1. 错排问题

求给定排列中错排出现的概率。

给定一个排列，若１－ｎ没有出现在原来的位置，则说明这是一个错排

这里算的贼麻烦，用的容斥原理

错排的递推公式为D(n) = (n - 1) \* (D(n - 1) + D(n - 2))

最终的公式是：（欧拉大神做出来的）ｎ的错排数量为[(n!/e)+0.5] 这里[]表示向下取整

ll zuhe(ll a, ll b){

if(a > b - a)

a = b - a;

ll res = 1;

for(ll i = 1; i <= a; i ++){

res = res \* (b - i + 1 )/i;

}

return res;

}

ll jie(ll n){

ll res = 1;

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

res \*= i;

}

return res;

}

1. 卡特兰数

对于包含ｎ个Ｘ和ｎ个Ｙ的字符串，满足任一前缀中Ｘ的数量大于Ｙ的数量的个数

卡特兰数公式：Cn = (2n)! / ((n+1)! \* n!)

递推公式：Cn = (2 \* (2n - 1) / (n + 1)) \* Cn-1 C0 = 1

ll mul(ll a, ll b){

ll res = 1;

while(b > 0){

if(b&1) res \*= a;

res %= mod;

a \*= a;

a %= mod;

b >>= 1;

}

res %= mod;

return res;

}

//直接使用公式计算

ll solve(ll n){

ll res = 1;

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

res \*= (n+i);

res %= mod;

res \*= mul(i, mod - 2);

res %= mod;

}

res \*= mul(n+1, mod - 2);

res %= mod;

return res;

}

ll ans[10110];

1. 高斯消元

TYPE: 高斯消元 异或消元

DETAIL: 给n个数，保证每个数的质因子最大不会超过2000，问有多少中取法能够使得取出的数的

乘机是完全平方数

TATICS: 将每一个数质因子分解，形成一个0,1矩阵，0表示当前质因子出现偶数次，否则出现奇

数次，然后进行高斯消元，注意消元的时候使用异或消元。

求出秩r，结果为 2^r - 1

#define CL(a) memset(a, 0, sizeof(a))

const int inf = 1e9+7;

const double eps = 1e-6;

const int maxn = 305, maxe = 2001;

int prime[303], isNotPrime[maxe], cnt;

void init(){

CL(prime);

CL(isNotPrime);

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

if(!isNotPrime[i]){

prime[cnt++] = i;

}

for(int j = 0; j < cnt && prime[j] \* i < maxe; j ++){

isNotPrime[i \* prime[j]] = 1;

if(i % prime[j] == 0)

break;

}

}

}

struct Mat{

int mat[maxn][maxn];

int n, m;

Mat(){

n = maxn, m = maxn;

}

Mat(int \_n):n(\_n), m(maxn){}

void init(){

CL(mat);

}

//求秩

int Rank(){

int i, j, k, col, max\_r, res = 0;

for(k = 0, col = 0; k < n && col < m; k ++, col++){

max\_r = k;

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

if(mat[i][col] > mat[max\_r][col]){

max\_r = i;

}

if(mat[max\_r][col] == 0){

k --;

continue;

}

if(k != max\_r){

for(j = col; j < m; j ++){

swap(mat[k][j], mat[max\_r][j]);

}

}

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

if(mat[i][col]){

for(int j = col; j < m; j ++){

mat[i][j] ^= mat[k][j];

}

}

}

res ++;

}

return res;

}

void print(){

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

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

printf("%d%c", mat[i][j], j == m - 1 ? '\n' : ' ');

}

}

}

};

ll mul(ll a, ll b){

ll res = 1;

while(b > 0){

if(b & 1) res \*= a;

res %= mod;

a \*= a;

a %= mod;

b >>= 1;

}

return res;

}

int main()

{

init();

for(i = 0; i < cnt; i ++){

if(prime[i] > 2000)

break;

printf("%d %d", prime[i-1], i); int T, n, tt = 1;

ll tmp;

scanf("%d", &T);

while(T --){

scanf("%d", &n);

Mat m(n);

m.init();

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

scanf("%I64d", &tmp);

