINE
**/

#include<bits/stdc++.h>
using namespace std;
typedef long long LL;

LL power(LL a, LL p, LL mod) {
    if (p==0)  return 1;
    LL ans = power(a, p/2, mod);
    ans = (ans * ans)%mod;
    if (p%2)   ans = (ans * a)%mod;
    return ans;
}

/** Find primitive root of p assuming p is prime.
if not, we must add calculation of phi(p)
Complexity : O(Ans * log (phi(n)) * log n + sqrt(p)) (if exists)
             O(p * log (phi(n)) * log n + sqrt(p)) (if does not
                 exist)
Returns -1 if not found
*/
int primitive_root(int p) {
    vector<int> factor;
    int phi = p-1, n = phi;

    for (int i=2; i*i<=n; ++i)
        if (n%i == 0) {
            factor.push_back (i);
```

```
            while (n%i==0) n/=i;
        }

    if (n>1) factor.push_back(n);

    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (int i=0; i<factor.size() && ok; ++i)
            ok &= power(res, phi/factor[i], p) != 1;
        if (ok) return res;
    }
    return -1;
}

/**
  Generates necessary info for NTT (for offline usage :3)
  returns maximum k such that 2^k divides mod
  ntt can only be applied for arrays not larger than this size
  mod MUST BE PRIME!!!!!

  We use that fact that primes the form p=c*2^k+1,
  there always exist the 2^k-th root of unity.
  It can be shown that g^c is such a 2^k-th root
  of unity, where g is a primitive root of p.
*/

int nttdata(int mod, int &root, int &inv, int &pw) {
    int c = 0, n = mod-1;
    while (n%2 == 0) c++, n/=2;
    pw = (mod-1)/n;
    int g = primitive_root(mod);
    root = power(g, n, mod);
    inv = power(root, mod-2, mod);
    return c;
}
```

```
struct NTT
{
    int N;
    vector<int> perm;

    int mod, root, inv, pw;

    NTT(int mod, int root, int inv, int pw) :
        mod(mod), root(root), inv(inv), pw(pw) {}

    void precalculate() {
        perm.resize(N);
        perm[0] = 0;

        for (int k=1; k<N; k<<=1) {
            for (int i=0; i<k; i++) {
                perm[i] <<= 1;
                perm[i+k] = 1 + perm[i];
            }
        }
    }

    void fft(vector<LL> &v, bool invert = false) {
        if (v.size() != perm.size()) {
            N = v.size();
            assert(N && (N&(N-1)) == 0);
            precalculate();
        }

        for (int i=0; i<N; i++)
            if (i < perm[i])
                swap(v[i], v[perm[i]]);

        for (int len = 2; len <= N; len <<= 1) {
```

```
        LL factor = invert ? inv : root;
        for (int i = len; i < pw; i <<= 1)
            factor = (factor * factor) % mod;

        for (int i=0; i<N; i+=len) {
            LL w = 1;
            for (int j=0; j<len/2; j++) {
                LL x = v[i+j], y = (w * v[i+j+len/2])%mod;
                v[i+j] = x+y;
                if (v[i+j] >= mod) v[i+j] -= mod;
                v[i+j+len/2] = x-y+mod;
                if (v[i+j+len/2] >= mod) v[i+j+len/2] -= mod;
                w = (w * factor)%mod;
            }
        }
    }
    if (invert) {
        LL n1 = power(N, mod-2, mod);
        for (LL &x : v) x=(x*n1)%mod;
    }
  }

  vector<LL> multiply(vector<LL> a, vector<LL> b) {
    int n = 1;
    while (n < a.size()+ b.size()) n<<=1;
    a.resize(n);
    b.resize(n);

    fft(a);
    fft(b);
    for (int i=0; i<n; i++) a[i] = (a[i] * b[i])%mod;
    fft(a, true);
    return a;
  }
};
```

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

    int mod = 7340033;
    int root, inv, pw;
    int k = nttdata(mod, root, inv, pw);

    vector<LL> left = {1,4,6,4,1};
    NTT ntt(mod, root, inv, pw);
    vector<LL> ans = ntt.multiply(left, left);
    for (LL x: ans)  cout<<x<<" ";
}
```

3.21 PSTree

```
/*
    Persistent Segment Tree Basic Code
    Memory: O(N+QlogN)
    Point Update, Range Query

    solves SPOJ MKTHNUM
    given a query (i,j,k)
    find the kth element in the sorted subarray [i,j]
*/

#include<bits/stdc++.h>
using namespace std;
const int MAXX = 100000;

struct Node {
    int cnt;
    Node *bam, *dan;
```

```
    Node(int c = 0, Node *b = NULL, Node *d = NULL) : cnt(c),
        bam(b), dan(d) {}

    void build(int l, int r) {
        if (l==r) return;
        int mid = (l+r)/2;
        bam = new Node();
        dan = new Node();
        bam->build(l, mid);
        dan->build(mid+1, r);
    }

    Node* update(int l, int r, int idx, int v) {
        if (idx < l || r < idx) return this;
        if (l==r) {
            Node *ret = new Node(cnt);
            ret->cnt += v;
            return ret;
        }

        Node* ret = new Node();
        int mid = (l+r)/2;
        ret->bam = bam->update(l, mid, idx, v);
        ret->dan = dan->update(mid+1, r, idx, v);
        ret->cnt = ret->bam->cnt + ret->dan->cnt;
        return ret;
    }
}* root[MAXX+7];

int getKth(Node* R, Node* L, int l, int r, int k) {
    assert(R->cnt+L->cnt >= k);
    if (l==r) {
        return l;
    }
    int bk = R->bam->cnt-L->bam->cnt;
    int mid = (l+r)/2;
    if (k <= bk) return getKth(R->bam, L->bam, l, mid, k);
    else return getKth(R->dan, L->dan, mid+1, r, k-bk);
}

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

    int n, m;
    cin >> n >> m;

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

    sort(s.begin(), s.end());
    map<int, int>mp;
    for (int i = 0; i < n; i++) mp[s[i]] = i;

    root[0] = new Node();
    root[0]->build(0, MAXX-1);

    for (int i = 1; i <= n; i++) {
        root[i] = root[i-1]->update(0, MAXX-1, mp[v[i-1]], 1);
    }

    while (m--) {
        int i, j, k;
        cin >> i >> j >> k;
        assert(1 <= k && k <= j-i+1);
        int x = getKth(root[j], root[i-1], 0, MAXX-1, k);
```

```
        cout << s[x] << '\n';
    }


    return 0;
}
```

3.22 PSTreeWithLazy

```
/*
    Persistent Segment Tree with Lazy Propagation

    The following code solves the problem:
        1. Add a, a+b, a+2b, ... to [l, r]
        2. Find the sum of [l, r]
*/

#include<bits/stdc++.h>
using namespace std;
typedef long long LL;

struct Node{
    LL sum; LL lazyA = 0, lazyB = 0; bool hasLazy = false;
    Node* bam, *dan;
    Node(LL s = 0, Node *b = NULL, Node *d = NULL) : sum(s),
        bam(b), dan(d) { }
    void init(int l, int r) {
        if (l==r) return;
        int mid = (l+r)/2;
        bam = new Node();
        dan = new Node();
        bam->init(l, mid);
        dan->init(mid+1, r);
    }
    void propagate(int l, int r) {
        if (!hasLazy) return;        ///important
        LL n = r-l+1;
        sum += n * lazyA;
        sum += ((n*(n-1))/2) * lazyB;
        if (l!=r) {
            bam = new Node(*bam);
            dan = new Node(*dan);
            int mid = (l+r)/2;
            bam->lazyA += lazyA;
            bam->lazyB += lazyB;
            dan->lazyA += lazyA+lazyB*(mid-l+1);
            dan->lazyB += lazyB;
            bam->hasLazy = true;
            dan->hasLazy = true;
        }
        lazyA = 0;
        lazyB = 0;
        hasLazy = false;
    }

    Node* update(int l, int r, int x, int y, LL a, LL b) {
        propagate(l, r);
        if (r < x || y < l) return this;
        if (x <= l && r <= y) {
            Node* rt = new Node(*this);
            rt->lazyA += a+(l-x)*b;
            rt->lazyB += b;
            rt->hasLazy = true;
            rt->propagate(l, r);
            return rt;
        }
        int mid = (l+r)/2;
        Node* rt = new Node();
        rt->bam = bam->update(l, mid, x, y, a, b);
```

```
        rt->dan = dan->update(mid+1, r, x, y, a, b);
        rt->sum = rt->bam->sum + rt->dan->sum;
        return rt;
    }
    LL query(int l, int r, int x, int y) {
        propagate(l, r);
        if (x <= l && r <= y) return sum;
        LL ans = 0;
        int mid = (l+r)/2;
        if (x <= mid) ans += bam->query(l, mid, x, y);
        if (mid < y) ans += dan->query(mid+1, r, x, y);
        return ans;
    }
};

int main()
{


    return 0;
}
```

