## Euler Totient Function

ll Euler\_Totient\_Function(ll n){

ll ans = n;

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

if(n %i==0) ans -= ans/i;

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

}

if(n>1) ans -=ans/n;

return ans;

}

## Geometria Computacional

const double EPS = 1e-8;

const double oo = 1e12;

const double PI = 3.141592653589793;

#define X real()

#define Y imag()

typedef complex<double> P;

typedef vector<P> Pol;

struct circle{

P p; double r;

circle(){}

circle(P x,double rr){

p=x, r = rr;

}

};

struct L: public vector <P>{ //Linea

L (P a, P b){

push\_back(a); push\_back(b);

}};

inline bool operator<(const P a, const P b){

return a.X!=b.X ?a.X<b.X :a.Y <b.Y;

}

double cross(P a, P b){//1

return imag(conj(a) \* b);

}

double dot(P a, P b){//2

return (conj(a)\*b).X;

}

//Orientacion de 3 puntos

int ccw(P a, P b, P c){ //3,1 2

b-=a; c-=a;

if(cross(b,c)>0) return +1;

if(cross(b,c)<0) return -1;

if(dot(b,c)<0) return +2;//c-a-b line

if(norm(b)<norm(c)) return -2;//a-b-c line

return 0;

}

//Interseccion de 2 rectas

bool intersectLL (L l, L m){//4,1

//non-parallel

return abs(cross(l[1]-l[0], m[1]-m[0])) > EPS

|| abs(cross(l[1]-l[0], m[0]-l[0])) < EPS;

} //same-line

//Punto interseccion recta recta

P crosspoint(L l, L m){ //5,1

double A = cross( l[1]-l[0], m[1]-m[0]);

double B = cross( l[1]-l[0], l[1]-m[0]);

if(abs(A)<EPS && abs(B)<EPS)

return m[0]; //Same line

if(abs(A)<EPS) return P(0,0);//parallels

return m[0] + B / A \* (m [1] - m [0]);

}

//Interseccion recta y segmento

bool intersectLS (L l, L s){//6, 1

//s[0] is left of l

return cross(l[1]-l[0], s[0]-l[0]) \*

cross(l[1]-l[0],s[1]-l[0])<EPS;

} //s[1] is right of l

//Interseccion recta y punto

bool intersectLP (L l, P p){//7,1

return abs(cross(l[1]-p, l[0]-p))<EPS;

}

//Interseccion de 2 segmento

bool intersectSS (L s, L t){//8,3

FOR(i,2)FOR(j,2) if(abs(s[i]-t[j])<EPS)

return 1; // same point

return ccw(s[0],s[1],t[0])\*ccw(s[0],s[1],t[1])<=0

&& ccw(t[0],t[1],s[0])\*ccw(t[0],t[1],s[1])<=0;

}

//Interseccion segmento y punto

bool intersectSP (L s,P p){//9

double a=abs(s[0]-p)+abs(s[1]-p);

return a-abs(s[1]-s[0])<EPS;

}

//Interseccion circulo circulo

pair<P, P> intersectCC(circle a,circle b) {

P x= b.p - a.p;

P A= conj(x), C = a.r\*a.r\*(x);

P B= (b.r\*b.r-a.r\*a.r-(x)\*conj(x));

P D= B\*B-4.0\*A\*C;

P z1= (-B+sqrt(D)) / (2.0\*A) +a.p;

P z2= (-B-sqrt(D)) / (2.0\*A) +a.p;

return pair<P, P>(z1, z2);

}

//Proyeccion punto recta

P projection(L l,P p){//10,2

double t=dot(p-l[0], l[0]-l[1])/norm(l[0]-l[1]);

return l[0] + t\*(l[0]-l[1]);

}

//Refleccion punto recta

P reflection(L l, P p){//11, 10

return p +(P(2,0) \*(projection(l,p)-p));

}

//Distancia recta punto

double distanceLP(L l,P p){//12, 10

return abs(p - projection(l,p));

}

//Distancia recta recta

double distanceLL(L a, L b){//13,4 12

if(intersectLL(a,b)) return 0;

return distanceLP(a,b[0]);

}

//Distancia recta segmento

double distanceLS(L l, L s){//14,7 12

if(intersectLS(l,s)) return 0;

return min(distanceLP(l,s[0]),distanceLP(l,s[1]));

