```
ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
```

Number Theory

el numero mas cercano no coprimo con n es n+p(n-p), donde p es primer primo que divide a n.

MCD y MCM

MCM(a,b)=a*b/MCD(a,b);

Si mcd(a,b) = d y a,b,c numeros enteros. Entonces se cumple

i) mcd(a/d ,b/d) = 1

ii) mcd(a + cb,b) = mcd(a,b)

Sean c,d enteros tales que c = dq + r. Entonces es (c,d) = (d,r) .

Congruencia

a≡b(mod m) si m divide a (a-b).

si a≡b(mod m) entonces a= km+b

a≡a(mod m) reflexiva

si a≡b(mod m) entonces b≡a(mod m) simetrica

si a≡b(mod m) y b≡c(mod m) entonces a≡c(mod m) transitiva

Si a,b,c,m son números enteros (m > 0) y a ≡ b(mod m), entonces es

i) a + c ≡ b + c(mod m)

ii) a - c ≡ b - c(mod m)

ii) ac congruente bc(mod m).

Si a,b,c,m son números enteros (m > 0) tales que ac ≡ bc(mod m) y d = (c,m), entonces es a ≡ b(mod m/d).

Si (c,m) = 1 y ac ≡ bc(mod m), entonces es a ≡ b(mod m). (a,b)->MCD(a,b)

Si a,b,c,d,m son números enteros (m > 0) tales que a ≡ b(mod m) y c ≡ d(mod m), entonces es

i) a ± c ≡ b ± d(mod m)

ii) ac ≡ bd(mod m).

Si a ≡ b(mod m), entonces a k ≡ b k (mod m) para todo k > 0.

Si m 1 ,m 2 ,...,m k son enteros positivos tales que a ≡ b(mod m i), entonces es

a ≡ b(mod [m 1 ,m 2 ,...,m k]). [a,b]->minimo comun multiplo de a y b

Si p es un número primo, entonces es

(p − 1)! ≡ −1(mod p)

Si n es un número entero positivo tal que

(n − 1)! ≡ −1(mod n),

entonces n es primo.

(El Pequeño Teorema de Fermat).

Sea a un número entero positivo y p un número primo que no lo divide, entonces es

a ^p−1 ≡ 1(mod p)

Sea a un número entero positivo y p un número primo que no lo divide, entonces a^ p−2 es inverso de a módulo p.

Sea a y b son enteros positivos y p un número primo que no divide a a, entonces la solución de la ecuación ax ≡ b(mod p) es x ≡ a^p−2 b(mod p).

(Teorema de Euler).

Si m es un entero positivo tal que (a,m) = 1, entonces

a φ(m) ≡ 1(mod m)

Funcion de euler

i) p es primo si y sólo si φ(p) = p − 1.

ii) Si α > 0 y p es primo, entonces es φ(p α) = p α − p α−1 .

Si (m,n) = 1, entonces es φ(mn) = φ(m)φ(n).

φ(n)=n (1 − 1/p1) (1 − 1/p2) ... (1 - 1/pn) pi->descomposición en factores primos de n

```
int phi(int a,int p[])
{
   int b=a;
   for(int i=0; p[i]*p[i]<=a; i++)
      if(a%p[i]==0)
      {
         b=b/p[i]*(p[i]-1);
         do a/=p[i];
         while(a%p[i]==0);
      }
   if(a>1) b=b/a*(a-1);
   return b;
}
```

φ(n) es par para todo valor positivo de n > 2.

Suma de los divisores

sum*=(prim[i]^(exp[i]+1)-1)/(prim[i]-1);

Producto de los divisores

ProdDiv = P = N^(D/2)=Sqrt(N^D) D=cantidad de divisores

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;
}
```

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 ;
}
```

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)

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;
}
```

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]
```

Propiedades

- Numero de secuencias correctas de paréntesis.
- Numero de arboles binaries no n+1 hojas
- Numero de triangulaciones de un poligono de n+2 lados
- Numero de formas de conectar 2n puntos en un circulo con cuerdas disjuntas
- Numero de caminos monótonos desde 0,0 hasta n,n (no diagonales)

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));
```

CANTIDAD DE DIGITOS DE N!

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

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

NOTES

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

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)

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;
}
```

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;
}
```

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;
```

En una secuencia de longitud n tiene que haber una subsecuencia monotona de al menos sqrt(N) elementos

Busqueda ternaria

Busqueda ternaria (para una funcion primero estrictamente creciente y luego estrictamente decreciente o viceversa)

```
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;
 }
```

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

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 ...
}
}
```

Probabilidad

pi->probabilidad de que ocurra el evento i;

P(todos eventos)=p1*p2*p3*...*pn; (independientes)

P(al menos uno)=1-((1-p1)*(1-p2)*...*(1-pn)); (independientes)

La probabilidad de aparicion simultanea de dos sucesos dependiente es igual al producto de la probabilidad de que aparezca el primer suceso por la probabilidad de que aparezca el segundo suponiendo que el primero sucedio

La probabilidad de que ocurra uno de los dos sucesos que se excluyen reciprocamente es la suma de las probabilidades de que ocurran cada uno

La probabilidad de que ocurra por lo menos uno de dos procesos simultaneos es la suma de las probabilidades de que ocurra cada uno menos la probabilidad de que ocurran simultaneamente(no importa si los sucesos son dependientes o independientes)

Valor esperado M=sumatoria(i*pi);

Propiedades

M(constante)=constante

M(cX)=c*M(X);

M(XY)=M(X)*M(Y);

M(X+Y)=M(X)+M(Y);

El valor esperado del numero de apariciones del evento A en n pruebas independientes es igual a n*probabilidad de que suceda el evneto en cada prueba

numeros de grundy

Sea y: Todas las posibles posiciones a las cuales nos podemos mover desde n.
G(n) = min(x>=0, x != G(y)).
Es decir, del conjunto de todos los números Grundy de las posiciones a las queme puedo mover desde n, el número Grundy de n va a ser el menor entero no negativo que no aparezca en dicho conjunto. Todas las posiciones terminales tienen números Grundy 0.
0 xor 1 = 1.
1 xor 1 = 0.
0 xor 0 = 0.
Para una serie de juegos, una posición es terminal si xor de los números de grundy de cada juego es igual a 0.

