**LIST :**

**AhoCorasick.cpp**

**SplayTree.cpp**

**PersistentSegmentTree.cpp**

**Geometry.cpp**

**SmallestEnclosingCircle.cpp**

**Zfunction.cpp**

**MinCostMaximumFlow.cpp**

**HeavyLightDecomposition.cpp**

**Bridge & Articulation.cpp**

**LCA.cpp**

**HopcroftKarp.cpp**

**Hungarian.cpp**

**ConvexHull.cpp**

**TenarySearch.cpp**

**BigInt.cpp**

**Kruskal.cpp**

**Lehmer – Count number of P number < n (1e10)**

**PhiFunction.cpp**

**Hash.cpp**

**KMP.cpp**

**Manacher.cpp**

**AhoCorasick.cpp**

const int maxn = 1e5 + 5;

const int maxc = 26;

struct node\_t {

node\_t\* p;

node\_t\* c[maxc];

node\_t \*bf, \*gf;

int key;

char val;

node\_t();

void clear();

} pool[maxn], \*ptr = pool;

node\_t::node\_t() {

clear();

}

void node\_t::clear() {

p = 0;

for (int i = 0; i < maxc; i++) c[i] = 0;

bf = gf = 0, key = -1, val = 0;

}

void clear() {

node\_t\* st = pool;

while (st != ptr) {

st->clear();

st++;

}

ptr = pool;

}

node\_t\* insert(node\_t\* x, char\* s, int key) {

while (\*s) {

int k = \*s - 'a';

if (!x->c[k]) {

x->c[k] = ptr++;

x->c[k]->p = x;

x->c[k]->val = k;

}

x = x->c[k];

s++;

}

x->key = key;

return x;

}

void pushlink(node\_t\* rt) {

static node\_t\* q[maxn];

int b = 0, e = 0;

q[e++] = rt;

while (b < e) {

node\_t\* x = q[b++];

for (int i = 0; i < maxc; i++) {

if (x->c[i]) q[e++] = x->c[i];

}

if (x == rt || x->p == rt) {

x->bf = rt;

x->gf = 0;

}

else {

x->bf = x->p->bf;

while (x->bf != rt && !x->bf->c[x->val]) x->bf = x->bf->bf;

if (x->bf->c[x->val]) x->bf = x->bf->c[x->val];

if (x->bf->key != -1) x->gf = x->bf;

else x->gf = x->bf->gf;

//x->info += x->bf->info;

}

}

}

void search(node\_t\* rt, char\* s) {

node\_t\* cur = rt;

while (\*s) {

if (cur == rt && !cur->c[\*s - 'a']) {

s++;

continue;

}

while (cur != rt && !cur->c[\*s - 'a']) cur = cur->bf;

cur = cur->c[\*s - 'a'];

if (!cur) {

cur = rt;

continue;

}

node\_t\* tmp = cur;

if (tmp->key == -1) tmp = tmp->gf;

while (tmp) {

cout << tmp->key << " ";

tmp = tmp->gf;

}

s++;

}

cout << "\n";

}

int main() {

node\_t\* rt = ptr++;

insert(rt, (char\*) "abab", 0);

insert(rt, (char\*) "aba", 1);

insert(rt, (char\*) "aabab", 2);

insert(rt, (char\*) "ababb", 3);

insert(rt, (char\*) "bab", 4);

pushlink(rt);

search(rt, (char\*) "aabab");

return 0;

}

**SplayTree.cpp**

struct node\_t {

node\_t \*p, \*l, \*r;

int key, size;

node\_t(int key) : p(0), l(0), r(0), key(key), size(1) {}

};

int size(node\_t\* x) {

return x ? x->size : 0;

}

int isrt(node\_t\* x) {

return !(x->p) || (x->p->l != x && x->p->r != x);

}

int left(node\_t\* x) {

return x->p->l == x;

}

void setchild(node\_t\* x, node\_t\* p, int l) {

(l ? p->l : p->r) = x;

if (x) x->p = p;

}

void push(node\_t\* x) {

}

void pull(node\_t\* x) {

x->size = size(x->l) + 1 + size(x->r);

}

void rotate(node\_t\* x) {

node\_t \*p = x->p, \*g = p->p;

int l = left(x);

setchild(l ? x->r : x->l, p, l);

if (!isrt(p)) setchild(x, g, left(p));

else x->p = g;

setchild(p, x, !l);

pull(p);

}

node\_t\* splay(node\_t\* x) {

push(x);

while (!isrt(x)) {

node\_t \*p = x->p, \*g = p->p;

if (g) push(g);

push(p), push(x);

if (!isrt(p)) rotate(left(x) != left(p) ? x : p);

rotate(x);

}

pull(x);

return x;

}

node\_t\* join(node\_t\* x, node\_t\* y) {

if (!x) return y;

while (x->r) push(x), x = x->r;

push(x);

setchild(y, x, 0);

return splay(x);

}

void split(node\_t\* t, node\_t\*& x, node\_t\*& y, int pos) {

if (pos < 0) {

x = 0, y = t;

return;

}

if (pos == size(t) - 1) {

x = t, y = 0;

return;

}

while (size(t->l) != pos) {

push(t);

if (size(t->l) > pos) {

t = t->l;

}

else {

pos -= size(t->l) + 1;

t = t->r;

}

}

splay(t);

x = t;

y = x->r;

x->r = y->p = 0;

pull(x);

}

void split(node\_t\* t, node\_t\*& x, node\_t\*& y, node\_t\*& z, int l, int r) {

split(t, x, y, l - 1), split(y, y, z, r - l);

}

void trace(node\_t\* x) {

if (!x) return;

push(x);

trace(x->l);

cout << x->key << " ";

trace(x->r);

}

int main() {

node\_t\* rt = 0;

for (int i = 0; i < 10; i++) {

rt = join(rt, new node\_t(i));

}

trace(rt); cout << "\n";

return 0;

