## Hopcroft Karp

int N,M,D[MAXV],Pair[MAXV],Q[MAXV];

bool BFS(){

int u,v,i,f =0,izq=0,der=0;

FORR(i,N+M) D[i] = 0;

FOR(i,N)

if(Pair[i]==-1)

Q[der++] = i;

while (izq < der){

u = Q[izq++];

for(i=last[u];i ;i=edges[i].next){

v = edges[i].v;

if (D[v + N]) continue;

D[v + N] = D[u] + 1;

if (Pair[v + N] != -1){

D[Pair[v + N]] = D[v + N] + 1;

Q[der++] = Pair[v + N];

} else f = 1;

}

}

return f;

}

int DFS(int u){

for(int v,i = last[u]; i; i=edges[i].next){

v = edges[i].v;

if (D[v+N] != D[u]+1) continue;

D[v + N] = 0;

if (Pair[v + N]==-1 || DFS(Pair[v + N])){

Pair[u] = v;

Pair[v + N] = u;

return 1;

}

}

return 0;

}

int Hopcroft\_Karp(){

int flow=0;

FORR(i,N+M) Pair[i] = -1;

while (BFS())

FOR(i,N)

if (Pair[i]==-1 &&DFS(i))

flow++;

return flow; }

## Hungarian

const ll oo = 1e18;

ll M[16][16]; int n;

// Para minimizar - M[i][j] < 0

ll Hungarian(){

int p,q;

ll xx,yy;

vector<ll> fx(n,oo),fy(n,0);

vector<int> x(n,-1),y(n,-1);

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

vector<int> t(n,-1),s(n+1,i);

for(p=q=0;p<=q && x[i]<0;++p)

for(int k=s[p],j=0;j<n && x[i]<0;++j)

if(fx[k]+fy[j]== M[k][j] && t[j]<0){

s[++q]=y[j],t[j]=k;

if(s[q]<0)

for(p=j;p>=0;j=p)

y[j] = k = t[j],

p = x[k],x[k]=j;

}

if(x[i]<0){

yy = oo;

FOR(k,q+1) FOR(j,n)

if(t[j]<0)

yy=min(yy,(fx[s[k]]+fy[j]-

M[s[k]][j]));

FOR(j,n)

fy[j] += (t[j]<0 ? 0:yy);

FOR(k,q+1) fx[s[k]] -=yy;

} else i++;

}

xx = 0;

FOR(i,n)

xx += M[i][x[i]];

return -xx;

}

## Max Flow Min Cost

int F[MAXV],Prev[MAXE];

ll D[MAXV],Phi[MAXV];

backEdge ( cap=0,cost = -cost )

Phi[i] = 0

bool Dikjstra()

F[i] = D[s] = 0

F[s] = D[i] = oo

D[u] > D[v] + edges[i].cost + Phi[v] - Phi[u]

F[u] = min(F[v] , edges[i].cap)

Prev[u] = i

ll MAX\_FLOW\_MIN\_COST(){

ll cost = 0,flujo = 0; int i;

while(Dikjstra()){

cost += (D[t] + Phi[t]) \* F[t];

flujo += F[t];

FOR(i,n)

if(F[i])

Phi[i] += D[i];

for(i = t; i != s; i = edges[Prev[i]].src){

edges[Prev[i]].cap -= F[t];

edges[Prev[i]^1].cap+= F[t];

}

}

return cost;

}

**Kth Shortest Path**

**int** n, d[MAXN]; **bool** marcas[MAXN];

Graph g,gt;

**int** **k\_shortestPath**(**int** s, **int** t, **int** k){

**int** i,u,v,p,pv;

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

d[i]=oo, marcas[i]=0;

d[t] = 0;

priority\_queue<Arco> Q;

Q.push(Arco(t, t, 0));

**while** (!Q.empty()) {

v = Q.top().u; Q.pop();

**if** (marcas[v]) **continue**;

marcas[v] = 1;

**for**(i=gt[v].size()-1;i>=0;i--){

u =gt[v][i].u;

p = gt[v][i].p;

**if** (d[u] > d[v] + p) {

d[u] = d[v] + p;

Q.push(Arco(v, u, d[u]));

}

}

}

**int** l = 0;

Q.push(Arco(-1,s,0));

**while** (!Q.empty()) {

v = Q.top().u;

pv = Q.top().p; Q.pop();

**if** (v == t && ++l == k)

**return** pv + d[s];

**for**(i=g[v].size()-1;i>=0;i--){

u = g[v][i].u;

p = g[v][i].p;

Q.push(Arco(v,u,pv+p-d[v]+d[u]));

}

}

**return** -1;

}

## Puntos Articulacion y CompBiConexas

int ndfs[MAXN], low[MAXN], pila[MAXN], top;

bool Art[MAXN];

vector<vector<int> > bi;

void Tarjan(int v, int cnt) {

int part = cnt > 1, u,i;

pila[top++] = v;

ndfs[v] = low[v] = cnt;

for(i=last[v];i ;i=edges[i].next) {

if (!ndfs[ u = edges[i].v ]) {

Tarjan(u, cnt + 1);

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

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

Art[v] = ++part == 2;

vector<int> A;

A.push\_back(v);

pila[top]=0;

while(pila[top]!= u)

A.push\_back(st[--top]);

bi.push\_back(A);

}

} else if (ndfs[u] != ndfs[v] - 1)

