Teaoria de Numeros

**public** **static** **long** minimoComumMultiplo(**long** a, **long** b) {

**return** (a \* b) / *maximoComunDivisor*(a, b);

}

**public** **static** **long** minimoComumMultiplo(ArrayList<Long> a) {

**long** temp = 0;

temp = *minimoComumMultiplo*(a.get(0), a.get(1));

**for** (**int** i = 2; i < a.size(); i++) {

temp = *minimoComumMultiplo*(temp, a.get(i));

}

**return** temp;

}

**public** **static** **long** minimoComumMultiplo2(Object[] objects) {

// a es un arreglo ordenado

**long** MCM = (Long) objects[objects.length - 1];

**long** temp = MCM;

**for** (**int** i = objects.length - 1, j = 2; i >= 0;) {

/\* opcional&& temp<=1000000 \*/

**if** (temp % (Long) objects[i] != 0) {

temp = MCM \* j;

j++;

i = objects.length - 1;

} **else** {

i--;

}

}

**return** temp;

}

**public** **static** **long** maximoComunDivisor(**long** a, **long** b) {

**if** (b % a == 0) {

**return** a;

} **else** {

**long** residuo = b % a;

**return** *maximoComunDivisor*(residuo, a);

}

}

**public** **static** **long** maximoComunDivisor(ArrayList<Long> a) {

**long** temp = 0;

temp = *maximoComunDivisor*(a.get(0), a.get(1));

**for** (**int** i = 2; i < a.size(); i++) {

temp = *maximoComunDivisor*(temp, a.get(i));

}

**return** temp;

}

**Reglas de Divisibilidad**

1. Un número es divisible por 2 si acaba en 0 o en cifra par.
2. Un número es divisible por 3 si lo es la suma de sus cifras.
3. Un número es divisible por 4 si lo es el número formado por sus dos últimas cifras.
4. Un número es divisible por 5 si acaba en 0 o en 5.
5. Un número es divisible por 6 si lo es por 2 y por 3 al mismo tiempo.
6. Un número es divisible por 8 si lo es el formado por sus tres últimas cifras.
7. Un número es divisible por 9 si lo es la suma de sus cifras.
8. Un número es divisible por 10 si acaba en 0.
9. Un número es divisible por 11 si lo es la diferencia entre la suma de las cifras que ocupan lugar impar y la suma de las que ocupan lugar par.
10. Un número es divisible por 25 si lo es el formado por sus dos últimas cifras.
11. Un número es divisible por 100, 1000, 10000,... si acaba en 00, 000, 0000,...
12. Para saber si un número es divisible por 7 se multiplica por 2 la cifra de las unidades y el resultado se resta del número que forman las cifras restantes. Este proceso se repite hasta que la diferencia está formada por una o dos cifras; si éstas son 0 o múltiplo de 7, el número inicial será divisible por 7.

**Números Primos**

**public** **class** **Criba** {//Para saber si un numero es Primo veo si n es impar y status en N es false;

**static** **int** *N* = 1000;

**static** **boolean**[] *status* = **new** **boolean**[1001];

**public** **static** **void** main(String[] args) {

**int** i, j, sqrtN;

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

*status*[i] = **false**;

}

sqrtN = (**int**) Math.*sqrt*((**double**) *N*);

**for** (i = 3; i <= sqrtN; i += 2) {

**if** (!*status*[i]) {

**for** (j = i \* i; j <= *N*; j += i + i) {

*status*[j] = **true**;

}

}

}

System.*out*.println(2);

**for** (i = 3; i <= *N*; i += 2) {

**if** (!*status*[i]) {

System.*out*.println(i);

}

}

}

}

public boolean[] **sieve**(int n) { **//TopCoder Es primo si prime[i];**

boolean[] prime=new boolean[n+1];

Arrays.fill(prime,true);

prime[0]=false;

prime[1]=false;

int m=(int) Math.sqrt(n);

for (int i=2; i<=m; i++)

if (prime[i])

for (int k=i\*i; k<=n; k+=i)

prime[k]=false;

return prime;

}

Si quisiera contar la cantidad de primos menores que n cada vez que ponga a alguien false incremento un contador y luego resto n –contador.