}

**PersistentSegmentTree.cpp**

const int MAXN = 3e6 + 5;

int ptr;

struct Node {

Node \*l, \*r;

int L, R, cnt;

Node();

} mem[MAXN], \*nil = mem + MAXN - 1;

Node::Node() {

l = r = nil;

L = R = -1;

cnt = 0;

}

Node\* newNode() {

return mem + (ptr++);

}

Node\* build(int L, int R) {

Node\* node = newNode();

node->L = L; node->R = R;

if (L == R) return node;

node->l = build(L, (L + R) >> 1);

node->r = build(((L + R) >> 1) + 1, R);

return node;

}

Node\* update(Node\* node, int i, int val) {

if (node->L > i || node->R < i) return node;

Node\* x = newNode();

x->L = node->L; x->R = node->R;

x->l = node->l; x->r = node->r;

if (node->L == node->R) {

x->cnt = node->cnt + val;

return x;

}

x->l = update(x->l, i, val);

x->r = update(x->r, i, val);

x->cnt = x->l->cnt + x->r->cnt;

return x;

}

int query(Node\* node, int L, int R) {

if (node->L > R || node->R < L) return 0;

if (node->L >= L && node->R <= R) return node->cnt;

return query(node->l, L, R) + query(node->r, L, R);

}

Node\* root[MAXN];

int main() {

int n = 10;

root[0] = build(0, n - 1);

root[1] = update(root[0], 2, 4);

root[2] = update(root[1], 3, 1);

cout << query(root[2], 0, 3) << "\n"; //Expected 5

return 0;

}

**Geometry.cpp**

#define RL double

#define EPS 1e-9

struct PT {

RL x, y;

PT() : x(0), y(0) {}

PT(RL x, RL y) : x(x), y(y) {}

PT(const PT& p) : x(p.x), y(p.y) {}

int operator < (const PT& rhs) const {return make\_pair(y, x) < make\_pair(rhs.y, rhs.x);}

int operator == (const PT& rhs) const {return make\_pair(y, x) == make\_pair(rhs.y, rhs.x);}

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 \* (RL c) const {return PT(x \* c, y \* c);}

PT operator / (RL c) const {return PT(x / c, y / c);}

};

RL cross(PT p, PT q) {return p.x \* q.y - p.y \* q.x;}

RL area(PT a, PT b, PT c) {return fabs(cross(a, b) + cross(b, c) + cross(c, a)) / 2;}

RL area2(PT a, PT b, PT c) {return cross(a, b) + cross(b, c) + cross(c, a);}

RL dot(PT p, PT q) {return p.x \* q.x + p.y \* q.y;}

RL dist(PT p, PT q) {return sqrt(dot(p - q, p - q));}

RL dist2(PT p, PT q) {return dot(p - q, p - q);}

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

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

PT RotateCCW(PT p, RL t) {return PT(p.x \* cos(t) - p.y \* sin(t), p.x \* sin(t) + p.y \* cos(t));}

int sign(RL x) {return x < -EPS ? -1 : x > EPS;}

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

ostream& operator << (ostream& os, const PT& p) {

os << "(" << p.x << "," << p.y << ")";

return os;

}

//Project c on Line(a, b)

PT ProjectPointLine(PT a, PT b, PT c) {

return a + (b - a) \* dot(c - a, b - a) / dot(b - a, b - a);

}

PT ProjectPointSegment(PT a, PT b, PT c) {

RL r = dot(b - a, b - a);

if (fabs(r) < EPS) return a;

r = dot(c - a, b - a) / r;

if (r < 0) return a;

if (r > 1) return b;

return a + (b - a) \* r;

}

RL DistancePointSegment(PT a, PT b, PT c) {

return dist(c, ProjectPointSegment(a, b, c));

}

//Compute distance between PT (x, y, z) and plane ax + by + cz = d

RL DistancePointPlane(RL x, RL y, RL z, RL a, RL b, RL c, RL d) {

return fabs(a \* x + b \* y + c \* z - d) / sqrt(a \* a + b \* b + c \* c);

}

//Determine if lines from a to b and c to d are parallel or collinear

int LinesParallel(PT a, PT b, PT c, PT d) {

return fabs(cross(b - a, c - d)) < EPS;

}

int LinesCollinear(PT a, PT b, PT c, PT d) {

return LinesParallel(a, b, c, d) && fabs(cross(a - b, a - c)) < EPS && fabs(cross(c - d, c - a)) < EPS;

}

//Determine if line segment from a to b intersects with line segment from c to d

int SegmentsIntersect(PT a, PT b, PT c, PT d) {

if (LinesCollinear(a, b, c, d)) {

if (dist2(a, c) < EPS || dist2(a, d) < EPS || dist2(b, c) < EPS || dist2(b, d) < EPS) return 1;

if (dot(c - a, c - b) > 0 && dot(d - a, d - b) > 0 && dot(c - b, d - b) > 0) return 0;

return 1;

}

if (cross(d - a, b - a) \* cross(c - a, b - a) > 0) return 0;

if (cross(a - c, d - c) \* cross(b - c, d - c) > 0) return 0;

return 1;

}

//Compute intersection of line passing through a and b

//with line passing through c and d, assuming that unique

//intersection exists; for segment intersection, check if

//segments intersect first

PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {

b = b - a; d = c - d; c = c - a;

return a + b \* cross(c, d) / cross(b, d);

}

//Compute center of circle given three points

PT ComputeCircleCenter(PT a, PT b, PT c) {

b = (a + b) / 2;

c = (a + c) / 2;

return ComputeLineIntersection(b, b + RotateCW90(a - b), c, c + RotateCW90(a - c));

}

//Determine if point is in a possibly non-convex polygon

//returns 1 for strictly interior points, 0 for

//strictly exterior points, and 0 or 1 for the remaining points.

