d83b #define MOD 1000000007  
77d8 #define MAX 100  
109a typedef long long int lli;  
   
 // returns a^x mod p  
cd7f lli exponent\_(lli a, lli x, lli p) {  
d14d lli ans = 1;  
6067 while (x > 0) {  
0925 if (x % 2 == 1)  
9956 ans = (ans \* a) % p;  
b46f x /= 2;  
5ebe a = (a \* a) % p;  
f288 }  
8194 return ans;  
a471 }  
   
b85b lli modular\_inverse\_(lli a, lli b, lli p) {  
95c1 return ((a % p) \* (exponent\_(b, p - 2, p) % p)) % p;  
0cd3 }  
   
 // return a % b (positive value)  
b7d9 int mod(int a, int b) {  
a055 return ((a%b) + b) % b;  
ad7f }  
   
 // computes gcd(a,b)  
9fdf int gcd(int a, int b) {  
5d42 while (b) { int t = a%b; a = b; b = t; }  
7a9c return a;  
e3a9 }  
   
 // computes lcm(a,b)  
b34f int lcm(int a, int b) {  
4682 return a / gcd(a, b)\*b;  
1249 }  
   
 // returns g = gcd(a, b); finds x, y such that d = ax + by  
4dba int extended\_euclid(int a, int b, int &x, int &y) {  
1c43 int xx = y = 0;  
40bb int yy = x = 1;  
45df while (b) {  
759e int q = a / b;  
19d1 int t = b; b = a%b; a = t;  
4a7f t = xx; xx = x - q\*xx; x = t;  
2d94 t = yy; yy = y - q\*yy; y = t;  
a111 }  
2960 return a;  
0136 }  
   
 // finds all solutions to ax = b (mod n)  
668c vector<int> modular\_linear\_equation\_solver(int a, int b, int n) {  
39ba int x, y;  
3ce5 vector<int> ret;  
19ca int g = extended\_euclid(a, n, x, y);  
fc4b if (!(b%g)) {  
a3d2 x = mod(x\*(b / g), n);  
d9e9 for (int i = 0; i < g; i++)  
edb3 ret.push\_back(mod(x + i\*(n / g), n));  
9f10 }  
a014 return ret;  
a57d }  
   
 // computes b such that ab = 1 (mod n), returns -1 on failure  
7375 int mod\_inverse(int a, int n) {  
1b05 int x, y;  
9955 int g = extended\_euclid(a, n, x, y);  
7da9 if (g > 1) return -1;  
d848 return mod(x, n);  
46bf }  
   
 // 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.  
f376 pair<int, int> chinese\_remainder\_theorem(int m1, int r1, int m2, int r2) {  
0313 int s, t;  
d54b int g = extended\_euclid(m1, m2, s, t);  
29cd if (r1%g != r2%g) return make\_pair(0, -1);  
0763 return make\_pair(mod(s\*r2\*m1 + t\*r1\*m2, m1\*m2) / g, m1\*m2 / g);  
1846 }  
   
 // Chinese remainder theorem: find z such that  
 // z % m[i] = r[i] for all i. Note that the solution is  
 // unique modulo M = lcm\_i (m[i]). Return (z, M). On  
 // failure, M = -1. Note that we do not require the a[i]'s  
 // to be relatively prime.  
e8e4 pair<int, int> chinese\_remainder\_theorem(const vector<int> &m, const vector<int> &r) {  
c8f4 pair<int, int> ret = make\_pair(r[0], m[0]);  
9909 for (int i = 1; i < m.size(); i++) {  
213f ret = chinese\_remainder\_theorem(ret.second, ret.first, m[i], r[i]);  
948d if (ret.second == -1) break;  
6cd9 }  
dce6 return ret;  
369a }  
   
 // computes x and y such that ax + by = c  
 // returns whether the solution exists  
6024 bool linear\_diophantine(int a, int b, int c, int &x, int &y) {  
21d0 if (!a && !b) {  
2b2f if (c) return false;  
085b x = 0; y = 0;  
a279 return true;  
cd6e }  
3b73 if (!a) {  
4265 if (c % b) return false;  
990e x = 0; y = c / b;  
095b return true;  
d837 }  
1918 if (!b) {  
86b2 if (c % a) return false;  
f0d7 x = c / a; y = 0;  
bb88 return true;  
45a1 }  
784b int g = gcd(a, b);  
4417 if (c % g) return false;  
9be5 x = c / g \* mod\_inverse(a / g, b / g);  
413a y = (c - a\*x) / b;  
60eb return true;  
5b7a }  
   
cbde int is\_prime[MAX + 1];  
baab vector<int> primes;  
0ed2 void calc\_primes() {  
42f4 for (int i = 2; i <= MAX; i++) is\_prime[i] = true;  
3719 for (int i = 2; i < MAX; i++) {  
b45f if (!is\_prime[i]) continue;  
4d91 for (int k = i + i; k\*k <= MAX; k += i)  
a573 is\_prime[k] = false;  
4d97 primes.push\_back(i);  
ba11 }  
8dad }  
   
9e09 lli fact[MAX];  
9e5c void calculate\_factorial() {  
8e38 fact[0] = 1;  
8c20 for (int i = 1; i < MAX; i++)  
fa88 fact[i] = fact[i - 1] \* i;  
47a9 }  
   
