PROPIEDADES DE FIBONACCI

$F_n = F_{n-2} + F_{n-1}$	$\sum_{i \le n} F_n = F_{n+2} - 1$
$\sum_{i \le n} F_i^2 = F_n * F_{n+1}$	$F_n^2 - F_{n-1} * F_{n+1} = -1^n$
$F_{2n} = F_n^2 + 2F_nF_{n-1}$	$F_{2n+1} = F_{n+1}^2 + F_n^2$
$F_{n+m} = F_{m+1} * F_n + F_m * F_{n-1}$	$\gcd(F_n, F_m) = F_{gcd(nm)}$ n>=3
$m \equiv 0 (mod\ n) \to F_m \equiv 0 (mod\ F_n)$	$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$

SQRT DESCOMPOSITION
EXAMPLE:Give an array a[] of size N, we query Q times. Each time we want to get the mode number(the value that
appears most often in a set of data) of subsequence a[l], a[l + 1] .. a[r].
En estos casos la idea seria,siendo S[l][r] la solucion para el intervalo l-r, poder transformar S[l][r] en un S[l'][r']
en tiempo lineal o logaritmico O((|l-l'|+|r-r'|) log N).
Teniendo esto lo que necesitamos es un orden conveniente para
las querys. Aqui es donde usamos la SQRT DESCOMPOSITION, consiste en descomponer el arreglo en bloques de sqrt(N). Teniendo
T = a el mayor numero <= que sqrt(N) ordenamos las query así:

```
bool cmp(Query A, Query B)
{
 if (A.l / T != B.l / T) return A.l / T < B.l / T;
 return A.r < B.r
}
```

Se puede probar que el costo total de todas las transformaciones N sqrt(N) log N.

COWPIC
los elementos estan numerados de 1 -> N.

```
for (int i = 0; i < N; i++) {
      fscanf (in, "%d", cow + i);
      loc [cow [i]] = i;
   }
```

inv = cantidad de pares invertidos calculados con ABI en O(N log N).

```
for (int i = 1; i <= N; i++) {
      inv += N - 1 - 2 * loc [i];
      best = min (best, inv);
   }
```

Para un grafo planar
Caras+Vertices=Aristas+Cantidad de componentes+1
Dos nodos pertenecen a una componente biconexa si hay dos caminos disjuntos(no tienen arista en común) entre ellos

TRABAJO CON BITS

Set union	Set intersection	Set subtraction	Set negation
A | B	A & B	A & ~B	ALL_BITS ^ A

Set bit	Clear bit	Test bit
A |= 1 << bit	A &= ~(1 << bit)	(A & 1 << bit) != 0

SumaDiv = FOR(i,k)
sum*=(prim[i]^(cant[i]+1)-1)/(prim[i]-1)
ProdDiv = P = N^(D/2)=Sqrt(N^D)

Numeros de Catalan
C[0]=C[1]=1;
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]

TODAS LAS MASCARAS S MENORES QUE M QUE CONTIENE SOLAMENTE LOS
BITS ACTIVOS EN M

```
void sub(int m){
int s = m ;
while  ( s > 0 ) {
... You can use the s ...
s = ( s - 1 )  & m ;
}
}
```

TODAS LAS MASCARAS S MENORES QUE M QUE CONTIENE SOLAMENTE LOS
BITS NO ACTIVOS EN M

```
void mask(int m){
int k = log2(m);
for(int s = (1<<k)-1; (s &= ~m) >= 0; s--){
... You can use the s ...
}
}
```

FACTORIAL MODULAR

```
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;
```

}

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

GRIRAR GRILLA 45 GRADOS

Matriz de N x M

X = X0 + Y0

Y = X0 – Y0 + max(N,M)

CANTIDAD DE DIGITOS DE N!

```
(ll)floor((log(2*M_PI*n)/2+n*(log(n)-1))/log(10))+1);
```

JOSEPHUS

```
int josephus(int n, int k) {
int f = 0;
for (int i = 0; i < n; i++) f = (f + k) % (i + 1);
return f + 1;
}

LL pot(LL b, int e){
   if (e == 0)
      return 1;
   if (e % 2 == 0){
      LL r = pot(b, e / 2);
      return (r * r) % MOD;
   }
   return (pot(b, e - 1) * b) % MOD;
}
//invMod(x) retorna el inverso modular de x en el sistema de restos
//de MOD, o sea, retorna un entero y tal q: (x * y) % MOD == 1.
LL invMod(LL x){
   //como MOD es primo entonces y = x^(MOD - 2)
   //ya q x^(MOD - 1) congruente con 1 mod MOD
   //por el Pequenno Teorema de Fermat.
   return pot(x, MOD - 2);
}
//bn(n, k) retorna el numero binomial correspondiente
//modulo MOD.
LL bn(int n, int k){
   LL denominador = (fact[k] * fact[n - k]) % MOD;
   LL inv = invMod(denominador);

   return (fact[n] * inv) % MOD;
}
```

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;
```

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
```

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.