int PointInPolygonSlow(const vector<PT>& p, PT q) {

int c = 0;

for (int i = 0; i < p.size(); i++) {

int j = (i + 1) % p.size();

if ((p[i].y <= q.y && q.y < p[j].y || p[j].y <= q.y && q.y < p[i].y) && q.x < p[i].x + (p[j].x - p[i].x) \* (q.y - p[i].y) / (p[j].y - p[i].y)) c = !c;

}

return c;

}

//Strictly inside convex Polygon

#define Det(a, b, c) ((b.x - a.x) \* (c.y - a.y) - (b.y - a.y) \* (c.x - a.x))

int PointInPolygon(vector<PT>& p, PT q) {

int a = 1, b = p.size() - 1, c;

if (Det(p[0], p[a], p[b]) > 0) swap(a, b);

//Allow on edge --> if (Det... > 0 || Det ... < 0)

if (Det(p[0], p[a], q) >= 0 || Det(p[0], p[b], q) <= 0) return 0;

while(abs(a - b) > 1) {

c = (a + b) / 2;

if (Det(p[0], p[c], q) > 0) b = c; else a = c;

}

//Alow on edge --> return Det... <= 0

return Det(p[a], p[b], q) < 0;

}

//Determine if point is on the boundary of a polygon

int PointOnPolygon(const vector<PT>& p, PT q) {

for (int i = 0; i < p.size(); i++) if (dist2(ProjectPointSegment(p[i], p[(i + 1) % p.size()], q), q) < EPS) return 1;

return 0;

}

//Compute intersection of line through points a and b with circle centered at c with radius r > 0

vector<PT> CircleLineIntersection(PT a, PT b, PT c, RL r) {

vector<PT> res;

b = b - a; a = a - c;

RL A = dot(b, b);

RL B = dot(a, b);

RL C = dot(a, a) - r \* r;

RL D = B \* B - A \* C;

if (D < -EPS) return res;

res.push\_back(c + a + b \* (-B + sqrt(D + EPS)) / A);

if (D > EPS) res.push\_back(c + a + b \* (-B - sqrt(D)) / A);

return res;

}

//Compute intersection of circle centered at a with radius r with circle centered at b with radius R

vector<PT> CircleCircleIntersection(PT a, PT b, RL r, RL R) {

vector<PT> res;

RL d = sqrt(dist2(a, b));

if (d > r + R || d + min(r, R) < max(r, R)) return res;

RL x = (d \* d - R \* R + r \* r) / (2 \* d);

RL y = sqrt(r \* r - x \* x);

PT v = (b - a) / d;

res.push\_back(a + v \* x + RotateCCW90(v) \* y);

if (y > 0) res.push\_back(a + v \* x - RotateCCW90(v) \* y);

return res;

}

//This code computes the area or centroid of a (possibly nonconvex)

//polygon, assuming that the coordinates are listed in a clockwise or

//counterclockwise fashion. Note that the centroid is often known as

//the "center of gravity" or "center of mass".

RL ComputeSignedArea(const vector<PT>& p) {

RL area = 0;

for(int i = 0; i < p.size(); i++) {

int j = (i + 1) % p.size();

area += p[i].x \* p[j].y - p[j].x \* p[i].y;

}

return area / 2.0;

}

RL ComputeArea(const vector<PT>& p) {

return fabs(ComputeSignedArea(p));

}

PT ComputeCentroid(const vector<PT>& p) {

PT c(0, 0);

RL scale = 6.0 \* ComputeSignedArea(p);

for (int i = 0; i < p.size(); i++) {

int j = (i + 1) % p.size();

c = c + (p[i] + p[j]) \* (p[i].x \* p[j].y - p[j].x \* p[i].y);

}

return c / scale;

}

//Tests whether or not a given polygon (in CW or CCW order) is simple

int IsSimple(const vector<PT>& p) {

for (int i = 0; i < p.size(); i++) {

for (int k = i + 1; k < p.size(); k++) {

int j = (i + 1) % p.size();

int l = (k + 1) % p.size();

if (i == l || j == k) continue;

if (SegmentsIntersect(p[i], p[j], p[k], p[l])) return 0;

}

}

return 1;

}

RL Angle(PT a) {

RL PI = acos((RL) - 1);

if (a.x == 0) {

if (a.y > 0) return PI / 2;

return 3 \* PI / 2;

}

if (a.y == 0) {

if (a.x > 0) return 0;

return PI;

}

RL res = atan(a.y / a.x);

if (a.x < 0) return res + PI;

if (a.y < 0) return res + 2 \* PI;

return res;

}

int main() {

}

**SmallestEnclosingCircle.cpp**

PT ProjectPointLine(PT a, PT b, PT c) {

return a + (b - a) \* dot(c - a, b - a) / dot(b - a, b - a);

}

RL DistancePointLine(PT a, PT b, PT c) {

return dist(c, ProjectPointLine(a, b, c));

}

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);

}

struct Line {

PT l, r;

Line() {}

Line(PT l, PT r) : l(l), r(r) {}

Line(const Line& rhs) : l(rhs.l), r(rhs.r) {}

};

Line Bisector(PT a, PT b) {

PT c = (a + b) / 2;

return Line(c, c + RotateCCW90(b - c));

}

struct Circle {

PT cen;

RL rad;

Circle() : rad(0) {}

Circle(PT cen, RL rad) : cen(cen), rad(rad) {}

Circle(const Circle& rhs) : cen(rhs.cen), rad(rhs.rad) {}

int operator < (const Circle& rhs) const {

return rad < rhs.rad;

}

};

Circle CircumCircle(PT a, PT b, PT c) {

if (sign(cross(a - c, b - c)) == 0) {

Circle res = Circle((a + b) / 2, dist(a, b) / 2);

res = max(res, Circle((b + c) / 2, dist(b, c) / 2));

res = max(res, Circle((c + a) / 2, dist(c, a) / 2));

return res;

}

Line ln1 = Bisector(a, b);

Line ln2 = Bisector(b, c);

PT cen = ComputeLineIntersection(ln1.l, ln1.r, ln2.l, ln2.r);

return Circle(cen, dist(cen, a));