}

//Distancia segmento punto

double distanceSP(L s, P p){//15, 10 9

const P r = projection(s,p);

if (intersectSP(s,r)) return abs(r-p);

return min( abs(s[0]-p), abs(s[1]-p) );

}

//distancia segmento segmento

double distanceSS (L s, L t) {//16,8 15

if (intersectSS(s, t)) return 0;

double a=oo,b=oo;

FOR(i,2) a=min(a, distanceSP(s,t[i]));

FOR(i,2) b=min(b, distanceSP(t,s[i]));

return min(a,b);

}

//Centro de circunferencia dado 3 puntos

P circunferenceCenter(P a, P b, P c){//17

P x =1.0/conj(b-a), y=1.0/conj(c-a);

return (y-x)/(conj(x)\*y-x\*conj(y)) +a;

}

//Angulo con el eje x

double anguloEjeX(P a){//18,1 2

P b = P(1,0);

if(dot(b,a)/(abs(a)\*abs(b))==1) return 0;

if(dot(b,a)/(abs(a)\*abs(b))==-1) return PI;

double aux=asin(cross(b,a)/(abs(a)\*abs(b)));

if(a.X<0 && a.Y>0) aux+=PI/2;

if(a.X<0 && a.Y<0) aux-=PI/2;

if(aux<0) aux += 2\*PI;

return aux;

}

//Angulo entre tres vectores

double anguloEntreVectores(P a, P b){//19,18

double aa = anguloEjeX(a);

double bb = anguloEjeX(b);

double r = bb - aa;

if (r<0) r+=2\*PI;

return r;

}

//Angulo entre tres puntos

double anguloEntre3Puntos(P a, P b, P c){//20,19

a-=b; c-=b;

return anguloEntreVectores(a,b);

}

Pol convexHull(Pol ps){//21,3

int t,i,n = ps.size(), k=0;

if (n < 3) return ps;

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

Pol ch (2\*n);

for(i=0;i<n;ch[k++]=ps[i++]) //lower

while(k>=2 && ccw(ch[k-2],ch[k-1],ps[i])<=0) --k;

for(i=n-2,t=k+1 ;i>=0; ch[k++]=ps[i--])// upper

while(k>=t && ccw(ch[k-2],ch[k-1], ps[i])<=0) --k;

ch.resize(k-1);

return ch;

}

int pointInPolygon(Pol pol, P p){//22, 1 2

bool in = false; int n=pol.size();

FOR(i,n){

P a= pol[i] - p, b= pol[(i+1)%n]-p;

if(a.Y > b.Y) swap(a,b);

if(a.Y<=0 && 0 < b.Y)

if (cross(a,b)<0) in = !in;

if(abs(cross(a,b))<=EPS &&dot(a,b)<=0)

return true; // ON

}

return in; // IN | OUT

}

pair <P,P> closestPair (Pol p) {//23

int i,n = p.size(), s=0, t=1, m=2;

vector<int> S(n); S[0]=0, S[1]=1;

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

double d = norm(p[s]-p[t]);

for(i =2;i<n; S[m++]=i++)

FOR(j,m){

if(norm(p[S[j]]-p[i])<d)

d=norm(p[s=S[j]]-p[t = i]);

if(p[S[j]].X < p[i].X-d)

S[j--] = S[--m];

}

return make\_pair( p[s], p[t] );

}

//max distance pair points, O(n)

double diameter(Pol pt) {//24, 1

int is=0,js=0, n=pt.size();

FAB(i,1,n){

if(pt[i].Y >pt[is].Y) is=i;

if(pt[i].Y <pt[js].Y) js=i;

}

double maxd=norm(pt[is]-pt[js]);

int i,maxi,j,maxj;

i = maxi = is; j = maxj = js;

do {

if(cross(pt[(i+1)%n]-pt[i],

pt[(j+1)%n]-pt[j])>=0)

j=(j+1)%n; else i=(i+1)%n;

if (norm(pt[i]-pt[j])>maxd){

maxd =norm(pt[i]-pt[j]);

maxi=i; maxj=j;

} }while(i!=is || j!=js);

return maxd;

}

double area(Pol pol) {//25, 1

double A=0; int n=pol.size();

FOR(i,n)

A+=cross(pol[i],pol[(i+1)%n]);

return A/2;

}

P rotate(P p1, double a){

double x=p1.real()\*cos(a)­p1.imag()\*sin(a);

double y=p1.real()\*sin(a)+p1.imag()\*cos(a);

return P(x,y);