3.23 Pollard-Rho

```
/**
Range: 10^18 (tested), should be okay up to 2^63-1

miller_rabin(n)
    returns 1 if prime, 0 otherwise
    Magic bases:
        n < 4,759,123,141      3 :  2, 7, 61
        n < 1,122,004,669,633  4 :  2, 13, 23, 1662803
        n < 3,474,749,660,383  6 :  2, 3, 5, 7, 11, 13
        n < 2^64               7 : 2, 325, 9375, 28178, 450775,
            9780504, 1795265022
    Identifies 70000 18 digit primes in 1 second on Toph

pollard_rho(n):
    If n is prime, returns n
    Otherwise returns a proper divisor of n
    Able to factorize ~120 18 digit semiprimes in 1 second on
        Toph
    Able to factorize ~700 15 digit semiprimes in 1 second on
        Toph

Note: for factorizing large number, do trial division upto
      cubic root and then call pollard rho once.
*/


#include<bits/stdc++.h>
#define LL long long
using namespace std;

LL mult(LL a, LL b, LL mod) {
    assert(b < mod && a < mod);
    long double x = a;
    uint64_t c = x * b / mod;
    int64_t r = (int64_t)(a * b - c * mod) % (int64_t)mod;
    return r < 0 ? r + mod : r;
}

LL power(LL x, LL p, LL mod){
        LL s=1, m=x;
        while(p) {
                if(p&1) s = mult(s, m, mod);
                p>>=1;
                m = mult(m, m, mod);
```

```
        }
        return s;
}

bool witness(LL a, LL n, LL u, int t){
        LL x = power(a,u,n);
        for(int i=0; i<t; i++) {
                LL nx = mult(x, x, n);
                if (nx==1 && x!=1 && x!=n-1) return 1;
                x = nx;
        }
        return x!=1;
}

vector<LL> bases = {2, 325, 9375, 28178, 450775, 9780504,
    1795265022};
bool miller_rabin(LL n) {
        if (n<2)   return 0;
        if (n%2==0)  return n==2;

        LL u = n-1;
        int t = 0;
        while(u%2==0)  u/=2, t++; // n-1 = u*2^t

        for (LL v: bases) {
                LL a = v%(n-1) + 1;
                if(witness(a, n, u, t)) return 0;
    }
        return 1;
}

LL gcd(LL u, LL v) {
    if (u == 0) return v;
    if (v == 0) return u;
    int shift = __builtin_ctzll(u | v);
    u >>= __builtin_ctzll(u);
    do {
        v >>= __builtin_ctz(v);
        if (u > v) swap(u, v);
        v = v - u;
    } while (v);
    return u << shift;
}

mt19937_64
    rng(chrono::steady_clock::now().time_since_epoch().count());
LL pollard_rho(LL n) {
    if (n==1)             return 1;
    if (n%2==0)           return 2;
        if (miller_rabin(n))  return n;

    while (true) {
        LL x = uniform_int_distribution<LL>(1, n-1)(rng);
        LL y = 2, res = 1;
        for (int sz=2; res == 1; sz*=2) {
            for (int i=0; i<sz && res<=1; i++) {
                x = mult(x, x, n) + 1;
                res = gcd(abs(x-y), n);
            }
            y = x;
        }
        if (res!=0 && res!=n) return res;
    }
}


///Solves UVA - 11476
const int MX = 2.2e5+7;
vector<int> primes;
bool isp[MX];
```

```
void sieve() {
    fill(isp+2, isp+MX, 1);
    for (int i=2; i<MX; i++)
        if (isp[i]) {
            primes.push_back(i);
            for (int j=2*i; j<MX; j+=i)
                isp[j] = 0;
        }
}

vector<LL> factorize(LL x) {
    vector<LL> ans;
    for (int p: primes) {
        if (1LL*p*p*p > x) break;
        while (x%p==0) {
            x/=p;
            ans.push_back(p);
        }
    }
    if (x > 1) {
        LL z = pollard_rho(x);
        ans.push_back(z);
        if (z < x) ans.push_back(x/z);
    }
    return ans;
}

int main()
{
//    freopen("in.txt", "r", stdin);
//    freopen("out-mine.txt", "w", stdout);
    sieve();
    int t;
    cin>>t;
    while (t--) {
        long long x;
        if (!(cin>>x))  break;
        vector<LL> ans = factorize(x);
        sort(ans.begin(), ans.end());

        vector<pair<LL, int>> ff;
        for (LL x: ans) {
            if (ff.size() && ff.back().first == x)
                ff.back().second++;
            else    ff.push_back({x, 1});
        }
        cout<<x<<" =";
        bool first = true;
        for (auto pr: ff) {
            if (!first) cout<<" *";
            first = false;
            cout<<" "<<pr.first;
            if (pr.second > 1)cout<<"^"<<pr.second;
        }
        cout<<endl;
    }
}
```

3.24 SegmentTree

```
/**
  Solves SPOJ HORRIBLE - Horrible Queries
  Range Increment Update, Range Sum Query
**/

#include <bits/stdc++.h>
#define LL long long
```

```
using namespace std;

const int N = 1e5+7;
int a[N];
LL tr[4*N];
LL lz[4*N];

///1. Merge left and right
LL combine (LL left, LL right) {
    return left + right;
}

///2. Push lazy down and merge lazy
void propagate(int u, int st, int en) {
    tr[u] += (en-st+1)*lz[u];
    if (st!=en) {
        lz[2*u] += lz[u];
        lz[2*u+1] += lz[u];
    }
    lz[u] = 0;
}


void build(int u, int st, int en) {
    if (st==en) {
        tr[u] = a[st];        ///3. Initialize
        lz[u] = 0;
    }
    else {
        int mid = (st+en)/2;
        build(2*u, st, mid);
        build(2*u+1, mid+1, en);
        tr[u] = combine(tr[2*u], tr[2*u+1]);
        lz[u] = 0;            ///3. Initialize
    }
}

void update(int u, int st, int en, int l, int r, int x) {
    propagate(u, st, en);
    if (r<st || en<l) return;
    else if (l<=st && en<=r) {
        lz[u] += x;           ///4. Merge lazy
        propagate(u, st, en);
    }
    else {
        int mid = (st+en)/2;
        update(2*u, st, mid, l, r, x);
        update(2*u+1, mid+1, en, l, r, x);
        tr[u] = combine(tr[2*u], tr[2*u+1]);
    }
}

LL query(int u, int st, int en, int l, int r) {
    propagate(u, st, en);
    if (r<st || en<l) return 0;      /// 5. Proper null value
    else if (l<=st && en<=r) return tr[u];
    else {
        int mid = (st+en)/2;
        return combine(query(2*u, st, mid, l, r), query(2*u+1,
            mid+1, en, l, r));
    }
}

void debug(int u, int st, int en) {
    cout<<"--->"<<u<<" "<<st<<" "<<en<<" "<<tr[u]<<"
        "<<lz[u]<<endl;
    if (st==en) return;
    int mid = (st+en)/2;
    debug(2*u, st, mid);
    debug(2*u+1, mid+1, en);
```

```
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int t;
    cin>>t;

    while (t--) {
        int n, q;
        cin>>n>>q;
        build(1,1,n);

        while (q--) {
            int c, l, r;
            cin>>c>>l>>r;

            if (c==1)  cout<<query(1, 1, n, l, r)<<"\n";
            else {
                int x;
                cin>>x;
                update(1, 1, n, l, r, x);
            }
//            debug(1, 1, n);
        }
    }
}
```

3.25 SqrtField

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

const int M = 1e9+7;
typedef long long LL;

LL mod(LL x) {
    LL ans = x%M;
    if (ans < 0)   ans += M;
    return ans;
}

template <LL X>
struct SqrtField {
    LL a, b, c;    /// (a + b*sqrt(X))/c;
    SqrtField(LL A=0, LL B=0, LL C=1) : a(A), b(B), c(C) {}


    SqrtField operator+(const SqrtField &y) {
        return SqrtField(mod(a*y.c + y.a*c), mod(b*y.c + y.b*c),
            mod(c*y.c));
    }

    SqrtField operator-(const SqrtField &y) {
        return SqrtField(mod(a*y.c - y.a*c), mod(b*y.c - y.b*c),
            mod(c*y.c));
    }

    SqrtField operator*(const SqrtField &y) {
        return SqrtField(mod(a*y.a + X*y.b*b), mod(a*y.b +
            b*y.a), mod(c*y.c));
    }

    SqrtField operator/(const SqrtField &y) {
        LL A = mod(a*y.a - X*y.b*b);
        LL B = mod(b*y.a - a*y.b);
        LL C = mod(y.a*y.a - X*y.b*y.b);
        A = mod(A*y.c);
```

```
        B = mod(B*y.c);
        C = mod(C*c);
        return SqrtField(A,B,C);
    }
};

template<LL X>
ostream& operator<<(ostream &os, const SqrtField<X> &x) {
    return os<<"("<<x.a<<"+"<<x.b<<"V"<<X<<")/"<<x.c;
}

int main() {
    SqrtField<2> c(5);        ///5
    SqrtField<2> b(0, 1);     ///3 sqrt(2)
    SqrtField<2> a(3,7,2);    ///(3+7*sqrt(2))/2

    cout<<a+b<<" "<<a-b<<" "<<a*b<<" "<<a/c<<endl;
    cout<<a*2<<" "<<a/2<<" "<<a+2<<" "<<a-1<<endl;
}
```

3.26 Suffix array

```
/**
 Suffix Array implementation with count sort.
 Source: E-MAXX
 Running time:
    Suffix Array Construction: O(NlogN)
    LCP Array Construction: O(NlogN)
    Suffix LCP: O(logN)
**/

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

typedef pair<int, int> PII;
typedef vector<int> VI;