Some useful series

1 + 2^2 + 3^2 + ... + n^2 = n * (n + 1) * (2*n + 1) / 6

1 + 2^3 + 3^3 + ... + n^3 = n * n * (n + 1) * (n + 1) / 4

1 + x^2 + x^3 + ... + x^k = (x^(k + 1) – 1) / (x - 1)

Joseph Problem

```
int joseph (int n, int k) {
int res = 0;
for (int i=1; i<=n; ++i)
res = (res + k) % i;
return res + 1;
}
```

Fibbonacci

```
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));
```

circular earthmover distance(restack)

```
int N;
    in >> N;
    vector<int> A(N + 1), B(N + 1);
    for (int i = 0; i < N; i++)
    {
        in >> A[i] >> B[i];
    }
    A[N] = A[0];
    B[N] = B[0];
    vector<int> S(N);
    S[0] = A[0] - B[0];
    for (int i = 1; i < N; i++)
        S[i] = A[i] + S[i - 1] - B[i];
    nth_element(S.begin(), S.begin() + N / 2, S.end());
    int m = S[N / 2];
    long long ans = 0;
    for (int i = 0; i < N; i++)
        ans += abs(S[i] - m);
    out << ans << endl;
```

$$
\begin{aligned}
a &= b^{\log_b a}\ , \\
\log_c(ab) &= \log_c a + \log_c b\ , \\
\log_b a^n &= n \log_b a\ , \\
\log_b a &= \frac{\log_c a}{\log_c b}\ , \\
\log_b(1/a) &= -\log_b a\ , \\
\log_b a &= \frac{1}{\log_a b}\ , \\
a^{\log_b c} &= c^{\log_b a}\ ,
\end{aligned}
$$

Gcd extendido e inverso modular

gcd(a,m) = 1 ⇐⇒ 1 = a.x + m.y

Luego x ≡ m a −1 , de modo que a tiene inverso mod m si y sólo si gcd(a,m) = 1. [Corolario: Z p es un cuerpo.] Para encontrar x e y, los rastreamos a trav´ es del algoritmo de Euclides:

```
p i i egcd ( int a , int b) {
 i f (b == 0) return make pair (1 , 0) ;
 else {
 p i i RES = egcd (b , a%b) ;
 return make pair (RES. second ,RES. f i
r s t -RES. second *(a/b) ) ;
 }
 }

 int inv ( int n , int m) {
 p i i EGCD = egcd (n , m) ;
 return ( (EGCD. f i r s t% m)+m)% m;
 }
```

Busqueda ternaria

```
double TS(double A, double B) {
double left = A, right = B;
while(abs(right−left) < EPS) {
double lt = (2.∗left + right) / 3;
double rt = (left + 2.∗right) / 3;
if(f(lt) < f(rt)) left = lt ;
else right = rt ;
}
return (left+right)/2;
}
```

```
// Shanks' Algorithm for the discrete logarithm problem O(sqrt(m))
// return x such that a^x = b mod m
intsolve ( int a, int b, int m ) {
	int n = ( int ) sqrt ( m + .0 ) + 1 ;
	int an = 1 ;
	for ( int i = 0 ; i < n ; ++ i )
		an = ( an * a ) % m ;
	map<int , int> vals ;
	for ( int i = 1 , cur = an ; i <= n ; ++ i ) {
		if ( ! vals. count ( cur ) )
			vals [ cur ] = i ;
		cur = ( cur * an ) % m ;
	}
	for ( int i = 0 , cur = b ; i <= n ; ++ i ) {
		if ( vals. count ( cur ) ) {
			int ans = vals [ cur ] * n - i ;
			if ( ans < m )
				return ans ;
		}
		cur = ( cur * a ) % m ;
	}
	return - 1 ;
}
boolean isConvex(int n, int[] x, int[] y){
  int pos = 0, neg = 0;
  for(int i = 0; i < n; i++){
    int prev = (i + n – 1) % n, next = (i + 1) % n;
    int pc = (x[next]-x[i])*(y[prev]-y[i]) -
   (x[prev]-x[i])*(y[next]-y[i]);
    if(pc < 0){
      neg++;
    }else if(pc > 0){
   pos++;
    }
  }
  return (neg == 0) || (pos == 0);
}
```