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

}

}

void ArtPoint\_CompBiconex(int N) {

FOR(i,N)

Art[i] = ndfs[i] = low[i] = 0;

bi.clear();

top=0; FOR(i,N)

if (!ndfs[i])

Tarjan(i, 1);

}

## Bridges y CompBiconexas

int num[MAXN], inS[MAXN];

vector< pair<int,int> > brdg;

vector< vector<int> > tecomp;

int S[MAXN],roots[MAXN];

int n,ndfs,topS,topR;

int i,m;

void visit(int v, int u) {

inS[v] = num[v] = ++ndfs;

S[topS++] = roots[topR++] = v;

for(int i=last[v];i;i=edges[i].next){

int w = edges[i].v;

if (!num[w])

visit(w, v);

else if (u != w && inS[w])

while (num[roots[topR-1]] > num[w])

topR--;

}

if (v == roots[topR - 1]) {

brdg.push\_back(make\_pair(u,v));

tecomp.push\_back(vector<int>());

while (1) {

int w = S[--topS];

inS[w] = false;

tecomp.back().push\_back(w);

if (v == w) break;

} topR--;

}

}

void Bridges\_CompBiCnx() {

FOR(i,n)

num[i] =inS[i] = 0;

topS = topR = 0;

brdg.clear(); tecomp.clear();

ndfs = 1;

FOR(i,n) if (!num[i]) {

visit(i, n);

brdg.pop\_back();

}

}

**LCA**

// T[i] - Padre de i

int N,T[MAXN], P[MAXN][20],L[MAXN];

void Preprocesar(int \*T){

FOR(i,N)

for(int j=0; 1<<j < N;j++)

P[i][j] = -1;

FOR(i,N) P[i][0] = T[i];

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

FOR(i,N)

if(P[i][j - 1]!= -1)

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

}

int QUERY(int p, int q){

int log=1;

if(L[p] < L[q]) swap(p,q);

while(1<<log <= L[p]) log++;

log--;

FORR(i,log)

if (L[p] - (1 << i) >= L[q])

p = P[p][i];

if (p == q) return p;

FORR(i,log)

if (P[p][i] != -1 && P[p][i] != P[q][i])

p = P[p][i], q = P[q][i];

return T[p];

}

**SCC**

bool visit[MAXV], mark[MAXV];

int lowlink[MAXV],vlink[MAXV],ndfs, pila[MAXV],top;

void Tarjan(int u){

vlink[u]=lowlink[u]=ndfs;

visit[u]=mark[u]=ndfs++;

pila[top++] = u;

for(int v,i=last[u];i ;i=edges[i].next)

if(!visit[ v = edges[i].v]){

Tarjan( v );

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

} else

if (mark[ v ])

lowlink[u] = min(lowlink[u],vlink[v]);

// Componente X

while(vlink[u] == lowlink[u]){

int v = pila[--top];

mark[v] = false;

if(v == u) break;

}

}

void SCC(){

ndfs = top = 1;

FOR(i,N)

visit[i]= false;

FOR(i,N)

if (!visit[i])

Tarjan(i);

}

**GCD Extendido**

ll GCDext(ll a, ll b, ll &x, ll &y){

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

if (b != 0){

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

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

}

return g;

}

**Inverso Modular**

ll invMod(ll a, ll m, ll &inv) {

ll x, y;

if (GCDext(a, m, x, y) != 1)

return 0 ; // noSolucion

inv = (x + m) % m;

return 1;

}

**Teorema del Resto Chino**

// x = a[i] mod m[i]

// if GCD(m[i],m[j]) != 1 -> noSolucion

int RestoChino(int n,ll \*a,ll \*m,ll \*x){

ll K = 1, inverso; \*x = 0;

FOR(i,n) K \*= m[i];

FOR(i,n){

invMod(K/m[i],m[i],inverso);

\*x += a[i]\* K/m[i]\* inverso;

}

\*x %= K;

return 1; // Tiene sol

}

## 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;

}

## Hungarian Extendido

int N,M,cx[MAX],cy[MAX],w[MAX][MAX];

// T[i][j] = cant de x para y

// w[i][j] = -w[i][j] para Minimizar

int T[MAX][MAX];

int lx[MAX],ly[MAX];

int S[MAX],SX[MAX],P[MAX];

void Inicializar(){

FOR(i,N){

lx[i]= -oo;

FOR(j,M){

if(lx[i]<w[i][j])

lx[i]= w[i][j];

T[i][j]= 0;

}

}

FOR(i,M) ly[i]= 0;

}

int HungarianExt(){ // 1

Inicializar();

int delta,f,j;

bool found,vx[MAX],vy[MAX];

FOR(u,N) while(cx[u]){ // 2

FOR(i,N) vx[i]=0, P[i] = -1;

FOR(i,M){ // 3

vy[i]=0 , SX[i]=u;

S[i] = lx[u]+ ly[i] -w[u][i];

} // 3

while(vx[u] = 1){ // 4

delta = oo,found = 0;

FOR(i,M) if(!vy[i]){ // 5

delta = min(S[i],delta);

if(S[i] == 0){ // 6

vy[i] = 1;

if(cy[i]){ // 7

f = min(cx[u],cy[i]);

for(j=SX[i];P[j]!=-1;j =SX[P[j]])

f= min(f, T[j][P[j]]);

cx[u] -=f, cy[i] -=f;

j= i; while(j!=-1){ // 8

T[SX[j]][j] += f;

if(P[SX[j]] != -1)

