

DOCUMENTO TÉCNICO DIVIDIR Y CONQUISTAR

Requisitos

Especificación 1

1. entrada n
2. imprimir n
3. si $n = 1$ entonces STOP
4. si n es impar entonces $n \rightarrow 3n + 1$
5. de lo contrario $n \rightarrow n/2$
6. GOTO 2 Dada la entrada 22, se imprimirá la siguiente secuencia de números:

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1.

Se conjetura que el algoritmo anterior terminará (cuando se imprima un 1) para cualquier valor de entrada integral. A pesar de la simplicidad del algoritmo, se desconoce si esta conjetura es verdadera. Sin embargo, se verifica para todos los enteros n tales que $0 < n < 1.000.000$.

Dada una entrada n , es posible determinar la cantidad de números impresos antes e incluyendo el 1 impreso. Para un n dado, esto se llama la longitud del ciclo de n . En el ejemplo anterior, la longitud del ciclo de 22 es 16. Para cualesquiera dos números i y j , determinar la longitud máxima del ciclo sobre todos los números entre e incluyendo tanto i como j .

Entrada

La entrada consistirá en una serie de pares de enteros i y j , un par de enteros por línea. Todos los enteros serán menores de 10,000 y mayores que 0. Debes procesar todos los pares de enteros y para cada par determinar la longitud máxima del ciclo sobre todos los enteros entre e incluyendo i y j . Puedes asumir que ninguna operación desborda un entero de 32 bits.

Salida

Para cada par de enteros de entrada i y j , debes imprimir i , j , y la longitud máxima del ciclo para los enteros entre e incluyendo i y j . Estos tres números deben estar separados por al menos un espacio con los tres números en una línea y con una línea de salida para cada línea de entrada. Los enteros i y j deben aparecer en la salida en el mismo orden en que aparecieron en la entrada y deben ser seguidos por la longitud máxima del ciclo (en la misma línea).

Especificación 2

Encuentra el número de formas en que un número entero dado, X , puede expresarse como la suma de las potencias N^{th} de números naturales únicos. Por ejemplo, si $X=3$ y $N=2$, tenemos que encontrar todas las combinaciones de cuadrados únicos que suman 13. La única solución es $2^2 + 3^2$.

Entrada

La primera línea contiene un entero X , donde $1 \leq X \leq 1000$

La segunda línea contiene un entero N , donde $2 \leq N \leq 10$

Salida

Salida de un solo entero, el número de combinaciones posibles calculadas con el número y la potencia respectiva.

Diseño

Estrategia1

Se establece la función a trozos del programa viendo todas las condiciones, donde la primera función es el problema en general ($3n1$), la segunda ($\#3n1$) que va a contar la cantidad de veces que se ejecuta el valor n . Las demás funciones ($3n1p$ y $3n1M$) se generan para almacenar grande información, es decir, cuando el n y m son grandes, y la cantidad de veces es alta, entonces recurre y guarda cálculos anteriores para que se use y no vuelva a repetir el cálculo.

Se aplica "Dividir y conquistar" en árboles de decisión para tomar el número de decisiones en que $3n1$ se repite para observar como se define la función a trozos. También para ver como se comporta esa función para las condiciones establecidas.

3N+1

La mejor sucesión entre N y M

$$3n1(n) = \begin{cases} n & , n=1 \\ n \cdot 3n1(n/2) & , n \% 2 = 0 \\ n \cdot 3n1(3n+1) & , n \% 2 \neq 0 \end{cases}$$