 // (r choose n) mod p  
a665 lli combinatorics\_(lli n, lli r, lli p) {  
2d25 calculate\_factorial();  
ac40 return (modular\_inverse\_(fact[n], (fact[r] \* fact[n - r]) % p, p)) % p;  
051f }  
   
 // pascal's triangle to get exact values for combinatorics.  
 // n choose k  
4d5e lli C[MAX];  
fd7c lli pascal\_comb\_(lli n, int k) {  
d6ca for (int j = 0; j < MAX; j++) C[j] = 0;  
f196 C[0] = 1; // nC0 is 1  
50ab for (int i = 1; i <= n; i++)  
6631 for (lli j = min(i, k); j > 0; j--)  
bb07 C[j] = C[j] + C[j - 1];  
accf return C[k];  
7d1b }  
   
 // A O(n^2) time and O(n^2) extra space method for Pascal's Triangle  
9966 lli pascal\_triangle[MAX][MAX];  
a65f void make\_pascal(int n) {  
12ec for (int line = 0; line < n; line++) {  
3229 for (int i = 0; i <= line; i++) {  
4386 if (line == i || i == 0)  
58de pascal\_triangle[line][i] = 1;  
6621 else  
6659 pascal\_triangle[line][i] = pascal\_triangle[line - 1][i - 1] + pascal\_triangle[line 1635 - 1][i];  
a8cc }  
653b }  
db54 }  
   
226e lli binomialCo(int N, int r) {  
1042 lli res = 1;  
b1e4 r = (r<(N - r)) ? r : (N - r);  
f7d9 for (int i = 1; i <= r; i++, N--) {  
4299 res \*= N;  
6c3a res /= i;  
d31c }  
8b17 return res;  
bc25 }  
   
 // we need to multiply the numbers in vector a and those in vector b  
 // then determine which product is bigger. return true if a > b;  
2270 bool compare\_product(vector<lli> a, vector<lli> b) {  
 // assumes all values in a and b are positive.  
da1b double logA = 0, logB = 0;  
949a for (int i = 0; i < a.size(); i++) logA += log10(a[i]);  
7d0e for (int i = 0; i < b.size(); i++) logB += log10(b[i]);  
9937 if (logA > logB) return true;  
1750 return false;  
80c7 }  
   
 // create list of primitive pythagorean triples  
7462 vector<pair<pair<int, int>, int> > pyth;  
ff0a void pythagorean\_triples(int N) {  
1479 for (int i = 1; i <= N; i++) {  
97d6 for (int k = 1; k < i; k++) {  
c77c if (gcd(i, k) != 1) continue;  
fbd9 if (i % 2 == 1 && k % 2 == 1) continue;  
   
72cc int a = i\*i - k\*k;  
041f int b = 2 \* i\*k;  
90d3 int c = i\*i + k\*k;  
a733 pyth.push\_back(make\_pair(make\_pair(a, b), c));  
9d44 }  
2497 }  
b959 }  
   
6353 int eulerPhiDirect(int n) {  
325b int result = n;  
ba1f for (int i = 2; i <= n; i++) {  
5050 if (is\_prime[i])  
071e result -= result / i;  
f045 }  
3c27 return result;  
399c }  
   
 // euler totient. with sieve  
a931 int eulerPhi[1000];  
4792 void eulerSieve(int N) {  
0bcb for (int i = 1; i <= N; i++)  
9b29 eulerPhi[i] = i;  
05cb for (int i = 1; i <= N; i++) {  
3d2d if (is\_prime[i]) {  
0251 for (int j = i; j <= N; j += i)  
b01d eulerPhi[j] -= eulerPhi[j] / i;  
edfd }  
ef12 }  
9705 }  
   
 // Fibonacci numbers.  
398d /\* Useful formulas.  
   
ab20 Every positive integer can be represented uniquely as a sum of two or more  
1fb7 distinct Fibonacci numbers. greedy.  
   
c887 F(n+1) \* F(n-1) - F(n) \* F(n) = (-1) ^ n  
5505 F(A+B) = F(A) \* F(B+1) + F(A-1) \* F(B)  
   
1df4 sum of first n fibonacci numbers = F(n+2) - 1  
   
4033 gcd(F(n), F(m)) = F(gcd(n, m))  
5cfe \*/  
a57c void multiply\_fibo(lli F[2][2], lli M[2][2]) {  
e462 lli x = F[0][0] \* M[0][0] + F[0][1] \* M[1][0];  
6342 lli y = F[0][0] \* M[0][1] + F[0][1] \* M[1][1];  
eb01 lli z = F[1][0] \* M[0][0] + F[1][1] \* M[1][0];  
a2c5 lli w = F[1][0] \* M[0][1] + F[1][1] \* M[1][1];  
   
e8c2 F[0][0] = x%MOD;  
e2e1 F[0][1] = y%MOD;  
6c49 F[1][0] = z%MOD;  
c9f3 F[1][1] = w%MOD;  
9e32 }  
   
7643 void power\_fibo(lli F[2][2], lli n) {  
98ee if (n == 0 || n == 1) return;  
bb91 lli M[2][2] = { { 1,1 },{ 1,0 } };  
   
