Number Theory

el número más cercano no coprimo con n es n+p o 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

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

Ecuaciones diofanticas

Sean a,b ∈ Z tales que d = (a,b). Entonces la ecuación ax + by = c tiene

solución si y sólo si d|c. En ese caso existen infinitas soluciones, siendo

todas ellas de la forma:

x = x0 + b/d \* n

y = y0 – a/d \* n

con x0 ,y0 solución particular de la ecuación dada.

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:

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 ; // no Solucion

inv = (x + m) % 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 binarios 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.

Sumatoria de F[i]^2 = F[n]\*F[n-1].

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

- Los numeros consecutivos de Fibonacci son primos entre si.

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

- 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){

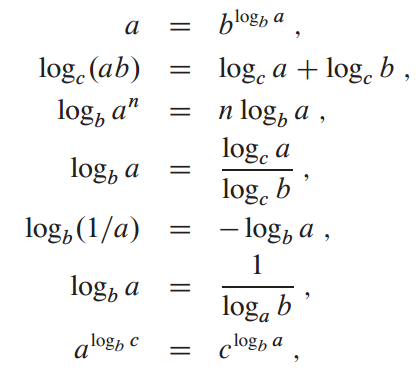
   pos++;

    }

  }

  return (neg == 0) || (pos == 0);

}



**NOTES**

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

GIRAR GRILLA 45 GRADOS

Matriz de N x M

X = X0 + Y0

Y = X0 – Y0 + max(N,M)

**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 reciprocamentees 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 evento 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.

Grafo

Grafos Bipartitos (Teorema de Konig). En un grafo bipartito G=(V,E), la maxima cardinalidad decualquier matching iguala la minima cardinalidad de cualquier vertex cover de G. Es decir, si se quiere encontrar el minimum vertex cover de cualquier grafo es suficiente con encontrar el maximum matching del grafo.

## Conjunto Parcialmente Ordenado (Partially Ordered Set):

Sea *S* un conjunto de elementos y  ≤  un orden parcial. O sea, para algunos elementos de x,y de *S* se tiene *x* ≤ *y*, y debe ser:

* Reflexiva (*x* ≤ *x*)
* Antisimétrica (si *x* ≤ *y* y *y* ≤ *x*, entonces *x* = *y*)
* Transitiva (si *x* ≤ *y* y *y* ≤ *z*, entonces *x* ≤ *z*)

Un ejemplo de conjunto parcialmente ordenado es el de los puntos (*x*, *y*) en el plano, con el operador (*x*1, *y*1) ≤ (*x*2, *y*2) si y solo si *x*1 ≤ *x*2 y *y*1 ≤ *y*2

## Cadena:

Se define una **cadena** *C* como un subconjunto de *S* {*x*1, *x*2, …, *xn* } tal que *x*1 ≤ *x*2 ≤ … ≤ *xn*.

## Anticadena:

Se define una **anticadena** *A* como un subconjunto de *S* tal que para todo par de elementos *x*,*y* de *A* no se cumple *x* ≤ *y* ni *y* ≤ *x*. O sea en una anticadena no existe un par de elementos que sean comparables. Al tamaño de la mayor anticadena se le llama ancho del conjunto y se va a denotar como **a(S)**.

## Partición:

Una **particion** *P* es un conjunto de cadenas de forma tal que cada elemento de S pertenece a exactamente una cadena. Al tamaño de la menor partición se le va a denotar como **mp(P)**.

# Teorena de Dilworth

El teorema plantea que el tamaño de la **mayor anticadena** en un conjunto finito parcialmente ordenado S es igual al tamaño de la **menor partición**, o sea que **a(S) = mp(S)**.

# Dual del teorema de Dilworth

El teorema dual de Dilworth plantea que el tamaño de la mayor cadena un conjunto parcialmente ordenado es igual a la menor cantidad de anticadenas en que se puede particionar.

C++

Leer hasta fin de linea : scanf(“[^\n]”,cad);

## C(n,k)%M Luca's Theorm

C(n,m)%M

sea (nk,nk-1,nk-2,..n0) el numero n en base M

sea (mk,mk-1,mk-2,..m0) el numero m en base M

entonces C(n,m)%M = (C(n0,m0)%M \* C(n1,m1)%M ... C(nk,mk)%M)%M;

Circles in a plane

//trace in a plane n circles that any one of them cut all the rest. Three circles

//can not pass through the same point. the number of regions that are limited in the plane

n\*(n-1)+2

Comparacion de floats

== if( fabs(x-y) <= eps )

!= if( fabs(x-y) > eps )

< if( (x-y) < eps )

<= if( (x-y) <= eps )

> if( (x-y) > eps )

>= if( (x-y) >= eps )

Cutting Circle

//the number of regions that is divided a circle after

//drawing all possible lines that connect n points

n\*n\*n\*n-6\*n\*n\*n+23\*n\*n-18\*n+24)/24

Desarranjo

//Cantidad de permutaciones de 1-n tal que ningun elemento caiga en su posicion

d(1) = 0, d(2) = 1

d(n) = (n-1)\*(d(n-1) + d(n-2))

Ideas de soluciones

8. Solution Ideas

• Dynamic Programming

– Drop a parameter, recover from others

– Swap answer and a parameter

– Parsing CFGs: CYK Algorithm

– Optimizations

\* Convex hull optimization

• dp[i] = minj < i {dp[j] + b[j]×a[i]}

• b[j] = b[j + 1]

• optionally a[i] = a[i + 1]

• O(n^2 ) to O(n)

\* Divide and conquer optimization

• dp[i][j] = mink < j {dp[i - 1][k] + C[k][j]}

• A[i][j] = A[i][j + 1]

