Informe
Matemática Discreta I
D. Same GCD's
Problema:1295D

Abel Molina Sánchez
Grupo 2-11
Ciencias de la Computación
Universidad de La Habana

26 de diciembre de 2020

1. Problema

D. Same GCD's

You are given two integers a and m. Calculate the number of integers x such that $0 \leq x < m$ and $gcd(a, m) = gcd(a + x, m)$.
Note: gcd(a,b) is the greatest common divisor of a and b.

Input:
The first line contains the single integer T ($1 \leq T \leq 50$) — the number of test cases.
Next T lines contain test cases — one per line.
Each line contains two integers a and m ($1 \leq a < m \leq 10^{10}$).

Output:
Print T integers — one per test case. For each test case print the number of appropriate x-s.

2. Insterpretación e idea

El problema plantea de forma directa que se necesita determinar la cantidad de valores x, $0 \leq x < m$ que cumplen que $mcd(a, m) = mcd(a + x, m)$.
Para determinar esta solución la primera idea de solución pasa por una iteración de fuerza bruta por todos los valores de x en cada caso y para cada uno determinar si se cumple la igualdad de los mcd's utilizando el algoritmo de Euclides. Esta solución intuitiva no pasa por ser muy eficiente, con lo cual, la siguiente idea pasa por demostrar que la cantida de valores x que cumplen la condición es igual a $\phi(m')$, donde $m' = m/mcd(a, m)$. Luego hay entonces que calcular $\phi(m')$ y para esto utilizaremos su fórmula en base a la descomposición en factores primos de m'.

Lema1: *Sean* $a, b \in N_+$ $a = da'$, $b = db'$. $d = mcd(a, b) \Rightarrow mcd(a', b') = 1$.
Demostración: Tenemos $a = da'$, $b = db'$, sea $d = mcd(a, b)$ y $k = mcd(a', b')$, supongamos que $k \neq 1$. Luego tenemos que $a = dka$" y $b = dkb$", con lo cual dk es divisor común de a y b, y $dk > d$ lo cual contradice que d sea el $mcd(a, b)$.

Lema2: *Sean* $a, b, m \in N_+$ $mcd(ma, mb) = m * mcd(a, b)$.
Demostración: Sea $d = mcd(a, b)$, tenemos que:
$ax + by = d$ (para $x, y \in Z$),
$max + mby = md$
$x(ma) + y(mb) = md$
$mcd(ma, mb) = m * mcd(a, b)$

Lema3: *Sean* $a, b, \in N_+$, $a|geqb$ $mcd(a, b) = mcd(a - b, b)$.
Demostración: Sea d un divisor común a y b.
Luego se tiene: $a = da'$, $b = db'$ y por lo tanto $a - b = da' - db' = d(a' - b')d$
Esto implica que d es un divisor común de $(a - b)$ y b.
Ahora en sentido contrario tengamos que d es un divisor común de $(a - b)$ y b.
Entonces:
$a - b = dp$, $b = dq$, con lo cual

$a = (a - b) + b = dp + dq = d(p + q)$,
con lo cual d es un divisor común de a y b.
Así, los pares (a, b) y $(a - b, b)$ tienen los mismos divisores comunes, en particular el *mcd*.

Lema4: *Sea $a \in N_+$, $\{r_1, r_2, ..., r_n\}$ un Sistema Residual Completo módulo n, $\{r_1 + a, r_2 + a, ..., r_n + a\}$ es también un Sistema Residual Completo módulo n.*
Demostración: Supongamos que existen índices $1 \leq i, j \leq n$ tales que:
$r_i + a \equiv r_j + a(n)$
entonces, n divide a $(r_i + a) - (r_j + a) = r_i - r_j$, con lo cual n divide a $r_i - r_j$ lo que implica que: $r_i \equiv r_j(n)$, lo cual es una contradicción con el hecho de que $\{r_1, r_2, .., r_n\}$ es un Sistema Residual Completo módulo n.

Se busca determinar la cantidad de valores $1 \leq x < m$ que cumplen que $mcd(a, m) = mcd(a+x, m)$. Luego tenemos que buscar una forma de determinar dicha cantidad. Para ello analizaremos las condiciones de los valores. Tenemos que los x_i forman el menor SRC(m), y tenemos que $a+x \leq m$ o $a+x > m$. Si $a+x > m$, tenemos que se cumple que $mcd(a+x, m) = mcd(a+x-m, m)$, luego uniendo estos resultados tenemos que dicha cantidad será el número de los $x' = (x+a)mod(n)$, $0 \leq x \leq m - 1$, tal mcd(x',m)=mcd(a,m).
Luego sea $d = mcd(a, m)$, tenemos que:
$a = da'$, $m = dm'$, con $mcd(a', m') = 1$.
Como se busca $mcd(a, m) = mcd(x', m)$, entonces, los números cumplirán que:
$d = mcd(x', m)$, lo que implica que $x' = dx$" y como hemos visto $m = dm'$, donde $mcd(x", m') = 1$.
Luego tenemos que:
$m = dm'$, $x' = dx$", y $1 \leq x' < m$, de donde $x' < m$, con lo cual, como $d = d$, $0 \leq x" < m'$.
luego, la cantidad de números sería la cantidad de los $x"$ menores m' primos relativos con m'. Pero esto es prácticamente por definición el valor de $\phi(m')$, faltaría demostrar que nunca se alcanzará la igualdad para m', y esto es que $m' > 1$, y esto se ve en el hecho de que $1 \leq a < m$, con lo cual $d = mcd(a, m) < m$ y por tanto $m/d = m' > 1$.
Luego hemos determinado que la cantidad bucada es igual a $\phi(m')$.