T[SX[j]][P[SX[j]]] -=f;

j= P[SX[j]];

} // 8

found = 1;

}else // 7

FOR(j,N)

if(!vx[j] && T[j][i]){ // 9

P[j]=i,vx[j]= 1;

FOR(k,M) if (!vy[k])

if(S[k]>lx[j]+ly[k]-w[j][k]){

S[k]= lx[j]+ly[k]-w[j][k];

SX[k]= j;

}

} // 9

break;} } // 6 5

if(found) break;

if(delta){

FOR(i,N) if(vx[i]) lx[i] -=delta;

FOR(i,M) if(vy[i]) ly[i] +=delta;

else

S[i] -=delta;

}

} } // 4 2

delta = 0;

FOR(i,N) FOR(j,M)

delta -= T[i][j]\*w[i][j];

return delta;

}

## Sieve Atkin

void Sieve\_Atkin(int N) {

FAB(x,1,sqrtN+1) FAB(y,1,sqrtN+1){

int n = 4\*x\*x + y\*y;

if (n<=N && (n%12==1||n%12==5))

isprime[n] ^= 1;

n =3\*x\*x +y\*y;

if(n<=N && n%12==7) isprime[n] ^= 1;

n = 3\*x\*x - y\*y;

if (n<=N && x > y && n%12==11)

isprime[n] ^= 1;

} isprime[2] = isprime[3] = true;

for(int k,n=5; n <= sqrtN; ++n)

if(isprime[n]) while((k+=n\*n)<=N)

isprime[k] = false;

}

## Edmons

int n, NewBase,Start, match[MAXN];// match[i]!=0

int izq,der,Q[MAXN],P[MAXN], C[MAXN];

bool marcas[MAXN] , X[MAXN];

int LCA(int u, int v) {

memset(X,0,sizeof(X));

while (true) {

u = C[u], X[u] = 1;

if (u == Start) break;

u = P[match[u]];

}

while (1)

if(X[v = C[v]]) break;

else v = P[match[v]];

return v;

}

void ResetTrace(int u) {

while (C[u] != NewBase) {

int v = match[u];

X[C[u]] = X[C[v]] = 1;

u = P[v];

if(C[u] != NewBase) P[u] = v;

}

}

void BlossomContract(int u, int v) {

NewBase = LCA(u, v);

memset(X, 0, sizeof(X));

ResetTrace(u), ResetTrace(v);

if (C[u] != NewBase) P[u] = v;

if (C[v] != NewBase) P[v] = u;

for(u = 1; u <= n; u++)

if (X[C[u]]) {

if (!marcas[u]) Q[der++]=u;

marcas[u] = C[u] = NewBase;

}

}

int FindPath() {

izq=0,der=0;

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

C[u]=u, marcas[u] =P[u]=0;

marcas[Start] = 1;

Q[der++]=Start;

while (izq<der) {

int u = Q[izq++];

for(int i=last[u];i ;i=edges[i].next){

int v =edges[i].v;

if(C[u] !=C[v]&& match[u]!=v)

if(v==Start ||(match[v] &&P[match[v]]))

BlossomContract(u, v);

else if (!P[v]) {

P[v] = u;

if (!match[v]) return v;

marcas[ match[v] ] = 1;

Q[der++]=march[v];

}

}

}

return 0;

}

void Aumentar(int u) {

int v, w; while (u) {

v = P[u], w = match[v];

match[v] = u,match[u] = v;

u = w;

}

} void Edmonds() {

memset(match, 0, sizeof(match));

for(Start= 1;Start<= n;Start++)

if (!match[Start])

Aumentar( FindPath() );

}

**Newton Raphson**

double NewtonRaphson(double n){

double x = 1, nx;

while (true) {

nx = (x + n / x) / 2;

if (fabs(x-nx) < EPS) break;

x = nx;

}

return x;

}

int NewtonRaphson(int n){

int a = 1; bool low = 0;

while(1) {

int nx=(a+n/a)/2;

if (a==nx || (nx>a &&low))

break;

low =nx<a, a=nx;

}

return a;

}

**Index Permutation**

ll fact[21];

**int** alpha[30]; // Ya Normalizado

ll **IndexPermutation**(**int** \*per,**int** n,**int** dif){

//n=len, dif -elems diferentes

**memset**(alpha,0,**sizeof**(alpha));

FOR(i,n) alpha[per[i]]++;

ll sol = 0, par;

FOR(i,n-1){

FAB(j,1,per[i]){

**if**(!alpha[j]) **continue**;

par = fact[n-i-1];

**for**(**int** k=1;k <= dif;k++)

par /=fact[alpha[k]-(k==j)];

sol += par;

}

--alpha[per[i]];

}

**return** sol;

}

**Complex**

**inline** P **operator\***(**const** P b)

{**return** P(x\*b.x-y\*b.y , x\*b.y + y\*b.x );}

**inline** P **operator/**(**const** P b){

P s = P( x\*b.x + y\*b.y , y\*b.x-x\*b.y);

s.x /= ( b.x\*b.x + b.y\*b.y);

s.y /= ( b.x\*b.x + b.y\*b.y);

**return** s;

}

**int** **Combinatoria** (**int** n, **int** k) {

**double** res = 1;

**for** (**int** i = 1; i <= k; + + i)

res = res \* (n - k + i) / i;