• O(kn^2 ) to O(knlogn)

• sufficient:C[a][c]+C[b][d]=C[a][d]+C[b][c],

a = b = c = d (QI)

\* Knuth optimization

• dp[i][j] = mini<k<j {dp[i][k]+dp[k][j]+C[i][j]}

• A[i][j - 1] = A[i][j] = A[i + 1][j]

• O(n^3 ) to O(n^2 )

• sufficient:QI and C[b][c] = C[a][d],a=b=c=d

• Greedy

• Randomized

• Optimizations

– Use bitset (/64)

– Switch order of loops (cache locality)

• Process queries offline

– Mo’s algorithm

• Square-root decomposition

• Precomputation

• Efficient simulation

– Mo’s algorithm

– Sqrt decomposition

– Store 2^k jump pointers

• Data structure techniques

– Sqrt buckets

– Store 2^k jump pointers

– 2^k merging trick

• Counting

– Inclusion-exclusion principle

– Generating functions

• Graphs

– Can we model the problem as a graph?

– Can we use any properties of the graph?

– Strongly connected components

– Cycles (or odd cycles)

– Bipartite (no odd cycles)

\* Bipartite matching

\* Hall’s marriage theorem

\* Stable Marriage

– Cut vertex/bridge

– Biconnected components

– Degrees of vertices (odd/even)

– Trees

\* Heavy-light decomposition

\* Centroid decomposition

\* Least common ancestor

\* Centers of the tree

– Eulerian path/circuit

– Chinese postman problem

– Topological sort

– (Min-Cost) Max Flow

– Min Cut

\* Maximum Density Subgraph

– Huffman Coding

– Min-Cost Arborescence

– Steiner Tree

– Kirchoff’s matrix tree theorem

– Prüfer sequences

– Lovász Toggle

– Look at the DFS tree (which has no cross-edges)

• Mathematics

– Is the function multiplicative?

– Look for a pattern

– Permutations

\* Consider the cycles of the permutation

– Functions

\* Sum of piecewise-linear functions is a

piecewise-linear function

\* Sum of convex(concave)functions is convex (concave)

– Modular arithmetic

\* Chinese Remainder Theorem

\* Linear Congruence

– Sieve

– System of linear equations

– Values to big to represent?

\* Compute using the logarithm

\* Divide everything by some large value

– Linear programming

\* Is the dual problem easier to solve?

• Logic

– 2-SAT

– XOR-SAT (Gauss elimination or Bipartite matching)

• Meet in the middle

• Only work with the smaller half (log(n))

• Strings

– Trie (maybe over something weird, like bits)

– Suffix array

– Suffix automaton (+DP?)

– Aho-Corasick

– eerTree

– Work with S + S

• Hashing

• Euler tour, tree to array

• Segment trees

– Lazy propagation

– Persistent

– Implicit

– Segment tree of X

• Geometry

– Minkowski sum (of convex sets)

– Rotating calipers

– Sweep line (horizontally or vertically?)

– Sweep angle

– Convex hull

• Fix a parameter (possibly the answer).

• Are there few distinct values?

• Binary search

• Sliding Window (+ Monotonic Queue)

• Computing a Convolution? Fast Fourier Transform

• Exact Cover (+ Algorithm X)

• Cycle-Finding

• What is the smallest set of values that identify the solution?

The cycle structure of the permutation? The powers of primes

in the factorization?

• Look at the complement problem

– Minimize something instead of maximizing

• Immediately enforce necessary conditions. (All values greater

than 0? Initialize them all to 1)

• Add large constant to negative numbers to make them positive

• Counting/Bucket sort

9. Debugging Tips

• Stack overflow? Recursive DFS on tree that is actually a long

path?

• Rounding negative numbers?

• Double

• Wrong Answer?

– Quitar el freopen,

– no mezclar cin con scanf

– Ver si hay que imprimir fin de linea

– Leer nuevamente el problema.

– Ver si es multiple casos, repetir el mismo caso varias

veces.

– long long

– Posibles Casos:

\* n = 0,n = -1,n = 1,n = 2^31 - 1 or n = -2^31

\* La lista esta vacia o con un solo elemento

\* n is even, n is odd

\* El Grafo esta vacion o contiene un solo vertice

\* El Grafo es un multigrafo (lazo o multiple aristas)

\* El Polygono es convexo o no

–Hay condicion inicial para los c

Josephus

int J( int n, int k ){

if( n == 1 )return 0;

return (J(n-1,k)+k)%n;

}

int main(){

int n, k;

cin >> n >> k;

cout << J(n,k)+1;

}

Minimo costo de a-b usando n aristas

//calcular en un grafo el menor costo de ir de un nodo A a un nodo B usando exactamente N aristas

//idea

//construir una matrix de adyacencia poniendo el costo de la arista y si no existe poniendo infinito

//modificar la multiplicacion de matrices por lo siguiente

//AB[i][j] = inf; N

//entonces AB[i][j] = min( AB[i][j], min( A[i][k]+B[k][j] ) )

// k=1

Trios Pitagoricos

trios pitagoricos primitivos son a,b,c tq gcd(a,b,c) == 1 y a\*a + b\*b == c\*c

y se cumple entonces que todo trio(ka, kb, kc) tambien es pitagorico

todos los trios pitagoricos primitivos son de la siguiente forma (n^2 - m^2, 2\*n\*m, n^2 + m^2 ) donde

0 < m < n y n y m son coprimos y al menos uno de ellos es par ejemplo m = 1, n = 2

( 2^2 - 1^2, 2\*2\*1, 2^2 + 1^2 ) = (3,4,5)