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

if(tmp % prime[j] == 0){

int ct = 0;

while(tmp % prime[j] == 0){

tmp /= prime[j];

ct ++;

}

m.mat[i][j] = ct & 1;

}

}

}

int r = n - m.Rank();

ll ans = mul(2, r) - 1;

ans = (ans + mod) % mod;

printf("Case #%d:\n%I64d\n",tt ++, ans);

}

return 0;

}

1. 欧拉函数 & 素数表

int prime[N], isNotPrime[N], cnt, phi[N]; //prime存放素数，phi为欧拉函数

void init() {

memset(prime, 0, sizeof(prime));

memset(isNotPrime, 0, sizeof(isNotPrime));

memset(phi, 0, sizeof(phi));

cnt = 0;

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

if (!isNotPrime[i]) {

prime[cnt++] = i;

phi[i] = i - 1;

}

for (int j = 0; j < cnt && prime[j] \* i < N; j++) {

isNotPrime[prime[j] \* i] = 1;

if (i % prime[j] != 0) {

phi[i\*prime[j]] = phi[i] \* (prime[j] - 1); //根据欧拉定理可得，当i是j的倍数时，phi[i\*j] = phi[i]\*(j-1); 其中i是素数

}

else{

phi[i\*prime[j]] = phi[i] \* prime[j]; //当不是倍数时，phi[i\*j] = phi[i]\*(j); 其中i是素数

break; // 保证每个数只被筛了一次

}

}

}

}

1. 模线性方程

ll exgcd(ll a, ll b, ll& x, ll& y) { //扩展欧几里得算法，求解ax + by = d = gcd(a, b)的解

if (b == 0) {

x = 1;

y = 0;

return a;

}

ll d = exgcd(b, a%b, x, y), tmp;

tmp = x;

x = y;

y = tmp - a / b\*y;

return d;

}

//求解模线性方程，ax = b (mod n)

//该方程当且仅当 b|d 时有d个解，每个解之间相差n/d

void modular\_linear(ll a, ll b, ll n) {

if (b == 0) {

puts("0");

return;

}

ll d, x, y;

d = exgcd(a, n, x, y);

if (b % d != 0) {

printf("FOREVER\n");

}

else {

x = (x\*b / d) % n;

x = (x % (n / d) + n / d) % (n / d); //求得最小的正数解 其余解为 (x+ i \* n / d) % n (i = 1,2,3...)

printf("%I64d\n", x);

}

}

1. 矩阵快速幂

const int N = 2;

const int mod = 10000;

struct mat {

int m[N][N];

void init() {

memset(m, 0, sizeof(m));

}

};

mat mul(mat a, mat b) {

mat c;

c.init();

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

for (int j = 0; j < N; j++)

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

c.m[i][j] += a.m[i][k] \* b.m[k][j];

c.m[i][j] %= mod;

}

return c;

}

mat multi(mat a,int n) {

mat c;

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

for (int j = 0; j < N; j++)

c.m[i][j] = (i == j);

if (n == 0) {

return c;

}

while (n > 0) { //与乘法快速幂类似

if (n & 1) c = mul(c, a);

a = mul(a,a);

n >>= 1;

}

return c;

}

计算几何基础模板

#define CL(a) memset(a, 0, sizeof(a))

const int inf = 1e9+7;

const int mod = 1e9+7;

const int maxn = 1e6+7;

const double eps = 1e-9;

struct Point{

double x, y;

Point(double \_x = 0, double \_y = 0):x(\_x), y(\_y){}

};

//向量与点等价，表示从原点到这个点的向量

typedef Point Vector;

Vector operator + (Vector A, Vector B){return Vector(A.x+B.x, A.y+B.y);}

Vector operator - (Vector A, Vector B){return Vector(A.x-B.x, A.y-B.y);}

Vector operator \* (Vector A, double p){return Vector(A.x\*p, A.y\*p);}

Vector operator / (Vector A, double p){return Vector(A.x/p, A.y/p);}

bool operator < (const Vector& A, const Vector& B){

return A.x < B.x || (A.x == B.x && A.y < B.y);

