**DINIC**

struct edge {

int to;

int f;

int c;

edge (int aa, int bb, int cc) {

to = aa;

f = bb;

c = cc;

}

};

int K;

int s, t;

int n, m;

vector<edge> edges;

vi g[MX];

int dist[MX];

bool bfs() {

queue<int> q;

q.push(s);

fill(dist, dist + n, INF);

dist[s] = 0;

while (!q.empty()) {

int v = q.front();

q.pop();

for (auto &id : g[v]) { {

int to = edges[id].to;

int can = edges[id].c - edges[id].f;

if (can < K || dist[to] != INF) continue;

dist[to] = dist[v] + 1;

if (to == t)

return true;

q.push(to);

}

}

return false;

}

int it[MX];

int dfs(int v, int mn = INF) {

if (v == t)

return mn;

while (it[v] != siz(g[v])) {

int id = g[v][it[v]];

int to = edges[id].to;

int can = edges[id].c - edges[id].f;

if (can >= K && dist[to] == dist[v] + 1) {

int res = dfs(to, min(mn, can));

if (res) {

edges[id].f += res;

edges[id^1].f -= res;

return res;

}

}

++it[v];

}

return 0;

}

int maxflow() {

int flow = 0;

K = 1 << 30;

while (K) {

while (bfs()) {

memset(it, 0, sizeof(int) \* n);

int delta;

while (delta = dfs(s))

flow += delta;

}

K >>= 1;

}

return flow;

}

**CENTROID DECOMPOSITION**

int lvl[MX];

int pr[MX];

int dfs(int v, int p, int sz, int &cent) {

int sum = 1;

for (auto to : g[v]) {

if (lvl[to] == -1 && to != p)

sum += dfs(to, v, sz, cent);

}

if ((p == -1 || sum \* 2 >= sz) && cent == -1)

cent = v;

return sum;

}

void build(int v, int sz, int level = 0, int last = -1) {

int cent = -1;

dfs(v, -1, sz, cent);

pr[cent] = last;

lvl[cent] = level;

for (auto to : g[cent]) {

if (lvl[to] == -1)

build(to, sz / 2, level + 1, cent);

}

}

**HLD**

int n;

vi g[MX];

int sz[MX], pr[MX];

void dfs(int v, int p) {

pr[v] = p;

sz[v] = 1;

int mx = 0;

for (int i = 0; i < siz(g[v]); ++i) {

int to = g[v][i];

if (to == p) continue;

dfs(to, v);

sz[v] += sz[to];

if (sz[to] > mx) {

mx = sz[to];

swap(g[v][0], g[v][i]);

}

}

}

int tin[MX], tout[MX], t = 0;

int up[MX];

void build(int v, int p) {

tin[v] = t++;

for (auto to : g[v]) {

if (to == p) continue;

up[to] = (to == g[v][0] ? up[v] : to);

build(to, v);

}

tout[v] = t;

}

bool is\_anc(int a, int b) {

return (tin[a] <= tin[b] && tout[a] >= tout[b]);

}

int ascend(int ja, int b, int &ans) {

while (!is\_anc(up[ja], b)) {

ans = max(ans, get(1, tin[up[ja]], tin[ja] + 1, 0, n));

ja = pr[up[ja]];

}

return ja;

}

int req(int a, int b) {

int ans = 0;

int ja = ascend(a, b, ans);

int jb = ascend(b, a, ans);

if (is\_anc(jb, ja))

swap(ja, jb);

return max(ans, get(1, tin[ja], tin[jb] + 1, 0, n));

}

**IMPLICT KEY TREAP**

struct nd {

int prior, key, sz;

nd \*l, \*r;

nd(int key): prior(rng()), sz(1), key(key), l(nullptr), r(nullptr) {};

};

int getsz(nd \* t) {

if (!t)

return 0;

return t->sz;

}

void upd(nd \*t) {

if (t)

t->sz = getsz(t->l) + getsz(t->r) + 1;

}

void push(nd \*t) {

}

pair<nd\*, nd\*> split(nd\* t, int k) {

if (!t)

return {nullptr, nullptr};

push(t);

if (getsz(t->l) >= k) {

auto splited = split(t->l, k);

t->l = splited.second;

upd(t);

return {splited.first, t};

} else {

auto splited = split(t->r, k - getsz(t->l) - 1);

t->r = splited.first;

upd(t);

return {t, splited.second};