**return** (**int**) (res + 0,01);

}

**Tiling Dominoes**

**double** res = 1;

**for**(**double** i = 1;i<=n;i++)

**for**(**double** j = 1;j<=k;j++){

**double** x = 4\*cos(PI\*i/(n+1))\*cos(PI\*i/(n+1));

x += 4\*cos(PI\*j/(k+1))\*cos(PI\*j/(k+1));

res \*= **pow**(x,0.25);

}

(ll)(res+0.000001);

**Digitos de N!, n > 3**

0.5\*log10(2\*n\*PI)+n\*log10(n/M\_E)+1;

## Miller Rabin

ll multMOD(ll x,ll y,ll mod){ //(x\*y)%mod

ll rx = 0; x %= mod, y %= mod;

for (int bx = 0; y >> bx; bx++){

if(bx) x = x + x;

if(x >= mod) x -= mod;

if((y >> bx) & 1) rx += x;

if (rx >= mod) rx -= mod;

}

return rx;

}

ll powMOD(ll a,ll b,ll mod){//(x^y)%mod

ll rx = 1;

for(int bx = 0;b; b>>=1,bx++){

if(bx) a = multMOD(a,a,mod);

if (b & 1)

rx = multMOD(rx,a,mod);

}

return rx;

}

ll f(ll x,ll mod){

ll rx= multMOD(x,x,mod) + 123;

while(rx >= mod) rx -= mod;

if(!rx) rx = 2;

return rx;

}

bool Miller\_Rabin(ll n ,ll iter){

ll m = n-1,b=2,z; int j,a=0;

while(!(m&1)) m>>=1, ++a;

while(iter--){

j =0; z = powMOD(b,m,n);

while(!(( !j && z==1)|| z==n-1))

if((j > 0 && z==1)|| ++j==a)

return 0;

else z = powMOD(z,2,n);

b = f(b,n);

}

return 1;

}

bool is\_prime(ll n){

if (n == 2) return true;

return n>1 && (n & 1) && Miller\_Rabin(n, 1);

}

## Pollard\_Rho

ll factores[70]; int nfactor;

ll pollard\_rho(ll c, ll num){

ll x = rand() % num;

ll i=1, k=2, y=x,comDiv;

do { i++;

if((x =multMOD(x, x, num)-c)<0)

x += num;

if(x == y) break;

comDiv =GCD((y-x +num) %num,num);

if(comDiv > 1 && comDiv < num )

return comDiv;

if(i ==k){

y = x; k <<= 1;

}

}while ( true );

return num;

}

void fFindFactor(ll num){

if ( is\_prime(num) ){

factores[nfactor++] = num;

return;

}

ll factor = num + 1;

while(factor >= num)

factor= pollard\_rho(rand()%(num-1)+ 1,num);

fFindFactor(factor);

fFindFactor(num / factor);

}

**Stable Matching**

vector<**int**> **stable\_matching** () {

vector<vector<**int**> > aux(N,vector<**int**>(N+1,N));

vector<**int**> matchW(N,N),proposedM(N);

FOR(i,N)FOR(j,N)

aux[i][orderW[i][j]]=j;

FOR(i,N) FAB(j,i,N){

**int** w=orderM[j][proposedM[j]++ ];

**if**(aux[w][j] <aux[w][matchW[w]])

swap(j,matchW[w]);

}

return matchW;

}

## 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;

}

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;

}

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;

}

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;

}

## KDtree

struct KDtree {

struct Node {

P p; Node \*l, \*r; Node(P pp){

p=pp,l=r =NULL;

}} \*root;

KDtree(){ root = NULL; }

#define cmp(d,p,q) (d ? p.X<q.X :p.Y<q.Y)

void insert(P p)

{ root=insert(root,0,p); }

void search(P ld,P ru,vector<P> &out)

{ search(root, 0 , ld, ru, out); }

Node \*insert(Node \*t,int d,P p) {

if (t == NULL)

return new Node(p);

if(t->p == p) return t; // Rep

if (cmp(d,p,t->p))

t->l = insert(t->l, !d, p);

else t->r = insert(t->r, !d, p);

return t;

}

void search(Node \*t,int d,P ld,P ru,

vector<P> &out){

if (t== NULL) return;

P p = t->p;

if(ld.X <= p.X && p.X <= ru.X)

if(ld.Y <= p.Y && p.Y <= ru.Y)

out.push\_back(p);

if(!cmp(d,p,ld))

search(t->l, !d, ld, ru, out);

if(!cmp(d,ru,p))

search(t->r, !d, ld, ru, out);

} };

## 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;

}

## Range Minimum Query

int DP[ MAXN ][20];

void RMQ(){

int i,j,k;

FOR(i,N) DP[i][0] = i;

for(j=1;(1<<j)<=N;j++)

for(i=0;i+(1<<j)-1<N;i++){

k=DP[i+(1 << (j-1))][j-1];

if (A[DP[i][j-1]]< A[k])

k = DP[i][j-1];

DP[i][j]= k;

}

}

int QUERY(int a,int b){

int k,m; if(a==b) return A[a-1];

for(k=0;(1<<k)<(b-a+1);k++);

k--; a--; b--;

m = DP[b-(1<<k)+1][k];

return min(A[DP[a][k]],A[m]);