Descomposicion en factores primos:

Primero lleno un arreglo con los primos menores que la Raiz del numero a descomponer mas uno o dos primos por acaso.

private static ArrayList<Long> **factoresPrimos**(Long number) {

ArrayList<Long> fP = new ArrayList<Long>();

try {

for (int i = 0; number > 1; i++) {

while (number % primos.get(i) == 0) {

number = number / primos.get(i));

fP.add(primos.get(i));

}

}

} catch (Exception e) {

fP.add(number);

}

return fP;

}

**Factores primos**

static ArrayList<Integer> facts(int x) {

boolean[] num = new boolean[x];

ArrayList<Integer> dP = new ArrayList<Integer>();

int[] primos = new int[x];

int pos = 0;

int sq = (int) Math.sqrt(x);

for (int i = 2; i <= sq; i++)

if (!num[i])

for (int j = 2 \* i; j < x; j += i)

num[j] = true;

for (int i = 2; i < x; i++)

if (!num[i])

primos[pos++] = i;

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

while (x % primos[i] == 0) {

dP.add(primos[i]);

x /= primos[i];

}

}

return dP;

}

**ARITMETICA MODULAR**

* (x + y) mod n = ((x mod n) + (y mod n)) mod n
* (x - y) mod n = ((x mod n) - (y mod n) + n) mod n
* xy mod n = (x mod n)(y mod n) mod n

**TEORIA DE NUMEROS**

* Dos números primos son primos relativos (cooprimos) su GCD es 1.
* Si **a** y **p** son cooprimos, entonces se cumple: **a^p – a** es divisible por **p**
* Cada entero positivo es la suma de a lo sumo 4 cuadrados.
* Todo primo de la forma 4n+1 se puede descomponer únicamente como suma de dos cuadrados.
* Todo entero es o bien un cubo o la suma de nueve cubos; además todo entero es o una cuarta potencia o la suma de como máximo 19 cuartas potencias.
* Todo entero par es la suma de dos primos y todo entero impar es o un primo o la suma de tres primos.
* Un primo de la forma 3n + 1 puede expresarse únicamente en la forma: x^2 + 3y^2.
* Para todo primo p, la cantidad (p-1)! + 1 es divisible por p; además si la cantidad es divisible por q, entonces q es primo*.*

**Fórmula para calcular el número de *Arboles Binarios* que se pueden formar con N nodos:**

2n! / (n! \* n! \* (n+1))

C (N / K) = N! / ((N-K)!\*K!)

P (N / K) = N! / (N-K)!

**Digitos de n!**

int factDigits(int n) {

return (int)(0.5\*Math.log10(2\*n\*Math.PI) + n\*Math.log10(n/Math.E)+1);

}

**Suma de una serie geometrica:**

S = a \* ( 1 – r ^ n ) / ( 1 - r)

**BIT OPERATIONS**

**number of bits 1 in “n” in binary.**

//via 1

int count(long n) {

int num = 0;

for (int i = 0; i < 32; i++)

if (n & (1 << i))

num++;

return num;

}

//via 2

int count(long n) {

int num = 0;

if (n)

do

num++;

while (n &= n - 1);

return num;

}

**// number of parity bits of a binary representation of a number of n.**

int parity(long n) {

n = ((0xFFFF0000 & n) >> 16) ^ (n & 0xFFFF);

n = ((0xFF00 & n) >> 8) ^ (n & 0xFF);

n = ((0xF0 & n) >> 4) ^ (n & 0xF);

n = ((12 & n) >> 2) ^ (n & 3);

n = ((2 & n) >> 1) ^ (n & 1);

return n;

}

**//Determine the least significant bit 1 in the binary representation of n.**

int low1(long n) {

return n ^ (n & (n - 1));

}

**// most significant bit with value 1 in the binary representation of n.**

int high1(long n) {

long num = 0;

if (!n)

return -1;

if (0xFFFF0000 & n) {

n = (0xFFFF0000 & n) >> 16;

num += 16;

}

if (0xFF00 & n) {

n = (0xFF00 & n) >> 8;

num += 8;

}

if (0xF0 & n) {

n = (0xF0 & n) >> 4;

num += 4;

}

if (12 & n) {

n = (12 & n) >> 2;

num += 2;

}

if (2 & n) {

n = (2 & n) >> 1;

num += 1;