}

}

nd\* merge(nd\* t1, nd\* t2) {

if (!t1)

return t2;

if (!t2)

return t1;

if (t1->prior > t2->prior) {

push(t1);

t1->r = merge(t1->r, t2);

upd(t1);

return t1;

} else {

push(t2);

t2->l = merge(t1, t2->l);

upd(t2);

return t2;

}

}

nd\* DeleteKth(nd\* t, int k) {

if (getsz(t->l) + 1 == k)

return merge(t->l, t->r);

if (getsz(t->l) >= k) {

t->l = DeleteKth(t->l, k);

upd(t);

return t;

} else {

t->r = DeleteKth(t->r, k - getsz(t->l) - 1);

upd(t);

return t;

}

}

nd\* Insert(nd\* t, int key, int place) {

nd \*key\_node = new nd(key);

auto splited = split(t, place - 1);

return merge(merge(splited.first, key\_node), splited.second);

}

nd\* PushBack(nd\* t, int key) { 

nd \* key\_node = new nd(key);

return merge(t, key\_node);

}

int GetKth(nd \*t, int k) {

auto split1 = split(t, k);

auto split2 = split(split1.first, k - 1);

nd\* ans\_node = split2.second;

push(ans\_node);

int answer = ans\_node->key;

t = merge(merge(split2.first, ans\_node), split1.second);

return answer;

}

void Print(nd \*t) {

int elem\_num = getsz(t);

rep(i, 1, elem\_num + 1) {

cout << GetKth(t, i) << ' ';

}

}

**TREAP**

struct nd {

int key;

int prior;

int sz;

nd \*l;

nd \*r;

nd (int x): key(x), prior(rng()), sz(1), l(nullptr), r(nullptr) {}

};

int getsz(nd \* t) {

if (!t)

return 0;

return t->sz;

}

void upd(nd \* t) {

t->sz = 1 + getsz(t->l) + getsz(t->r);

}

void push(nd \* t) {

}

pair<nd\*, nd\*> splitx(nd \* t, int x) {

if (!t)

return {nullptr, nullptr};

push(t);

if (t->key > x) {

auto spl = splitx(t->l, x);

t->l = spl.second;

upd(t);

return {spl.first, t};

} else {

auto spl = splitx(t->r, x);

t->r = spl.first; 

upd(t);

return {t, spl.second};

}

}

pair<nd\*, nd\*> splitk(nd \* t, int k) {

if (!t)

return {nullptr, nullptr};

push(t);

if (getsz(t->l) >= k) {

auto spl = splitk(t->l, k);

t->l = spl.second;

upd(t);

return {spl.first, t};

} else {

auto spl = splitk(t->r, k - getsz(t->l) - 1);

t->r = spl.first;

upd(t);

return {t, spl.second};

}

}

nd \* merge(nd \* t1, nd \* t2) {

if (!t1)

return t2;

if (!t2)

return t1;

if (t1->prior > t2->prior) {

push(t1);

t1->r = merge(t1->r, t2);

upd(t1);

return t1;

} else {

push(t2);

t2->l = merge(t1, t2->l);

upd(t2);

return t2;

}

}

int max(nd \* t) {

push(t);

if (t->r)

return max(t->r);

return t->key;

}

int min(nd \* t) {

push(t);

if (t->l)

return min(t->l);

return t->key;

}

nd \* insert(nd \* t, int x) {

nd \* key = new nd(x);

auto spl = splitx(t, x);

auto spl2 = splitx(spl.first, x - 1);

return merge(merge(spl2.first, key), spl.second);

}

nd \* erasex(nd \* t, int x) {

auto spl = splitx(t, x);

auto spl2 = splitx(spl.first, x - 1);

return merge(spl2.first, spl.second);

}

nd \* erasek(nd \* t, int k) {

auto spl = splitk(t, k);

auto spl2 = splitk(spl.first, k - 1);

return merge(spl2.first, spl.second);

}

int getk(nd \* &t, int k) {

auto spl = splitk(t, k);

auto spl2 = splitk(spl.first, k - 1);

int res = spl2.second->key;

t = merge(merge(spl2.first, spl2.second), spl.second);

return res;

}

int lessx(nd \* &t, int x) {

auto spl = splitx(t, x - 1);

int res = getsz(spl.first);

t = merge(spl.first, spl.second);

return res;

}

int first\_more(nd \*&t, int x) {

auto spl = splitx(t, x);

int res;

if (!spl.second)

res = INF;

else

res = min(spl.second);

t = merge(spl.first, spl.second);

return res;