}

typedef vector <P> Tr;

Tr make\_tr(P a,P b,P c){

Tr r(3);

r[0]=a; r[1]=b; r[2]=c;

return r;

}

bool tr\_contains(Tr t,P p){

return ccw(t[0],t[1],p)>=0 &&

ccw(t[1],t[2],p)>=0 &&

ccw(t[2],t[0],p)>=0;

}

bool ear\_Q(int i,int j,int k,Pol pol){

Tr t = make\_tr(pol[i], pol[j], pol[k]);

if (ccw(t[0],t[1],t[2])<=0) return false;

for (int m=0; m<pol.size(); ++m)

if (m!=i && m!=j && m!=k)

if (tr\_contains(t, pol[m]))

return false;

return true;

}

void triangulate(Pol pol, vector<Tr> &t){

int n=pol.size();

vector<int> l, r;

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

l.push\_back((i­1+n)%n);

r.push\_back((i+1+n)%n);

}

int i=n­1;

while (t.size()<n­2){

i = r[i];

if (ear\_Q(l[i],i,r[i],pol)){

t.push\_back(make\_tr(pol[l[i]],pol[i],pol

[r[i]]));

l[r[i]]=l[i];

r[l[i]]=r[i];

}

}

}

pair<P,P> CCInter(P c1, double r1, P c2,

double r2){

P A=conj(c2­c1);

P B=(r2\*r2­r1\*r1­(c2­c1)\*conj(c2­c1)),

C=r1\*r1\*(c2­c1);

P D = B\*B­ 4.0\*A\*C;

P z1 = (­B+sqrt(D))/(2.0\*A)+c1;

P z2=(­B­sqrt(D))/(2.0\*A)+c1;

return pair<point, point>(z1,z2);

}

//Geometria 3D

struct P3 {

double x, y, z;

P3(double X = 0, double Y = 0, double Z =

0): x(X), y(Y), z(Z) { }

};

struct V3 {

double x, y, z;

V3(double X=0, double Y=0, double Z=0):

x(X), y(Y), z(Z) { }

V3(P3 p) { x = p.x; y = p.y; z = p.z; }

V3(P3 p, P3 q) { x = q.x ­ p.x; y = q.y ­ p.y;

z = q.z ­ p.z; }

};

P3 operator + (const P3 &p, const V3 &v){

return P3(p.x+v.x,p.y+v.y,p.z+v.z);}

P3 operator + (const P3 &p, const P3 &q){

return P3(p.x+q.x,p.y+q.y,p.z+q.z);}

P3 operator ­ (const P3 &p, const V3 &v){

return P3(p.x­v.x, p.y­v.y, p.z­v.z);}

P3 operator ­ (const P3 &p, const P3 &q){

return P3(p.x­q.x, p.y­q.y, p.z­q.z);}

V3 operator + (const V3 &u, const V3 &v){

return V3(u.x+v.x, u.y+v.y, u.z+v.z);}

V3 operator ­ (const V3 &u, const V3 &v){

return V3(u.x­v.x, u.y­v.y, u.z­v.z);}

V3 operator \* (const double &a, const V3 &v){

return V3(a\*v.x, a\*v.y, a\*v.z);}

double dot(const V3 u, const V3 v){

return u.x\*v.x+u.y\*v.y+u.z\*v.z;}

V3 cross(const V3 u, const V3 v){

return V3(u.y\*v.z­u.z\*v.y,u.z\*v.x­

u.x\*v.z,u.x\*v.y­u.y\*v.x);

}

double norma(const V3 v){

return sqrt(dot(v, v));}

struct recta{

P3 a, b;

recta(P3 A, P3 B): a(A), b(B) { }

recta(P3 P, V3 V): a(P) { b = P + V; }

};

struct semirecta{

P3 a, b;

semirecta(P3 A, P3 B): a(A), b(B) { }

semirecta(P3 P, V3 V): a(P) { b=P+V; }

};

struct segmento {

P3 a, b;

segmento(P3 A, P3 B): a(A), b(B) { }

};

struct triangulo {

P3 a, b, c;

triangulo(P3 A,P3 B,P3 C):a(A),b(B),c(C) { }

};

double distancia(const P3 a, const P3 b){

return norma(V3(a, b));}

double distancia(const P3 p, const recta r){

V3 v(r.a, r.b), w(r.a, p);

return norma(cross(v, w)) / norma(v);