4cb1 power\_fibo(F, n / 2);  
b245 multiply\_fibo(F, F);  
b5cd if (n % 2 != 0)  
427c multiply\_fibo(F, M);  
e26e }  
   
1168 lli fib(lli n) {  
f225 lli F[2][2] = { { 1,1 },{ 1,0 } };  
7f73 if (n == 0)  
c3bc return 0;  
a345 power\_fibo(F, n - 1);  
3887 return F[0][0];  
39b9 }  
   
 // the n-th s-gonal number;  
68b4 lli polygonal\_number(int n, int s) {  
74e4 return ((s - 2) \* (n) \* (n - 1)) / 2 + n;  
23da }  
   
a949 lli catalan\_number(int n) {  
f2af lli comb = pascal\_comb\_(2 \* n, n); // could use binomialCo (2\*n, n)  
a6a6 return comb / (n + 1);  
3548 }  
   
47e8 lli secStirling[MAX][MAX];  
6519 lli second\_kind\_stirling(int n, int k) { // initialize secStirling to -1 for all values  
a8c9 if (secStirling[n][k] >= 0) return secStirling[n][k];  
dafd if (n == 0 && k == 0) return secStirling[n][k] = 1;  
1311 else if (n == 0 || k == 0) return secStirling[n][k] = 0;  
c1f1 return secStirling[n][k] = k \* second\_kind\_stirling(n - 1, k) + second\_kind\_stirling(n - 1, k -c799 1);  
ce41 }  
   
613d lli firstStirling[MAX][MAX];  
8a05 lli first\_kind\_stirling(int n, int k) { // initialize firstStirling to -1 for all values  
761e if (firstStirling[n][k] >= 0) return firstStirling[n][k];  
0e4a if (n == 0 && k == 0) return firstStirling[n][k] = 1;  
559d else if (n == 0 || k == 0) return firstStirling[n][k] = 0;  
9fce return firstStirling[n][k] = (n - 1) \* first\_kind\_stirling(n - 1, k) + first\_kind\_stirling(n - 311b 1, k - 1);  
d9f5 }  
   
 // evaluate (1^m + 2^m + ... + n^m) % MOD  
e1dd lli power\_sum(int n, int m) {  
9904 lli a = exponent\_(n + 1, m + 1, MOD) - 1;  
bb28 lli b = 0;  
05db for (int i = 1; i <= n; i++) {  
a14c lli x = exponent\_(i + 1, m + 1, MOD);  
7eb1 lli y = exponent\_(i, m + 1, MOD);  
612e lli z = (m + 1) \* exponent\_(i, m, MOD) % MOD;  
86dd b += (((x - y) % MOD) - z) % MOD;  
ec4b }  
424a lli ans = modular\_inverse\_(a - b, m + 1, MOD);  
6b0f return (ans + MOD) % MOD;  
7b25 }

6907 #define EPS 1e-9  
de6c #define INF 1e100  
8a54 #define PI acos(-1)  
   
4cd1 struct PT {  
3b5c double x, y;  
571b PT() {}  
5e08 PT(double x, double y) : x(x), y(y) {}  
27b2 PT(const PT &p) : x(p.x), y(p.y) {}  
e3d5 bool operator == (PT other) const {  
a424 return (fabs(x - other.x) < EPS && (fabs(y - other.y) < EPS));  
255c }  
f07e PT operator + (const PT &p) const { return PT(x + p.x, y + p.y); }  
5c43 PT operator - (const PT &p) const { return PT(x - p.x, y - p.y); }  
13dc PT operator \* (double c) const { return PT(x\*c, y\*c); }  
e053 PT operator / (double c) const { return PT(x / c, y / c); }  
fcc0 };  
   
70ba double dot(PT p, PT q) { return p.x\*q.x + p.y\*q.y; }  
6aae double dist2(PT p, PT q) { return dot(p - q, p - q); }  
6630 double dist(PT p, PT q) { return sqrt(dist2(p, q)); }  
3ed7 double cross(PT p, PT q) { return p.x\*q.y - p.y\*q.x; }  
   
 // points to lines.  
765e struct line { double a, b, c; };

1f3c void pointsToLine(PT p1, PT p2, line &l) {  
434b if (fabs(p1.x - p2.x) < EPS) {  
8136 l.a = 1.0; l.b = 0.0; l.c = -p1.x;  
b474 }  
5e3c else {  
f2e7 l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);  
0d9c l.b = 1.0;  
f866 l.c = -(double)(l.a \* p1.x) - p1.y;  
37be }  
f1c0 }  
   
   
 // vector stuff. ---------------------------------  
4611 struct vec {  
39ec double x, y;  
62ab vec(double \_x, double \_y) : x(\_x), y(\_y) {}  
42e0 };  
   
75d1 vec toVec(PT a, PT b) { // convert 2 points to vector a->b  
368f return vec(b.x - a.x, b.y - a.y);  
79c9 }  
   
ab6b double cross(vec a, vec b) { return a.x \* b.y - a.y \* b.x; }  
1228 double dot(vec a, vec b) { return (a.x \* b.x + a.y \* b.y); }  
e8eb double norm\_sq(vec v) { return v.x \* v.x + v.y \* v.y; }  
   
de0e vec scale(vec v, double s) { // nonnegative s = [<1 .. 1 .. >1]  
3f7e return vec(v.x \* s, v.y \* s);  
f186 }  
   
e71f PT translate(PT p, vec v) { // translate p according to v  
3fd0 return PT(p.x + v.x, p.y + v.y);  
8183 }  
   