}

int lower\_bound(nd \*&t, int x) {

auto spl = splitx(t, x - 1);

int res;

if (!spl.second)

res = INF;

else

res = min(spl.second);

t = merge(spl.first, spl.second);

return res;

}

int first\_less(nd \*&t, int x) {

auto spl = splitx(t, x - 1);

int res;

if (!spl.first)

res = INF;

else

res = max(spl.first);

t = merge(spl.first, spl.second);

return res;

}

void print(nd \* t, bool end = true) {

if (!t)

return;

print(t->l, false);

cout << t->key << ' ' << t->sz << " ";

print(t->r, false);

if (end) cout << endl;

}

**PERSISTENT SEGMENT TREE**

struct nd {

int l;

int r;

int sum;

nd (int \_l = 0, int \_r = 0, int \_sum = 0) {

l = \_l;

r = \_r;

sum = \_sum;

}

};

nd t[MX \* 20];

int rt[MX];

int sz = 0;

int init() {

t[sz] = nd();

return sz++;

}

void build(int v, int l, int r) {

if (l + 1 == r)

return;

int m = (l + r) / 2;

build(t[v].l = init(), l, m);

build(t[v].r = init(), m, r);

}

int get(int v1, int v2, int k, int cl, int cr) {

if (cl + 1 == cr)

return cl;

int m = (cl + cr) / 2;

int sumleft = t[t[v2].l].sum - t[t[v1].l].sum;

if (sumleft >= k)

return get(t[v1].l, t[v2].l, k, cl, m);

return get(t[v1].r, t[v2].r, k - sumleft, m, cr);

}

int upd(int v, int i, int cl, int cr) {

int u = init();

if (cl + 1 == cr) {

t[u].sum = t[v].sum + 1;

return u;

}

t[u] = t[v];

int m = (cl + cr) / 2;

if (i < m) {

t[u].l = upd(t[v].l, i, cl, m);

} else {

t[u].r = upd(t[v].r, i, m, cr); 

}

t[u].sum = t[t[u].l].sum + t[t[u].r].sum;

return u;

}

**EXT\_GCD**

int gcd(int a, int b, int &x, int &y) {

if (b == 0) {

x = 1;

y = 0;

return a;

}

int g = gcd(b, a % b, y, x);

y -= x \* (a / b);

return g;

}

**RABIN-MILLER**

ll mult(ll a, ll b, ll mod) {

\_\_int128 res = a;

res \*= b;

return res % mod;

}

ll pw(ll a, ll b, ll mod) {

if (b == 0) return 1;

if (b & 1) return mult(a, pw(a, b - 1, mod), mod);

return pw(mult(a, a, mod), b / 2, mod);

}

bool is\_prob\_prime(ll x, int iters) {

if (x == 1)

return false;

if (x == 2)

return true;

if ((x & 1) == 0)

return false;

int k = \_\_builtin\_ctzll(x - 1);

ll m = (x - 1) / (1 << k);

for (int it = 0; it < iters; ++it) {

ll num = rng() % (x - 1) + 1;

ll cur = pw(num, m, x);

if (cur == 1 || cur == x - 1)

continue;

bool good = false;

for (int pw2 = 1; pw2 < k; ++pw2) {

cur = mult(cur, cur, x);

if (cur == x - 1) {

good = true;

break;

}

}

if (!good)

return false;

}

return true;

}

**POLLARD**

inline \_\_int128 sub(\_\_int128 a, \_\_int128 b) {

return a - b;

}

inline \_\_uint128\_t f(\_\_uint128\_t x) {

return (x + 1) \* (x + 1);

}

inline \_\_int128 abs(\_\_int128 x) {

if (x < 0)

return -x;

return x;

}

ull gcd(ull a, ull b) {

while (b) {

a %= b;

swap(a, b);

}

return a;

}

map<ull, ull> divs;

ull pollard(ull x, ull seed = 1) {

ull i = seed;

ull j = seed;

ull res = 1;

int it = 1e5;

while ((res == 1 || res == x) && it--) {

i = f(i) % x;

j = f(j) % x;

j = f(j) % x;

ull val = abs(sub(i, j));

res = gcd(val, x);

}

return res;

}

void slow(ull x) {

ull d = 2;

while (d \* d <= x) {

while (x % d == 0) {

++divs[d];

x /= d;

}

++d;

}

if (x != 1)

++divs[x];