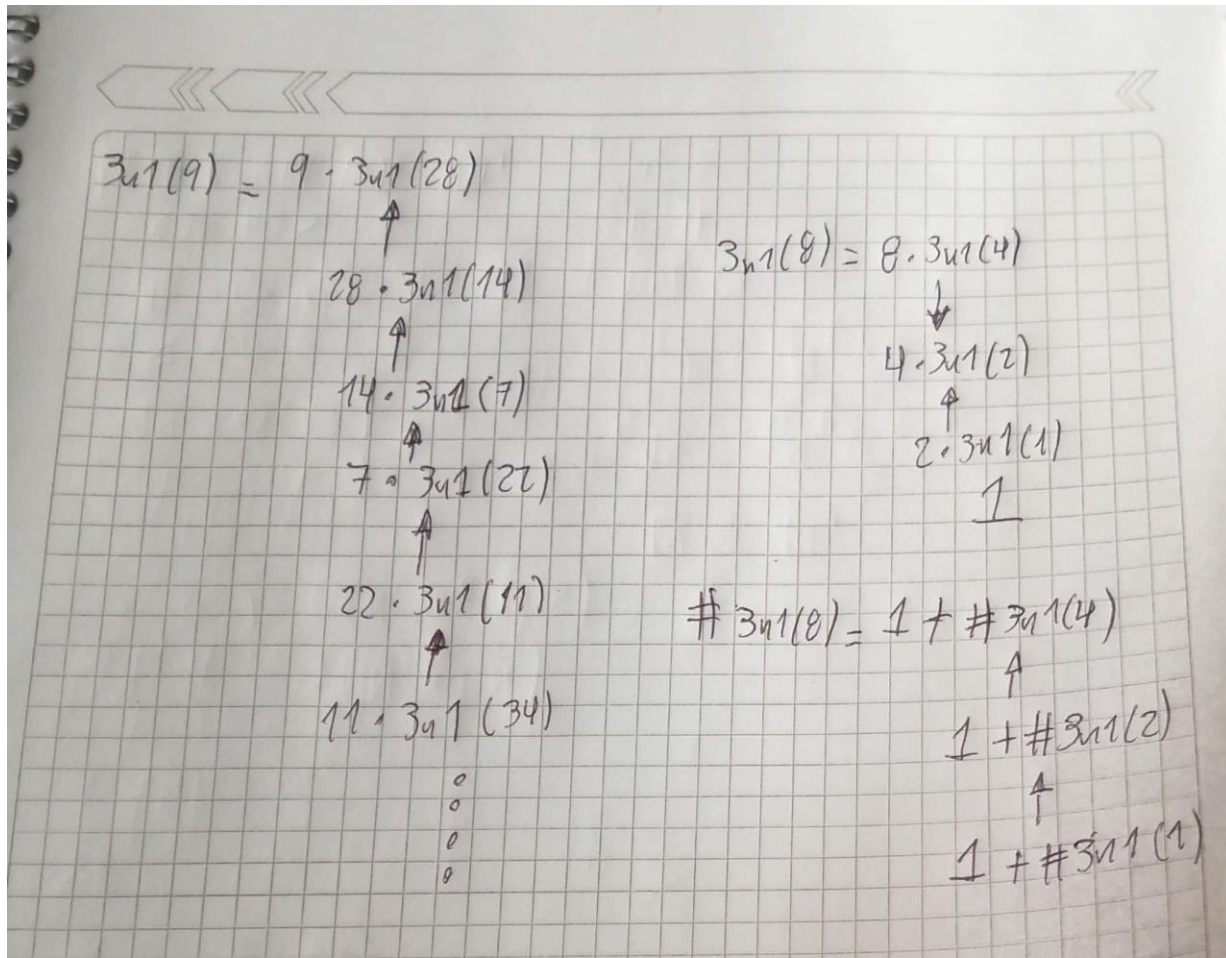
$$\#3n1(n) = \begin{cases} 1 & , n=1 \\ 1 + \#3n1(n/2) & , n \% 2 = 0 \\ 1 + \#3n1(3n+1) & , n \% 2 \neq 0 \end{cases}$$

$$\#3n1P(n, M) = \begin{cases} 1 & , n=1 \\ 1 + \#3n1P(n/2, M) & , n \% 2 = 0 \\ 1 + \#3n1P(3n+1, M) & , n \% 2 \neq 0 \end{cases}$$

$$\#3n1M(n, M) = \begin{cases} M[n] & , [n] \in M \\ M[n] = \#3n1P(n, M) & , [n] \notin M \\ M[n] & \end{cases}$$

5. 5 16 8 4 2 1

$$Max3n1(M, M) = \sum_{i=1}^M \#3n1(i, M)$$



Estrategia2

Se establece la función a trozos del programa viendo todas las condiciones, donde la función es el problema en general (PSum). Las demás funciones (P'Sum y PSumM) se generan para almacenar grande información, es decir, cuando el número n y la potencia k pueden tomar valores grandes. Este ejercicio se planteó en términos anteriores, también, en el árbol de decisión se detalla que cada término necesita el anterior hasta el punto en el que no haya decisión y donde $T(\text{total})$ va a ser el encargado en decir de cuántas formas se puede sumar el número a la potencia.