```
#define iszero(t) (t.len==1&&t.s[0]==0)
#define setlen(l,t) t.len=l;while(t.len>1&&t.s[t.len-1]==0) t.len--
const int maxlen=100;
struct bigint
{
    int len,s[maxlen];
    bigint() { *this=0; }
    bigint(int a) { *this=a; }
    bigint(const char *a) { *this=a; }
    bigint operator=(int);
    bigint operator=(const char*);
    bigint operator=(const bigint&); //Optional
    friend ostream& operator<<(ostream&,const bigint&);
    bigint operator+(const bigint&);
    bigint operator-(const bigint&);
    bigint operator*(const bigint&);
    bigint operator/(const bigint&); //Require - cmp
    bigint operator%(const bigint&); //Require - cmp
    static int cmp(const bigint&,const bigint&);
    static bigint sqrt(const bigint&); //Require - * cmp
};
bigint bigint::operator=(int a)
{
    len=0;
    do
    {
        s[len++]=a%10;
        a/=10;
    }
    while(a>0);
    return *this;
}
bigint bigint::operator=(const char *a)
{
    len=strlen(a);
    for(int i=0; i<len; i++) s[i]=a[len-i-1]-'0';
    return *this;
}
bigint bigint::operator=(const bigint &a)
{
    len=a.len;
    memcpy(s,a.s,sizeof(*s)*len);
    return *this;
}
ostream& operator<<(ostream &os,const bigint &a)
{
    for(int i=a.len-1; i>=0; i--) os<<a.s[i];
    return os;
}
bigint bigint::operator+(const bigint &a)
{
    bigint b;
    b.s[b.len=max(len,a.len)]=0;
    for(int i=0; i<b.len; i++) b.s[i]=(i<len?s[i]:0)+(i<a.len?a.s[i]:0);
    for(int i=0; i<b.len; i++)
        if(b.s[i]>=10)
        {
```

```
            b.s[i]-=10;
            b.s[i+1]++;
        }
    if(b.s[b.len]) b.len++;
    return b;
}
//Make sure *this>=a
bigint bigint::operator-(const bigint &a)
{
    bigint b;
    for(int i=0; i<len; i++) b.s[i]=s[i]-(i<a.len?a.s[i]:0);
    for(int i=0; i<len; i++)
        if(b.s[i]<0)
        {
            b.s[i]+=10;
            b.s[i+1]--;
        }
    setlen(len,b);
    return b;
}
bigint bigint::operator*(const bigint &a)
{
    bigint b;
    memset(b.s,0,sizeof(*s)*(len+a.len+1));
    for(int i=0; i<len; i++)
        for(int j=0; j<a.len; j++) b.s[i+j]+=s[i]*a.s[j];
    for(int i=0; i<len+a.len; i++)
    {
        b.s[i+1]+=b.s[i]/10;
        b.s[i]%=10;
    }
    setlen(len+a.len+1,b);
    return b;
}
bigint bigint::operator/(const bigint &a)
{
    bigint b,c;
    for(int i=len-1; i>=0; i--)
    {
        if(!iszero(b))
        {
            for(int j=b.len; j>0; j--) b.s[j]=b.s[j-1];
            b.len++;
        }
        b.s[0]=s[i];
        c.s[i]=0;
        while(cmp(b,a)>=0)
        {
            b=b-a;
            c.s[i]++;
        }
    }
    setlen(len,c);
    return c;
}
bigint bigint::operator%(const bigint &a)
{
    bigint b;
    for(int i=len-1; i>=0; i--)
    {
        if(!iszero(b))
        {
            for(int j=b.len; j>0; j--) b.s[j]=b.s[j-1];
            b.len++;
        }
        b.s[0]=s[i];
        while(cmp(b,a)>=0) b=b-a;
    }
    return b;
}
```

```
int bigint::cmp(const bigint &a,const bigint &b)
{
   if(a.len<b.len) return -1;
   else if(a.len>b.len) return 1;
   for(int i=a.len-1; i>=0; i--)
      if(a.s[i]!=b.s[i]) return a.s[i]-b.s[i];
   return 0;
}
bigint bigint::sqrt(const bigint &a)
{
   int n=(a.len-1)/2,p;
   bigint b,d;
   b.len=n+1;
   for(int i=n; i>=0; i--)
   {
      if(!iszero(d))
      {
         for(int j=d.len+1; j>1; j--) d.s[j]=d.s[j-2];
         d.s[0]=a.s[i*2];
         d.s[1]=a.s[i*2+1];
         d.len+=2;
      }
      else d=a.s[i*2]+(i*2+1<a.len?a.s[i*2+1]*10:0);
      bigint c;
      c.s[1]=0;
      for(int j=1; j<=n-i; j++)
      {
         c.s[j]+=b.s[i+j]<<1;
         if(c.s[j]>=10)
         {
            c.s[j+1]=1;
            c.s[j]-=10;
         }
         else c.s[j+1]=0;
      }
      c.len=n-i+1+c.s[n-i+1];
      for(p=1;; p++)
      {
         c.s[0]=p;
         if(cmp(d,c*p)<0) break;
      }
      b.s[i]=c.s[0]=p-1;
      d=d-c*(p-1);
   }
   return b;
}

scanf("[^\n]",cad);
```