}

void factor(ull x) {

if (x <= 1e12) {

slow(x);

return;

}

ull d = pollard(x);

if (d == 1 || d == x) {

divs[x]++;

return;

}

factor(d);

factor(x / d);

}

**FFT mod 998244353**

int bitrev[NN];

int w[NN];

void init() {

int W = 2;

while (true) {

ll x = W;

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

x = mult(x, x);

if (x == MOD - 1) break;

++W;

}

w[0] = 1;

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

bitrev[i] = (bitrev[i >> 1] >> 1) ^ ((i & 1) << (LOG - 1));

w[i] = mult(w[i - 1], W);

}

}

void fft(vi &a, int k) {

int L = 1 << k;

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

int x = bitrev[i] >> (LOG - k);

if (x > i) {

swap(a[i], a[x]);

}

}

rep(lvl, 0, k) {

int len = 1 << lvl;

for (int i = 0; i < L; i += (len << 1)) {

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

int x = a[i + j];

int y = mult(w[j << (LOG - 1 - lvl)], a[i + j + len]);

a[i + j] = add(x, y);

a[i + j + len] = sub(x, y);

}

}

}

}

void invfft(vi & c, int k) {

fft(c, k);

int L = 1 << k;

reverse(c.begin() + 1, c.begin() + L);

int RL = rev(L);

for (auto&x : c)

x = mult(x, RL);

}

vi mult(vi a, vi b) {

int k = 0;

while ((1 << k) < siz(a) + siz(b) - 1)

++k;

int L = 1 << k;

a.resize(L, 0);

b.resize(L, 0);

fft(a, k);

fft(b, k);

vi c(L);

for (int i = 0; i < L; ++i)

c[i] = mult(a[i], b[i]);

invfft(c, k);

return c;

}

**FFT ANY MOD**

int bitrev[NN];

cd w[NN];

void init() {

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

bitrev[i] = ((i & 1) << (LOG - 1)) ^ (bitrev[i >> 1] >> 1);

w[i] = complex(cos(2 \* PI \* i / N), sin(2 \* PI \* i / N));

}

}

void fft(vector<cd> & a, int k) {

int L = 1 << k;

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

int x = bitrev[i] >> (LOG - k);

if (i > x)

swap(a[i], a[x]);

}

rep(lvl, 0, k) {

int len = 1 << lvl;

for (int i = 0; i < L; i += len << 1) {

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

cd x = a[i + j];

cd y = a[i + j + len] \* w[j << (LOG - 1 - lvl)];

a[i + j] = x + y;

a[i + j + len] = x - y;

}

}

}

}

void invfft(vector<cd> & a, int k) {

fft(a, k);

int L = 1 << k;

reverse(a.begin() + 1, a.begin() + L);

for (auto&x : a) {

x /= L;

x = cd(((long long)round(x.real())) % MOD, 0);

}

}

void norm(vector<ll> & x) {

while (siz(x) && !x.back())

x.ppb;

}

vector<ll> mult(vector<ll> & a, vector<ll> & b) {

norm(a);

norm(b);

int k = 0;

while ((1 << k) < siz(a) + siz(b) - 1)

++k;

int L = 1 << k;

vector<vector<cd>> ac(2, vector<cd>(L));

vector<vector<cd>> bc(2, vector<cd>(L));

for (int i = 0; i < siz(a); ++i) {

ac[0][i] = a[i] & (BASE - 1);

ac[1][i] = a[i] >> LOGB;

}

for (int i = 0; i < siz(b); ++i) {

bc[0][i] = b[i] & (BASE - 1);

bc[1][i] = b[i] >> LOGB;

}

fft(bc[0], k);

fft(bc[1], k);

fft(ac[0], k);

fft(ac[1], k);

vector<ll> res(L, 0);

vector<cd> c(L);

for (int i = 0; i < 2; ++i) {

for (int j = 0; j < 2; ++j) {

rep(len, 0, L)

c[len] = ac[i][len] \* bc[j][len];

invfft(c, k);

rep(len, 0, L)

res[len] = add(res[len], mult(Q[i + j], (int)round(c[len].real()) % MOD));

}

}

norm(res);

return res;

}

**FFT COMPLEX**

int bitrev[NN];

cd w[NN];

void init() {

for (int i = 0; i < N; ++I) {

bitrev[i] = ((i & 1) << (LOG - 1)) ^ (bitrev[i >> 1] >> 1);

w[i] = complex(cos(2 \* PI \* i / N), sin(2 \* PI \* i / N));