}

Circle Enclose(vector<PT>& p) {

random\_shuffle(p.begin(), p.end());

int n = p.size();

Circle c(p[0], 0);

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

if (sign(dist(c.cen, p[i]), c.rad) > 0) {

c = Circle(p[i], 0);

for (int j = 0; j < i; j++) {

if (sign(dist(c.cen, p[j]), c.rad) > 0) {

c = Circle((p[i] + p[j]) / 2, dist(p[i], p[j]) / 2);

for (int k = 0; k < j; k++) {

if (sign(dist(c.cen, p[k]), c.rad) > 0) {

c = CircumCircle(p[i], p[j], p[k]);

}

}

}

}

}

}

return c;

}

int Inside(vector<PT>& p, Circle c) {

for (int i = 0; i < p.size(); i++) {

if (sign(dist(c.cen, p[i]), c.rad) > 0) return 0;

}

return 1;

}

int main() {

map<pair<int, int>, int> hs;

vector<PT> p;

for (int i = 0; i < 100000; i++) {

int x = rand();

int y = rand();

if (!hs.count(make\_pair(x, y))) {

hs[make\_pair(x, y)];

p.push\_back(PT(x, y));

}

}

Circle c = Enclose(p);

cout << fixed << setprecision(3) << c.cen << " " << c.rad << "\n";

return 0;

}

**Zfunction.cpp**

struct ZFuntion {

int n;

vector<int> f; //the length of the longest substr begins at i

void build(char\* s) {

n = strlen(s), f.resize(n);

int l = 0, r = 0;

f[0] = n;

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

if (i > r) {

l = r = i;

while (r < n && s[r] == s[r - l]) r++;

f[i] = r - l, r--;

}

else {

int k = i - l;

if (f[k] < r - i + 1) f[i] = f[k];

else {

l = i;

while (r < n && s[r] == s[r - l]) r++;

f[i] = r - l, r--;

}

}

}

}

} zf;

int main() {

zf.build((char\*) "stringsrandom");

for (int i = 0; i < zf.n; i++) {

cout << zf.f[i] << " \n"[i == zf.n - 1];

}

return 0;

}

**MinCostMaximumFlow.cpp**

/\*

\* Complexity: O(min(E^2 \* V \* logV, E \* logV \* Flow))

\*/

template<class TF, class TC, TF Foo, TC Coo> struct MinCostMaximumFlow {

static const int MAXV = 1e5 + 5;

static const int MAXE = 1e6 + 5;

int n, s, t, E;

int adj[MAXE], nxt[MAXE], lst[MAXV], frm[MAXV];

TF cap[MAXE], flw[MAXE], totalFlow;

TC cst[MAXE], pot[MAXE], dst[MAXV], totalCost;

void init(int n, int s, int t) {

this->n = n, this->s = s, this->t = t;

fill\_n(lst, n, -1), E = 0;

}

void add(int u, int v, TF ca, TC co) {

adj[E] = v, cap[E] = ca, flw[E] = 0, cst[E] = +co, nxt[E] = lst[u], lst[u] = E++;

adj[E] = u, cap[E] = 0, flw[E] = 0, cst[E] = -co, nxt[E] = lst[v], lst[v] = E++;

}

void bellman() {

fill\_n(pot, n, 0);

while (1) {

int found = 0;

for (int u = 0; u < n; u++) for (int e = lst[u]; e != -1; e = nxt[e]) if (flw[e] < cap[e]) {

int v = adj[e];

if (pot[v] > pot[u] + cst[e]) {

pot[v] = pot[u] + cst[e];

found = 1;

}

}

if (!found) break;

}

}

int dijkstra() {

priority\_queue<pair<TC, int> > que;

fill\_n(dst, n, Coo), dst[s] = 0;

que.push(make\_pair(-dst[s], s));

while (que.size()) {

TC dnow = -que.top().first;

int u = que.top().second;

que.pop();

if (dst[u] < dnow) continue;

for (int e = lst[u]; e != -1; e = nxt[e]) if (flw[e] < cap[e]) {

int v = adj[e];

TC dnxt = dnow + cst[e] + pot[u] - pot[v];

if (dst[v] > dnxt) {

dst[v] = dnxt;

frm[v] = e;

que.push(make\_pair(-dnxt, v));

}

}

}

return dst[t] < Coo;

}

TC mincost() {

totalCost = 0, totalFlow = 0;

bellman();

while (1) {

if (!dijkstra()) break;

TF mn = Foo;

for (int v = t, e = frm[v]; v != s; v = adj[e ^ 1], e = frm[v]) mn = min(mn, cap[e] - flw[e]);

for (int v = t, e = frm[v]; v != s; v = adj[e ^ 1], e = frm[v]) {

flw[e] += mn;

flw[e ^ 1] -= mn;

}

totalFlow += mn;

totalCost += mn \* (dst[t] - pot[s] + pot[t]);

for (int u = 0; u < n; u++) pot[u] += dst[u];

}

return totalCost;

}

};

MinCostMaximumFlow<int, int, (int) 1e9, (int) 1e9> mcmf;

int main() {

mcmf.init(3, 0, 2);

mcmf.add(0, 1, 1, 3);

mcmf.add(1, 2, 1, 10);

mcmf.add(0, 2, 1, 100);

cout << mcmf.mincost() << "\n";

return 0;

}

**HeavyLightDecomposition.cpp**

const int maxn = 1e5 + 5;

int n;

vector<int> adj[maxn];

int size[maxn];

int lev[maxn];

int p[maxn];

int heavy[maxn];

int num[maxn];

int head[maxn];

int cnt;

void upd(int l, int r, int n, int val) {} //Need to modify

int query(int l, int r, int n) {return 0;} //Need to modify

void firstdfs(int u, int dad) {

size[u] = 1, heavy[u] = -1;

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

int v = adj[u][i];

if (v != dad) {

p[v] = u, lev[v] = lev[u] + 1;

firstdfs(v, u);

size[u] += size[v];

if (heavy[u] == -1 || size[heavy[u]] < size[v]) {

heavy[u] = v;