 // -------------------------------------------------------------------  
   
 // rotate a point CCW or CW around the origin  
801c PT RotateCCW90(PT p) { return PT(-p.y, p.x); }  
6248 PT RotateCW90(PT p) { return PT(p.y, -p.x); }  
5862 PT RotateCCW(PT p, double t) { // by certain angle t  
c936 return PT(p.x\*cos(t) - p.y\*sin(t), p.x\*sin(t) + p.y\*cos(t));  
b634 }  
 // returns true if point r is on the left side of line pq  
bdd5 bool is\_ccw(PT p, PT q, PT r) {  
667a return cross(toVec(p, q), toVec(p, r)) > 0;  
d34e }  
   
fe3d double angle(PT a, PT o, PT b) { // returns angle aob in rad  
e359 vec oa = toVec(o, a), ob = toVec(o, b);  
fdc7 return acos(dot(oa, ob) / sqrt(norm\_sq(oa) \* norm\_sq(ob)));  
3f93 }  
 // returns true if point r is on the same line as the line pq  
0b49 bool collinear(PT p, PT q, PT r) {  
8c31 return fabs(cross(toVec(p, q), toVec(p, r))) < EPS;  
8c1c }  
   
   
 // which side of a line formed by points a,b is point c on?  
0d45 double point\_on\_side(PT a, PT b, PT c) {  
aa17 return ((b.x - a.x)\*(c.y - a.y) - (b.y - a.y)\*(c.x - a.x));  
bd2d }  
   
 // project point c onto line through a and b  
 // assuming a != b  
da7c PT ProjectPointLine(PT a, PT b, PT c) {  
df6b return a + (b - a)\*dot(c - a, b - a) / dot(b - a, b - a);  
5e86 }  
   
 // project point c onto line segment through a and b  
570f PT ProjectPointSegment(PT a, PT b, PT c) {  
f259 double r = dot(b - a, b - a);  
5389 if (fabs(r) < EPS) return a;  
b6e9 r = dot(c - a, b - a) / r;  
8fca if (r < 0) return a;  
33a5 if (r > 1) return b;  
7f7c return a + (b - a)\*r;  
e386 }  
   
 // compute distance from c to segment between a and b  
ed09 double DistancePointSegment(PT a, PT b, PT c) {  
b173 return dist2(c, ProjectPointSegment(a, b, c));  
9df6 }  
   
 // compute distance between point (x,y,z) and plane ax+by+cz=d  
ac51 double DistancePointPlane(double x, double y, double z,  
9c44 double a, double b, double c, double d)  
2499 {  
afe6 return fabs(a\*x + b\*y + c\*z - d) / sqrt(a\*a + b\*b + c\*c);  
3502 }  
   
 // determine if lines from a to b and c to d are parallel or collinear  
6a8c bool LinesParallel(PT a, PT b, PT c, PT d) {  
75a8 return fabs(cross(b - a, c - d)) < EPS;  
43d0 }  
   
9dba bool areParallel(line l1, line l2) { // check coefficients a & b  
dab8 return (fabs(l1.a - l2.a) < EPS) && (fabs(l1.b - l2.b) < EPS);  
c6a8 }  
   
0fe7 bool LinesCollinear(PT a, PT b, PT c, PT d) {  
514c return LinesParallel(a, b, c, d)  
b16f && fabs(cross(a - b, a - c)) < EPS  
7661 && fabs(cross(c - d, c - a)) < EPS;  
0bce }  
   
9630 bool collinear(int ax, int ay, int bx, int by, int cx, int cy) {  
c04c bool result = false;  
6889 if (ay == by)  
5d83 result = (by == cy);  
7d87 else if (ax == bx)  
48c9 result = (bx == cx);  
1720 else  
143c result = ((by - ay)\*(cx - bx) == (cy - by)\*(bx - ax));  
ce1b return result;  
de0d }  
   
 //true if (x2,y2) lies between (x1,y1) and (x3,y3), otherwise false  
b7d2 bool between(int x1, int y1, int x2, int y2, int x3, int y3) {  
b9bd bool xbetween = (x1<x3) ? (x1 <= x2 && x2 <= x3) : (x3 <= x2 && x2 <= x1);  
6a83 bool ybetween = (y1<y3) ? (y1 <= y2 && y2 <= y3) : (y3 <= y2 && y2 <= y1);  
3225 return xbetween && ybetween && collinear(x1, y1, x2, y2, x3, y3);  
29ec }  
   
 // returns true (+ intersection point) if two lines are intersect  
60ea bool areIntersect(line l1, line l2, PT &p) {  
5476 if (areParallel(l1, l2)) return false;  
259d p.x = (l2.b \* l1.c - l1.b \* l2.c) / (l2.a \* l1.b - l1.a \* l2.b);  
246e if (fabs(l1.b) > EPS) p.y = -(l1.a \* p.x + l1.c);  
9d73 else p.y = -(l2.a \* p.x + l2.c);  
f353 return true;  
9fe7 }  
   