}

return 1 << num;

**//Index of the most significant bit 1 in the binary representation of n.**

int indexHigh1(long n) {

n = n | (n >> 1);

n = n | (n >> 2);

n = n | (n >> 4);

n = n | (n >> 8);

n = n | (n >> 16);

return count(n) - 1;

}

//Determine if n is a power of 2

int isTwoPower(long n) {

return (n & (n - 1)) == 0;

}

**CALCULAR EL DETERMINANTE DE UNA MATRIZ**

const double EPS = 1E-9;

int n;

scanf("%d", &n);

vector<vector<double> > a(n, vector<double>(n));

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

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

scanf("%lf", &a[i][j]);

double det = 1;

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

int k = i;

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

if (abs(a[j][i]) > abs(a[k][i]))

k = j;

if (abs(a[k][i]) < EPS) {

det = 0;

break; }

swap(a[i], a[k]);

if (i != k)

det = -det;

det \*= a[i][i];

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

a[i][j] /= a[i][i];

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

if (j != i && abs(a[j][i]) > EPS)

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

a[j][k] -= a[i][k] \* a[j][i];}

cout << det;

**BUSCAR DIVISORES DE UN NUMERO**

ll divisor[1000000];

ll ndiv;

void find\_div(ll N) {

ll i, p, old;

if (N < 0)

N = -N;

ndiv = 0;

divisor[ndiv++] = 1;

for (p = 2; p <= 46337 && p \* p <= N; p++) {

old = ndiv;

while (N % p == 0) {

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

ll tmp = divisor[ndiv - old] \* p;

divisor[ndiv] = tmp;

ndiv++;}

N /= p;}}

if (N > 1) {

old = ndiv;

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

divisor[ndiv] = divisor[ndiv - old] \* N;

ndiv++; }}

//sort(divisor, divisor + ndiv); }

**GAUSS PARA SISTEMA DE ECUACIONES DE NxN**

int n; //cantidad de variables

scanf("%d", &n);

vector<vector<double> > a(n, vector<double>(n + 1));

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

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

scanf("%lf", &a[i][j]);

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

int k = i;

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

if (abs(a[j][i]) > abs(a[k][i]))

k = j;

swap(a[i], a[k]);

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

a[i][j] /= a[i][i];

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

if (j != i)

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

a[j][k] -= a[i][k] \* a[j][i]; }

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

printf("%.15lf\n", a[i][n]);

**CANTIDAD DE DIGITOS EN N!**

//para n > 3

int n = 20;

int PI = 2 \* acos(0);

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

**CANTIDAD DE NUMEROS PALINDROMES**

//cant de palindromes con ‘n’ o menos digitos

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

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

**CANTIDAD DE NUMEROS PRIMOS EN N!**

//p es el numero primo, to es el limite

//aux inicialmente = p, inicialmente s = 0

#define get\_cnt\_primes(p, to)\

while(aux <= to) {\

s += to / aux;\

aux \*= p;\

}

**DESCOMPONER NÚMERO EN SUMADOS SIN REPETICION**

//primera llamada DescEnSum(n, 1, 0, "", n);

//array es un StringBuffer para acelerar la impresión en Java

void DescEnSum(int n, int pos, int ninc, String s, int numero) {

if (ninc == numero) {

if(array[numero] == null)

array[numero] = new StringBuffer();

array[numero].append(numero + " = " + s.substring(0, s.length() - 3) + "\n");

}

for (int i = pos; i <= n; i++)

DescomponEnSumando(n - i, i, ninc + i, s + i + " + ", numero);}

**CANTIDAD DE DIGITOS ENTRE 0-9 DESDE 1 HASTA N (DIGIT COUNT)**

typedef long long ll;

//sol es un arreglo de int, donde va a estar la respuesta

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; } }

**CANTIDAD DE DIVISORES, SUMA DIVISORES, MULT DIVISORES**

N=p^a\*q^b\*r^c //descomposicion de N

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)