Se aplica "Dividir y conquistar" en árboles de decisión para tomar el número de decisiones en que PSum se repite para observar cómo se define la función a trozos. También para ver cómo se comporta esa función para las condiciones establecidas.

Power Sum

$L = \text{Potencia}$
 $T = \text{total}$

$$T = (X_0)^k + (X_1)^k + \dots + (X_n)^k$$

$0 < X_i < \sqrt[k]{T}$

Árbol de decisión

$T = (1)^k + \dots$

$T = (-)^k + \dots$

T_0

$T_1 = T_0 - (1)^k$

$T_1 = T_0$

$T_2 = T_1 - (2)^k$

$T_2 = T_1$

$T_2 = T_1 - (2)^k$

$T_2 = T_1$

1

1

0

0

$(T \neq 0)$

$(T = n^k)$

$T < 0$

$T < (n^k)$

1 2 3 4 ... 5

$n = 1$

Variable Variable

$P_{sum}(T, n, k) =$

1

0

$T = 0 \vee T = n^k$

$T < 0 \vee T < n^k$

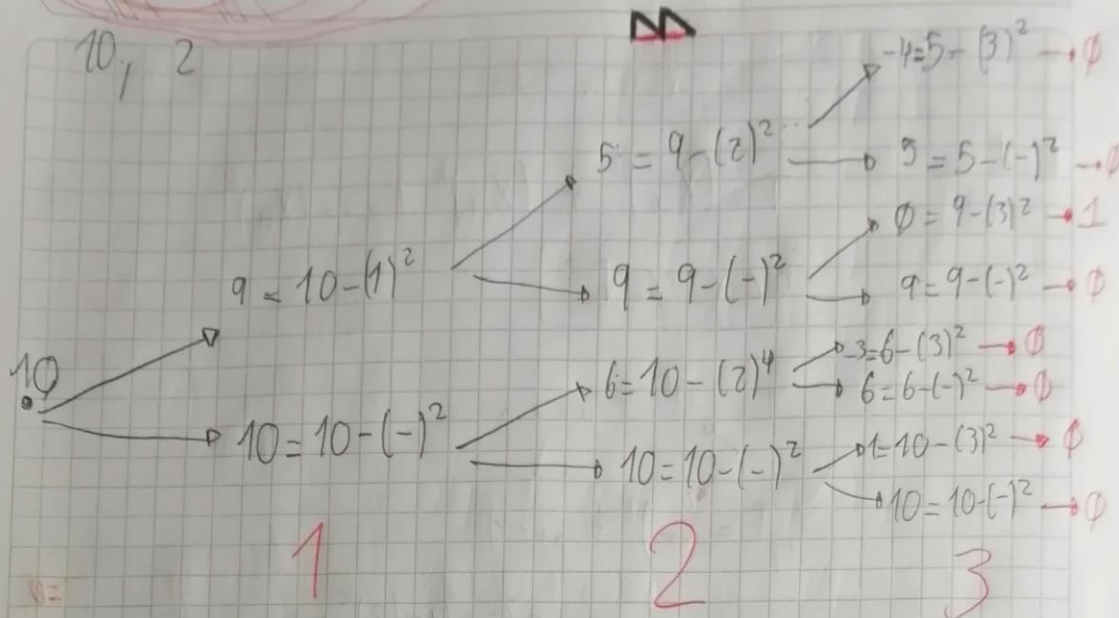
1 2 3 4 ... 5

↑

$P_{sum}(T - (n)^k, n+1, k) + P_{sum}(T, n+1, k) \quad T > 0 \wedge T > n^k$

$$T = (x_0)^k + \dots + (x_n)^k$$

10, 2



$$P_{\text{SUM}}(T, n, k, M) = \begin{cases} 1 & T = \emptyset \vee T \leq n^k \\ 0 & T \not\leq \emptyset \vee T \not\leq n^k \end{cases}$$

$$P_{\text{sum}}(T(n)^k, n+1, k, M) + P_{\text{sum}}(T, n+1, k, M) \quad T > 0$$

A $T(n)^k$

$$P_{\text{sum}} = \begin{cases} M[(T, n, k)] & (T, n, k) \text{ in } M \\ M[(T, n, k)] = P_{\text{sum}}(T, n, k, M), M[(T, n, k)] & (T, n, k) \text{ not in } M \end{cases}$$