}

}

void fft(vector<cd> & a, int k) {

int L = 1 << k;

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

int x = bitrev[i] >> (LOG - k);

if (i > x)

swap(a[i], a[x]);

}

rep(lvl, 0, k) {

int len = 1 << lvl;

for (int i = 0; i < L; i += len << 1) {

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

cd x = a[i + j];

cd y = a[i + j + len] \* w[j << (LOG - 1 - lvl)];

a[i + j] = x + y;

a[i + j + len] = x - y;

}

}

}

}

void invfft(vector<cd> & a, int k) {

fft(a, k);

int L = 1 << k;

reverse(a.begin() + 1, a.begin() + L);

for (auto &x : a) {

x /= L;

}

}

vector<ll> mult(vector<ll> & a, vector<ll> & b) {

int k = 0;

while ((1 << k) < siz(a) + siz(b) - 1)

++k;

int L = 1 << k;

vector<cd> ac(L);

vector<cd> bc(L);

vector<cd> cc(L);

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

ac[i] = a[i];

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

bc[i] = b[i];

fft(ac, k);

fft(bc, k);

for (int i = 0; i < L; ++i)

cc[i] = ac[i] \* bc[i];

invfft(cc, k);

vector<ll> c(L);

for (int i = 0; i < L; ++i)

c[i] = (long long) round(cc[i].real());

return c;

}

**GEOM DOUBLE**

int sgn(double x) {

if (x > EPS)

return 1;

if (x < -EPS)

return -1;

return 0;

}

double fix(double x) {

if (x > 0)

x = min(x, 1 - EPS);

if (x < 0)

x = max(x, -1 + EPS);

return x;

}

bool eq(double a, double b) {

return abs(a - b) < EPS;

}

bool lt(double a, double b) {

return a + EPS < b;

}

bool mt(double a, double b) {

return a - EPS > b;

}

struct pt {

double x;

double y;

pt (double \_x = 0, double \_y = 0) {

x = \_x;

y = \_y;

}

pt (pt a, pt b) {

x = b.x - a.x;

y = b.y - a.y;

}

pt perp() {

return pt(-y, x);

}

pt turn(double a) {

return pt(x \* cos(a) - y \* sin(a), x \* sin(a) + y \* cos(a));

}

double len() {

return sqrt(x \* x + y \* y);

}

pt operator + (pt p) {

return pt(x + p.x, y + p.y);

}

double operator \* (pt p) {

return x \* p.y - y \* p.x;

}

double operator ^ (pt p) {

return x \* p.x + y \* p.y;

}

bool operator < (const pt & p) const {

return lt(x, p.x) || (eq(x, p.x) && lt(y, p.y));

}

};

istream& operator >> (istream & cin, pt & p) {

cin >> p.x >> p.y;

return cin;

}

ostream& operator << (ostream & cout, pt & p) {

cout << p.x << ' ' << p.y;

return cout;

}

#define vec pt

double dist(pt a, pt b) {

return pt(a, b).len();

}

double ang(pt a, pt b) {

return atan2(a \* b, a ^ b);

}

double segdist(pt a, pt b, pt c) {

double res = min(dist(a, c), dist(b, c));

vec ab(a, b);

vec ac(a, c);

vec ba(b, a);

vec bc(b, c);

if ((ab ^ ac) > 0 && (ba ^ bc) > 0)

return abs(ac \* bc) / dist(a, b);

return res;

}

bool cmp(pt a, pt b) {

if (eq(a \* b, 0))

return a.len() < b.len();

return a \* b > 0;

}

vector<pt> build\_hull(vector<pt> pts) {

int n = pts.size();

pt p = pts[0];

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

mine(p, pts[i]);

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

pts[i] = pt(p, pts[i]);

sort(all(pts), cmp);

vector<pt> hull = {pts[0]};

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

while (hull.size() > 1) {

if (pt(hull[siz(hull) - 2], hull.back()) \* pt(hull.back(), pts[i]) < EPS)

hull.ppb;

else

break;

}

hull.pb(pts[i]);

}

for (auto &x : hull) {

x = x + p;

dbg(x);

}

return hull;

}

struct line {

double a;

double b;

double c;

line (double \_a = 0, double \_b = 0, double \_c = 0) {

a = \_a;

b = \_b;

c = \_c;