/// Equivalence Class INFO
vector<VI> c;
VI sort_cyclic_shifts(const string &s)
{
    int n = s.size();
    const int alphabet = 256;
    VI p(n), cnt(alphabet, 0);

    c.clear();
    c.emplace_back();
    c[0].resize(n);

    for (int i=0; i<n; i++)      cnt[s[i]]++;
    for (int i=1; i<alphabet; i++) cnt[i] += cnt[i-1];
    for (int i=0; i<n; i++)      p[--cnt[s[i]]] = i;

    c[0][p[0]] = 0;
    int classes = 1;

    for (int i=1; i<n; i++) {
        if (s[p[i]] != s[p[i-1]]) classes++;
        c[0][p[i]] = classes - 1;
    }

    VI pn(n), cn(n);
    cnt.resize(n);

    for (int h=0; (1<<h) < n; h++) {
        for (int i=0; i<n; i++) {
            pn[i] = p[i] - (1<<h);
            if (pn[i] < 0) pn[i] += n;
        }
```

```
        fill(cnt.begin(), cnt.end(), 0);

        /// radix sort
        for (int i = 0; i < n; i++)      cnt[c[h][pn[i]]]++;
        for (int i = 1; i < classes; i++) cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)   p[--cnt[c[h][pn[i]]]] =
            pn[i];

        cn[p[0]] = 0;
        classes = 1;

        for (int i=1; i<n; i++) {
            PII cur = {c[h][p[i]], c[h][(p[i] + (1<<h))%n]};
            PII prev = {c[h][p[i-1]], c[h][(p[i-1] + (1<<h))%n]};
            if (cur != prev)  ++classes;
            cn[p[i]] = classes - 1;
        }
        c.push_back(cn);
    }
    return p;
}

VI suffix_array_construction(string s)
{
    s += "!";
    VI sorted_shifts = sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}

/// LCP between the ith and jth (i != j) suffix of the STRING
int suffixLCP(int i, int j)
{
    assert(i != j);
    int log_n = c.size()-1;
    int ans = 0;
    for (int k = log_n; k >= 0; k--) {
        if (c[k][i] == c[k][j]) {
            ans += 1 << k;
            i += 1 << k;
            j += 1 << k;
        }
    }
    return ans;
}

VI lcp_construction(const string &s, const VI &sa)
{
    int n = s.size();
    VI rank(n, 0);
    VI lcp(n-1, 0);

    for (int i=0; i<n; i++)
        rank[sa[i]] = i;

    for (int i=0, k=0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }

        int j = sa[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k]) k++;
        lcp[rank[i]] = k;
        if (k) k--;
    }
    return lcp;
}
```

```
const int MX = 1e6+7, K = 20;
int lg[MX];

void pre()
{
    lg[1] = 0;
    for (int i=2; i<MX; i++)
        lg[i] = lg[i/2]+1;
}

struct RMQ{
    int N;
    VI v[K];
    RMQ(const VI &a) {
        N = a.size();
        v[0] = a;

        for (int k = 0; (1<<(k+1)) <= N; k++) {
            v[k+1].resize(N);
            for (int i = 0; i-1+(1<<(k+1)) < N; i++) {
                v[k+1][i] = min(v[k][i], v[k][i+(1<<k)]);
            }
        }
    }

    int findMin(int i, int j) {
        int k = lg[j-i+1];
        return min(v[k][i], v[k][j+1-(1<<k)]);
    }
};

/// Solves SPOJ-SARRAY. Given a string, find its suffix array
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);

    string s;
    cin>>s;

    vector<int> sa = suffix_array_construction(s);
    for (int i: sa)
        cout<<i<<"\n";
}
```

3.27 SuffixAutomata

```
/**
    Linear Time Suffix Automata contruction.
    Build Complexity: O(n * alphabet)

    To achieve better build complexity and linear space,
    use map for transitions.
**/


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


const int MAXN = 1e5+7, ALPHA = 26;
int len[2*MAXN], link[2*MAXN], nxt[2*MAXN][ALPHA];
int sz;
int last;

void sa_init() {
    memset(nxt, -1, sizeof nxt);

    len[0] = 0;
```

```
    link[0] = -1;
    sz = 1;
    last = 0;
}

void add(char ch) {
    int c = ch-'a';

    int cur = sz++;                          //create new node
    len[cur] = len[last]+1;

    int u = last;
    while (u != -1 && nxt[u][c] == -1) {
        nxt[u][c] = cur;
        u = link[u];
    }

    if (u == -1) {
        link[cur] = 0;
    }
    else {
        int v = nxt[u][c];
        if (len[v] == len[u]+1) {
            link[cur] = v;
        }
        else {
            int clone = sz++;                //create node by
                cloning
            len[clone] = 1 + len[u];
            link[clone] = link[v];

            for (int i=0; i<ALPHA; i++)
                nxt[clone][i] = nxt[v][i];

            while (u != -1 && nxt[u][c] == v) {
                nxt[u][c] = clone;
                u = link[u];
            }

            link[v] = link[cur] = clone;
        }
    }
    last = cur;
}

vector<int> edge[2*MAXN];
///Optional, Call after adding all characters
void makeEdge() {
    for (int i=0; i<sz; i++) {
        edge[i].clear();
        for (int j=0; j<ALPHA; j++)
            if (nxt[i][j]!=-1)
                edge[i].push_back(j);
    }
}

// The following code solves SPOJ SUBLEX
// Given a string S, you have to answer some queries:
// If all distinct substrings of string S were sorted
// lexicographically, which one will be the K-th smallest?

long long dp[2*MAXN];
bool vis[2*MAXN];

void dfs(int u) {
    if (vis[u]) return;
    vis[u] = 1;
    dp[u] = 1;
    for (int i: edge[u]) {
        if (nxt[u][i] == -1)  continue;
```

```
        dfs(nxt[u][i]);
        dp[u] += dp[nxt[u][i]];
    }
}

void go(int u, long long rem, string &s) {
    if (rem == 1) return;
    long long sum = 1;
    for (int i: edge[u]) {
        if (nxt[u][i] == -1) continue;
        if (sum + dp[nxt[u][i]] < rem) {
            sum += dp[nxt[u][i]];
        }
        else {
            s += ('a' + i);
            go(nxt[u][i], rem-sum, s);
            return;
        }
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    string s;
    cin>>s;

    sa_init();
    for (char c: s) add(c);
    makeEdge();


    dfs(0);
    int q;
    cin>>q;

    while (q--) {
        long long x;
        cin>>x;
        x++;
        string s;
        go(0, x, s);
        cout<<s<<"\n";
    }
}
```

3.28 Treap

```
/*
Solves CF 863D
Given an array
2 types of updates
    1. Reverse a segment
    2. Cyclically shift a segment to the right
*/

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

typedef struct item * pitem;
struct item {
    int prior, value, cnt;
    bool rev;

    item(int value):prior(rand()), value(value) {
        cnt = 0;
        rev = 0;
        l = r = nullptr;
```

```
    }

    pitem l, r;
};

namespace Treap {

    int cnt (pitem it) {
        return it != nullptr? it->cnt : 0;
    }

    void upd_cnt (pitem it) {
        if (it!=nullptr)
            it->cnt = cnt(it->l) + cnt(it->r) + 1;
    }

    void push (pitem it) {
        if (it != nullptr && it->rev == true) {
            it->rev = false;
            swap (it->l, it->r);
            if (it->l) it->l->rev ^= true;
            if (it->r) it->r->rev ^= true;
        }
    }

    void merge (pitem & t, pitem l, pitem r) {
        push (l);
        push (r);
        if (l==nullptr || r==nullptr)
            t = (l!=nullptr) ? l : r;
        else if (l->prior > r->prior)
            merge (l->r, l->r, r), t = l;
        else
            merge (r->l, l, r->l), t = r;
        upd_cnt (t);
    }

    void split (pitem t, pitem & l, pitem & r, int key, int add
        = 0) {
        if (t==nullptr) {
            l = r = nullptr;
            return;
        }
        push (t);
        int cur_key = add + cnt(t->l);

        if (key <= cur_key)
            split (t->l, l, t->l, key, add), r = t;
        else
            split (t->r, t->r, r, key, add + 1 + cnt(t->l)), l =
                t;
        upd_cnt (t);
    }

    void reverse (pitem &t, int l, int r) {
        pitem t1, t2, t3;
        split (t, t1, t2, l);
        split (t2, t2, t3, r-l+1);
        t2->rev ^= true;
        merge (t, t1, t2);
        merge (t, t, t3);
    }


    void insert (pitem & t, int key, int value) {

        pitem x = new item(value);

        pitem L, R;
        split(t, L, R, key);
```

```
    merge(L, L, x);
    merge(t, L, R);

    upd_cnt(t);
}

void erase (pitem & t, int key) {
    assert(cnt(t) > key);

    pitem L, MID, R;
    split(t, L, MID, key);
    split(MID, MID, R, 1);
    merge(t, L, R);

    delete MID;

    upd_cnt(t);
}


void cyclicShift(pitem &t, int l, int r)
{
    assert(0 <= l && r < cnt(t));

    pitem L, MID, R;
    split(t, L, MID, r);
    split(MID, MID, R, 1);
    merge(t, L, R);

    assert(MID!=nullptr);
    insert(t, l, MID->value);
    delete MID;
    upd_cnt(t);
}
```