3000 10000 10000
3000 3000
4000

Casos de prueba 1

Entrada	Justificación	Salida
1 10	Cualesquiera valores	1 10 20
100 200	Cualesquiera valores	100 200 125
201 210	Cualesquiera valores	201 210 89
1 1	Caso base	1 1 1
300 999	Cualesquiera valores	300 999 179
900 1000	Cualesquiera valores	900 1000 174

The screenshot shows a code editor with two files open: `3N+1.py` and `salida3N+1.txt`. The `3N+1.py` file contains the following input data:

```

1 1 10
2 100 200
3 201 210
4 1 1
5 300 999
6 900 1000

```

The `salida3N+1.txt` file contains the following output data:

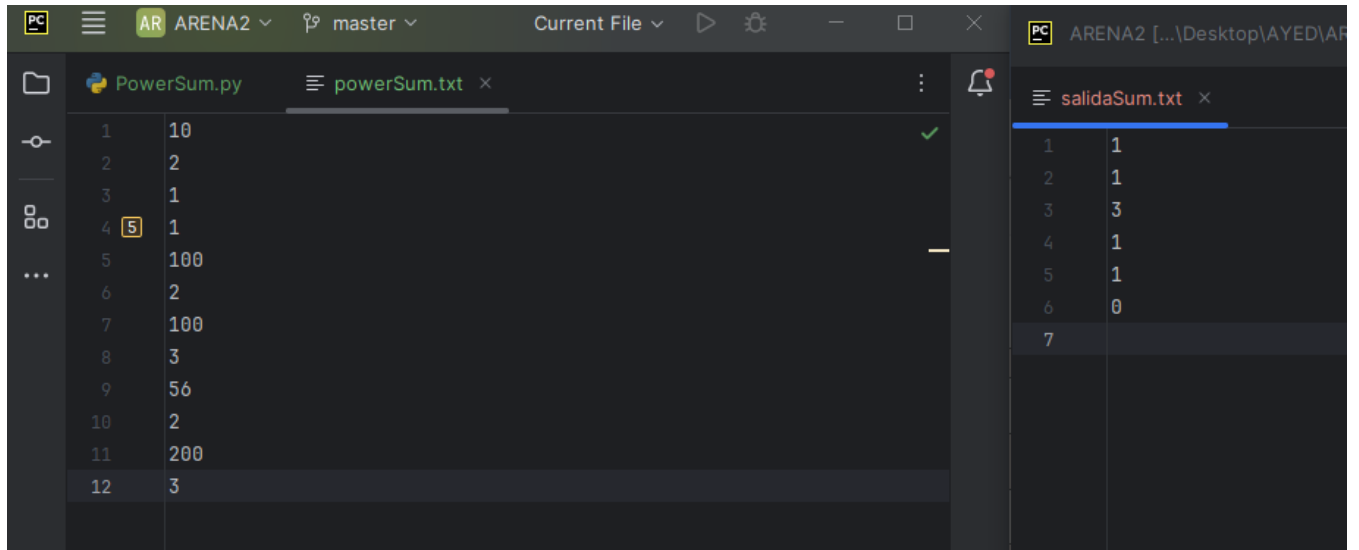
```

1 1 10 20
2 100 200 125
3 201 210 89
4 1 1 1
5 300 999 179
6 900 1000 174
7

```

Casos de prueba 2

Entrada	Justificación	Salida
10 2	Cualesquiera valores	1
1 1	Caso base	1
100 2	Cualesquiera valores	3
100 3	Cualesquiera valores	1
56 2	Cualesquiera valores	1
200 3	Cualesquiera valores	0



```
PC AR ARENA2 master Current File
PowerSum.py powerSum.txt
1 10
2 2
3 1
4 1
5 100
6 2
7 100
8 3
9 56
10 2
11 200
12 3

PC ARENA2 [...]Desktop\AYED\AF
salidaSum.txt
1 1
2 1
3 3
4 1
5 1
6 0
7
```

Análisis

Temporal

Para los dos programas como el método para resolverlo es con árbol de expansión, y las funciones que lo definen no son fraccionarios, entonces, la manera de calcular la complejidad de los programas es por observación (# de decisiones) que en este caso se puede ver que es 2^n .