 // line segment p-q intersect with line A-B.  
f92a PT lineIntersectSeg(PT p, PT q, PT A, PT B) {  
06a8 double a = B.y - A.y;  
91b4 double b = A.x - B.x;  
cf90 double c = B.x \* A.y - A.x \* B.y;  
dc65 double u = fabs(a \* p.x + b \* p.y + c);  
e2a0 double v = fabs(a \* q.x + b \* q.y + c);  
e504 return PT((p.x \* v + q.x \* u) / (u + v), (p.y \* v + q.y \* u) / (u + v));  
2755 }  
   
 // determine if line segment from a to b intersects with  
 // line segment from c to d  
14ad bool SegmentsIntersect(PT a, PT b, PT c, PT d) {  
065e if (LinesCollinear(a, b, c, d)) {  
c6d7 if (dist2(a, c) < EPS || dist2(a, d) < EPS ||  
77c3 dist2(b, c) < EPS || dist2(b, d) < EPS) return true;  
a14a if (dot(c - a, c - b) > 0 && dot(d - a, d - b) > 0 && dot(c - b, d - b) > 0)  
9083 return false;  
1348 return true;  
40e7 }  
e3ce if (cross(d - a, b - a) \* cross(c - a, b - a) > 0) return false;  
95b8 if (cross(a - c, d - c) \* cross(b - c, d - c) > 0) return false;  
6542 return true;  
1357 }  
   
 // 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  
1272 PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {  
73b0 b = b - a; d = c - d; c = c - a;  
3cfd assert(dot(b, b) > EPS && dot(d, d) > EPS);  
703f return a + b\*cross(c, d) / cross(b, d);  
fbfc }  
   
 // compute center of circle given three points  
38a4 PT ComputeCircleCenter(PT a, PT b, PT c) {  
9f96 b = (a + b) / 2;  
2652 c = (a + c) / 2;  
92b5 return ComputeLineIntersection(b, b + RotateCW90(a - b), c, c + RotateCW90(a - c));  
ace8 }  
   
 // determine if point is on the boundary of a polygon  
9dfc bool PointOnPolygon(const vector<PT> &p, PT q) {  
3ac0 for (int i = 0; i < p.size(); i++)  
39e3 if (dist2(ProjectPointSegment(p[i], p[(i + 1) % p.size()], q), q) < EPS)  
d21a return true;  
a584 return false;  
2551 }  
   
 // returns true if point is on or inside polygon.  
9dfa bool PointInPolygon(const vector<PT> &p, PT q) {  
f354 if (PointOnPolygon(p, q)) return true;  
   
398a bool c = 0;  
ece5 for (int i = 0; i < p.size(); i++) {  
844d int j = (i + 1) % p.size();  
9526 if ((p[i].y <= q.y && q.y < p[j].y ||  
d4c7 p[j].y <= q.y && q.y < p[i].y) &&  
cbaa q.x < p[i].x + (p[j].x - p[i].x) \* (q.y - p[i].y) / (p[j].y - p[i].y))  
f561 c = !c;  
0fd6 }  
fece return c;  
778b }  
   
 // cuts polygon Q along the line formed by point a -> point b  
 // (note: the last point must be the same as the first point)  
2a17 vector<PT> cutPolygon(PT a, PT b, const vector<PT> &Q) {  
0aa2 vector<PT> P;  
c87c for (int i = 0; i < (int)Q.size(); i++) {  
7aa1 double left1 = cross(toVec(a, b), toVec(a, Q[i]));  
f2d0 double left2 = 0;  
5ba6 if (i != (int)Q.size() - 1)  
00cb left2 = cross(toVec(a, b), toVec(a, Q[i + 1]));  
2653 if (left1 > -EPS) P.push\_back(Q[i]);  
0f87 if (left1 \* left2 < -EPS)  
63e5 P.push\_back(lineIntersectSeg(Q[i], Q[i + 1], a, b));  
2b62 }  
98dd if (!P.empty() && !(P.back() == P.front()))  
b411 P.push\_back(P.front());  
a174 return P;  
a576 }  
   
 // compute intersection of line through points a and b with  
 // circle centered at c with radius r > 0  
3426 vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {  
db06 vector<PT> ret;  
8ea4 b = b - a;  
c9ab a = a - c;  
ebf8 double A = dot(b, b);  
82ea double B = dot(a, b);  
df4b double C = dot(a, a) - r\*r;  
ed70 double D = B\*B - A\*C;  
00b8 if (D < -EPS) return ret;  
f38a ret.push\_back(c + a + b\*(-B + sqrt(D + EPS)) / A);  
7b59 if (D > EPS)  
685e ret.push\_back(c + a + b\*(-B - sqrt(D)) / A);  
1dc7 return ret;  
3893 }  
   