}

## 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 |\_0 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;

}

}

## TREAP

**srand( time( 0 ) );**

#define size(r) buff[r].ch[2]

#define hijo(r,i) buff[r].ch[i]

#define PR(r) buff[r].ch[4]

#define key(r) buff[r].ch[3]

struct Treap {

struct Nodo {

int ch[5];

Nodo() {}

Nodo( int key ){

ch[0]=ch[1]=0, ch[4]=rand();

ch[2]=1, ch[3]=key;

}

} buff[MAXNODES];

int root, nodes;

void update\_size( int root ) {

size(root) = 1 +

size(hijo(root,0))+ size(hijo(root,1));

}

void rotate(int &root,bool dir) {

int tmp = hijo(root,dir);

hijo(root,dir)= hijo(tmp,1 - dir);

hijo(tmp,1-dir)= root;

update\_size(root);

update\_size(tmp);

root = tmp;

}

void insert(int &root,int val){

if ( root == 0 ) {

buff[root= ++nodes]=Nodo(val);

return;

}

if (val == key(root)) return;

bool dir = !( val < key(root) );

insert( hijo(root,dir), val );

if(PR(root) >PR(hijo(root,dir)))

rotate( root, dir );

update\_size( root );

}

void erase(int &root,int val){

if (root==0 ) return;

if ( val != key(root)) {

bool dir= !(val<key(root));

erase( hijo(root,dir), val );

} else {

int L = hijo(root,0);

int R = hijo(root,1);

if (L) if(R)

rotate(root, PR(L)>PR(R));

else rotate(root,0);

else if(R) rotate(root,1);

else { root = 0; return ; }

erase( root, val );

}

update\_size( root );

}

int countLessThan(int root,int val){

int cant = 0;

while(root) {

bool dir= !(val<key(root));

if( dir ) {

cant+=size(hijo(root,0));

if(val<=key(root))

return cant;

cant++;

}

root = hijo(root,dir);

}

return cant;

}

int findKth( int root, int kth ) {

while(root) {

int v=hijo(root,0);

if(kth< size(v)) root=v;

else {

kth -=size(v)+ 1;

if(kth <0) return key(root);

root=hijo(root,1);

}

}

return -1;

}

};

## KMP

int pi[MAXN]; // prefix function

void PreKMP(char \*P,int n){

int q,k=0; pi[1] = 0;

for(q =2;q <=n; pi[q++] =k){

while(k && (P[k]!=P[q-1]))

k=pi[k];

if(P[k]==P[q-1]) k++;

}

}

void KMP(char \*T,int n,char \*P,int m){

int i,q=0; PreKMP(P,m);

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

while((q>0) && (P[q]!=T[i-1]))

q = pi[q];

if(P[q]==T[i-1]) q++;

if(q==m) q = pi[q];//found

}

## }

## Manacher

int rad[2\*MAX];

void Manacher(char \*s,int n){

int i=0,j=0,k;

while(i < 2 \* n - 1 ) {

while(i >= j && i+j+1< 2\*n &&

s[(i-j)/ 2]==s[(i+j+1)/2])

j++;

rad[i] = j, k = 1;

while(k <=rad[i] && rad[i-k]!=rad[i]-k){

rad[i+k ]=min(rad[i-k],rad[i]-k);

k++;

}

j = max(j-k,0),i +=k;

}

}

**ZAlgorithm**

int Z[MAX]; // Z[i]=SA[i]%SA[0]

void zAlgorithm(char \*S,int n){

int g=0,f=0; Z[0] = n;

FAB(i,1,n)

if(i<g && Z[i-f]!=(g-i))

Z[i]=min(Z[i-f],g-i);

else{

g = max(g, f = i);

while(g<n && S[g]==S[g-f]) g++;

Z[i] = g - f;

}

}

**Suffix Array**

int wa[MAXN], wb[MAXN], we[MAXN], wv[MAXN];

int SA[MAXN];

int cmp(int \*r,int a,int b,int l){

return r[a]==r[b] && r[a+l]==r[b+l];

}

void SuffixArray(char \*cad, int N) {

N++; int j, p, \*x=wa, \*y=wb,range=256;

memset(we, 0, range\*sizeof(int));

FOR(i,N) we[x[i]=cad[i]]++;

FAB(i,1,range) we[i]+=we[i-1];

FORR(i,N-1) SA[--we[x[i]]]=i;

for(j=p=1; p<N; j<<=1, range = p) {

p=0; FAB(i,N-j,N) y[p++]=i;

FOR(i,N) if (SA[i]>=j)

y[p++]=SA[i]-j;

FOR(i,N) wv[i]=x[y[i]];

memset(we, 0, range\*sizeof(int));

FOR(i,N) we[wv[i]]++;

FAB(i,1,range) we[i] +=we[i-1];

FORR(i,N-1) SA[--we[wv[i]]]= y[i];

swap(x, y);

x[SA[0]]=0, p = 1;

FAB(i,1,N)

if(cmp(y,SA[i],SA[i-1],j))

x[SA[i]]= p-1;

else x[SA[i]]=p++;

} N--;

}

int rank[MAXN], lcp[MAXN];

void findLCP(char \*cad,int N) {

int j, k=0;

FAB(i,1,N+1) rank[SA[i]] = i;

FOR(i,N){

if(k) k--; j=SA[rank[i]-1];

while(cad[i+k]==cad[j+k]) k++;

lcp[rank[i]]=k;