}

//判断浮点数的正负

int dcmp(double x){

if(fabs(x) < eps) return 0;

else return x < 0 ? -1 : 1;

}

bool operator == (const Vector& A, const Vector& B){

return dcmp(A.x-B.x) == 0 && dcmp(A.y-B.y) == 0;

}

/\* 求极角

向量(x, y) 的级角为 atan2(y, x);

\*/

//点积

double Dot(Vector A, Vector B){return A.x\*B.x + A.y\*B.y;}

//利用点积求长度

double Length(Vector A){return sqrt(Dot(A, A));}

//求两个向量的角度

double Angle(Vector A, Vector B){

return acos(Dot(A, B)/ Length(A)/Length(B));

}

//叉积

double Cross(Vector A, Vector B){return A.x\*B.y - A.y\*B.x;}

double Area(Point A, Point B, Point C){return Cross((B-A), C-A)/2;}

//将向量旋转一定的角度

Vector Rotate(Vector A, double rad){

return Vector(A.x\*cos(rad)-A.y\*sin(rad) ,A.x\*sin(rad) + A.y\*cos(rad));

}

//求一个向量的法线，即旋转90°再单位化

Vector Normal(Vector A){

double L = Length(A);

return Vector(-A.y/L, A.x/L);

}

//直线可以表示成一个起点P和方向向量v l: P+tv t为参数

//若已知直线上的两个点，则参数方程为A+(B-A)t

//对于上面的参数方程，线段 0 < t < 1 射线 t > 0

//求直线交点

//调用前确保两条之前有且只有一个交点，当且仅当Cross(v, w) != 0;

Point GetLineIntersection(Point P, Vector v, Point Q, Vector w){

Vector u = P - Q;

double t = Cross(w, u) / Cross(v, w);

return P + v \* t;

}

//点到直线距离，利用叉积

double DistanceToLine(Point P, Point A, Point B){

Vector v1 = B - A, v2 = P - A;

return fabs(Cross(v1, v2)) / Length(v1);

}

//点到线段距离，需要考虑点到线段的垂线是否在线段上

double DistanceToSegment(Point P, Point A, Point B){

if(A == B) return Length(P-A);

Vector v1 = B - A, v2 = P - A, v3 = P - B;

if(dcmp(Dot(v1, v2)) < 0) return Length(v2);

else if (dcmp(Dot(v1, v3)) > 0) return Length(v3);

else return fabs(Cross(v1, v2)) / Length(v1);

}

//求点在直线上的投影

Point GetLineProjection(Point P, Point A, Point B){

Vector v = B - A;

return A + v\*(Dot(v, P-A) / Dot(v, v));

}

//判断点是否在线段上（可以在端点）

bool onSegment(Point p, Point a1, Point a2){

return dcmp(Cross(a1-p, a2-p) == 0) && dcmp(Dot(a1-p, a2-p)) <= 0;

}

//判断两条线段是否相交，不包含端点相交的情况

bool IsSegmaIntersection(Point a1, Point a2, Point b1, Point b2){

double c1 = Cross(a2-a1, b1-a1);

double c2 = Cross(a2-a1, b2-a1);

double c3 = Cross(b2-b1, a1-b1);

double c4 = Cross(b2-b1, a2-b1);

return dcmp(c1)\*dcmp(c2) < 0 && dcmp(c3) \* dcmp(c4) < 0;

}

//多边形求面积，适用于凸多边形和凹多边形

double ConvexPolygonArea(Point \*p, int n){

double res = 0;

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

res += Cross(p[i] - p[0], p[i+1] - p[0]);

return res / 2;

}

//多边形求面积，适用于凸多边形和凹多边形

double ConvexPolygonArea(Point \*p, int n){

double res = 0;

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

res += Cross(p[i] - p[0], p[i+1] - p[0]);

return res / 2;