}

}

}

}

void dfs(int u, int h, int p) {

num[u] = cnt++, head[u] = h;

if (~heavy[u]) {

dfs(heavy[u], h, u);

}

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

int v = adj[u][i];

if (v != p && v != heavy[u]) {

dfs(v, v, u);

}

}

}

void upd(int u, int v, int val) {

int hu = head[u], hv = head[v];

while (hu != hv) {

if (lev[hu] < lev[hv]) swap(u, v), swap(hu, hv);

upd(num[hu], num[u], n, val);

u = p[hu], hu = head[u];

}

if (lev[u] > lev[v]) swap(u, v);

upd(num[u], num[v], n, val); //upd(num[u] + 1, num[v]) for edge path

}

int query(int u, int v) {

int res = 0;

int hu = head[u], hv = head[v];

while (hu != hv) {

if (lev[hu] < lev[hv]) swap(u, v), swap(hu, hv);

res += query(num[hu], num[u], n);

u = p[hu], hu = head[u];

}

if (lev[u] > lev[v]) swap(u, v);

res += query(num[u], num[v], n); //query(num[u] + 1, num[v]) for edge path

return res;

}

int main() {

return 0;

}

**Bridge & Articulation.cpp**

const int MAXN = 100000 + 5;

int n;

vector<int> adj[MAXN];

int num[MAXN];

int low[MAXN];

int tms;

int root, nchild;

void dfs(int u, int p = -1) {

num[u] = low[u] = ++tms;

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

int v = adj[u][i];

if (!num[v]) {

if (u == root) nchild++;

dfs(v, u);

if (low[v] >= num[u]) {

//u is an articulation point

}

if (low[v] > num[u]) {

//u -> v is a bridge

}

low[u] = min(low[u], low[v]);

}

else if (v != p) {

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

}

}

}

int main() {

dfs(root = 0);

if (nchild > 1) {

//root is an articulation point

}

return 0;

}

**LCA.cpp**

const int N = 100005;

int n, Root, l[N], P[20][N];

int level(int u) {

if (u==Root) return l[u]=1;

if (l[u]==0) l[u]=level(P[0][u])+1;

return l[u];

}

int lca(int x, int y) {

for (int k=19; k>=0; k--)

if (l[P[k][x]]>=l[y])

x=P[k][x];

for (int k=19; k>=0; k--)

if (l[P[k][y]]>=l[x])

y=P[k][y];

for (int k=19; k>=0; k--)

if (P[k][x]!=P[k][y])

{ x=P[k][x]; y=P[k][y]; }

while (x!=y)

{ x=P[0][x]; y=P[0][y]; }

return x;

}

void solve() {

scanf("%d", &n);

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

int p; scanf("%d", &p);

while (p-->0) {

int q; scanf("%d", &q);

P[0][q] = i;

}

}

for (int i=1; i<=n; i++)

if (P[0][i]==0) Root=i;

for (int i=1; i<=n; i++)

level(i); // done l

for (int k=1; k<=19; k++)

for (int i=1; i<=n; i++)

P[k][i] = P[k-1][P[k-1][i]];

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

while (m-->0) {

int x, y;

scanf("%d%d", &x, &y);

printf("%d\n", lca(x, y));

}

}

main() {

int t; scanf("%d", &t);

for (int i=1; i<=t; i++) {

printf("Case %d:\n", i);

solve();

for (int j=1; j<=n; j++)

{ l[j]=0; P[0][j]=0; }

}

}

**HopcroftKarp.cpp ( Cặp ghép )**

/\*

\* Complexity: O(E \* sqrt(V))

\*/

struct HopcroftKarp {

static const int MAXV = 1e3 + 5;

static const int MAXE = 1e6 + 5;

int nx, ny, E, adj[MAXE], nxt[MAXE], lst[MAXV], ptr[MAXV], lev[MAXV], que[MAXV], matx[MAXV], maty[MAXV];

void init(int nx, int ny) {

this->nx = nx, this->ny = ny;

E = 0, fill\_n(lst, nx, -1);

fill\_n(matx, nx, -1), fill\_n(maty, ny, -1);

}

void add(int x, int y) {

adj[E] = y, nxt[E] = lst[x], lst[x] = E++;

}

int bfs() {

int qsize = 0;

for (int x = 0; x < nx; x++) if (matx[x] != -1) lev[x] = 0;

else {

lev[x] = 1;

que[qsize++] = x;

}

int found = 0;

for (int i = 0; i < qsize; i++) {

for (int x = que[i], e = lst[x]; ~e; e = nxt[e]) {

int y = adj[e];

if (!~maty[y]) found = 1;

else if (!lev[maty[y]]) {

lev[maty[y]] = lev[x] + 1;

que[qsize++] = maty[y];

}

}

}

return found;

}

int dfs(int x) {

for (int& e = ptr[x]; ~e; e = nxt[e]) {

int y = adj[e];

if (!~maty[y] || (lev[maty[y]] == lev[x] + 1 && dfs(maty[y]))) {

matx[x] = y;

maty[y] = x;

return 1;

}

}

return 0;

}

int maxmat() {

int res = 0;

while (bfs()) {

for (int x = 0; x < nx; x++) ptr[x] = lst[x];

for (int x = 0; x < nx; x++) if (!~matx[x]) res += dfs(x);

}

return res;

}

} hopkarp;

int main() {

hopkarp.init(3, 3);

hopkarp.add(0, 1);

hopkarp.add(0, 2);

hopkarp.add(1, 2);

cout << hopkarp.maxmat() << "\n";

return 0;

}

**Hungarian.cpp ( Cặp ghép )**

/\*

\* Complexity: O(n^3)

\* Indexing from 1

\*/

template<class T, T oo> struct Hungary {

static const int MAXN = 1000 + 5;

static const int MAXM = 1000 + 5;

int nx, ny, maty[MAXM], frm[MAXM], used[MAXM];