}}

## SDAWG

const int alfa = 27;

struct SDAWG{

struct state {

int length, edges[alfa],suf;

bool solid[alfa];

state() { length = suf = 0;

memset(edges,0,sizeof(edges));

memset(solid,0,sizeof(solid));

}

};

vector<state> aut;

void setedge(int a, int b, int ch, int solid) {

aut[a].edges[ch] = b;

aut[a].solid[ch] = solid;

if (aut[b].length <= aut[a].length)

aut[b].length = aut[a].length + 1;

}

SDAWG(char\* s) {

aut.push\_back(state());

aut.push\_back(state());

int i=-1,current = 1, sink = 1,newsink;

int newnode, v, w, a;

aut[1].suf = aut[0].suf = 0;

while(s[++i]) {

a = Convertir(s[i]);

newsink = ++current;

aut.push\_back(state());

setedge(sink, newsink, a, 1);

w = aut[sink].suf;

while (w && aut[w].edges[a] == 0) {

setedge(w, newsink, a, 0);

w = aut[w].suf;

}

v = aut[w].edges[a];

if (w == 0)

aut[newsink].suf = 1;

else if (aut[w].solid[a])

aut[newsink].suf = v;

else {

newnode = ++current;

aut.push\_back(state());

FOR(j,alfa){

aut[newnode].edges[j] =aut[v].edges[j];

setedge(w, newnode, a, 1);

w = aut[w].suf;

aut[newnode].suf = aut[v].suf;

aut[newsink].suf = aut[v].suf = newnode;

while(w && !aut[w].solid[a]){

setedge(w,newnode,a,aut[w].solid[a]);

w =aut[w].suf;

}

}

sink = newsink;

}

}

}

int FDM(char \*s) {

int best= 0,len= 0,i=-1,w= 1;

while(s[++i]) {

int a = Convertir(s[i]);

if (aut[w].edges[a])

len++, w= aut[w].edges[a];

else {

while (true) {

w= aut[w].suf;

if(w== 0||aut[w].edges[a])

break;

}

if (w== 0) len= 0,w= 1;

else {

len= aut[w].length + 1;

w= aut[w].edges[a];

}

}

if(len >best) best=len;

}

return best;

}

};

## Aho Corasick

#define F(x,y) T[x].next[y]

const int alfa = 27;

struct Aho\_Corasick{

struct PMA {

int suf,next[alfa],accept;

PMA() {accept=-1;

memset(next,0,sizeof(next));suf = 0;}

};

int root,size,father[MAXN],Q[MAXN];

vector<PMA> T;

Aho\_Corasick(){

T.push\_back(PMA());

T.push\_back(PMA());

root = size = 1;

}

void Add(char \*p,int id){

int t = root,i = -1;

while( p[++i]){

int c = Convertir(p[i]);

if (F(t,c) == 0 )

F(t,c) = ++size, T.push\_back(PMA());

t = F(t,c);

}

if(T[t].accept != -1 )

father[id] = T[t].accept;

else

T[t].accept = father[id] = id;

}

void buildPMA() {

T.push\_back(PMA());

int izq= 0,der= 0,c= 0;

while(++c<alfa)

if (F(root,c)) {

F( F(root,c) ,0) = root;

Q[der++] = F(root,c);

} else

F(root,c) = root;

while (izq < der){

int t = Q[izq++];

for ( c = 1 ; c < alfa ; ++c)

if (F(t,c)) {

Q[der++] = F(t,c);

int r = F(t,0);

while (!F(r,c)) r =F(r,0);

F(F(t,c),0)= F(r,c);

if (T[F(F(t,c),0)].accept != -1)

T[F(t,c)].suf =F(F(t,c),0);

else

T[F(t,c)].suf =T[F(F(t,c),0)].suf;

}

}

}

void match(char \*S,int \*cant) {

int v = root,i = -1;

while (S[++i]){

int c = Convertir(S[i]);

while (!F(v,c)) v = F(v,0);

v = F(v,c);

if(T[v].accept != - 1)

cant[T[v].accept]++;

for (int u= T[v].suf; u; u= T[u].suf)

if(T[u].accept != - 1)

cant[T[u].accept]++;

}

}

};

**Joseph**

**int** **joseph** (**int** n, **int** k) {

**int** res = 0;

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

res = (res + k) % i;

**return** res + 1;

}

## Expresiones

import javax.script.\*;

ScriptEngineManager manager = new ScriptEngineManager();

ScriptEngine motor = manager.getEngineByName("js");

motor.put("VARIABLE", valor);

motor.eval(Expresion);

## Expresiones Regulares

import java.util.regex.\*;

Pattern pattern = Pattern.compile(expresion);

Matcher matcher = pattern.matcher(patron);

if (matcher.matches())

[abc] -> a, b, or c (simple class)

[^abc] -> Any character except a, b, or c (negation)

[a-zA-Z] -> a hasta z or A hasta Z, inclusive (range)

[a-d[m-p]] -> a hasta d, or m hasta p: [a-dm-p] (union)

[a-z&&[def]] -> d, e, or f (intersection)

[a-z&&[^bc]] -> a through z, except for b and c: [ad-z] (subtraction)

[a-z&&[^m-p]] -> a through z, and not m through p: [a-lq-z](subtraction)

. -> Any character

\d -> A digit: [0-9]

\D -> A non-digit: [^0-9]