```
    void output (pitem t, vector< int >&v) {
        if (t==nullptr) return;
        push (t);
        output (t->l, v);
        v.push_back(t->value);
        output (t->r, v);
    }

    void output2 (pitem t) {
        if (t==nullptr) return;
        push (t);
        cout << "(";
        output2 (t->l);
        cout << (t->value);
        output2 (t->r);
        cout << ")";
    }
}

int main()
{
    std::ios_base::sync_with_stdio(false);

    srand(time(0));

    pitem tr = nullptr;

    int n, q, m;
    cin >> n >> q >> m;

    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        Treap::insert(tr, i, x);
    }
```

```
    while (q--) {
        int t, l, r;
        cin >> t >> l >> r;
        l--; r--;
        if (l==r) continue;
        if (t==1) Treap::cyclicShift(tr, l, r);
        else Treap::reverse(tr, l, r);
    }


    vector< int >b;
    Treap::output(tr, b);

    while (m--) {
        int x;
        cin >> x; x--;
        cout << b[x] << " ";
    }


    return 0;
}
```

3.29 TwoSAT

```
/**
    2 SAT
    Complexity: Linear
    Source: Repon da
**/

#include <bits/stdc++.h>
using namespace std;
const int N = 101;
// 1 based Indexing.
// call init(n) -> n is number of variable
struct TwoSAT{
    int n,nn;
    vector<int> G[2*N] , R[2*N], top; //G - original graph,R
        -Reverse graph,top-topological
    int col[2*N],vis[2*N],ok[2*N]; // col- sccNo, ok -> value
        assignment.
    void init(int n)
    {
        this->n=n;
        this->nn = n+n;
        for(int i= 0; i <= nn; i ++) G[i].clear(), R[i].clear(),
            col[i] = 0, vis[i] = 0, ok[i] = 0;
        top.clear();
    }

    int inv(int no) { return (no <= n) ? no + n: no - n; }
    void add(int u,int v)
    {
        G[u].push_back(v);
        R[v].push_back(u);
    }
    void OR(int u,int v){
        add(inv(u), v);
        add(inv(v), u);
    }
    void AND(int u,int v){
        add(u,v);
        add(v,u);
    }
    void XOR(int u,int v){
        add(inv(v), u);
```

```
    add(u, inv(v));
    add(inv(u), v);
    add(v, inv(u));
}
void XNOR(int u,int v){
    add(u,v);
    add(v,u);
    add(inv(u), inv(v));
    add(inv(v), inv(u));
}
void force_true(int x) {
    add(inv(x),x);
}
void force_false(int x) {
    add(x,inv(x));
}
void dfs(int u)
{
    vis[u] = 1;
    for(int i = 0; i < G[u].size(); i ++ ) {
        int v = G[u][i];
        if(vis[v] == 0 ) dfs(v);
    }
    top.push_back(u);
}

void dfs1(int u,int color)
{
    col[u] = color;
    if(u <= n) ok[u] = 1;
    else ok[u - n] = 0;
    for(int i = 0 ;i < R[u].size(); i ++ ) {
        if(col[R[u][i]] == 0 )dfs1(R[u][i], color);
    }
}
    void FindScc()
    {

        for(int i = 1; i <= nn ; i ++ ) {
            if(vis[i] == 0 ) dfs(i);
        }

        int color = 0;
        reverse(top.begin(), top.end());
        for(auto u: top) {
            if(col[u] == 0) dfs1(u , ++ color);
        }


    }
    void solve()
    {
        FindScc();
        for(int i=1;i<=n;i++){
            if(col[i] == col[n + i]) {
                printf("Impossible\n");
                return;
            }
        }
        vector<int> v;
        for(int i = 1; i <= n; i ++ ) {
            if(ok[i]) v.push_back(i); // All 1'
        }
        cout << v.size() << endl;
        for(auto x: v) printf("%d ",x); puts("");

    }
};

int main()
```

```
{

    return 0;
}
```

3.30 Z

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

vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for (int i=1; i<n; i++) {
        if (i<=r)  z[i] = min(r-i+1, z[i-l]);
        while (i+z[i]<n && s[i+z[i]] == s[z[i]]) z[i]++;
        if (i+z[i]-1>r) l = i, r = i+z[i]-1;
    }
    return z;
}


///Solves SPOJ QUERYSTR
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    int t;
    cin>>t;
    while (t--) {
        string s;
        cin>>s;
        int n = s.size();
        reverse(s.begin(), s.end());
        vector<int> ans = z_function(s);
        ans[0] = n;

        int q;
        cin>>q;
        while (q--) {
            int x;
            cin>>x;
            x = n-x;
            cout<<ans[x]<<"\n";
        }
    }
}
```

3.31 betterUnorderdMap

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

const int N = 2e5;

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
            chrono::steady_clock::now().time_since_epoch().count();
```

```
        return splitmix64(x + FIXED_RANDOM);
    }
};

void insert_numbers(long long x) {
    clock_t begin = clock();
    unordered_map<long long, int, custom_hash> numbers;

    for (int i = 1; i <= N; i++)
        numbers[i * x] = i;

    long long sum = 0;

    for (auto &entry : numbers)
        sum += (entry.first / x) * entry.second;

    printf("x = %lld: %.3lf seconds, sum = %lld\n", x, (double)
        (clock() - begin) / CLOCKS_PER_SEC, sum);
}

int main() {
    insert_numbers(107897);
    insert_numbers(126271);
}
```

3.32 dynamicCHT

```
/*
struct line :
    m -> slope
    c -> y intercept
    xleft -> stores the left point of the line till which it is
        optimal
    type = 0 -> lines to be stored
    type = 1 -> query type

double meet(line x, line y)
    returns the x coordinate of intersection of x and y

struct cht:
    cht() -> constructor
    void addline(m, c) -> adds line(m, c) to hull
    long long getbest(long long x) -> finds optimal line for
        given x and
                                        returns value of y(x) on
                                            that line

cht stores the lower hull of lines
to store the upper hull of lines
    negate m and c in the constructor of line
    negate the output of getbest(x)
*/



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

/// CHT code courtesy : rajat1603

struct line{
        long long m , c;
        double xleft;
        bool type;
        line(long long _m , long long _c){
                m = _m;
                c = _c;
                type = 0;
        }
```

```
	bool operator < (const line &other) const{
		if(other.type){
			return xleft < other.xleft;
		}
		return m > other.m;
	}
};
double meet(line x , line y){
	return 1.0 * (y.c - x.c) / (x.m - y.m);
}
struct cht{	/// stores the lower hull of lines
	set < line > hull;
	cht(){
		hull.clear();
	}
	typedef set < line > :: iterator ite;
	bool hasleft(ite node){
		return node != hull.begin();
	}
	bool hasright(ite node){
		return node != prev(hull.end());
	}
	void updateborder(ite node){
		if(hasright(node)){
			line temp = *next(node);
			hull.erase(temp);
			temp.xleft = meet(*node , temp);
			hull.insert(temp);
		}
		if(hasleft(node)){
			line temp = *node;
			temp.xleft = meet(*prev(node) , temp);
			hull.erase(node);
			hull.insert(temp);
		}
		else{
			line temp = *node;
			hull.erase(node);
			temp.xleft = -1e18;
			hull.insert(temp);
		}
	}
	bool useless(line left , line middle , line right){
		return meet(left , middle) > meet(middle , right);
	}
	bool useless(ite node){
		if(hasleft(node) && hasright(node)){
			return useless(*prev(node) , *node ,
			    *next(node));
		}
		return 0;
	}
	void addline(long long m , long long c){
		line temp = line(m, c);
		auto it = hull.lower_bound(temp);
		if(it != hull.end() && it -> m == m){
			if(it -> c > c){
				hull.erase(it);
			}
			else{
				return;
			}
		}
		hull.insert(temp);
		it = hull.find(temp);
		if(useless(it)){
			hull.erase(it);
			return;
		}
		while(hasleft(it) && useless(prev(it))){
```

```
                hull.erase(prev(it));
            }
            while(hasright(it) && useless(next(it))){
                hull.erase(next(it));
            }
            updateborder(it);
        }
        long long getbest(long long x){
            if(hull.empty()){
                return -1e18;
            }

            line query(0 , 0);
            query.xleft = x;
            query.type = 1;

            auto it = hull.lower_bound(query);
            it = prev(it);
            return it -> m * x + it -> c;
        }
};
```

3.33 fft anymod

```
/**
Iterative Implementation of Discrete Fast Fourier Transform
Can Handle any mod that fits in int
Complexity: O(N log N)

Possible Optimizations:
1. Remove leading zeros (No, seriously!!!!)
2. Writing a custom complex class reduces runtime.
3. If there are multiple testcases of similar sizes, it
   might be faster to precalculate the roots of unity.
4. If mod is < 2^26 use RT = 13 and comment out long double
**/

#include<bits/stdc++.h>
#define double long double
using namespace std;

typedef complex<double> CD;
typedef long long LL;
const double PI = acos(-1);

const int RT = 16;
struct FFT
{
    int N;
    vector<int> perm;

    void precalculate() {
        perm.resize(N);
        perm[0] = 0;

        for (int k=1; k<N; k<<=1) {
            for (int i=0; i<k; i++) {
                perm[i] <<= 1;
                perm[i+k] = 1 + perm[i];
            }
        }
    }

    void fft(vector<CD> &v, bool invert = false) {
        if (v.size() != perm.size()) {
            N = v.size();
            assert(N && (N&(N-1)) == 0);
            precalculate();
        }
```

