Proyecto 1: Búsqueda de Tripletes

Miguel Ochoa Hernández, Cesar Fernando Laguna Ambriz, Eduardo Armando Villarreal García
Maestria: Ciencias Computacionales
Universidad Autónoma de Guadalajara

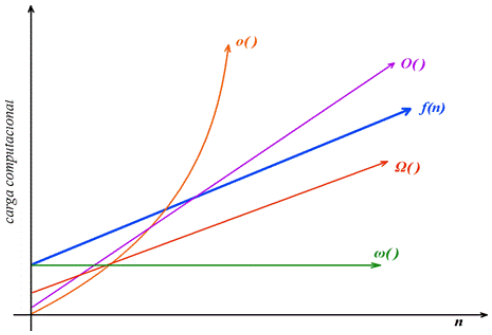
***Resumen*- De acuerdo al análisis y diseño de algoritmos se pretende encontrar la forma más eficiente de resolver un problema con el fin de encontrar el modo más eficiente de resolver dicho problema, Villalpando (p8) refiere que el principal análisis asintótico en informática es de comparar funciones reales de una variable natural, para poder decir cuál tiene mejor comportamiento asintótico, es decir, cuál es menor cuando la variable independiente es suficientemente grande.**

Dicho lo anterior podemos inferir en el número de operaciones y de ese modo encontrar o saber de qué orden es y por lo tanto dependiendo el orden tomara menos tiempo o más tiempo, si es de orden menor tomara menos y si es de orden mayor le tomara más tiempo.

I. Introducción

En el análisis asintótico de algoritmos tenemos un conjunto de asíntotas para encontrar su orden de complejidad, para ello tenemos un conjunto de notaciones:

$\theta(1)$	$constant$
$\theta(\log(n))$	$logaritmica$
$\theta((\log(n))^c)$	$polilogaritmico$
$\theta(n)$	$lineal$
$\theta(n^2)$	$quadratica$
$\theta(n^c)$	$polinomial$
$\theta(c^n)$	$exponencial$



Según la notación y grafica anterior podemos inferir que a medida que cambia el orden de entrada de una función el algoritmo crece conforme al tiempo.

(Cárdenas Abril 2013) Describe que La notación asintótica es de suma importancia en ciencias de la computación para determinar el tiempo de ejecución de los algoritmos y/o hacer comparaciones entre ellos. Sirve de parámetro de referencia estándar, ya que no hay un modelo o máquina universal contra el que se puedan evaluar todos los algoritmos en su tiempo de ejecución o corrimiento.

II. Antecedente Algoritmo

A. Escriba un programa que genere un millón de números enteros Aleatorios entre -100 y 100.
   1. Escriba los primeros 1,000 números en un archivo 1Knums.txt.
   2. Escriba los primeros 2,000 números en un archivo 2Knums.txt.
   3. Escriba los primeros 5,000 números en un archivo 5Knums.txt.
   4. Escriba los primeros 10,000 números en un archivo 10Knums.txt.
   5. Escriba los primeros 100,000 números en un archivo 100Knums.txt.
   6. Escriba el millón de números en un archivo 1Mnums.txt.

B. Para cada uno de los archivos generados en el punto anterior, encuentre todas las sumas de tres números que den como resultado cero.
   1. ¿Cuáles son esos tripletes? Escriba en un archivo los tripletes encontrados.
   2. ¿Cuántos tripletes encontró?
   3. ¿Cuánto tiempo se tardó en encontrar esos tripletes?

C. Reporte los resultados de estas corridas en un documento.

D. Utilice la función randomize() o equivalente para alimentar con una semilla diferente los números aleatorios y realice 100 corridas diferentes generando solamente 100 mil números aleatorios entre -100 y 100.

E. Para cada uno de los archivos generados en el punto anterior, encuentre todas las sumas de tres números que den como resultado cero.
   1. ¿Cuáles son esos tripletes? Escriba en un archivo los tripletes encontrados.
   2. ¿Cuántos tripletes encontró?
   3. ¿Cuánto tiempo se tardó en encontrar esos tripletes?

F. Reporte los resultados de estas corridas en el documento anterior.

III. Propuesta de Solución

La propuesta de solución para el ejercicio dado, se centra principalmente en la utilización de diccionarios, ya que para buscar en diccionarios se utiliza un algoritmo llamado hash, que se basa en realizar un cálculo numérico sobre la clave del elemento, y tiene una propiedad muy interesante: sin importar

cuántos elementos tenga el diccionario, el tiempo de búsqueda es siempre aproximadamente igual.