}

double distancia(P3 p, semirecta s){

V3 v(s.a, s.b), w(s.a, p);

if (dot(v,w)<=0) return distancia(p, s.a);

return distancia(p, recta(s.a, s.b));

}

double distancia(P3 p, segmento s){

V3 v(s.a, s.b), w(s.a, p);

double c1 = dot(v, w), c2 = dot(v, v);

if (c1 <= 0) return distancia(p, s.a);

if (c2 <= c1) return distancia(p, s.b);

return distancia(p, s.a + (c1/c2)\*v);

}

double distancia(recta r, recta s){

V3 u(r.a, r.b), v(s.a, s.b), w(r.a, s.a);

double a=dot(u,u),b=dot(u,v),c=dot(v,v),

d=dot(u,w),e=dot(v,w);

double D = a\*c ­ b\*b, sc, tc;

if (D < EPS) {

sc = 0;

tc = (b > c) ? d/b : e/c;

} else {

sc = (b\*e ­ c\*d) / D;

tc = (a\*e ­ b\*d) / D;

}

V3 dP = w + (sc \* u) ­ (tc \* v);

return norma(dP);

}

double distancia(segmento r, segmento s){

V3 u(r.a, r.b), v(s.a, s.b), w(s.a, r.a);

double a=dot(u,u),b=dot(u,v),c=dot(v,v),

d=dot(u,w),e=dot(v,w);

double D = a\*c ­ b\*b;

double sc, sN, sD = D;

double tc, tN, tD = D;

if (D < EPS) {

sN = 0;sD = 1;tN = e;tD = c;

} else {

sN = (b\*e – c\*d);

tN = (a\*e ­ b\*d);

if (sN < 0) {

sN = 0;tN = e;tD = c;

} else if (sN > sD) {

sN = sD;tN = e + b;tD = c;

}

}

if (tN < 0) {

tN = 0;

if (­d < 0) {

sN = 0;

} else if (­d > a) {

sN = sD;

} else {

sN = ­d;

sD = a;

}

} else if (tN > tD) {

tN = tD;

if ((­d + b) < 0) {

sN = 0;

} else if (­d + b > a) {

sN = sD;

} else {

sN = ­d + b;

sD = a;

}

}

sc = fabs(sN) < EPS ? 0 : sN / sD;

tc = fabs(tN) < EPS ? 0 : tN / tD;

V3 dP = w + (sc \* u) ­ (tc \* v);

return norma(dP);

}

V3 projecao(V3 u, V3 v) {

return (dot(v, u) / dot(u, u)) \* u;

}

bool between(P3 a, P3 b, P3 p) {

return dot(V3(p ­ a), V3(p ­ b)) < EPS;

}

double linedist(P3 a, P3 b, P3 p) {

P3 proj=a+projecao(V3(a, b), V3(a, p));

if (between(a, b, proj)) {

return norma(V3(proj, p));

} else {

return min(norma(V3(a,p)),norma(V3(b,p)));

}

}

double distancia(P3 p, triangulo T) {

V3 X(T.a, T.b), Y(T.a, T.c), P(T.a, p);

V3 PP = P ­ projecao(cross(X, Y), P);

P3 PPP = T.a + PP;

V3 R1 = cross(V3(T.a, T.b), V3(T.a, PPP));

V3 R2 = cross(V3(T.b, T.c), V3(T.b, PPP));

V3 R3 = cross(V3(T.c, T.a), V3(T.c, PPP));

if (dot(R1,R2)>­EPS && dot(R2,R3)>­EPS &&

dot(R1,R3)>­EPS){

return norma(V3(PPP, p));

} else {

return min(linedist(T.a,T.b,p),

min(linedist(T.b,T.c,p),linedist(T.c,T.a,p)));

}

}

## Minimal Enclosing Circle

double distSqr(P &p1, P &p2){

return (p1.X-p2.X)\*(p1.X-p2.X) +

(p1.Y-p2.Y)\*(p1.Y-p2.Y);

}

bool contain(circle c,P p){

return distSqr(c.p,p)<= c.r\*c.r;

}

circle findCircle(P a,P b){

P p( real(a+b)/2.0 , imag(a+b)/2.0);

return circle( p, sqrt(distSqr(a,p)));

