Informe
Matemática Discreta I
Two Divisors
Problema:1366D

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

26 de diciembre de 2020

1. Problema

D.Two Divisors

You are given n integers $a_1, a_2, \ldots, a_n$.
For each a_i find its two divisors $d1 > 1$ and $d2 > 1$ such that $gcd(d1 + d2, ai) = 1$ (where $gcd(a, b)$ is the greatest common divisor of a and b) or say that there is no such pair.

Input:
The first line contains single integer $n(1 \leq n \leq 5 * 10^5)$ — the size of the array a. The second line contains n integers $a_1, a_2, \ldots, a_n$ $(2 \leq a_i \leq 10^7)$ — the array a.

Output:
To speed up the output, print two lines with n integers in each line.
The $i-th$ integers in the first and second lines should be corresponding divisors $d1 > 1$ and $d2 > 1$ such that $gcd(d1 + d2, ai) = 1$ or -1 and -1 if there is no such pair. If there are multiple answers, print any of them.

2. Interpretación e Idea

El problema plantea el hecho de encontrar para cada número a_i si posee al menos dos divisores $d1$, $d2$ que cumplen que $mcd(d1 + d2, a_i) = 1$.
La primera idea de solución y a modo de fuerza bruta sería determinar todos los divisores de a_i y luego determinar dos a dos si cumplen que $mcd(d_i + d_j, a_i) = 1$ utilizando el algoritmo de Euclides para el cálculo del mcd.
Luego en busca de mejorar la eficiencia de la solución vamos a analizar la descomposición en factores primos de a_i, para determinar que para cuando $a_i = p_i^{\alpha_i}$ con p_i primo, no existirán tales divisores; y en caso contrario vamos a demostrar que escogiendo un $d1 = p_j$, p_j factor primo de a_i, y $d_2 = a_i/p_j^{\alpha_j}$, tendremos que $d1$ y $d2$ son divisores que cumplen el criterio.
Luego a la hora de la implementación apoyandonos en el algoritmo de la criba de Eratosthenes vamos a predeterminar para cada valor posible de la entrada su menor factor primo. Luego una vez tenido este, lo tomamos como $d1$ y de la forma descrita anteriormente obtenemos $d2$.

3. Demostración

Teorema1: *Sea* $n = \prod_{i=1}^{k} p_i^{\alpha_i}$ *la descomposición en factores primos de* $n \in N_+$, *si* $k = 1$, *no existen dos divisores* $d1$, $d2$ *de* n *que cumplen que* $mcd(d1 + d2, n) = 1$.
Demostración: Como $k = 1$, $n = p_1^{\alpha_1}$, luego tendremos que todo dividsor q de n es divisible por p_1, porque, sea un q', $q'|n$, si q' es primo $q' = p_1$, luego no puede ser primo, entonces, q' compuesto y sea, q" el divisor no trivial más pequeño de q', tenemos que q" es primo, y q"$|n$ entonces q" $= p_1$.
Luego sean dos divisores cualesquiera de n, $d1$ y $d2$, $d1 = p_1 * d1'$, $d2 = p_1 * d2'$, luego,
$d1 + d2 = p_1(d1' + d2')$ y por tanto $p_1|d1 + d2$. Con lo cual queda demostrado.