3. Pseudocódigo

```
solve(a,m):
    d = mcd(a,m)
    return phi(m/d)

phi(m):
    phi = m
    div = 2
    while div*div <= m:
        if m%div==0:
            while m%div==0:
                m/= div
            phi -= phi/div
        div++
    if m > 1:
        phi -= phi/m
    return phi
```

Tenemos que, sea $n \in N$, $\phi(n) = n \prod_{i=1}^{k}(1 - \frac{1}{p_i})$, con lo cual necesitamos analizar que se detectan los factores primos de m en la ejecución.
Sea $n \in N_+$, n tiene a lo sumo un factor primo mayor que $\sqrt{n}$.
Supongamos que tiene al menos dos, sean p y q . Se tiene que $p > \sqrt{n}$ y $q > \sqrt{n}$, con lo cual $p * q > \sqrt{n} * \sqrt{n}$, con lo cual $p * q > n$, lo cual contradice que p y q sean factores primos de n.

Luego, tenemos que los divisores de cualquier número n compuesto son $1 \leq d_i \leq n$. Quitando los divisores triviales, se cumple que serán: $1 < d_i < n$. Luego tenemos que ningún número compuesto cumplirá la condición $m\,\%div == 0$, ya que antes habrían sido analizados sus divisores, y los divisores de un número a son divisores de $a * k\ \forall k \in N$. Por tanto tenemos que se detectan los factores primos de m durante la iteración del ciclo, excepto a lo sumo uno, esto es, que haya uno mayor que $\sqrt{n}$.

Veamos que en caso de poseerlo, este factor se incluye en la condición if final. Digamos que el número n posee un factor mayor que $\sqrt{n}$, sea ahora $n = \prod_{i=1}^{j} p_i^{\alpha_i}$ la descomposición en primos de n. Asumamos que los p_i están ordenados de menor a mayor en la productoria(el orden de los factores no altera el producto). Luego tendremos que $p_j > \sqrt{n}$, tenemos que $\alpha_j = 1$ porque de lo contrario tendríamos al menos $p_j^2 > n$. Luego, sea $q = \prod_{i=1}^{j-1} p_i^{\alpha_i}$, tenemos que $q|n$, $q * p_j = n$, luego $n/q = p_j$, y como en el ciclo se analizan los p_i, $i \in [1, ..., j-1]$, entonces tenemos que en la condición if final detectamos a $p_j > \sqrt{n}$.
Por tanto luego de terminada la ejecución del algoritmo y por la fórmula de $\phi(m)$, habremos determinado el valor de ϕ para la entrada.
En el caso peor cada factor primo aparecería en la descomposición con exponente $\alpha_i = 1$, con lo cual se completa un ciclo hasta la iteración $i = \sqrt{m}$, con lo cual la complejidad temporal de $phi(m)$

es $O(\sqrt{m})$.
Luego, para analizar la complejidad temporal de la solución de cada test, tenemos que la complejidad de determinar el mcd(a,m) por el algoritmo de Euclides será $O(log(m))$, y la complejidad temporal de *phi* será en $O(\sqrt{m'})$, donde $m' = m/mcd(a, m)$, luego la complejidad de la solución de cada test será de $O(logm+\sqrt{m'})$ de donde terminar completa la ejecución será $O(T*(logm+\sqrt{m'}))$.

4. Tester

Para la realización del tester, se utiliza la solución de fuerza bruta del problema para determinar la solución esperada para cada entrada. En la carpeta *./test* dentro de *./test-casos-prueba* se deben encontrar por cada caso de prueba 3 archivos: casoX.in, casoXexpected.out y casoX.out; X identifica el número del caso, el .in contiene los valores de entrada del caso, ..expected.out debe contener la solución dada por el algoritmo de fuerza bruta(incluido dentro de la carpeta tester), y el casoX.out debe tener la solución dada por el algoritmo de solución.
Para generar las entradas de los casos se creó el *generator.py* en el cual se da la opción de crear un caso de prueba de forma manual o generar uno aleatorio utilizando el método *randint* de la librería *random* de *python*.
Para comprobar la soluciones se debe ejecutar el script *checker.py* el cual por cada caso de prueba determina que la solución ofrecida por el algoritmo coincide con la esperada y por lo tanto es solución. Hay que tener en cuenta que *./test* debe incluir los 3 archivos de cada caso, y que *checker.py* analizará todos los casos incluidos en *./test*. En el caso de prueba número 9 se utilizan como entrada del algoritmo los 30 primeros números de fibonacci. Los casos del 2 al 8 son tomados del Codeforce.