T cst[MAXN][MAXM], fx[MAXN], fy[MAXM], dst[MAXM];

void init(int nx, int ny) {

this->nx = nx, this->ny = ny;

fill\_n(fx, nx + 1, 0), fill\_n(fy, ny + 1, 0);

fill\_n(maty, nx + 1, 0);

for (int i = 0; i <= nx; i++) {

fill\_n(cst[i], ny + 1, oo);

}

}

void add(int x, int y, int c) {

cst[x][y] = c;

}

T mincost() {

for (int x = 1; x <= nx; x++) {

int y0 = 0;

maty[0] = x;

for (int y = 0; y <= ny; y++) {

dst[y] = oo + 1;

used[y] = 0;

}

do {

used[y0] = 1;

int x0 = maty[y0], y1;

T delta = oo + 1;

for (int y = 1; y <= ny; y++) if (!used[y]) {

T curdst = cst[x0][y] - fx[x0] - fy[y];

if (dst[y] > curdst) {

dst[y] = curdst;

frm[y] = y0;

}

if (delta > dst[y]) {

delta = dst[y];

y1 = y;

}

}

for (int y = 0; y <= ny; y++) if (used[y]) {

fx[maty[y]] += delta;

fy[y] -= delta;

}

else {

dst[y] -= delta;

}

y0 = y1;

}

while (maty[y0] != 0);

do {

int y1 = frm[y0];

maty[y0] = maty[y1];

y0 = y1;

}

while (y0);

}

T res = 0;

for (int y = 1; y <= ny; y++) {

int x = maty[y];

if (cst[x][y] < oo) res += cst[x][y];

}

return res;

}

};

Hungary<int, (int) 1e9> hungary;

int main() {

hungary.init(3, 3);

hungary.add(1, 2, 3);

hungary.add(1, 2, 1);

cout << hungary.mincost() << "\n";

return 0;

}

**ConvexHull.cpp**

#define RL double

#define EPS 1e-9

struct PT {

RL x, y;

PT() : x(0), y(0) {}

PT(RL x, RL y) : x(x), y(y) {}

PT(const PT& p) : x(p.x), y(p.y) {}

int operator < (const PT& rhs) const {return make\_pair(y, x) < make\_pair(rhs.y, rhs.x);}

int operator == (const PT& rhs) const {return make\_pair(y, x) == make\_pair(rhs.y, rhs.x);}

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 \* (RL c) const {return PT(x \* c, y \* c);}

PT operator / (RL c) const {return PT(x / c, y / c);}

};

RL cross(PT p, PT q) {return p.x \* q.y - p.y \* q.x;}

RL area(PT a, PT b, PT c) {return fabs(cross(a, b) + cross(b, c) + cross(c, a)) / 2;}

RL area2(PT a, PT b, PT c) {return cross(a, b) + cross(b, c) + cross(c, a);}

RL dot(PT p, PT q) {return p.x \* q.x + p.y \* q.y;}

RL dist(PT p, PT q) {return sqrt(dot(p - q, p - q));}

RL dist2(PT p, PT q) {return dot(p - q, p - q);}

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

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

PT RotateCCW(PT p, RL t) {return PT(p.x \* cos(t) - p.y \* sin(t), p.x \* sin(t) + p.y \* cos(t));}

int sign(RL x) {return x < -EPS ? -1 : x > EPS;}

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

ostream& operator << (ostream& os, const PT& p) {

os << "(" << p.x << "," << p.y << ")";

return os;

}

//Remove degenerate

#define REMOVE\_REDUNDANT

#ifdef REMOVE\_REDUNDANT

bool between(const PT& a, const PT& b, const PT& c) {

return (fabs(area2(a, b, c)) < EPS && (a.x - b.x) \* (c.x - b.x) <= 0 && (a.y - b.y) \* (c.y - b.y) <= 0);

}

#endif

void ConvexHull(vector<PT>& pts) {

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

pts.erase(unique(pts.begin(), pts.end()), pts.end());

vector<PT> up, dn;

for (int i = 0; i < pts.size(); i++) {

while (up.size() > 1 && area2(up[up.size() - 2], up.back(), pts[i]) >= 0) up.pop\_back();

while (dn.size() > 1 && area2(dn[dn.size() - 2], dn.back(), pts[i]) <= 0) dn.pop\_back();

up.push\_back(pts[i]);

dn.push\_back(pts[i]);

}

pts = dn;

for (int i = up.size() - 2; i >= 1; i--) pts.push\_back(up[i]);

#ifdef REMOVE\_REDUNDANT

if (pts.size() <= 2) return;

dn.clear();

dn.push\_back(pts[0]);

dn.push\_back(pts[1]);

for (int i = 2; i < pts.size(); i++) {

if (between(dn[dn.size() - 2], dn[dn.size() - 1], pts[i])) dn.pop\_back();

dn.push\_back(pts[i]);

}

if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {

dn[0] = dn.back();

dn.pop\_back();

}

pts = dn;

#endif

}

int main() {

return 0;

}

**TenarySearch.cpp**

#include<bits/stdc++.h>

using namespace std;

double x, y;

double calc(double m) {

//return (x-2\*m)\*(y-2\*m)\*m;

//caculation here

}

void solve() {

cin >> x >> y;

double l = 0, r = min(x, y)/2;

double del = 1e-9;

double m1,m2;

while ( abs( r - l ) > del ) {

m1 = l + ( r - l ) / 3;

m2 = r - ( r - l ) / 3;

if ( calc(m1) > calc(m2) ) r = m2;

else if ( calc(m1) < calc(m2) ) l = m1;

else {

l = m1;

r = m2;

}

}

double h = (l+r)/2.0;

cout << fixed << setprecision(11) << (x-2\*h)\*(y-2\*h)\*h << '\n';

}

int main() {

int T;

cin >> T;

while (T--) solve();