}

line(pt p1, pt p2) {

a = p1.y - p2.y;

b = p2.x - p1.x;

c = -a \* p1.x - b \* p1.y;

}

double dist(pt p) {

return abs(p.x \* a + p.y \* b + c) / sqrt(a \* a + b \* b);

}

};

pt inter(line f1, line f2) {

double x = f1.c \* f2.b - f1.b \* f2.c;

double y = f1.a \* f2.c - f1.c \* f2.a;

double d = f1.a \* f2.b - f1.b \* f2.a;

return pt(x / -d, y / -d);

}

struct cir {

pt C;

double r;

cir (pt \_C, double \_r) {

C = \_C;

r = \_r;

}

cir (pt a, pt b, pt c) {

pt mab((a.x + b.x) / 2, (a.y + b.y) / 2);

vec ab = pt(a, b).perp();

line l1(mab, mab + ab);

pt mac((a.x + c.x) / 2, (a.y + c.y) / 2);

vec ac = pt(a, c).perp();

line l2(mac, mac + ac);

C = inter(l1, l2);

r = dist(C, a);

assert(eq(r, dist(C, a)) && eq(r, dist(C, b)) && eq(r, dist(C, c)));

}

vector<pt> tang(pt p) {

double d = dist(p, C);

if (eq(r, d))

return {p};

double L = sqrt(d \* d - r \* r);

double a = acos(fix(L / d));

vec pc(p, C);

pc = pc \* (L / d);

return {p + pc.turn(a), p + pc.turn(-a)};

}

};

**GEOM INT**

struct pt {

int x;

int y;

pt (int \_x = 0, int \_y = 0) {

x = \_x;

y = \_y;

}

pt (pt a, pt b) {

x = b.x - a.x;

y = b.y - a.y;

}

pt perp() {

return pt(-y, x);

}

ll len2() {

return x \* x + y \* y;

}

double len() {

return sqrt(len2());

}

pt operator + (pt p) {

return pt(x + p.x, y + p.y);

}

ll operator \* (pt p) {

return x \* p.y - y \* p.x;

}

ll operator ^ (pt p) {

return x \* p.x + y \* p.y;

}

bool operator < (const pt & p) const {

return mp(x, y) < mp(p.x, p.y);

}

};

#define vec pt

istream& operator >> (istream & cin, pt & p) {

cin >> p.x >> p.y;

return cin;

}

ostream& operator << (ostream & cout, pt & p) {

cout << p.x << ' ' << p.y;

return cout;

}

ll dist2(pt a, pt b) {

return pt(a, b).len2();

}

double dist(pt a, pt b) {

return sqrt(dist2(a, b));

}

double ang(pt a, pt b) {

return atan2(a \* b, a ^ b);

}

double segdist(pt a, pt b, pt c) {

double res = min(dist(a, c), dist(b, c));

vec ab(a, b);

vec ac(a, c);

vec ba(b, a);

vec bc(b, c);

if ((ab ^ ac) > 0 && (ba ^ bc) > 0)

return abs(ac \* bc) / dist(a, b);

return res;

}

bool cmp(pt a, pt b) {

if (a \* b == 0)

return a.len2() < b.len2();

return a \* b > 0;

}

vector<pt> build\_hull(vector<pt> pts) {

int n = pts.size();

pt p = pts[0];

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

mine(p, pts[i]);

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

pts[i] = pt(p, pts[i]);

sort(all(pts), cmp);

vector<pt> hull = {pts[0]};

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

while (hull.size() > 1) {

if (pt(hull[siz(hull) - 2], hull.back()) \* pt(hull.back(), pts[i]) < EPS)

hull.ppb;

else

break;

}

hull.pb(pts[i]);

}

for (auto &x : hull) {

x = x + p;

}

return hull;

}

struct line {

int a;

int b;

int c;

line (int \_a = 0, int \_b = 0, int \_c = 0) {

a = \_a;

b = \_b;

c = \_c;

}

line(pt p1, pt p2) {

a = p1.y - p2.y;

b = p2.x – p1.x; 

c = -a \* p1.x - b \* p1.y;

}

double dist(pt p) {

return abs(p.x \* a + p.y \* b + c) / sqrt(a \* a + b \* b);

}

};

struct cir {

pt C;

double r;

cir (pt \_C, double \_r) {

C = \_C;

r = \_r;

}

};

**Manacher`s algorithm**

int l = 0, r = 0;