 // compute intersection of circle centered at a with radius r  
 // with circle centered at b with radius R  
07fd vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {  
b7c9 vector<PT> ret;  
f2a9 double d = sqrt(dist2(a, b));  
63b5 if (d > r + R || d + min(r, R) < max(r, R)) return ret;  
de46 double x = (d\*d - R\*R + r\*r) / (2 \* d);  
a961 double y = sqrt(r\*r - x\*x);  
241e PT v = (b - a) / d;  
5e78 ret.push\_back(a + v\*x + RotateCCW90(v)\*y);  
3633 if (y > 0)  
174c ret.push\_back(a + v\*x - RotateCCW90(v)\*y);  
8203 return ret;  
1c6d }  
   
 // This code computes the area or a (possibly nonconvex)  
 // polygon, assuming that the coordinates are listed in a clockwise or  
 // counterclockwise fashion.  
881c double ComputeSignedArea(const vector<PT> &p) {  
f475 double area = 0;  
137e for (int i = 0; i < p.size(); i++) {  
3bb4 int j = (i + 1) % p.size();  
da33 area += p[i].x\*p[j].y - p[j].x\*p[i].y;  
9eac }  
984a return area / 2.0;  
54bf }  
   
1eaa double ComputeArea(const vector<PT> &p) {  
98f6 return fabs(ComputeSignedArea(p));  
b4ba }  
   
20d5 PT ComputeCentroid(const vector<PT> &p) {  
874e PT c(0, 0);  
1fcb double scale = 6.0 \* ComputeSignedArea(p);  
6ea9 for (int i = 0; i < p.size(); i++) {  
4c69 int j = (i + 1) % p.size();  
9467 c = c + (p[i] + p[j])\*(p[i].x\*p[j].y - p[j].x\*p[i].y);  
3cde }  
e1db return c / scale;  
df73 }  
   
 // tests whether or not a given polygon (in CW or CCW order) is simple  
d819 bool IsSimple(const vector<PT> &p) {  
cb26 for (int i = 0; i < p.size(); i++) {  
2360 for (int k = i + 1; k < p.size(); k++) {  
d8b5 int j = (i + 1) % p.size();  
870f int l = (k + 1) % p.size();  
ff99 if (i == l || j == k) continue;  
9733 if (SegmentsIntersect(p[i], p[j], p[k], p[l]))  
f710 return false;  
87c5 }  
2c43 }  
9231 return true;  
8cec }  
   
 // returns true if we always make the same turn while examining  
 // all the edges of the polygon one by one  
2200 bool isConvex(const vector<PT> &P) {  
de2d int sz = (int)P.size();  
e17c if (sz <= 3) return false;  
7d36 bool isLeft = is\_ccw(P[0], P[1], P[2]);  
a221 for (int i = 1; i < sz - 1; i++)  
4a19 if (is\_ccw(P[i], P[i + 1], P[(i + 2) == sz ? 1 : i + 2]) != isLeft)  
25bd return false;  
6e22 return true;  
130c }  
   
   
 // reflect p3 across line formed by p1, p2  
2a1e PT calc\_refl(PT p1, PT p2, PT p3) {  
c586 double delx = p2.x - p1.x;  
03f5 double dely = p2.y - p1.y;  
6ffc double u = (-delx\*(p3.y - p1.y) + dely\*(p3.x - p1.x)) / (delx\*delx + dely\*dely);  
548e PT ans;  
38bb ans.x = p3.x - dely \* 2 \* u;  
2ee8 ans.y = p3.y + delx \* 2 \* u;  
b68c return ans;  
65c9 }  
   
43a7 double distToLines(PT a, PT b, PT c, PT d) {  
4947 if (SegmentsIntersect(a, b, c, d))  
f78a return 0;  
9a6a double ans = min(DistancePointSegment(c, d, a), DistancePointSegment(c, d, b));  
20dc ans = min(ans, DistancePointSegment(a, b, c));  
f567 ans = min(ans, DistancePointSegment(a, b, d));  
406f return ans;  
7a7e }  
   
 // trianlge stuff ------------------  
   
9bf5 bool canFormTriangle(double a, double b, double c) {  
8133 return (a + b > c) && (a + c > b) && (b + c > a);  
9c74 }  
   
db7b double area(double x1, double y1, double x2, double y2, double x3, double y3) {  
433a return 0.5 \* (x1 \* y2 + x2 \* y3 + x3 \* y1 - x1 \* y3 - x2 \* y1 - x3 \* y2);  
d264 }  
   
d3c6 double area(double ab, double bc, double ca) { // Heron's fomula  
ce8d double s = 0.5 \* (ab + bc + ca);  
c196 return sqrt(s) \* sqrt(s - ab) \* sqrt(s - bc) \* sqrt(s - ca);  
b671 }  
   
7987 double area(PT a, PT b, PT c) {  
8eb9 return area(dist(a, b), dist(b, c), dist(c, a));  
cc08 }  
   
f4e2 double perimeter(double ab, double ac, double bc) {  
9ac9 return ab + bc + ac;  
4cab }  
   
 // circle stuff. ----------------------------  
   
 // Descartes Circle Theorem  
 // given the radii of three circles, determine the radius of the fourth circle  
 // that is tangent to all three of them.  
0e65 double descartes\_circle(double r1, double r2, double r3) {  
ce80 double s1 = 1.0 / r1 + 1.0 / r2 + 1.0 / r3;  
5e06 double s2 = 1.0 / (r1\*r2) + 1.0 / (r1\*r3) + 1.0 / (r2\*r3);  
c665 return s1 + sqrt(s2); // substract here if you want internally tangent circle  
2e4e }  
   