```
    for (int i=0; i<N; i++)
        if (i < perm[i])
            swap(v[i], v[perm[i]]);

    for (int len = 2; len <= N; len <<= 1) {
        double angle = 2 * PI / len;
        if (invert) angle = -angle;
        CD factor = polar(1.0L, angle);

        for (int i=0; i<N; i+=len) {
            CD w(1);
            for (int j=0; j<len/2; j++) {
                CD x = v[i+j], y = w * v[i+j+len/2];
                v[i+j] = x+y;
                v[i+j+len/2] = x-y;
                w *= factor;
            }
        }
    }
    if (invert)
        for (CD &x : v) x/=N;
}

vector<LL> multiply(const vector<LL> &a, const vector<LL>
    &b, int M) {
    int n = 1;
    while (n < a.size()+ b.size()) n<<=1;
    vector<CD> al(n), ar(n), bl(n), br(n);

    for (int i=0; i<a.size(); i++) {
        LL k = a[i]%M;
        al[i] = k >> RT;
        ar[i] = k & ((1<<RT)-1);
    }
    for (int i=0; i<b.size(); i++) {
        LL k = b[i]%M;
        bl[i] = k >>RT;
        br[i] = k & ((1<<RT)-1);
    }

    fft(al); fft(ar);
    fft(bl); fft(br);

    for (int i=0; i<n; i++) {
        CD ll = (al[i] * bl[i]);
        CD lr = (al[i] * br[i]);
        CD rl = (ar[i] * bl[i]);
        CD rr = (ar[i] * br[i]);
        al[i] = ll; ar[i] = lr;
        bl[i] = rl; br[i] = rr;
    }

    fft(al, true); fft(ar, true);
    fft(bl, true); fft(br, true);

    vector<LL> ans(n);
    for (int i=0; i<n; i++) {
        LL right = round(br[i].real());
        right %= M;
        LL mid = round(round(bl[i].real()) +
            round(ar[i].real()));
        mid = ((mid%M)<<RT)%M;
        LL left = round(al[i].real());
        left = ((left%M)<<(2*RT))%M;
        ans[i] = (left+mid+right)%M;
    }
    return ans;
}
```

```
};

int main()
{
    int t;
    cin>>t;
    FFT fft;

    while (t--)
    {
        int n;
        cin>>n;

        vector<LL> a(n+1), b(n+1), ans;
        for (int i=0; i<=n; i++) cin>>a[n-i];
        for (int i=0; i<=n; i++) cin>>b[n-i];

        ans = fft.multiply(a, b, 1e9+7);

        for (int i=2*n; i>=0; i--)
            cout<<ans[i]<<" ";
        cout<<endl;
    }
}
```

3.34 hashing with gp hash table

```
#include<bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef pair<int, int> pii;
#define M 1000005

constexpr uint64_t mod = (1ull<<61) - 1;
uint64_t modmul(uint64_t a, uint64_t b){
        uint64_t l1 = (uint32_t)a, h1 = a>>32, l2 = (uint32_t)b,
            h2 = b>>32;
        uint64_t l = l1*l2, m = l1*h2 + l2*h1, h = h1*h2;
        uint64_t ret = (l&mod) + (l>>61) + (h << 3) + (m >> 29) +
            (m << 35 >> 3) + 1;
        ret = (ret & mod) + (ret>>61);
        ret = (ret & mod) + (ret>>61);
        return ret-1;
}
uint64_t modsum(uint64_t a, uint64_t b){
        ull x = a+b;
        if(a+b >= mod) x -= mod;
        return x;
}
uint64_t doMod(ull x)
{
    if(x >= mod) x -= mod;
    return x;
}

const ll B1 = 104723;
const ll B2 = 104729;
const ll MOD1 = 1000000007;
const ll MOD2 = 100000007;

ll pb1[M], pb2[M];

void init()
{
    pb1[0] = pb2[0] = 1;
```

```
    for(int i = 1; i < M; i++) pb1[i] = (pb1[i-1]*B1)%MOD1,
        pb2[i] = (pb2[i-1]*B2)%MOD2;
}

struct myHash
{
    ll h1, h2;

    myHash(): h1(0), h2(0) {}
    myHash(ll h1, ll h2): h1(h1%MOD1), h2(h2%MOD2) {}
    myHash add(ll v1, ll v2) { return myHash(h1*B1+v1,
        h2*B2+v2); }
    myHash operator-(const myHash& rhs) { return
        myHash(h1-rhs.h1+MOD1, h2-rhs.h2+MOD2); }
    bool operator==(const myHash& rhs) const { return
        make_pair(h1, h2) == make_pair(rhs.h1, rhs.h2); }
    bool operator<(const myHash& rhs) const { return
        make_pair(h1, h2) < make_pair(rhs.h1, rhs.h2); }
};

ostream& operator<<(ostream& dout, myHash h)
{
    return dout<<"h1 = "<<h.h1<<", h2 = "<<h.h2;
}

vector<myHash> getAll(string str)
{
    int n = str.size();
    vector<myHash>ans(n);

    myHash h;
    for(int i = 0; i < str.size(); i++) h = h.add(str[i],
        str[i]), ans[i] = h;

    return ans;
}

myHash getTot(string str)
{
    if(str.size() == 0) return myHash();

    vector<myHash>shob = getAll(str);
    return shob[shob.size()-1];
}

myHash getRange(const vector<myHash>& shob, int l, int r)
{
    myHash lft = l? shob[l-1]: myHash();
    myHash rht = shob[r];

    lft = myHash(lft.h1*pb1[r-l+1], lft.h2*pb2[r-l+1]);
    return rht-lft;
}

struct gpHash
{
    ll operator()(myHash h) const { return h.h1*1000000000+h.h2;
        } // return h.h1^1023240101 in case of single hashing
};

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

    init();

    gp_hash_table<myHash, int, gpHash>ma;

    return 0;
```

```
}
```

3.35 manacher

```
#include<bits/stdc++.h>
#define VI vector<int>
using namespace std;

///0-based indexing
///p[0][i] is the maximum length of half palindrome around half
    index i
///p[1][i] is the maximum length of half palindrome around
    characters i
VI p[2];

void manacher(const string s) {
       int n = s.size();
       p[0] = VI(n+1);
       p[1] = VI(n);

       for (int z=0; z<2; z++)
       for (int i=0, l=0, r=0; i<n; i++)
       {
           int t = r - i + !z;
           if (i<r) p[z][i] = min(t, p[z][l+t]);
           int L = i-p[z][i], R = i+p[z][i] - !z;
           while (L>=1 && R+1<n && s[L-1] == s[R+1])
               p[z][i]++, L--, R++;
           if (R>r) l=L, r=R;
       }
}

bool ispalin(int l, int r)
{
    int mid = (l+r+1)/2;
    int sz = r-l+1;
    bool b = sz%2;
    int len = p[b][mid];
    len = 2*len + b;
    return len>=sz;
}

int main()
{
    int n;
    cin>>n;

    string s;
    cin>>s;

    manacher(s);

    int ans = 0;
    for (int i=0; i<=n; i++) ans = max(ans, p[0][i]*2);
    for (int i=0; i<n; i++)  ans = max(p[1][i]*2+1, ans);

    cout<<ans<<endl;

}
```

3.36 palindromicTree

```
/*************************************************************************
    Palindrome tree. Useful structure to deal with palindromes
       in strings. O(N)
    This code counts number of palindrome substrings of the
       string.
    Based on problem 1750 from informatics.mccme.ru:
```

```
   http://informatics.mccme.ru/moodle/mod/statements/view.php?chapterid=1750
*******************************************************************************/

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

const int MAXN = 105000;

struct node {
    int next[26];
    int len;
    int sufflink;
    int num;
};

int len;
string s;
node tree[MAXN];
int num;          // node 1 - root with len -1, node 2 - root
    with len 0
int suff;         // max suffix palindrome
long long ans;

bool addLetter(int pos) {
    int cur = suff, curlen = 0;
    int let = s[pos] - 'a';

    while (true) {
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] ==
            s[pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {
        suff = tree[cur].next[let];
        return false;
    }

    num++;
    suff = num;
    tree[num].len = tree[cur].len + 2;
    tree[cur].next[let] = num;

    if (tree[num].len == 1) {
        tree[num].sufflink = 2;
        tree[num].num = 1;
        return true;
    }

    while (true) {
        cur = tree[cur].sufflink;
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] ==
            s[pos]) {
            tree[num].sufflink = tree[cur].next[let];
            break;
        }
    }

    tree[num].num = 1 + tree[tree[num].sufflink].num;

    return true;
}

void initTree() {
    memset(tree, 0, sizeof tree); ///MUST if there are test cases
    num = 2; suff = 2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
```