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

if (r >= i)

d1[i] = min(r - i + 1, d1[l + r - i]);

while (i + d1[i] < n && i - d1[i] >= 0 && s[i + d1[i]] == s[i - d1[i]])

++d1[i];

if (i + d1[i] - 1 > r) {

r = i + d1[i] - 1;

l = i - d1[i] + 1;

}

}

l = 0, r = -1;

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

if (r >= i)

d2[i] = min(r - i + 1, d2[l + r - i + 1]);

while (i + d2[i] < n && i - d2[i] - 1 >= 0 && s[i + d2[i]] == s[i - 1 - d2[i]])

++d2[i];

if (i + d2[i] - 1 > r) {

r = i + d2[i] - 1;

l = i - d2[i];

}

}

**LCP**

int lcp[MX];

int pos[MX];

void build\_lcp(str s) {

s += '#';

int n = siz(s);

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

pos[p[i]] = i;

int cur = 0;

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

if (pos[i] == n - 1)

continue;

cur = max(0, cur - 1);

int j = p[pos[i] + 1];

int sz = min(n - i, n - j);

while (cur < sz && s[i + cur] == s[j + cur])

++cur;

lcp[pos[i]] = cur;

}

}

**SUFFIX ARRAY**

int cnt[MX];

int c[MX];

int cn[MX];

int p[MX];

int pn[MX];

int classes;

void sufarr(str s) {

s += '#';

int n = s.size();

for (auto &x : s)

++cnt[x];

partial\_sum(cnt, cnt + MX, cnt);

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

p[--cnt[s[i]]] = i;

classes = 1;

c[p[0]] = 0;

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

if (s[p[i]] != s[p[i - 1]]) ++classes;

c[p[i]] = classes - 1;

}

for (int k = 0; (1 << k) < n; ++k) {

memset(cnt, 0, classes \* sizeof(int));

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

pn[i] = p[i] - (1 << k);

if (pn[i] < 0)

pn[i] += n;

++cnt[c[i]];

}

partial\_sum(cnt, cnt + classes, cnt);

fi1(n, 0)

p[--cnt[c[pn[i]]]] = pn[i];

classes = 1;

cn[p[0]] = 0;

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

int lw = p[i - 1];

int rw = lw + (1 << k);

int l = p[i];

int r = l + (1 << k);

if (rw >= n) rw -= n;

if (r >= n) r -= n;

classes += c[l] != c[lw] || c[r] != c[rw];

cn[p[i]] = classes - 1;

}

memcpy(c, cn, n \* sizeof(int));

}

}

**SUFFIX AUTOMATON**

struct node {

int pr;

int s;

int to[K];

node () {

pr = -1;

s = -1;

memset(to, -1, sizeof(to));

}

};

vector<node> t;

int add(int a, char ch) {

int b = t.size();

t.push\_back(node());

t[b].pr = a;

t[b].s = 0;

while (a != -1) {

if (t[a].to[ch] == -1) {

t[a].to[ch] = b;

a = t[a].s;

continue;

}

int c = t[a].to[ch];

if (t[c].pr == a) {

t[b].s = c;

break;

}

int d = t.size();

t.push\_back(node());

t[d].pr = a;

t[d].s = t[c].s;

t[b].s = t[c].s = d;

memcpy(t[d].to, t[c].to, sizeof(t[d].to));

for (; a != -1 && t[a].to[ch] == c; a = t[a].s)

t[a].to[ch] = d;

break;

}

return b;

}

int last;

void init() {

last = 0;

t.push\_back(node());

}

void addstring(str & s) {

for (auto &x : s) {

last = add(last, x);

}

bool findstring(str & s) {

int cur = 0;

for (auto &x : s) {

cur = t[cur].to[x];

if (cur == -1)

return false;

}

return true;

}

**KOPEL ALLOC**

const int MAX\_MEM = 1e8;

int mpos = 0;

char mem[MAX\_MEM];

inline void \* operator new ( size\_t n ) {

char \*res = mem + mpos;

mpos += n;

assert(mpos <= MAX\_MEM);

return (void \*)res;

}

**TEMPLATE**

/\*

#pragma GCC optimize("Ofast")

#pragma GCC optimize("O3")

#pragma GCC optimize("unroll-loops")

#pragma GCC target("sse,sse2,sse3,ssse3,sse4,sse4.1,sse4.2,popcnt,abm,mmx,avx,avx2,tune=native")