}

circle findCircle(P pa,P pb,P pc) {

double a,b,c,x,y,r,d;

c = sqrt(distSqr(pa , pb));

b = sqrt(distSqr(pa , pc));

a = sqrt(distSqr(pb , pc));

if (b==0 || c==0 || a\*a>= b\*b+c\*c)

return findCircle(pb,pc);

if (b\*b >= a\*a+c\*c)

return findCircle(pa,pc);

if (c\*c >= a\*a+b\*b)

return findCircle(pa,pb);

d = real(pb-pa)\*imag(pc-pa);

d = 2 \* (d - imag(pb-pa)\*real(pc-pa));

x = (imag(pc-pa)\*c\*c-imag(pb-pa)\*b\*b)/d;

y = (real(pb-pa)\*b\*b-real(pc-pa)\*c\*c)/d;

x += real(pa), y += imag(pa);

r= sqrt(pow(real(pa)-x,2)+ pow(imag(pa)-y,2));

return circle(P(x,y),r);

}

P points[MAXN], R[3];

circle sed(int n,int nr){

circle c;

if(nr == 3)

c = findCircle(R[0],R[1],R[2]);

else if (n == 0 && nr==2)

c = findCircle(R[0], R[1]);

else if(n==1 && nr == 0)

c = circle(points[0],0);

else if(n == 1 && nr == 1)

c = findCircle(R[0],points[0]);

else{

c = sed(n-1, nr);

if(!contain(c,points[n-1])){

R[nr++] = (points[n-1]);

c = sed(n-1, nr);

}

}

return c;

}

## Salto del Caballo

ll SaltoCaballo(ll x1,ll y1,ll x2,ll y2){

ll dx =abs(x2-x1);

ll dy =abs(y2-y1);

ll lb= max(dx+1 , dy + 1)/2;

lb = max(lb, (dx + dy + 2)/3);

while((lb % 2) != (dx+ dy)%2) lb++;

if(abs(dx)==1 && !dy) return 3;

if(abs(dy)==1 && !dx) return 3;

if(abs(dx)==2 && abs(dy)==2) return 4;

return lb;

}

## Day Of Week

int DayOfWeek(int d, int m, int y){

if(m<3) y--, m+=10; else m -=2;

int c= y/100; y %= 100;

c =y- 2 \* c+ d+ y/4 +c/4;

return((int)(2.6\*m-0.2)+c+7)%7;

}

## Catalan

C[n] => FOR(k=0,n-1) C[k] \* C[n-1-k]

C[n] => Comb(2\*n,n) / (n + 1)

C[n] => 2\*(2\*n-3)/n \* C[n-1]

## Fact Mod

int factMod (int n, int p) {

int res = 1,i;

while (n > 1) {

if ((n/p) & 1)

res = (res \* (p-1)) % p;

for (i=n%p; i > 1;i--)

res = (res \* i) % p;

n /= p;

}

return res % p;

}

## Fibonacci

-Sumatoria de F[1..n]=F[n+2]-1.

- Si n es divisible por m entonces Fn es divisible por Fm

- Los nmeros consecutivos de Fibonacci son primos entre si.

- Si N es Fibonacci => (5\*N\*N + 4 || 5\*N\*N 4) es un cuadrado

- Suma de n terminos partiendo del k-simo + k = F[k+n+1]

- gcd(F[p], F[n]) = F[gcd(p,n)] = F[1] = 1

- Cantidad num fibonacci hasta n

floor((log10(n)+ (log10(5)/2))/log10(1.6180));

// \_ \_ ^ n \_ \_

//a b | 0 1 | = |fib(n-1) fib(n) |

//c d |\_1 1\_| |\_fib(n) fib(n+1)\_|

struct matrix{

ll a, b, c, d;

matrix(ll a, ll b, ll c, ll d) :

a(a), b(b), c(c), d(d) {}

const matrix operator\*(const matrix &t){

ll A =a\*t.a+ b\*t.c;

ll B =a\*t.b+ b\*t.d;

ll C =c\*t.a+ d\*t.c;

ll D =c\*t.b+ d\*t.d;

return matrix(A,B,C,D);

}

};

matrix pow(const matrix &p, int n){

if (n == 1) return p;

matrix k = pow(p, n/2);

matrix ans = k\*k;

if (n & 1) ans = ans \* p;

return ans;

}

## Kth Permutacion

int N; // N grupos

char grupo[22];//caract del grupo

int cantgrupo[22], quitar;