```
}

int main() {
    //assert(freopen("input.txt", "r", stdin));
    //assert(freopen("output.txt", "w", stdout));

    cin >> s;
    len = s.size();

    initTree();

    for (int i = 0; i < len; i++) {
        addLetter(i);
        ans += tree[suff].num;
    }

    cout << ans << endl;

    return 0;
}
```

3.37 suffix$_a$rray$_r$eponda

```
/***************************** END OF TEMPLATE
   ******************************/
/*****************************************************************************************
 *     Suffix Array Implementation O(nlog n)
 *     Definition : suffix(i) => substring [i,n-1]
 *     Input : STRING (inpStr) , sigma => Character Set Size
 Output:
 sa => ith smallest suffix of the string
 rak => rak[i] indicates the position of suffix(i) in the suffix
    array
 height => height[i] indicates the LCP of i-1 and i th suffix
 *     LCP of suffix(i) & suffix(j) = { L = rak[i], R = rak[j] ,
    min(height[L+1, R]);}
 *****************************************************************************/
#include<bits/stdc++.h>
using namespace std;


const int maxn = 5e5+5;
int wa[maxn],wb[maxn],wv[maxn],wc[maxn];
int r[maxn],sa[maxn],rak[maxn],
   height[maxn],dp[maxn][22],jump[maxn], SIGMA = 0 ;

int cmp(int *r,int a,int b,int l){return r[a] == r[b] && r[a+l]
   == r[b+l];}
void da(int *r,int *sa,int n,int m)
{
    int i,j,p,*x=wa,*y=wb,*t;
    for( i=0;i<m;i++) wc[i]=0;
    for( i=0;i<n;i++) wc[x[i]=r[i]] ++;
    for( i=1;i<m;i++) wc[i] += wc[i-1];
    for( i= n-1;i>=0;i--)sa[--wc[x[i]]] = i;
    for( j= 1,p=1;p<n;j*=2,m=p){
        for(p=0,i=n-j;i<n;i++)y[p++] = i;
        for(i=0;i<n;i++)if(sa[i] >= j) y[p++] = sa[i] - j;
        for(i=0;i<n;i++)wv[i] = x[y[i]];
        for(i=0;i<m;i++) wc[i] = 0;
        for(i=0;i<n;i++) wc[wv[i]] ++;
        for(i=1;i<m;i++) wc[i] += wc[i-1];
        for(i=n-1;i>=0;i--) sa[--wc[wv[i]]] = y[i];
        for(t=x,x=y,y=t,p=1,x[sa[0]] = 0,i=1;i<n;i++) x[sa[i]]=
            cmp(y,sa[i-1],sa[i],j) ? p-1:p++;
    }
}
void calheight(int *r,int *sa,int n)
{
```

```
    int i,j,k=0;
    for(i=1;i<=n;i++) rak[sa[i]] = i;
    for(i=0;i<n;height[rak[i++]] = k ) {
        for(k?k--:0, j=sa[rak[i]-1] ; r[i+k] == r[j+k] ; k ++) ;
    }
}

void initRMQ(int n)
{
    for(int i= 0;i<=n;i++) dp[i][0] = height[i];
    for(int j= 1; (1<<j) <= n; j ++ ){
        for(int i = 0; i + (1<<j) - 1 <= n ; i ++ ) {
            dp[i][j] = min(dp[i][j-1] , dp[i + (1<<(j-1))][j-1]);
        }
    }
    for(int i = 1;i <= n;i ++ ) {
        int k = 0;
        while((1 << (k+1)) <= i) k++;
        jump[i] = k;
    }

}
int askRMQ(int L,int R)
{
    int k = jump[R-L+1];
    return min(dp[L][k], dp[R - (1<<k) + 1][k]);
}

char s[maxn];
int main()
{
    scanf("%s",s);
    int n = strlen(s);
    for(int i = 0; i < n; i ++) {
        r[i] = s[i]-'a' + 1;
        SIGMA = max(SIGMA, r[i]);
    }
    r[n] = 0;
    da(r,sa,n+1,SIGMA + 1); // don't forget SIGMA + 1. It will
        ruin you.
    calheight(r,sa,n);

    /// will sort string of length n+1
    for (int i = 0; i <= n; i++) {
        printf("%d ", sa[i]);
    }
    printf("\n");
    for (int i = 0; i <= n; i++) {
        printf("%d ", height[i]);
    }
    printf("\n");

}


/*=========================================================================
```

4 Template

4.1 AhoCorasick

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

const int sigma = 26;
struct Vertex {
    int next[sigma];  /// indices of child node
```

```
    bool leaf = false; /// true if the char is a last char in a
        string
    int p = -1;        /// index of parent node
    char pch;          /// parent character

    int link = -1;     /// suffix link for a vertex p is a edge
        that
                       /// points to the longest proper suffix of
                           the
                       /// string corresponding to the vertex p.

    int go[sigma];     /// save the values of go(int v, char ch)
        in the array

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(next, next+sigma, -1);
        fill(go, go+sigma, -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].leaf = true;
}
```

```
int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}

/// returns the node to go from node 'v' with edge 'ch'

int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}

void bfs()
{
    queue< int >q;
    q.push(0);

    while (!q.empty()) {
        int u = q.front(); q.pop();
        get_link(u);
```

```
        if (t[t[u].link].leaf) t[u].leaf = true;
        /// ^^leaf variable is TRUE if it is the end char in the
            string
        /// OR if any of the nodes connected through the suffix
            links is TRUE

        /// Alternative is to keep a frequency of the end nodes
            (leaf==TRUE)
        /// connected up through the suffix links

        for (int v = 0; v < sigma; v++) {
            if (t[u].next[v]==-1) continue;
            q.push(t[u].next[v]);
        }
    }
}

int main()
{

    return 0;
}
```

4.2 Dinic

```
/**
 Adjacency list implementation of Dinic's blocking flow
    algorithm.
 For Undirected graphs, add each edge in both directions.
 This is very fast in practice, and only loses to push-relabel
    flow.

 Source: Stanford Notebook
 Running time: O(V^2 E)
 Memory : O(V+E)

 INPUT:
    - graph, constructed using AddEdge()
    - source and sink

 OUTPUT:
    - maximum flow value
    - To obtain actual flow values, look at edges with capacity
       > 0
      (zero capacity edges are residual edges).
**/

#include<bits/stdc++.h>
using namespace std;
typedef long long LL;

struct Edge {
    int u, v;
    LL cap, flow;
    Edge() {}
    Edge(int u, int v, LL cap): u(u), v(v), cap(cap), flow(0) {}
};

struct Dinic {
    int N;
    vector<Edge> edges;
    vector<vector<int>> adj;
    vector<int> d, pt;

    Dinic(int N): N(N), edges(0), adj(N), d(N), pt(N) {}

    void AddEdge(int u, int v, LL cap) {
        if (u != v) {
            edges.emplace_back(u, v, cap);
```

```
        adj[u].emplace_back(edges.size() - 1);
        edges.emplace_back(v, u, 0);
        adj[v].emplace_back(edges.size() - 1);
    }
}

bool BFS(int S, int T) {
    queue<int> q({S});
    fill(d.begin(), d.end(), N + 1);
    d[S] = 0;
    while(!q.empty()) {
        int u = q.front(); q.pop();
        if (u == T) break;
        for (int k: adj[u]) {
            Edge &e = edges[k];
            if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
                d[e.v] = d[e.u] + 1;
                q.emplace(e.v);
            }
        }
    }
    return d[T] != N + 1;
}

LL DFS(int u, int T, LL flow = -1) {
    if (u == T || flow == 0) return flow;
    for (int &i = pt[u]; i < adj[u].size(); ++i) {
        Edge &e = edges[adj[u][i]];
        Edge &oe = edges[adj[u][i]^1];
        if (d[e.v] == d[e.u] + 1) {
            LL amt = e.cap - e.flow;
            if (flow != -1 && amt > flow) amt = flow;
            if (LL pushed = DFS(e.v, T, amt)) {
                e.flow += pushed;
                oe.flow -= pushed;
                return pushed;
            }
        }
    }
    return 0;
}

LL MaxFlow(int S, int T) {
    LL total = 0;
    while (BFS(S, T)) {
        fill(pt.begin(), pt.end(), 0);
        while (LL flow = DFS(S, T))
            total += flow;
    }
    return total;
}
};

// BEGIN CUT
// The following code solves SPOJ problem #4110: Fast Maximum
   Flow (FASTFLOW)

int main()
{
    int N, E;
    scanf("%d %d", &N, &E);
    Dinic dinic(N);
    for(int i = 0; i < E; i++)
    {
        int u, v;
        LL cap;
        scanf("%d %d %lld", &u, &v, &cap);
        dinic.AddEdge(u - 1, v - 1, cap);
        dinic.AddEdge(v - 1, u - 1, cap);
    }
```

```
    printf("%lld\n", dinic.MaxFlow(0, N - 1));
    return 0;
}

// END CUT
```

4.3 EdmondsKarp

```
/**
 Adjacency list implementation of Edmonds-Karp algorithm.
 For Undirected graphs, add the commented line in AddEdge.
 Source : Emaxx (modified)

 Running time: O(VE^2)
 Memory: O(V^2)

 INPUT:
    - graph, constructed using AddEdge()
    - source and sink

 OUTPUT:
    - maximum flow value
    - To obtain actual flow values, either keep an extra matrix,
      or save a copy of the original capacity matrix and subtract
**/

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

typedef long long LL;
typedef pair<int, LL> PIL;
typedef vector<int> VI;
typedef vector<VI> VVI;
typedef vector<LL> VL;
typedef vector<VL> VVL;

const LL INF = 1e18+7;

struct EdmondsKarp
{
    int n;
    VVL cap;
    VVI adj;

    EdmondsKarp(int nn) {
        n = nn;
        cap = VVL(n, VL(n));
        adj = VVI(n);
    }

    void AddEdge(int u, int v, LL c) {
        if (u==v)  return;
        adj[u].push_back(v);
        adj[v].push_back(u);
        cap[u][v] += c;
        ///cap[v][u] += c;    // For Undirected Graphs.
    }

    LL bfs(int s, int t) {
        VI par(n, -1);
        par[s] = -2;
        queue<PIL> q;
        q.push({s, INF});
        LL ans = 0;

        while (!q.empty()) {
            int u = q.front().first;
            LL flow = q.front().second;
```

```
            q.pop();

            for (int v : adj[u]) {
                if (par[v] == -1 && cap[u][v]) {
                    par[v] = u;
                    LL new_flow = min(flow, cap[u][v]);
                    if (v == t) {ans = new_flow; break;}
                    q.push({v, new_flow});
                }
            }
        }

        if (ans == 0) return 0;
        int cur = t;
        while (cur != s) {
            int prev = par[cur];
            cap[prev][cur] -= ans;
            cap[cur][prev] += ans;
            cur = prev;
        }
        return ans;
    }

    LL MaxFlow(int s, int t) {
        LL flow = 0;
        while (LL new_flow = bfs(s, t))
            flow += new_flow;
        return flow;
    }
};

///Solves SPOJ MTOTALF - Total Flow
///Given a graph find the max flow between two nodes
int main()
{
    int m;
    cin>>m;

    EdmondsKarp solver(130);
    for (int i=0; i<m; i++) {
        char u, v;
        int cap;
        cin>>u>>v>>cap;
        solver.AddEdge(u, v, cap);
    }

    cout<<solver.MaxFlow('A', 'Z')<<endl;
}
```