Teorema2: *Sea* $p_1^{\alpha_1}p_2^{\alpha_2}....p_k^{\alpha_k}$ *la descomposición en factores primos de* $n \in N_+$*, si* $k > 1$*, existen dos divisores* $d1$*,* $d2$ *de* n *que cumplen que* $mcd(d1 + d2, n) = 1$*.*
Demostración: Sea $n = p_1^{\alpha_1}p_2^{\alpha_2}....p_k^{\alpha_k}$, $k > 1$,
$n = \prod_{i=1}^{k} p_i^{\alpha_i}$,
sea p_j arbitrario, uno de los factores primos de n, tenemos:
$n = p_j^{\alpha_j} * \prod_{i \neq j}^{k} p_i^{\alpha_i}$
sea $q_j = \prod_{i \neq j}^{k} p_i^{\alpha_i}$
$n = p_j^{\alpha_j} * q_j$
Sea ahora $p = p_i, i \in [1, .., k]$ arbitrario, uno de los factores primos de n. tenemos que $p_j \nmid q_j$, porque sabemos que si $p_j|q_j \Rightarrow p_j|p_i$ para un $i \neq j \in [1, .., k]$, lo cual sería una contradicción con el hecho de que los p_i son primos.
a su vez, $q_j \nmid p_j$, porque entonces $q_j = p_j$, no puede ser 1 porque $k > 1$, y por tanto los $p_i \in q_j$ dividen a p_j, lo cual es una contradicción con que p_j es primo.
Por tanto tenemos que p divide a uno entre p_j y q_j, si $p|p_j$, $p = p_j$ y $p_j \nmid q_j$; y si $p \nmid p_j$, $p = p_i \neq p_j$, por tanto $p|q_j$. Sin pérdida de generalidad digamos que $p|q_j$.
Luego tenemos:
$q_j \equiv 0(p)$
$p_j \equiv r(p), r \neq 0$
$p_j + q_j \equiv r + 0(p)$
$p_j + q_j \equiv r(p), r \neq 0$
Luego $p \nmid p_j + q_j$ [1]
Luego tenemos que ningún p_i divide a $p_j + q_j$.
Luego supongamos que $mcd(p_j + q_j, n) = d > 1$.
$d|n$ y $d|p_j + q_j$,
Si d es primo es una contradicción con [1], por tanto analicemos para cuando d es compuesto.
sea ahora p' el menor divisor no trivial de d, sabemos que p' es primo.
$p'|d$, luego $p'|n$ y $p'|p_j+q_j$, pero entonces p' es un factor primo de n, con lo cual se contradice [1]. Luego, hemos encontrado dos divisores, $d1 = p_j$ y $d2 = q_j$ de n que cumplen que $mcd(d1 + d2, n) = 1$, y a su vez hemos determinado una forma de hallarlos teniendo en cuenta la elección arbitraria de p_j.

4. Algoritmo y complejidad temporal

```
solve(n,a):
    N = 10**7+1
    min_prime = [0]*N
    sieve_e(N)
    d1,d2 = [-1]*n,[-1]*n
    for i in [0,...,n-1]:
        p = min_prime[ai]
        if ai is prime:
            continue
        while p | ai:
            ai = ai/p
        if ai > 1
            d1[i],d2[i] = p,ai
     return d1,d2
```

```
|sieve_e(N):
|    p = 2
|    while p*p <= N:
|        if min_prime[p]:
|            continue
|        for i = p*p; i < N; i+=p:
|            if not min_prime[i]:
|                min_prime[i] = p
|        p++
```

Veamos que la criba nos coloca en $min - prime[i]$ el valor del menor factor primo de i o 0 si i es primo. En un principio todos los números en el rango hasta N se marcan como primos, ahora tenemos que, los divisores de un número compuesto c son $\leq c$. En particular sus factores primos. A su vez tenemos que cualquir número compuesto n tiene al menos un factor primo $\leq \sqrt{n}$. Para demostrarlo supongamos lo contrario, esto es, tengamos el número n con factorización en primos $n = p_1 p_2 ... p_r$.
Supongamos que $p_i > \sqrt{n}\ \forall i = 1, .., r$. Entonces tendríamos que :
$p_1 * p_2 .... * p_r > (\sqrt{n})^r = n^{r/2} \geq n$ si $r \geq 2$ y tenemos que n es compuesto con lo cual $r \geq 2$ y llegamos a una contradicción.
Luego se tiene que se alcanzan los primos $\leq N$ en el ciclo hasta $\sqrt{N}$ y para cada $n < N$ sabemos que $\sqrt{n} < \sqrt{N}$. Veamos entonces que para cada valor compuesto se actualiza este valor, y esto ocurre en el ciclo interno. Para eso necesitamos ver que luego de determinar los múltiplos de los primos $2, 3, .., P_k$, tenemos que los números compuestos que quedan sin analizar tienen factores solamente mayores que P_k, de lo contrario habrían sido analizados como múltiplos de un $p_i \leq P_k$. A su vez se tiene que el menor de estos compuestos es $(P_{k+1})^2$, ya que todos los múltiplos de P_{k+1} fueron analizados con los múltiplos de los $p_i \leq P_k$. A su vez quedarán en 0 los primos en el rango P_k, P_{k+1} ya que no tienen divisores no triviales por lo cual no serán múltiplo de ningún p_i. Para finalizar es claro que hasta $\sqrt{N}$ se alcanzan todos los compuestos ya que a lo sumo $P_k = \sqrt{N}$ y se tendría que $(P_k)^2 = N$. Con lo cual luego de finalizado se habrán analizado todos los números compuestos en el rango hasta N y los primos quedarán en 0.
Luego se elige $d1 = min - prime[i]$ para cada a_i y garantizamos en el último ciclo while que se tenga $d2 = ai/p^\alpha$ que será solución si $a_i \neq p^\alpha$ por lo demostrado en los teoremas 1 y 2.