//FOR(i,N) quitar \*= fac[cantgrupo[i]]

void KthPermutacion(int k,int quedan){

if (quedan == 0) return;

int total = fact[quedan - 1];

int inicio = 0, fin = 0;

FOR(i,N){

if (cantgrupo[i] == 0) continue;

fin += (cantgrupo[i] \* total) / quitar;

if (fin > k){

quitar /= cantgrupo[i]--;

cout << grupo[i];

KthPermutacion(k-inicio,quedan-1);

}

else inicio = fin;

}

}

**Digit Count**

void DigitCount(int n,ll \*sol){

ll aux=n, sum=0,p=1,d;

while(aux){

d = aux % 10, aux /= 10;

sol[d] += ((n%p)+1);

for(int i=0;i<d;i++) sol[i]+=p;

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

sol[i] += sum\*d;

sol[0] -= p;

sum = p + 10 \* sum;

p \*= 10;

}

}

**Triangle Counting - TJU**

**inline** **bool** **upper**(pnt a) {

**return** imag(a)>0 ||(imag(a)== 0&& eal(a)>0);

}

**inline** **bool** **compare\_angle**(pnt a, pnt b) {

**if** (upper(a) && !upper(b)) **return** **true**;

**if** (!upper(a) && upper(b)) **return** **false**;

**return** cross(a,b) > 0;

}

**inline** **bool** **same\_half**(pnt a, pnt b) {

ll cr = cross(b,a);

**if**(cr < 0) **return** 1;

**if**(cr == 0 && dot(b,a) > 0) **return** 1;

**return** 0;

}

**int** n;

pnt arr[100001];

**int** **main**() {

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

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

**scanf**("%lld%lld",&arr[i].real(),&arr[i].imag());

sort(arr, arr+n, compare\_angle);

ll sol = ll(n) \* (n - 1) / 2 \* (n - 2) / 3;

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

**while**((j + 1)%n != i &&

same\_half(arr[i],arr[(j+1)%n]))

j = (j + 1)%n;

ll cc = (j - i + n)%n;

sol -= cc\*(cc-1)/2;

**if**(i == j) ++j;

}

cout << sol << endl;

**return** 0;

}

**Grirar Grilla 45 grados**

r = (max(col, filas) << 1) + 10;

c = (max(col, filas) << 1) + 10;

xx = x + y + 5;

yy = x - y + filas + 5;

**Teoria de Numeros**

N=p^a\*q^b\*r^c

**CantDiv** = D = (a+1)\*(b+1)\*(c+1)

**SumaDiv** = FOR(i,k)

sum\*=(prim[i]^(cant[i]+1)-1)/(prim[i]-1)

**ProdDiv** = P = N^(D/2)=Sqrt(N^D)

**Cant de Palindromes de <= N Digitos**

a(n) = 2 \*(10^(n/2) -1) si n es par

a(n) = 11\*(10^(n-1)/2)-2 si n es impar

**Rotar Punto**

P RotarPunto(P p,**double** ang){

**double** x=p.x\***cos**(pi\*ang)-p.y\***sin**(pi\*ang);

**double** y=p.x\***sin**(pi\*ang)+p.y\***cos**(pi\*ang);

**return** P(x,y)

}

**Número Ciclomático**

M : cantidad de Aristas

N: # de vértices

P:# de componentes conexas.

NC = M – N + P cantidad de ciclos.

Número de Estabilidad Interna:

Un conjunto de vértices se dice que es

interiormente estable si dos vértices

cualesquiera del conjunto no son adyacentes.

El mayor subconjunto interiormente estable de

un grafo es conocido como número de

estabilidad interna. Lo designaremos por I.

En todo grafo se cumple la siguiente

relación:

I(G) \* NC(G) = Total de vértices de la red.

**Teoria de numeros**

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

int g = a; x = 1; y = 0;

if (b != 0){

g = extGcd(b, a%b, y, x);

y ­= (a/b)\*x;

}

return g;

}

bool mExtGcd(int a,int b,int c,int &x,int &y){

int r = extGcd(a,b,x,y);

if (c%r != 0) return false;

x\*=c/r; y\*=c/r;

return true;

}

vector<int> primes;

int MAX = 1000000;

**Inverso multiplicativo**

a\*inv == 1 (mod m)

bool invMult(long long a, long long m, long

long &inv) {

long long x, y, r;

r = extGcd(a, m, x, y);

if (r!=1) return false;

inv = x;

if (inv<0) inv += m;

return true;