 // given points of intersection between two circles, get the centers of the circles.  
772e bool circle2PtsRad(PT p1, PT p2, double r, PT &c, bool flip) {  
dd66 if (flip) swap(p1, p2);  
3bff double d2 = (p1.x - p2.x) \* (p1.x - p2.x) +  
ff80 (p1.y - p2.y) \* (p1.y - p2.y);  
a559 double det = r \* r / d2 - 0.25;  
071b if (det < 0.0) return false;  
c4d5 double h = sqrt(det);  
aae3 c.x = (p1.x + p2.x) \* 0.5 + (p1.y - p2.y) \* h;  
2154 c.y = (p1.y + p2.y) \* 0.5 + (p2.x - p1.x) \* h;  
bc2b return true; // to get the other center, reverse p1 and p2  
5d9c }  
   
 // The Great-Circle Distance between any two points A and B on sphere  
 // is the shortest distance along a path on the surface of the sphere  
f3fb double greater\_circle\_distance(double pLat, double pLong, double qLat, double qLong, double radius)dfe4 {  
d5cd pLat \*= PI / 180; pLong \*= PI / 180; // conversion from degree to radian  
47d9 qLat \*= PI / 180; qLong \*= PI / 180;  
fe7d return radius \* acos(cos(pLat)\*cos(pLong)\*cos(qLat)\*cos(qLong) +  
9237 cos(pLat)\*sin(pLong)\*cos(qLat)\*sin(qLong) +  
73c7 sin(pLat)\*sin(qLat));  
3be3 }  
   
6560 double circle\_area(double R) {  
51f7 return PI \* R\*R;  
baf2 }  
   
5c8c double circle\_segment\_area(double R, double d) {  
8774 if (d == 0) return circle\_area(R) / 2;  
   
de6f double hh = (R\*R - d\*d);  
9213 double theta = 2 \* acos((R\*R + d\*d - hh) / (2 \* R\*d));  
c368 return R\*R \* (theta - sin(theta)) / 2;  
4666 }  
   
 // return volume of cap sphere with radius R, cap height h  
4329 double spherical\_cap\_volume(double R, double h) {  
1ec9 return PI \* h \* h \* (3.0 \* R - h) / 3.0;  
488b }  
   
5a5e double spherical\_cap\_area(double R, double h) {  
d00f return 2 \* PI \* R \* h;  
7efd }  
   
   
b916 double rInCircle(double ab, double bc, double ca) {  
b58c return area(ab, bc, ca) / (0.5 \* perimeter(ab, bc, ca));  
65d1 }  
   
fad1 double rInCircle(PT a, PT b, PT c) {  
bce1 return rInCircle(dist(a, b), dist(b, c), dist(c, a));  
0d9a }  
   
 // returns 1 if there is an inCircle center, returns 0 otherwise  
 // if this function returns 1, ctr will be the inCircle center  
 // and r is the same as rInCircle  
e763 int inCircle(PT p1, PT p2, PT p3, PT &ctr, double &r) {  
b1ef r = rInCircle(p1, p2, p3);  
13fd if (fabs(r) < EPS) return 0; // no inCircle center  
   
0698 line l1, l2; // compute these two angle bisectors  
70a3 double ratio = dist(p1, p2) / dist(p1, p3);  
947e PT p = translate(p2, scale(toVec(p2, p3), ratio / (1 + ratio)));  
e3a6 pointsToLine(p1, p, l1);  
   
cfa2 ratio = dist(p2, p1) / dist(p2, p3);  
7200 p = translate(p1, scale(toVec(p1, p3), ratio / (1 + ratio)));  
1a9d pointsToLine(p2, p, l2);  
   
24ff areIntersect(l1, l2, ctr); // get their intersection point  
47e7 return 1;  
3a42 }  
   
566e double rCircumCircle(double ab, double bc, double ca) {  
b419 return ab \* bc \* ca / (4.0 \* area(ab, bc, ca));  
cddd }  
   
3daa double rCircumCircle(PT a, PT b, PT c) {  
1a55 return rCircumCircle(dist(a, b), dist(b, c), dist(c, a));  
a8af }  
   
 // returns 1 if there is a circumCenter center, returns 0 otherwise  
 // if this function returns 1, ctr will be the circumCircle center  
 // and r is the same as rCircumCircle  
28f7 int circumCircle(PT p1, PT p2, PT p3, PT &ctr, double &r) {  
8520 double a = p2.x - p1.x, b = p2.y - p1.y;  
8697 double c = p3.x - p1.x, d = p3.y - p1.y;  
882f double e = a \* (p1.x + p2.x) + b \* (p1.y + p2.y);  
9c86 double f = c \* (p1.x + p3.x) + d \* (p1.y + p3.y);  
8703 double g = 2.0 \* (a \* (p3.y - p2.y) - b \* (p3.x - p2.x));  
c120 if (fabs(g) < EPS) return 0;  
   
7eb3 ctr.x = (d\*e - b\*f) / g;  
2169 ctr.y = (a\*f - c\*e) / g;  
2859 r = dist(p1, ctr);  
e1eb return 1;  
5f72 }  
   