4.4 FFT

```
/**
Iterative Implementation of Discrete Fast Fourier Transform
Complexity: O(N log N)

Possible Optimizations:
1. Remove leading zeros (No, seriously!!!!)
2. Writing a custom complex class reduces runtime.
3. If there are multiple testcases of similar sizes, it
   might be faster to precalculate the roots of unity.
**/

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

typedef complex<double> CD;
typedef long long LL;
const double PI = acos(-1);
```

```
struct FFT
{
    int N;
    vector<int> perm;

    void precalculate() {
        perm.resize(N);
        perm[0] = 0;

        for (int k=1; k<N; k<<=1) {
            for (int i=0; i<k; i++) {
                perm[i] <<= 1;
                perm[i+k] = 1 + perm[i];
            }
        }
    }

    void fft(vector<CD> &v, bool invert = false) {
        if (v.size() != perm.size()) {
            N = v.size();
            assert(N && (N&(N-1)) == 0);
            precalculate();
        }

        for (int i=0; i<N; i++)
            if (i < perm[i])
                swap(v[i], v[perm[i]]);

        for (int len = 2; len <= N; len <<= 1) {
            double angle = 2 * PI / len;
            if (invert) angle = -angle;
            CD factor = polar(1.0, angle);

            for (int i=0; i<N; i+=len) {
                CD w(1);
                for (int j=0; j<len/2; j++) {
                    CD x = v[i+j], y = w * v[i+j+len/2];
                    v[i+j] = x+y;
                    v[i+j+len/2] = x-y;
                    w *= factor;
                }
            }
        }
        if (invert)
            for (CD &x : v) x/=N;
    }

    vector<LL> multiply(const vector<LL> &a, const vector<LL>
        &b) {
        vector<CD> fa(a.begin(), a.end()), fb(b.begin(), b.end());

        int n = 1;
        while (n < a.size()+ b.size()) n<<=1;
        fa.resize(n);
        fb.resize(n);

        fft(fa);
        fft(fb);
        for (int i=0; i<n; i++) fa[i] *= fb[i];
        fft(fa, true);

        vector<LL> ans(n);
        for (int i=0; i<n; i++)
            ans[i] = round(fa[i].real());
        return ans;
    }
};

///Solves SPOJ POLYMUL
///Given two polynomials, find their product
```

```
int main()
{
    int t;
    cin>>t;
    FFT fft;

    while (t--)
    {
        int n;
        cin>>n;

        vector<LL> a(n+1), b(n+1), ans;
        for (int i=0; i<=n; i++) cin>>a[n-i];
        for (int i=0; i<=n; i++) cin>>b[n-i];

        ans = fft.multiply(a, b);

        for (int i=2*n; i>=0; i--)
            cout<<ans[i]<<" ";
        cout<<endl;
    }
}
```

4.5 FordFulkerson

```
/**
 Adjacency list implementation of Ford-Fulkerson algorithm.
 For Undirected graphs, add each edge in both directions

 Running time: O(E * flow)
 Memory : O(V+E)

 INPUT:
     - graph, constructed using AddEdge()
     - source and sink

 OUTPUT:
     - maximum flow value
     - To obtain actual flow values, look at edges with capacity
        > 0
       (zero capacity edges are residual edges).
**/

#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef vector<int> VI;
typedef vector<VI> VVI;
const LL INF = 1e18+7;

struct Edge {
  int u, v;
  LL cap, flow;
  Edge() {}
  Edge(int u, int v, LL cap): u(u), v(v), cap(cap), flow(0) {}
};

struct FordFulkerson
{
    int n;
    VVI adj;
    vector<Edge> edges;
    vector<bool> vis;

    FordFulkerson(int nn) {
        n = nn;
        adj = vector<VI> (n);
        vis = vector<bool> (n);
```

```
}

void AddEdge(int u, int v, LL c) {
    if (u==v)  return;
    edges.emplace_back(u, v, c);
    adj[u].emplace_back(edges.size() - 1);
    edges.emplace_back(v, u, 0);
    adj[v].emplace_back(edges.size() - 1);
}

LL dfs(int u, int t, LL flow=INF) {
    if (u==t)  return flow;
    vis[u] = 1;

    for (int i: adj[u]) {
        Edge &e = edges[i];
        Edge &oe = edges[i^1];
        if (!vis[e.v] && e.cap > e.flow) {
            LL cur = min(flow, e.cap-e.flow);
            LL newflow = dfs(e.v, t, cur);
            if (newflow == 0) continue;
            e.flow += newflow;
            oe.flow -= newflow;
            return newflow;
        }
    }
    return 0;
}

LL MaxFlow(int s, int t) {
    LL flow = 0;
    while (true) {
        fill(vis.begin(), vis.end(), 0);
        LL new_flow = dfs(s, t);
        if (new_flow == 0) break;
            flow += new_flow;
        }
        return flow;
    }
};

///Solves SPOJ MTOTALF - Total Flow
///Given a graph find the max flow between two nodes
int main()
{
    int m;
    cin>>m;

    FordFulkerson solver(130);
    for (int i=0; i<m; i++) {
        char u, v;
        int cap;
        cin>>u>>v>>cap;
        solver.AddEdge(u, v, cap);
    }
    cout<<solver.MaxFlow('A', 'Z')<<endl;;
}
```

4.6 Gauss

```
#include <bits/stdc++.h>
#define VD vector<double>
#define VVD vector<VD>
using namespace std;

/**
Gauss-Jordan Elimination for solving multi-variable
Linear Equations. If there are multiple solutions,
outputs a arbitrary valid solution. If some of the
```

```
variable are unsolvable, they are assigned nan.

INPUT: 2D vector representing the Augmented Matrix form
OUTPUT: 1D vector containing the solutions
Complexity: O(n^3)

Note: There must be as exactly as many equations as
variables, if there are less equations, add all zero rows
to make n nows. If there are more than n equations, keep
n (preferably unique) equations and check the others using
that solution.

*/

void print(VVD mat)
{
    int n=mat.size();
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++)
            cout<<mat[i][j]<<" ";
        cout<<" | "<<mat[i][n]<<endl;
    }
}

const double EPS = 1e-9;
VD Gauss(VVD a)
{
    int n=a.size();

    for (int i=0; i<n; i++) {
        int id=i;
        for (int j=i; j<n; j++)
            if (abs(a[j][i]) > abs(a[id][i]))
                id = j;
        swap(a[i], a[id]);
        if (abs(a[i][i]) < EPS) continue;

        for (int j=i+1; j<n; j++)
            for (int k=n; k>=i; k--)
                a[j][k] -= a[i][k] * (a[j][i] / a[i][i]);
    }

    for (int i=0; i<n; i++) {
        if (abs(a[i][i]) < EPS) continue;
        for (int j=0; j<i; j++)
            for (int k=n; k>=i; k--)
                a[j][k] -= a[i][k] * (a[j][i] / a[i][i]);
    }

    VD sol(n);
    vector<bool> bad(n);

    for (int i=n-1; i>=0; i--)
    {
        if (abs(a[i][i]) < EPS && abs(a[i][n]) > EPS) bad[i]=1;
        else if (abs(a[i][i]) < EPS)   sol[i]=0;
        else    sol[i] = a[i][n]/a[i][i];

        for(int j=i; j<n; j++)
            if( abs(a[i][j]) > EPS && bad[j])
                bad[i]=1;
    }

    for (int i=0; i<n; i++)
        if (bad[i])    sol[i]=NAN;

    return sol;
}
```

```
int main()
{
    int n;
    cin>>n;
    VVD mat (n, VD (n+1));
    for (int i=0; i<n; i++)
        for (int j=0; j<=n; j++)
            cin>>mat[i][j];

    VD sol = Gauss(mat);

    for (int i=0; i<n; i++)
        cout<<sol[i]<<endl;
}
```