}

**a\*x == b (mod n)**

bool MLE(long long a, long long b, long long

n, long long &x){

long long d, xx, y;

d = extGcd(a,n,xx,y);

if (b%d) return false;

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

return true;

}

**Teorema del resto chino x == r[i] (mod m[i])**

bool TRC (vector<long long> r, vector<long

long> m, long long &x, long long &M){

int n=r.size();

long long inv;

x=0; M=1;

for (int i=0; i<n; i++) M\*=m[i];

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

if (!invMult(M/m[i],m[i],inv)) return

false;

x+=r[i]\*(M/m[i])\*inv;

}

x = (x%M);

return true;

}

**Euler’s totient theorem**

If n is a positive integer and a is coprime to

n, then a^phi(n) == 1 (mod n)

**Teorema de Wilson**

Si p es un número primo, entonces (p–1)! == ­1

mod(p).

**Fermat's little theorem**

If p is a prime number, then for any integer a

that is coprime to p, we have a^p a (mod p) ≡

**Discrete logarithm theorem**

Si g es una raiz primitiva de Zn entonces la

ecuacion g^x == g^y (mod n) se cumple si y

solo si se cumple x == y mod(phi(n).

g es una raiz primitiva mod n si las potencias

de g modulo n van por todos los coprimos de n.

La raiz primitiva existe si n = 2, 4, p^k o

2\*p^k donde p es un primo impar.

Para comprbar que g es una raiz primitiva de n

solo tenemos q comprobar que g^d != 1 mod(n)

para todo primo p que divide a phi(n), d =

phi(n)/p.

**Cantidad de digitos de n!**

(long long)floor( (log(2\*acos(­1)\*a)/2 +

a\*(log(a)­1) ) /log(10) ) + 1);

**Probabilidad**

P (E1 E2 ) + P (E1 E2 ) = P (E1 ) + P ∪ ∩

(E2 ). Entonces si E1 y E2 son mutuamente

exclusivos, P (E1 E2 ) = P (E1 ) + P (E2 ). ∪

Probabilidad de que ocurra el evento E1 dado

que ha ocurrido el evento E2

P (E1 |E2 ) = P (E1 E2 )/P (E2 ) ∩

**Teorema de Bayes**

P (E1 |E2 ) = P (E1)\*P (E2 |E1 )/P (E2 )

**Bernoulli**

Una prueba de Bernoulli es aquella que puede

terner 2 resultados exito o fallo. Si la

probabilidad de exito de una prueba de

Bernoulli es p, la probabilidad de q ocurran k

exitos en una secuencia de n eventos

idependientes es: C(n,k)\*(p^k)\*(1­p)^(n­k).

**m^(n)**

m^(n) = m(m − 1)(m − 2) ∙ ∙ ∙ (m − n + 1).

**Stirling number of the second kind**

{m,n} = (1/n!) \* (­1)^(n­k)\*(n,k)\*(k^m) Σ

**Classical occupancy problem**

En una urna con m bolas numeradas de 1 a m.

Suponga que extraemos n bolas una por una, con

remplazamientos. La probabilidad de que hallan

sido extraidas exactamente t bolas diferentes

es:

P1(m,n,t) = {n,t}\*(m^(t))/(m^n) .

**Problema del cumpleanno**

En una urna con m bolas numeradas de 1 a m.

Suponga que extraemos n bolas una por una, con

remplazamientos. La probabilidad de que halla

una coincidencia es:

P2(m,n) = 1 – P1(m,n,n) = 1­(m^(n))/(m^n) ≈

1­exp(­(n\*n)/(2\*m)). exp(x) = e^x.

Si sacamos n1 bolas de una urna y n2 bolas de

otra con remplaso, la probabilidad de

coincidencia es:

P3(m,n1,n2) = 1­(1/m^(n1+n2))\*

Σ(m^(t1+t2)\*{n1,t1}\*{n2,t2}) 1­exp(­ ≈

(n\*n)/m).

Si sacamos n1 bolas de una urna y n2 bolas de

otra sin remplaso, la probabilidad de

coincidencia es:

P4(m,n1,n2) = 1 – (m^((n1+n2)))/

(m^(n1)+m^(n2)).

Si sacamos n1 bolas de una urna con remplazo y

n2 bolas de otra sin remplaso, la probabilidad

de coincidencia es:

P5(m,n1,n2) = 1­(1­n2/m)^n1.