}

**BigInt.cpp**

const int base = 10;

const int nblock = 9;

const int blockbase = (int) round(pow(base, nblock));

struct Bigint {

vector<int> a;

int sign;

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;

if (v < 0)

sign = -1, v = -v;

for (; v > 0; v = v / blockbase)

a.push\_back(v % blockbase);

}

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] >= blockbase;

if (carry) res.a[i] -= blockbase;

}

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] += blockbase;

}

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 / blockbase);

a[i] = (int) (cur % blockbase);

}

trim();

}

void operator \*= (long long v) {

if (v >= (long long) blockbase \* blockbase) {

\*this \*= Bigint(v);

}

int a = v / blockbase;

int b = v % blockbase;

\*this = \*this \* a \* blockbase + \*this \* b;

}

Bigint operator \* (int v) const {

Bigint res = \*this;

res \*= v;

return res;

}

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 = blockbase / (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 \*= blockbase;

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) blockbase \* 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) blockbase;

a[i] = (int) (cur / v);

rem = (int) (cur % v);

}

trim();

}

void operator /= (long long v) {

\*this /= Bigint(v);

}

Bigint operator / (int v) const {

Bigint res = \*this;

res /= v;

return res;

}

Bigint operator / (long long 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) blockbase) % v;

return m \* sign;

}

long long operator % (long long v) const {

return (\*this % Bigint(v)).longValue();

}

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 \* blockbase + 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 -= nblock) {

int x = 0;

for (int j = max(pos, i - nblock + 1); j <= i; j++) x = x \* base + 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(nblock) << 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] \* base;

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;

}

static vector<long long> karatsuba(vector<long long>& a, vector<long long>& b) {

int n = a.size();

vector<long long> res(n << 1);

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;

vector<long long> a1(a.begin(), a.begin() + k);

vector<long long> a2(a.begin() + k, a.end());

vector<long long> b1(b.begin(), b.begin() + k);

vector<long long> b2(b.begin() + k, b.end());

vector<long long> a1b1 = karatsuba(a1, b1);

vector<long long> a2b2 = karatsuba(a2, b2);

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

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

vector<long long> r = karatsuba(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 {

int r = 6;

int t = round(pow(base, r));

vector<int> ar = convert\_base(this->a, nblock, r);

vector<int> br = convert\_base(v.a, nblock, r);

vector<long long> a(ar.begin(), ar.end());

vector<long long> b(br.begin(), br.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);

vector<long long> c = karatsuba(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 % t));

carry = (int) (cur / t);

}

res.a = convert\_base(res.a, r, nblock);

res.trim();

return res;

}

friend Bigint sqrt(const Bigint& a) {

Bigint x0 = a, x1 = (a + 1) / 2;

while (x1 < x0) {

x0 = x1;

x1 = (x1 + a / x1) / 2;

}

return x0;

}

friend Bigint pow(Bigint a, Bigint b) {

if (b == Bigint(0)) return Bigint(1);

Bigint T = pow(a, b / 2);

if (b % 2 == 0) return T \* T;

return T \* T \* a;

}

friend Bigint pow(Bigint a, int b) {

return pow(a, (Bigint(b)));

}

friend int log(Bigint a, int n) {

int res = 0;

while (a > Bigint(1)) {

res++;

a /= n;

}

return res;

}

template<class T> friend Bigint operator + (const T& v, const Bigint& a) {

return a + v;

}

template<class T> friend Bigint operator - (const T& v, const Bigint& a) {

return -a + v;

}

template<class T> friend Bigint operator \* (const T& v, const Bigint& a) {

return a \* v;

}

template<class T> friend Bigint operator / (const T& v, const Bigint& a) {

return Bigint(v) / a;

}

Bigint operator ++() {

(\*this) += 1;

return \*this;

}

Bigint operator ++(int) {

(\*this) += 1;

return \*this - 1;

}

Bigint operator --() {

(\*this) -= 1;

return \*this;

}

Bigint operator --(int) {

(\*this) -= 1;

return \*this + 1;

}

};

int main() {

Bigint n = Bigint("123456789");

Bigint m = Bigint("987654321");

cout << n \* m << "\n";

cout << 5 + m << "\n";

cout << 5 - m << "\n";

cout << 5 \* m << "\n";

cout << 5 / m << "\n";

Bigint k = 1;

cout << ++k << "\n";

cout << k++ << "\n";

cout << --k << "\n";

cout << k-- << "\n";

return 0;

}

**Kruskal.cpp**

#include <stdio.h>

#include <vector>

#include <algorithm>

using namespace std;

int par[230997];

int anc(int p){ if (par[p]==p) return p; else return par[p]=anc(par[p]); }

void join(int p, int q){ par[anc(p)]=anc(q); }

typedef pair<int, int> ii;

typedef pair<int, ii> iii;

#define X first

#define Y second

vector<iii> edge;

int n, m;

main(){

int i, p, q, w, r=0;

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

for (i=1; i<=n; i++) par[i]=i;

while (m--){

scanf("%d%d%d", &p, &q, &w);

edge.push\_back(iii(w, ii(p, q)));

}

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

vector<iii>::iterator it;

for (it=edge.begin(); it!=edge.end(); it++){

if (anc(it->Y.X) != anc(it->Y.Y)){

join(it->Y.X, it->Y.Y);

r += it->X;

}

}

printf("%d\n", r);

}

**Lehmer – Count number of P number < n (1e10)**

#define long long long

const int N = 100005;

const int M = 1000000007;

bool np[N];

int p[N], pp=0;

void eratos() {

np[0]=np[1]=true;

for (int i=2; i\*i<N; i++) if (!np[i])

for (int j=i\*i; j<N; j+=i) np[j]=true;

for (int i=2; i<N; i++)

if (!np[i]) p[++pp]=i;

}

long power(long a, long k) {

long P = 1;

while (k) {

if (k&1) P=P\*a;

k/=2; a=a\*a;

}

return P;

}

long power(long a, long k, long M) {

long P=1;

for (a=a%M; k; k/=2)