Para el análisis temporal de la solución tenemos que el algoritmo de la criba de Eratosthenes es $O(NloglogN)$[1], donde tenemos que N es el límite superior de la entrada del algoritmo que será $a_i \leq 10^7$. Luego, tenemos que obtener $d1 = p$ se hace en $O(1)$. En el segundo ciclo para la búsqueda de $d2$ tenemos el peor caso cuando a_i es de la forma p_i^k y tenemos que se realiza la mayor

[1]Time-complexity-of-Sieve-Eratosthenes-GeeksforGeeks.pdf en la bibliografía, tomado del sitio *www.geeksforgeeks.org*

cantidad de iteraciones cuando es de la forma 2^k. Luego tenemos que determinar la solución para los n a_i es $O(nlogN)$. Luego las inicializaciones son $O(N)$ y $O(n)$ al igual que devolver la solución que será $O(n)$. Luego tenemos que complejidad temporal de la solución será $O(NloglogN+nlogN)$.

5. Tester

Se incluye un *generador.py* para generar casos de forma aleatoria y de forma manual. Para analizar en los casos en la carpeta *./tests* se debe encontrar el archivo *casoX.in* y el archivo *casoX.out*, este último debe contener la respuesta dada por el algoritmo a la entrada en el .in.
Para analizar la correctitud de las soluciones se incluye el script *checker.py* que analiza cada caso de prueba que se encuentra en *./tests*.
Para determinar la correctitud de la solución, en primer lugar se leen las listas de entrada a, y las listas de solución $d1$ y $d2$ del algoritmo.
Por cada caso se itera por las listas, si se tiene que los divisores para a_i son distintos de -1, se comprueba, empleando el método $gdc(a, b)$ de la librería *math* de *python* si el $mcd(d1 + d2, a_i) \neq 1$ y que $d1|a_i$ y $d2|a_i$. En caso de que la solución dada por el algoritmo sea que no existen $d1$ y $d2$ para a_i, entonces se utiliza la solución de fuerza bruta para determinar que esto es correcto. Para ello se tiene el método *brute-force* que crea una lista para almacenar los divisores no triviales de a_i en cada caso. Para esto, se itera desde $div = 2$ mientras $div \leq \sqrt{a_i}$ y si $div|a_i$ se agrega a lista de divisores, y si $div \neq \sqrt{a_i}$, se agrega $div2 = a_i/div$ a la lista. Luego se comprueba que el $mcd(div_i + div_j, a_i) \neq 1$ para cualquier par de divisores (div_i, div_j).
Se incluyen algunos casos de pruebas, como por ejemplo el *caso4* donde se analiza el algoritmo para todos los números en el rango $[1, 5 * 10^5]$[2]

[2]Probablemente al momento de recibir el informe haya 2 o 3 casos solamente pues los casos grandes generan ficheros de mucho peso para el envío por correo. La idea inicial fue analizar todos los rangos de 500000 naturales desde 1 hasta 10^7 y se cubrirían todas las posibles entradas del problema.