**Sudoku**

//se llama inicialmente con i = 0 y j = 0 y en cells

//0 en los desconocidos y el valor en los conocidos.

//si retorna true al final la matriz queda llena

//con la solucion.

static boolean solve(int i, int j, int[][] cells) {

if (i == 9) {

i = 0;

if (++j == 9)

return true;

}

if (cells[i][j] != 0) // skip filled cells

return solve(i+1,j,cells);

for (int val = 1; val <= 9; ++val) {

if (legal(i,j,val,cells)) {

cells[i][j] = val;

if (solve(i+1,j,cells))

return true;

}

}

cells[i][j] = 0; // reset on backtrack

return false;

}

static boolean legal(int i, int j, int val, int[][] cells) {

for (int k = 0; k < 9; ++k) // row

if (val == cells[k][j])

return false;

for (int k = 0; k < 9; ++k) // col

if (val == cells[i][k])

return false;

int boxRowOffset = (i / 3)\*3;

int boxColOffset = (j / 3)\*3;

45

for (int k = 0; k < 3; ++k) // box

for (int m = 0; m < 3; ++m)

if (val == cells[boxRowOffset+k][boxColOffset+m])

return false;

return true; // no violations, so it's legal

}

**Algorithm: Bignum (MULT)**

#include <cstdio>

#include <algorithm>

#include <cstring>

using namespace std;

int i, j, ln, ln1, d, m, r, tot, d1, S[100];

char n[100], n1[100];

int conv (char a) {

return a - 48;

}

int main() {

scanf ("%s %s", &n, &n1);

ln = strlen (n) - 1;

ln1 = strlen (n1) - 1;

for (i = ln1; i >= 0; i--) {

d = conv(n1[i]);

m = ln1 - i;

for (j = ln; j >= 0; j--) {

d1 = conv (n[j]);

tot = d1 \* d;

r = tot % 10;

S[m] = (S[m] + r) % 10;

S[++m] += tot / 10;

}

}

for (i = m - 1; i >= 0; i--)

printf ("%d", S[i]);

system ("pause");

return 0;

}

**Algorithm: Bignum (RESTA)**

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <ios>

using namespace std;

#define BASE 10

#define MAXD 100 + 1

char A[MAXD], B[MAXD];

int C[MAXD];

int szA, szB, szC, na, nb;

int main ()

{

//Entrar los dos números y lo acepta como cadena

scanf( "%s%s", A, B );

//Tomo la longitud de las dos cadenas

szA = strlen( A );

szB = strlen( B );

reverse( A, A + szA ); //Invierto el orden de las cadenas

reverse( B, B + szB );

int llevar = 0;

//hago el ciclo hasta la longitud mayo de ellas

int aaa = szA;

if ( szB > szA )

aaa = szB;

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

{

na = ( i >= szA ) ? 0 : A[i] - '0';

nb = ( i >= szB ) ? 0 : B[i] - '0';

C[szC] = na - nb - llevar;

if ( C[szC] < 0 ) {

C[szC] += BASE;

llevar = 1;

} else llevar = 0;

szC++;

}

int j = aaa;

while ( !C[j] ) j--;

for ( ; j >= 0; j-- ) printf( "%d", C[j] );

cout<<endl;

system ("pause");

return 0;

}

**Algorithm: Bignum (SUM)**

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

#define BASE 10

#define MAXD 100 + 1

char A[MAXD], B[MAXD];

int C[MAXD];

int szA, szB, szC, na, nb;

int main ()

{

//Entrar los dos números y lo acepta como cadena

scanf( "%s%s", A, B );

//Tomo la longitud de las dos cadenas

szA = strlen( A );

szB = strlen( B );

int may = max (szA, szB);

reverse( A, A + szA ); //Invierto el orden de las cadenas

reverse( B, B + szB );

int llevar = 0;

//hago el ciclo hasta la longitud de la mayor de ellas may = ( szA >? szB )

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

{

na = ( i >= szA ) ? 0 : A[i] - '0';

nb = ( i >= szB ) ? 0 : B[i] - '0';

C[szC] = na + nb + llevar;

llevar = C[szC] / BASE;

//resto de dividir por 10

C[szC] %= BASE;

szC++;

}

//si me quedo al final con algo que no sea cero

if ( llevar )

C[szC++] = llevar;

for ( int i = szC - 1; i >= 0; i-- )

printf( "%d", C[i] );

cout<<endl;

system ("pause");

return 0;

}