**CRIBA DE ERASTOTENES**

inline void Erastotenes() {

for (int i = 2; i < MAXN; ++i)

primes[i] = i;

for (int i = 2; i \* i < MAXN; ++i)

if (primes[i])

for (int j = i \* i; j < MAXN; j += i)

primes[j] = 0;

remove(primes, primes + MAXN, 0); }

**EULER TOTIEN FUNCTION**

//EulerTotientFunction(n) es la cantidad de numeros de 1 a n coprimos con n

ll EulerTotientFunction(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; }

**EXPONENCIACION MODULAR**

|  |  |
| --- | --- |
| int modulo(long a, long b, long c) {  long x = 1, y = a;  while (b > 0) {  if (b % 2 == 1) {  x = (x \* y) % c;}  y = (y \* y) % c;  b /= 2;}  return (int) (x % c); } | int bigmod(int b, int p, int m){  int mask = 1;  int pow2 = b % m;  int r = 1;  while (mask){  if (p & mask) r = (r \* pow2) % m;  pow2 = (pow2 \* pow2) % m;  mask <<= 1;  }  return r; } |

**EUCLIDES EXTENDIDO**

//para division modular se llama de la sgte forma:

//a = MOD, b = den, x = 1, y = 0

//la respuesta es ((((y + MOD) % MOD) \* num)) % MOD

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

ll g = a;

x = 1;

y = 0;

if (b != 0) {

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

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

return g; }

**CALCULAR N! MOD P**

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; }

**GCD**

int GCD(int a, int b) {

int r;

while (b != 0) {

a = a % b;

r = a;

a = b;

b = r;}

return a; }

**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; }

**NEWTON PAPHSON**

//calcular la raiz cuadrada de un numero

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; }

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]

**POLARD RO**

//descomponer en factores primos, un numero de hasta 64 bit

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 = mul(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); }

**RABIN MILLER**

//chequear si un numero es primo, hasta 64 bit

typedef unsigned long long ull;

#define MAXP 10

const int prime[MAXP] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 };

ull N;

ull mul(ull a, ull b, ull m) {

if (!b)

return 0;

ull ret = 2 \* mul(a, b / 2, m) % m;

if (b % 2)

ret = (ret + a) % m;

return ret; }

ull power(ull x, ull n, ull m) {

ull ret = 1;

for (; n; x = mul(x, x, m), n /= 2)

if (n % 2)

ret = mul(ret, x, m);

return ret; }

bool rabin\_miller(ull p) {

if (p < 2 || (p > 2 && !(p % 2)))

return false;

for (int i = 0; i < MAXP && prime[i] < p; i++)

if (power(prime[i], p - 1, p) != 1)

return false;

return true; }

**CANTIDAD DE CEROS AL FINAL DE N!**

long N, B, i, j, p, c, noz, k;

while (true) {

st = new StringTokenizer(cin.readLine());

N = Long.parseLong(st.nextToken());

B = Long.parseLong(st.nextToken());

if (N == 0 && B == 0)

break;

noz = N;

j = B;

boolean test = true;

for (i = 2; i <= B && j > 1; i++) {

if(test && isPrime(j, 4))

if(i <= j)

i = j;

if (j % i == 0) {

test = true;

p = 0;

while (j % i == 0) {

p++;

j /= i;}

c = 0;

k = N;

while (k / i > 0) {

c += k / i;

k /= i;}

noz = Math.min(noz, c / p);

} else

test = false; }

System.out.println(noz); }

**CALCULAR TERMINO DE FIBBONACCI MODULO MOD**

long fibo(long n) {

long[][] fib = { { 1, 1 }, { 1, 0 } };

long[][] ret = { { 1, 0 }, { 0, 1 } };

long[][] tmp = { { 0, 0 }, { 0, 0 } };

while (n > 0) {

if (n % 2 == 1) {

tmp[0][0] = 0; tmp[0][1] = 0; tmp[1][0] = 0; tmp[1][1] = 0;

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

for (int k = 0; k < 2; k++)

tmp[i][j] = (tmp[i][j] + ret[i][k] \* fib[k][j])% MOD;

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

ret[i][j] = tmp[i][j];}

tmp[0][0] = 0;tmp[0][1] = 0;tmp[1][0] = 0;tmp[1][1] = 0;

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

for (int k = 0; k < 2; k++)

tmp[i][j] = (tmp[i][j] + fib[i][k] \* fib[k][j]) % MOD;

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

fib[i][j] = tmp[i][j];

n /= 2; }

return ret[0][1] % MOD; }