\s -> A whitespace character: [ \t\n\x0B\f\r]

\S -> A non-whitespace character: [^\s]

\w -> A word character: [a-zA-Z\_0-9]

\W -> A non-word character: [^\w]

\p{Punct} -> One of !"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~

\p{Lower} -> A lower-case alphabetic character: [a-z]

\p{Upper} -> An upper-case alphabetic character:[A-Z]

\p{Alpha} -> An alphabetic character

\p{Digit} -> A decimal digit: [0-9]

\p{Alnum} -> An alphanumeric character:[\p{Alpha}\p{Digit}]

\p{XDigit} -> A hexadecimal digit: [0-9a-fA-F]

\p{Space} -> A whitespace character: [ \t\n\x0B\f\r]

X? -> X, once or not at all

X\* -> X, zero or more times

X+ -> X, one or more times

X{n} -> X, exactly n times

X{n,} -> X, at least n times

X{n,m}-> X, at least n but not more than m times

X|Y -> Either X or Y

**Paint**

Path2D.Double p = new Path2D.Double();

//Crear path

p.moveTo(x1, 0);

p.lineTo(x1, y);

p.lineTo(x2, y);

p.lineTo(x2, 0);

p.lineTo(x1, 0);

p.closePath();

//Compilar

Area area = new Area(p);

//Sacar

PathIterator iter = area.getPathIterator(null);

while (!iter.isDone()) {

double[] buf = new double[6];

switch (iter.currentSegment(buf)) {

case PathIterator.SEG\_MOVETO:

case PathIterator.SEG\_LINETO:

points.add(new Point2D.Double(buf[0],buf[1]));

break;

case PathIterator.SEG\_CLOSE:

totArea += computePolygonArea(points);

points.clear();

break;

}

iter.next();

}

**Area de Rectangulos**

struct Event{

int x,y1,y2,v;

Event(){}

Event(int vv,int ww,int mm,int nn)

{x=vv;y1=ww;y2=mm;v=nn;}

bool operator<(const Event& a) const{

return x < a.x;}

} E[MAXN \* 2];

int n,c,d,v,V[1000000],A[1000000];

void update(int index,int a,int b){

if(a > d || b < c) return;

if(a >= c && b <= d) {

V[index] += v; A[index] = 0;

if(V[index] > 0) A[index] =b-a+1;

else if(a != b)

A[index]=A[2\*index]+A[2\*index+1];

return;

}

update(2\*index,a,(a+b)/2);

update(2\*index+1,(a+b)/2+1,b);

A[index]=0;

if(V[index]>0) A[index]=b-a+1;

else if(a != b)

A[index]=A[2\*index]+A[2\*index+1];

}

int x1,x2,y1,y2,ptr,sol;

scanf("%d",&n);

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

scanf("%d%d%d%d",&x1,&y1,&x2,&y2);++y1;++y2;

E[ptr++] = Event(x1,y1,y2, 1);

E[ptr++] = Event(x2,y1,y2,-1);

}

sort(E,E+(2\*n));

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

if(i!=0)

sol += A[1] \* (E[i].x - E[i-1].x);

c = E[i].y1 + 1;

d = E[i].y2;

v = E[i].v;

update(1,1,30001);

}

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

**void** **output\_tandem**(string s, **int** shift, **bool** left,

**int** cntr, **int** l, **int** l1, **int** l2) { **int** pos;

**if** (left) pos = cntr-l1;

**else** pos = cntr-l1-l2-l1+1;

cout<<"["<<shift+pos<<".."<<shift+pos+2\*l-1;

cout << "] = " << s.substr (pos, 2\*l) << endl;

}

**void** **output\_tandems**(string s,**int** shift,**bool** left,

**int** cntr,**int** l,**int** k1,**int** k2){

**for** (**int** l1=1; l1<=l; ++l1) {

**if** (left && l1 == l) **break**;

**if** (l1 <= k1 && l-l1 <= k2)

output\_tandem (s,shift,left, cntr,l,l1,l-l1);

}

}

**int** **get\_z**(vector<**int**> & z, **int** i) {

**return** 0<=i && i<(**int**)z.size() ? z[i] :0;}

**void** **find\_tandems** (string s, **int** shift =0) {

**int** n=(**int**) s.length();

**if** (n == 1) **return**;

**int** nu = n/2, nv = n-nu;

string u = s.substr (0, nu),

v = s.substr (nu);

string ru = string (u.rbegin(), u.rend()),

rv = string (v.rbegin(), v.rend());

find\_tandems (u, shift);

find\_tandems (v, shift + nu);

vector<**int**> z1 = z\_function (ru),

z2 =z\_function(v + '#' + u),

z3 =z\_function(ru + '#' + rv),

z4 =z\_function(v);

**for** (**int** cntr=0; cntr<n; ++cntr) {

**int** l, k1, k2;

**if** (cntr < nu) { l = nu - cntr;

k1 = get\_z (z1, nu-cntr);

k2 = get\_z (z2, nv+1+cntr);

}**else** { l = cntr - nu + 1;

k1 = get\_z (z3, nu+1 + nv-1-(cntr-nu));

k2 = get\_z (z4, (cntr-nu)+1);