}

1. 求多边形面积

//任选一个点，按照逆时针或顺时针方向，与每两个点依次做叉积，所得结果除以2就是多边形的面积

1. 判断两直线是否相交

//判断两个直线是否相交，只需要没个线段两个端点分别位于另一条线段的两侧。

//对于AB CD 两条线段，若AB向量与AC向量叉乘大于零，则说明C点位于AB直线的逆时针方向，若等于零说明C点在AB之线上，若小于零说明C点在AB直线的顺时针方向，通过这个特点来判断两点是否位于一条直线的同一侧。

bool intersect(point x1, point y1, point x2, point y2){

double d1, d2, d3, d4;

//d1,d2表示x1，y1是否位于x2->y2这条向量的两侧

d1 = (y2-x2)\*(x1-x2);

d2 = (y2-x2)\*(y1-x2);

//d3,d4同理

d3 = (y1-x1)\*(x2-x1);

d4 = (y1-x1)\*(y2-x1);

if(d1\*d2<0 && d3\*d4<0)

return true;

return false;

}

1. 求两个圆相交面积

const int N = 30;

const double eps = 1e-8;

const double Pi = acos(-1.0);

struct point{

double x, y;

point(){}

point(double \_x, double \_y):x(\_x), y(\_y){}

point operator + (point t){

return point(x+t.x, y+t.y);

}

point operator - (point t){

return point(x-t.x, y-t.y);

}

point operator \* (double a){

return point(x\*a, y\*a);

}

point operator / (double a){

return point(x/a, y/a);

}

double operator \* (point t){ // 向量叉积，返回向量|a|\*|b|\*sin(θ)方向垂直u,v 且遵循右手定则

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

}

double operator ^ (point t){ // 向量点积，返回值 |a|\*|b|\*cos(θ)

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

}

double len(){

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

}

};

struct circle{

//圆心

point o;

//半径

double r;

circle(){}

circle(point \_o, double \_r):o(\_o), r(\_r){}

double area(){

return Pi\*r\*r;

}

//求两个圆相交面积

double interarea(circle t){

double d = (t.o-o).len();

//相离的情况

if(d > r + t.r + eps )

return 0;

//内含的情况

if(t.r+d < r - eps)

return t.area();

if(r+d < t.r - eps)

return area();

double xita = acos((r\*r+d\*d-t.r\*t.r)/(2\*r\*d));

double arf = acos((t.r\*t.r+d\*d-r\*r)/(2\*t.r\*d));

//a圆对应扇形的面积

double S1 = xita\*r\*r;

//b圆对应扇形的面积

double S2 = arf\*t.r\*t.r;

//两圆圆心和交点构成的四边行的面积

double S3 = d\*r\*sin(xita);

//相交面积

return S1+S2-S3;

}

}cr[N];

1. 凸包

//凸包求法，按照x轴排序，则第一个点一定是凸包的顶点。枚举后面每一个点，若当前栈中只有一个点，则直接入栈，否则判断是否发生右旋，若发生右旋，将栈顶的元素弹出，直到发生左旋，讲当前点入栈，从前向后扫一边，从后向前扫一边就能求出凸包

void Convex(point \*p, int& n){

int i, j, r, top, m;

sort(p, p+n, cmp);

//先入队前两个点

s[0] = p[0];

s[1] = p[1];

top = 1;

for(i = 2; i < n; i ++){

//这里判断是否发生左旋的时候>=0，若等于0说明当前点是凸包边上的一个顶点，可以不用考虑，如果要求凸包所有的顶点的话则改成>0

while(top>0 && (p[i]-s[top-1])\*(s[top]-s[top-1]) >= 0)top--;

s[++top] = p[i];

}

m = top;

//最后一个点当前一定位于栈顶

s[++top] = p[n-2];

for(i = n-3; i >= 0; i --){

while(top>m && (p[i]-s[top-1]) \* (s[top]-s[top-1]) >= 0)top--;

s[++top] = p[i];

}

//当前的栈顶是第一个点，top就是凸包的顶点数

n = top;

}

int main(){

int n;

while(~scanf("%d", &n) && n){

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

scanf("%lf%lf", &no[i].x, &no[i].y);

}

Convex(no, n);

double ans = 0;

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

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

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

ans = max( fabs((s[i]-s[j]) \* (s[i]-s[k])) / 2, ans);

}

}

}

printf("%.2f\n", ans);

}

return 0;

}