Código1

```
1 from sys import stdin #Biblioteca para leer entradas estándar
2
3 memo = {}
4
5 2 usages new *
6 def numTNOne(n): #Función que describe 3N+1 en general
7     if n==1:
8         return 1
9     if n%2 == 0:
10        return 1 + numTNOne(n // 2)
11    return 1 + numTNOne(3 * n + 1)
12
13 2 usages new *
14 def numTNOneP(n, M): #Función que servirá de soporte para almacenar información y después guardarlo en la memoria
15     if n==1:
16         return 1
17     if n%2 == 0:
18         return 1 + numTNOneMemo(n // 2, M)
19     return 1 + numTNOneMemo(3 * n + 1, M)
20
21 2 usages new *
22 def numTNOneMemo(n, M): #Función memoria
23     if n in M.keys():
24         return M[n]
25     M[n] = numTNOneP(n, M)
26     return M[n]
```

```
23
24 1 usage new *
25 def map_numTNOneMemo(n): #Función que recurrirá a guardar los datos en memo, que se creó como diccionario
26     if n in memo.keys():
27         return memo[n]
28     memo[n] = numTNOneP(n, memo)
29     return memo[n]
30
31 1 usage new *
32 def maxNumTNOne(n,m): #Retorna el máximo valor de hacer los cálculos del número n y m
33     return max(map(map_numTNOneMemo, range(n,m+1)))
34
35 1 usage new *
36 def main():
37     line = stdin.readline().strip() #Leer la línea y quita los espacios para poner la entrada estándar
38     while line:
39         n,m = map(int, line.split()) #Poner n,m en una sola línea separados por espacio
40         print(n,m, maxNumTNOne(min(n,m), max(n,m)))
41         line = stdin.readline().strip() #Para que no quede en un bucle
42
43 main()
```

Código2

```
1 import sys #Libreria para entradas estándar
2 3 usages new *
3 def pSum(total, n_termino, potencia): #Power Sum en general con las condiciones
4     n_potenciado = n_termino ** potencia
5     if total == 0 or total == n_potenciado:
6         return 1
7     if total < 0 or total < n_potenciado:
8         return 0
9     return (pSum(total - n_potenciado, n_termino+1, potencia) +
10            pSum(total, n_termino+1, potencia))
11
12 1 usage new *
13 def pSumP(total, n_termino, potencia, M = {}): #PowerSum de almacenamiento de información en la memoria
14     n_potenciado = n_termino ** potencia
15     if total == 0 or total == n_potenciado:
16         return 1
17     if total < 0 or total < n_potenciado:
18         return 0
19     return (pSumM(total - n_potenciado, n_termino+1, potencia, M) +
20            pSumM(total, n_termino+1, potencia, M))
21
22 2 usages new *
23 def pSumM(total, n_termino, potencia, M = {}): #PowerSum memoria para no repetir cálculos
24     if (total, n_termino, potencia) in M.keys():
25         return M[(total, n_termino, potencia)]
26     M[(total, n_termino, potencia)] = pSumP(total, n_termino, potencia, M)
27     return M[(total, n_termino, potencia)]
28
29 pSumP()
```

```
1 usage new *
26 def main():
27     lines = sys.stdin.readlines() #Lee entradas estándar por consola
28     for i in range(0, len(lines), 2): #Poner las entradas por línea separada y sin espacio de por medio
29         total = int(lines[i].strip()) #Convierte a entero el total quitando el espacio
30         potencia = int(lines[i+1].strip()) #Convierte la potencia en entero quitando el espacio
31         print(pSum(total, n_termino: 1, potencia))
32
33 main()
```

Documentación

Dentro de los códigos correspondientes.

Fuentes

/PowerSum.py /powerSum.txt /salidaSum.txt

/3N+1.py /3N+1.txt /salida3N+1.txt