{ if (k&1) P=P\*a%M; a=a\*a%M; }

return P;

}

long root(long n, long k) {

long x = pow(n, 1.0/k);

while (power(x, k)%M==power(x, k, M) && power(x, k)<n) x++;

while (power(x, k)%M!=power(x, k, M) || power(x, k)>n) x--;

return x;

}

map<long, long> Phi[N];

long phi(long x, int a) {

if (Phi[a].count(x)) return Phi[a][x];

if (a==1) return (x+1)/2;

long Result = phi(x, a-1) - phi(x/p[a], a-1);

return Phi[a][x] = Result;

}

long pi(long x) {

if (x<N)

return upper\_bound(p+1, p+pp+1, x) - (p+1);

long a = pi(root(x, 4));

long b = pi(root(x, 2));

long c = pi(root(x, 3));

long Sum = phi(x, a) + (b+a-2)\*(b-a+1)/2;

for (int i=a+1; i<=b; i++)

Sum -= pi(x/p[i]);

for (int i=a+1; i<=c; i++) {

long bi = pi(root(x/p[i], 2));

for (int j=i; j<=bi; j++)

Sum -= pi(x/p[i]/p[j]) - (j-1);

}

return Sum;

}

main(){

eratos();

long n;

while (cin >> n)

cout << pi(n) << endl;

}

**PhiFunction.cpp**

#define long long long

long Power[230997][15]; // positive

long power(int a, int k){

if (k==0) return 1;

if (Power[a][k] > 0) return Power[a][k];

long p=power(a, k/2);

if (k&1) return Power[a][k] = p\*p\*a;

else return Power[a][k] = p\*p;

}

long phi(int p, int k){

// phi of p^k with p is a prime

if (k==0) return 1;

return (p-1)\*power(p, k-1);

}

long Phi[230997]; // positive

long phi(int m){

int i, k, om=m;

long r=1;

if (Phi[om] > 0) return Phi[om];

for (i=2; i\*i<=m; i++){

k=0;

while (m%i==0) { k++; m/=i; }

r \*= phi(i, k);

}

if (m>1) r \*= phi(m, 1);

return Phi[om] = r;

}

int n;

main(){

int i;

long r=0;

for (;;){

scanf("%d", &n);

if (n==0) return 0;

r = phi(n);

printf("%lld\n", r);

}

}

**Hash.cpp**

#define long long long

const int N=1000006, BASE=1000000007;

int m, n;

char a[N], b[N];

long A[N], B[N], M[N];

void hash\_build(char a[], int n, long H[]) {

for (int i=1; i<=n; i++)

H[i] = (H[i-1] \* M[1] + a[i]) % BASE;

}

long hash\_range(long H[], int L, int R) {

return (H[R] - H[L-1]\*M[R-L+1] + 1LL\*BASE\*BASE) % BASE;

}

main() {

M[0]=1; M[1]=2309;

for (int i=2; i<N; i++)

M[i] = M[i-1] \* M[1] % BASE;

scanf("%s", a+1); m=strlen(a+1);

scanf("%s", b+1); n=strlen(b+1);

hash\_build(a, m, A);

hash\_build(b, n, B);

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

if (hash\_range(A, i, i+n-1) == B[n])

printf("%d ", i);

}

printf("\n");

}

**KMP.cpp**

#include <bits/stdc++.h>

using namespace std;

/\*

\* Complexity: O(N)

\*/

struct KMP {

string s;

int n;

vector<int> link;

void build(char\* str) {

s = str, n = (int) s.size();

link.resize(n);

int cur = link[0] = -1;

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

while (cur >= 0 && s[i] != s[cur + 1]) cur = link[cur];

if (s[i] == s[cur + 1]) cur++;

link[i] = cur;

}

}

vector<int> query(char\* t) {

vector<int> res;

int k = strlen(t);

int cur = -1;

for (int i = 0; i < k; i++) {

while (cur >= 0 && t[i] != s[cur + 1]) cur = link[cur];

if (t[i] == s[cur + 1]) cur++;

if (cur == n - 1) {

res.push\_back(i - n + 1);

cur = link[cur];

}

}

return res;

}

} kmp;

int main() {

kmp.build((char\*) "123123");

vector<int> res = kmp.query((char\*) "412312312312312312341231231231234");

for (int i = 0; i < res.size(); i++) cout << res[i] << " \n"[i == res.size() - 1]; //Expected 1 4 7 10 13 20 23 26

return 0;

}

**Manacher.cpp**

#include <bits/stdc++.h>

using namespace std;

/\*

\* Complexity: O(N)

\*/

struct Manachar {

int n;

vector<int> d; //Radius of odd palindromes

vector<int> e; //Radius of even palindromes

int build(char\* s) {

n = strlen(s), d.resize(n), e.resize(n);

int res = 0;

int l = 0, r = -1;

for (int i = 0; i < n; ++i) {

int k = (i > r) ? 1 : min(d[l + r - i], r - i) + 1;

while (i - k >= 0 && i + k < n && s[i - k] == s[i + k]) k++;

d[i] = --k;

res = max(res, k + k + 1);

if (r < i + k) {

l = i - k;

r = i + k;

}

}

l = 0; r = -1;

for (int i = 0; i < n; ++i) {

int k = (i > r) ? 1 : min(e[l + r - i + 1], r - i + 1) + 1;

while (i - k >= 0 && i + k - 1 < n && s[i - k] == s[i + k - 1]) k++;

e[i] = --k;

res = max(res, k + k);

if (r < i + k - 1) {

l = i - k;

r = i + k - 1;

}

}

return res;

}

} mnc;

int main() {

cout << mnc.build((char\*) "aaadefede") << "\n";

for (int i = 0; i < mnc.n; i++) {

cout << mnc.d[i] << " \n"[i == mnc.n - 1];

}

for (int i = 0; i < mnc.n; i++) {

cout << mnc.e[i] << " \n"[i == mnc.n - 1];

}

return 0;

}