4.7 Hashing

```
/**
Simple Library for String Hashing
Uses Rolling Double Hash.
Hash(abc........z) = a*p^n + b*p^(n-1) + ..... + z

Cautions:
1. You may assign any integer values to characters.
   Common is 'a' = 1, 'b' = 2 ..........
   Here the ascii values of characters is used.
   But never assign any character the value 0.
   For example if 'a' = 0; 'abc' and 'bc' has the same hash.

2. Single Hashing with an unusual mod is often enough,
   but will surely fail for good enough judge-data.
   In order to convert to Single Hash -
       o Delete operator overloads (optional)
       o Replace all PLL with LL
       o Change mp pairs to appropriate value

Some Primes:
1000000007, 1000000009, 1000000861, 1000099999 ( < 2^30 )
1088888881, 1111211111, 1500000001, 1481481481 ( < 2^31 )
2147483647 (2^31-1),
**/

#include <bits/stdc++.h>
#define ff first
#define ss second
#define mp make_pair
using namespace std;
typedef long long LL;
typedef pair<LL, LL> PLL;

const PLL M=mp(1e9+7, 1e9+9); ///Should be large primes
const LL base=347;            ///Should be a prime larger than
   highest value
const int N = 1e6+7;          ///Highest length of string

ostream& operator<<(ostream& os, PLL hash) {
    return os<<"("<<hash.ff<<", "<<hash.ss<<")";
}

PLL operator+ (PLL a, LL x)  {return mp(a.ff + x, a.ss + x);}
PLL operator- (PLL a, LL x)  {return mp(a.ff - x, a.ss - x);}
PLL operator* (PLL a, LL x)  {return mp(a.ff * x, a.ss * x);}
PLL operator+ (PLL a, PLL x) {return mp(a.ff + x.ff, a.ss +
   x.ss);}
PLL operator- (PLL a, PLL x) {return mp(a.ff - x.ff, a.ss -
   x.ss);}
PLL operator* (PLL a, PLL x) {return mp(a.ff * x.ff, a.ss *
   x.ss);}
```

```
PLL operator% (PLL a, PLL m) {return mp(a.ff % m.ff, a.ss %
    m.ss);}

PLL power (PLL a, LL p) {
    if (p==0)  return mp(1,1);
    PLL ans = power(a, p/2);
    ans = (ans * ans)%M;
    if (p%2)   ans = (ans*a)%M;
    return ans;
}

///Magic!!!!!!!
PLL inverse(PLL a) {
    return power(a, (M.ff-1)*(M.ss-1)-1);
}

PLL pb[N];     ///powers of base mod M
PLL invb;

///Call pre before everything
void hashPre() {
    pb[0] = mp(1,1);
    for (int i=1; i<N; i++)
        pb[i] = (pb[i-1] * base)%M;
    invb = inverse(pb[1]);
}

///Calculates Hash of a string
PLL Hash (string s) {
    PLL ans = mp(0,0);
    for (int i=0; i<s.size(); i++)
        ans=(ans*base + s[i])%M;
    return ans;
}
```

```
///appends c to string
PLL append(PLL cur, char c) {
    return (cur*base + c)%M;
}

///prepends c to string with size k
PLL prepend(PLL cur, int k, char c) {
    return (pb[k]*c + cur)%M;
}

///replaces the i-th (0-indexed) character from right from a to
    b;
PLL replace(PLL cur, int i, char a, char b) {
    cur = (cur + pb[i] * (b-a))%M;
    return (cur + M)%M;
}

///Erases c from the back of the string
PLL pop_back(PLL hash, char c) {
    return (((hash-c)*invb)%M+M)%M;
}

///Erases c from front of the string with size len
PLL pop_front(PLL hash, int len, char c) {
    return ((hash - pb[len-1]*c)%M+M)%M;
}

///concatenates two strings where length of the right is k
PLL concat(PLL left, PLL right, int k) {
    return (left*pb[k] + right)%M;
}

///Calculates hash of string with size len repeated cnt times
///This is O(log n). For O(1), pre-calculate inverses
PLL repeat(PLL hash, int len, LL cnt) {
```

```
    PLL mul = (pb[len*cnt] - 1) * inverse(pb[len]-1);
    mul = (mul%M+M)%M;
    PLL ans = (hash*mul)%M;

    if (pb[len].ff == 1)  ans.ff = hash.ff*cnt;
    if (pb[len].ss == 1)  ans.ss = hash.ss*cnt;
    return ans;
}

///Calculates hashes of all prefixes of s including empty prefix
vector<PLL> hashList(string s) {
    int n = s.size();
    vector<PLL> ans(n+1);
    ans[0] = mp(0,0);

    for (int i=1; i<=n; i++)
        ans[i] = (ans[i-1] * base + s[i-1])%M;
    return ans;
}

///Calculates hash of substring s[l..r] (1 indexed)
PLL substringHash(const vector<PLL> &hashlist, int l, int r) {
    int len = (r-l+1);
    return ((hashlist[r] - hashlist[l-1]*pb[len])%M+M)%M;
}


///Solves LightOJ 1255-Substring Frequency
///You are given two strings A and B. You have to find
///the number of times B occurs as a substring of A.
char buffer[N];
int main()
{
    hashPre();
    int t;
    scanf("%d", &t);

    for (int cs=1; cs<=t; ++cs)
    {
        string a, b;
        scanf("%s", buffer); a = buffer;
        scanf("%s", buffer); b = buffer;
        int na = a.size(), nb = b.size();

        PLL hb = Hash(b);
        vector<PLL> ha = hashList(a);
        int ans = 0;

        for (int i=1; i+nb-1<=na; i++)
            if (substringHash(ha, i, i+nb-1) == hb) ans++;
        printf("Case %d: %d\n", cs, ans);
    }
}
```

4.8 KMP

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

/// builds the prefix automaton in O(N*alphabet)
vector< vector< int > >automaton;
void buildAutomaton(const string& s)
{
    int n = s.size(), k = 0;

    vector< int >zer(26, 0);
    for (int i = 0; i <= n; i++) automaton.push_back(zer);

    automaton[0][s[0]-'a'] = 1;
```

```
    for (int i = 1; i <= n; i++) {
        automaton[i] = automaton[k];

        if (i < n) {
            automaton[i][s[i]-'a'] = i+1;
            k = automaton[k][s[i]-'a'];
        }
    }
}

/// everything 1-indexed
/// v[i] = 0 -> empty string matched
/// v[i] = k -> prefix s[0..(k-1)] matched
vector<int> prefixFunction(const string& s)
{
    int n = s.size(), k = 0;

    vector< int >v(n+1);
    v[1] = 0;

    for (int i = 2; i <= n; i++) {
        while (k > 0 && s[k]!=s[i-1]) k = v[k];
        if (s[k]==s[i-1]) k++;
        v[i] = k;
    }
    return v;
}

int kmpMatcher(const string& text, const string& pattern)
{
    vector<int> pi = prefixFunction(pattern);
    int matchCount = 0, k = 0;

    for (int i = 0; i < text.size(); i++) {
        while (k > 0 && text[i]!=pattern[k]) k = pi[k];
        if (text[i]==pattern[k]) k++;

        if (k==pattern.size()) { /// full pattern match found
            matchCount++;
            k = pi[k];
        }
    }
    return matchCount;
}

/// LightOJ 1255 - Substring Frequency
/// You are given two non-empty strings A and B, both
/// contain lower case English alphabets. You have to
/// find the number of times B occurs as a substring of A.

int main()
{
    std::ios_base::sync_with_stdio(false);
    int t;
    cin >> t;
    for (int ti = 1; ti <= t; ti++) {
        string A, B;
        cin >> A >> B;
        cout << "Case " << ti << ": " << kmpMatcher(A, B) << endl;
    }
    return 0;
}
```

4.9 Linear sieve

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

const int N = 1e8;
```

```
int lp[N+1];
vector<int> pr;

void pre()
{
    for (int i=2; i<=N; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            pr.push_back (i);
        }
        for (int j=0; j<(int)pr.size() && pr[j]<=lp[i] &&
            i*pr[j]<=N; ++j)
            lp[i * pr[j]] = pr[j];
    }
}

int main()
{
    pre();
    cout<<pr.size()<<endl;
}
```

4.10 Trie

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

const int N=1e6+10;
int tr[N][26];
int pop[N];
int sz=0;

void init()
{
    memset(tr, -1, sizeof tr);
    sz=0;
}

void insert(string &s)
{
    pop[0]++;
    for (int i=0, cur=0; i<s.size(); i++) {
        int c=s[i]-'a';
        if(tr[cur][c] == -1) tr[cur][c]=++sz;
        cur = tr[cur][c];
        pop[cur]++;
    }
}

int count(string &s)     //Prefix count
{
    int cur=0;
    for (int i=0; i<s.size(); i++) {
        int c=s[i]-'a';
        if(tr[cur][c] == -1) return 0;
        cur = tr[cur][c];
    }
    return pop[cur];
}

/**
The following code solves SPOJ ADAINDEX
Basically given a list of strings find how
many times a query string appears as prefix.
**/

int main()
{
    ios::sync_with_stdio(0);
```

```
    cin.tie(0);

    int n, m;
    cin>>n>>m;
    init();
    for (int i=0; i<n; i++)
    {
        string s;
        cin>>s;
        insert(s);
    }

    for (int i=0; i<m; i++)
    {
        string s;
        cin>>s;
        cout<<count(s)<<endl;
    }
}
```

4.11 ost

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>

using namespace std;
using namespace __gnu_pbds;
using namespace __gnu_cxx;

// Order Statistic Tree

/* Special functions:

        find_by_order(k) --> returns iterator to the kth largest
            element counting from 0
        order_of_key(val) --> returns the number of items in a
            set that are strictly smaller than our item
*/

typedef
    tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node
ordered_set;


int main(void)
{
    std::ios::sync_with_stdio(false);



    return 0;
}
```