}

**if** (k1 + k2 >= l) output\_tandems(s,shift,cntr<nu,cntr,l,k1,k2);

} }

**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;

}

}

**int** **LIS**(**int** n,**int** \*a){

**int** i,l, r, c, p[MAXN], b[MAXN], m = 1;

**for** (b[0]=0,i=1; i < n; i++){

**if** (a[b[m-1]] < a[i]){

p[i] = b[m-1];

b[m++] = i;

**continue**;

}

l = 0, r = m - 1;

**while** (l < r){

c = (l + r) / 2;

**if** (a[b[c]] < a[i])

l =c+1;

**else** r =c;

}

**if** (a[i] < a[b[l]]) {

p[i] = (l > 0)? b[l-1] : -1;

b[l] = i;

} **else**

p[i] = -1;

}

**return** m;

}

**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;

**FFT**

**void** **fft** (P \*a, **int** n,**bool** invert) {

**for** (**int** i=1, j=0; i<n; ++i) {

**int** bit = n >> 1;

**for** (; j>=bit; bit>>=1)

j -= bit;

j += bit;

**if** (i < j) swap (a[i], a[j]);

}

**for** (**int** len=2; len <= n; len <<= 1) {

**double** ang = 2\*PI/len \* (invert ? -1 : 1);

P wlen (**cos**(ang), **sin**(ang));

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

P w (1,0);

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

P u = a[i+j], v = a[i+j+len/2] \* w;

a[i+j] = u + v;

a[i+j+len/2] = u - v;

w = w \* wlen;

}

}

}

**if** (invert)

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

a[i].x /= n, a[i].y /= n;

}

**int** n1[MAXN], n2[MAXN], sol[MAXN], ln1,ln2,lsol,lfa,lfb;

P fa[MAXN], fb[MAXN];

**void** **multiply** () {

**int** n = 1,i, x; lfa = lfb = 0;

**for**(i=0;i<ln1;i++)

fa[lfa++] = P(n1[i],0);

**for**(i=0;i<ln2;i++)

fb[lfb++] = P(n2[i],0);

n=4<<(**int**)(**log**((**double**)max(ln1,ln2))/**log**(2.0));

**while**(lfa < n || lfb < n){

**if** (lfa < n) fa[lfa++] = P(0,0);

**if** (lfb < n) fb[lfb++] = P(0,0);

}

lfa = lfb = n;

fft (fa,n, **false**), fft (fb,n, **false**);

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

fa[i] = fa[i] \* fb[i];

fft (fa,n, **true**);

**for** (x=i=0, lsol=-1; i<n; ++i){

sol[i] = x + **int** (fa[i].x + 0.5);

x = sol[i] / 10;

sol[i] %= 10;

**if** (sol[i]) lsol = i;

}

}

**Fast Input**

**const** **int** bz = 10240;

**char** bf[bz + 1], \*ppp = bf;

**int** ch, sg, bt = 0;

**#define** GET(c) { \

**if**(ppp-bf==bt && (bt==0 || bt==bz)) { \

bt = fread(bf,1,bz,stdin); ppp=bf; }\

**if**(ppp-bf==bt && (bt>0 && bt<bz)) { \

bf[0] = 0; ppp=bf; } \

c = \*ppp++; \

}

**#define** number(n) { \

n=sg=0; **do** { GET(ch); } \

**while**(!isdigit(ch) && ch!='-'); \

**if**(ch=='-') { sg=1; GET(ch); } \

**while**(isdigit(ch)){n=10\*n+ch-48;GET(ch); } \

**if**(sg){ n= -n;} \

}

**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;

**Karp Rabin**

#define REHASH(a,b,h) ((((h)-(a)\*d)<<1)+(b))

void KR(char \*x, int m, char \*y, int n) {

int d, hx, hy, i, j = 0;

/\* Preprocessing \*/

/\* computes d = 2^(m-1) with

the left-shift operator \*/

for (d = i = 1; i < m; ++i)

d = (d<<1);

for (hy = hx = i = 0; i < m; ++i) {

hx = ((hx<<1) + x[i]);

hy = ((hy<<1) + y[i]);

}

/\* Searching \*/

while (j <= n-m) {

if (hx == hy && memcmp(x, y + j, m) == 0)

OUTPUT(j);

hy = REHASH(y[j], y[j + m], hy);

++j;

}

}

**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)

}

**LCS**

**struct** node {

**int** value;

node \*next;

**node**(**int** v,node \*n) :

value(v), next(n) { }

};

**#define** index\_of(as, x) \

distance(as.begin(),lower\_bound(as.begin(),as.end(),x))

**const** **int** oo = 99999999 ;

vector<**int**> **lcs\_hs**( vector<**int**> a, vector<**int**> b) {

**int** n = a.size(), m = b.size();

map< **int** , vector< **int** > > M;

**for** (**int** j= m-1 ;j >= 0; --j)

M[b[j]].push\_back(j);

vector<**int**> xs(n+ 1, oo); xs[ 0 ] = -oo;

vector<node\*> link(n+1);

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

**if** (M.count(a[i])) {

vector< **int** > ys = M[a[i]];

**for** (**int** j = 0;j< ys.size(); ++j) {

**int** k =index\_of(xs, ys[j]);

xs[k] =ys[j];

link[k]=**new** node(b[ ys[j]], link[k-1]);

}

}

vector<**int**> c;

**int** l =index\_of(xs, oo - 1 ) - 1 ;

**for** (node \*p= link[l]; p; p=p->next)

c.push\_back(p->value);

reverse(c.begin(), c.end());

**return** c;

}