\*/

#include <iostream>

#include <cstdio>

#include <vector>

#include <cmath>

#include <map>

#include <algorithm>

#include <string>

#include <utility>

#include <set>

#include <stack>

#include <deque>

#include <ctime>

#include <random>

#include <cassert>

#include <cmath>

#include <climits>

#include <queue>

#include <cstring>

#include <bitset>

#include <iomanip>

#include <chrono>

#ifdef LOCAL

#define dbg(x) cout << #x << " : " << x << endl;

#else

#define dbg(x)

#endif

#define int long long

#define pb push\_back

#define ppb pop\_back()

#define mp make\_pair

#define lb lower\_bound

#define ub upper\_bound

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

#define sz(a) (int)a.size()

#define siz(a) (int)a.size()

#define fi first

#define se second

#define cinv(v) for (auto& x: v) cin >> x

template <typename T1, typename T2> inline bool chkmin(T1 &x, const T2 &y) {if (x > y) {x = y; return 1;} return 0;}

template <typename T1, typename T2> inline bool chkmax(T1 &x, const T2 &y) {if (x < y) {x = y; return 1;} return 0;}

#define mine chkmin

#define maxe chkmax

using namespace std;

typedef long long ll;

typedef unsigned long long ull;

typedef char ch;

typedef string str;

typedef pair<int, int> pii;

typedef vector<int> vi;

typedef vector<vi> vvi;

typedef vector<pii> vpii;

typedef vector<vpii> vvpii;

typedef vector<ch> vch;

typedef vector<str> vs;

const int MOD = (int)1e9 + 7;

const int INF = (int)1e9 + 50;

const long long BIG = (long long)2e18 + 50;

const int MX = 200010;

const double EPS = 1e-9;

mt19937 rng(chrono::steady\_clock::now().time\_since\_epoch().count());

ostream& operator<< (ostream &out, const vi &b) {

for (auto k : b) out << k << " ";

return out;

}

istream& operator>> (istream& in, pii& b) {

in >> b.first >> b.second;

return in;

}

ostream& operator<< (ostream& out, const pii& b) {

out << "{" << b.first << ", " << b.second << "}";

return out;

}

int32\_t main() {

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

cout.tie(0);

}

**comp**

#!/bin/bash

g++ -Wall -Wextra -pedantic -std=c++17 -DLOCAL -O2 -Wshadow -Wconversion -Wshift-overflow=2 -fsanitize=address -fsanitize=undefined -fsanitize=bounds $1.cpp -o $1

**bld**

#!/bin/bash

comp $1

if [ "$2" == "" ]; then

./$1

else ./$1 < $2

fi

**run**

#!/bin/bash

if [ "$2" == "" ]; then

./$1

else ./$1 < $2

fi

**stress**

#!/bin/bash

#$1 - wrong, $2 - slow, $3 - gen

comp $1

comp $2

comp $3

for ((i=1; i<100000; ++i)); do

./$3 > input

./$1 <input> wrong.txt

./$2 <input> slow.txt

diff slow.txt wrong.txt > /dev/null || break

echo "Test $i OK"

done

echo "Test $i WA"

echo "input:"

cat input

echo "correct answer"

cat slow.txt

echo "wrong answer"

cat wrong.txt

**fcomp**

#!/bin/bash

g++ -std=c++17 -DLOCAL -O2 $1.cpp -o $1

**fbld**

#!/bin/bash

fcomp $1

if [ "$2" == "" ]; then

./$1

else ./$1 < $2

fi

**fstress**

#!/bin/bash

#$1 - wrong, $2 - slow, $3 - gen

fcomp $1

fcomp $2

fcomp $3

for ((i=1; i<100000; ++i)); do

./$3 > input

./$1 <input> wrong.txt

./$2 <input> slow.txt

diff slow.txt wrong.txt > /dev/null || break

echo "Test $i OK"

done

echo "Test $i WA"

echo "input:"

cat input

echo "correct answer"

cat slow.txt

echo "wrong answer"

cat wrong.txt

NAMES -> PAGES

dinic 1

centroid 2

hld 2

ДДНК 3

ДД 5

персдо 8

расширенный евклид 9

поллард 10

FFT mod 998244353 11

FFT ANY MOD 12

FFT COMPLEX 13

GEOM DOUBLE 15

GEOM INT 18

АЛГОРИТМ МАНАКЕРА 20

LCP 20

СУФМАС 21

СУФАВТОМАТ 21

КОПЕЛЬ 23

ШАБЛОН 23

СКРИПТЫ 25