 // returns true if point d is inside the circumCircle defined by a,b,c  
16ca int inCircumCircle(PT a, PT b, PT c, PT d) {  
b11e return (a.x - d.x) \* (b.y - d.y) \* ((c.x - d.x) \* (c.x - d.x) + (c.y - d.y) \* (c.y - d.y)) +  
24de (a.y - d.y) \* ((b.x - d.x) \* (b.x - d.x) + (b.y - d.y) \* (b.y - d.y)) \* (c.x - d.x) +  
b0c7 ((a.x - d.x) \* (a.x - d.x) + (a.y - d.y) \* (a.y - d.y)) \* (b.x - d.x) \* (c.y - d.y) -  
d6c2 ((a.x - d.x) \* (a.x - d.x) + (a.y - d.y) \* (a.y - d.y)) \* (b.y - d.y) \* (c.x - d.x) -  
54a9 (a.y - d.y) \* (b.x - d.x) \* ((c.x - d.x) \* (c.x - d.x) + (c.y - d.y) \* (c.y - d.y)) -  
820b (a.x - d.x) \* ((b.x - d.x) \* (b.x - d.x) + (b.y - d.y) \* (b.y - d.y)) \* (c.y - d.y) > 0 ? 1 8aad : 0;  
6c28 }  
   
 // Convex Hull. ------------------------------------------  
   
1248 PT pivot;  
eff3 bool angleCmp(PT a, PT b) { // angle sorting function  
0166 if (collinear(pivot, a, b))  
a0af return dist(pivot, a) < dist(pivot, b);  
b0ab double d1x = a.x - pivot.x, d1y = a.y - pivot.y;  
f0bb double d2x = b.x - pivot.x, d2y = b.y - pivot.y;  
02cb return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0;  
586b }  
   
3338 vector<PT> convex\_hull(vector<PT> P) {  
4e96 int i, j, n = (int)P.size();  
6edd if (n <= 3) {  
db0a if (!(P[0] == P[n - 1])) P.push\_back(P[0]);  
ba1b return P;  
ddad }  
   
94aa int P0 = 0;  
7b91 for (i = 1; i < n; i++)  
0794 if (P[i].y < P[P0].y || (P[i].y == P[P0].y && P[i].x > P[P0].x))  
8d1e P0 = i;  
   
bc9c PT temp = P[0]; P[0] = P[P0]; P[P0] = temp;  
57fa pivot = P[0];  
4267 sort(++P.begin(), P.end(), angleCmp);  
   
b3d0 vector<PT> S;  
383f S.push\_back(P[n - 1]);  
32f8 S.push\_back(P[0]);  
8338 S.push\_back(P[1]);  
179b i = 2;  
ca6f while (i < n) {  
9d91 j = (int)S.size() - 1;  
ba43 if (is\_ccw(S[j - 1], S[j], P[i]))  
6b12 S.push\_back(P[i++]);  
a51d else  
5b33 S.pop\_back();  
9aa4 }  
0c5a return S;  
d01f }

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Algebra \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

binomial coefficient O(n) line 171

catalan numbers line 278

chinese remainder theorem special case line 74

chinese remainder theorem general line 86

combinatorics (n choose k) line 140

compare large products line 183

diophantine (ax + bx = c) line 97

exponentiation by squaring. line 6

extended euclid line 38

euler totient directly determined line 208

euler totient sieve line 219

factorial line 133

fibonacci line 243-271

gcd line 27

lcm line 33

modular inverse (a/b) % p line 17

mod inverse - find b s.t. (ab = 1 mod(m)) line 64

modulus line 22

modular linear equation solver ax = b mod(n) line 51

pascal's triangle (n^2) line 148, 158

primes line 122

primitive pythagorean triples line 194

polygonal numbers line 274

power sum (1^m + 2^m + ... + n^m) % MOD line 302

stirling number first kind line 284

stirling number second kind line 293

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Geometry \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

area of polygon (signed) line 632

area of polygon (unsigned) line 641

angle formed by lines line 429

angle comparator for sorting line 849

circle area line 760

circle segment area line 764

convex hull line 857

circle - get center given 3 pts on circle line 550

circle with line intersection line 599

circle with circle intersection line 616

circle-circle point of intersection -> center line 737

circumcircle radius line 810, 814

circumcircle center line 821

centroid line 645

Descartes circle theorem line 730

great circle distance line 751

incircle center line 793

incircle, radius line 782, 786

line, pair of points into a line line 380

lines parallel (point form) line 473

lines parallel (line form) line 477

lines collinear (point form) line 481

lines intersect (line form) line 506

lines intersect (point form) line 543

PT line 360-377

project point to line line 446

project point to segment line 351

point on a segment line 499

point in circumcircle

point - segment distance line 461

point - plane distance line 466

points collinear line 434

points collinear (no struct) line 487

point on Polygon line 557

point in Polygon line 565

polygon cut line 581

polygon is simple line 656

polygon is convex line 671

reflect point across line line 693

rotate angle line 419-425

spherical cap Volume line 773

spherical cap surface area line 777

segments intersect line 515, 526

triangle, can form given side lengths line 704

triangle area line 708,712,717

triangle, perimeter line 721

vector line 395-414