El algoritmo se basa en vaciar un vector con los números a un diccionario, que usa como llave los números del vector y como valor de nodo se guarda la cantidad de veces que dicho numero apareció dentro del vector:

```
for i in vector:
  numbs[i] += 0
```

Gracias a esto podemos crear tripletes haciendo búsquedas dentro de un diccionario con solo M llaves dependientemente de la cantidad de elementos N que tenga el vector. El proceso de búsqueda de triples consta de tres, uno principal que itera P veces hasta que no se encuentren más tripletes en el diccionario, un ciclo interno que itera sobre el rango de elementos M del diccionario, empezando por la llave con el menor valor; al final hay un último bucle que se encarga de recorrer el mismo rango de elementos M del diccionario sin embargo esta vez se hace sentido inverso del signo del elemento de la iteración del bucle anterior tomando los negativos desde el menor elemento del rango M hasta -1y los positivos empezando desde 0 hasta el mayor elemento positivo del rango M.

```
domain = list(range(lower, higher + 1))
while not done:
    done = True
    for i in domain:
        if nums[i] < 0:
            continue
        inv = domain if i < 0 else list(reverse(domain))
        for j in inv:
            if nums[j] < 0:
                continue
            done = find(i, j, (i+j) * -1)
            if done:
                save(i, j, (i+j) * -1)
```

La selección de números para un triplete se realiza de la siguiente manera, el primer elemento se obtiene del segundo ciclo y se considera un elemento valido si tiene más de una ocurrencia dentro del diccionario es decir, `if nums[i] > 0` , el segundo y el tercer elemento del triplete se seleccionan dentro del último ciclo, y se darán como válidos sí y solo sí el segundo elemento tienen una ocurrencia mayor a cero dentro del diccionario y el inverso de la suma de los primeros dos elementos, es decir el tercer número, es un elemento valido dentro del diccionario con una ocurrencia mayor a cero. Por ultimo para obtener la cantidad de tripletes encontrados se toman en cuenta los siguientes tres casos:

- Todos los números encontrados son diferentes: Aquí se revisa cuál de los tres elementos tiene el menor número de ocurrencias y se resta dicha cantidad de los nodos de los elementos en el diccionario.
- Los tres elementos son iguales: Solo aplica para 0 y el número de tripletes está determinado por la división entera de la cantidad de veces que tiene el diccionario multiplicado por tres.
- Dos de los elementos son iguales: En este caso la cantidad de tripletes se determinada sacando el menor valor entre la cantidad que de veces que aparece el elemento diferente y la división entera entre dos del número de veces que aparecen los elementos repetidos en el diccionario, sacando el menor valor al número no repetido se le resta el valor seleccionado mientras que a uno de los nodos repetidos se le resta el valor mínimo multiplicado por dos.

Una vez obtenido el algoritmo se responderá a las preguntas que se expone en la problemática anterior.

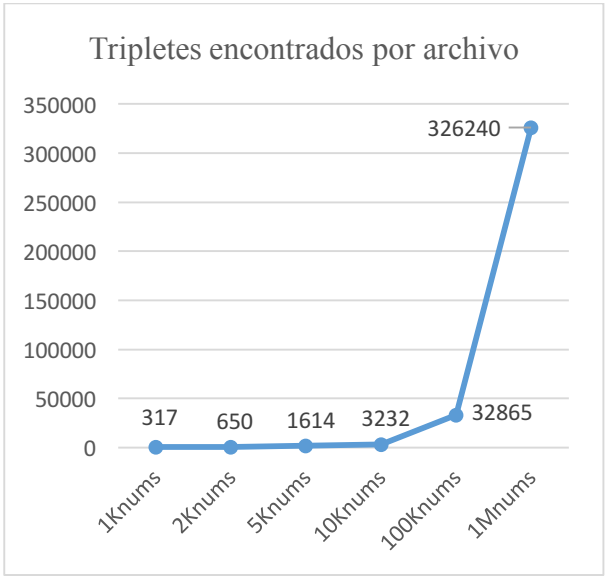


Fig. 1.1 Numero de Tripletes (la suma es 0), encontrados por el algoritmo al hacer la lectura de cada archivo, según la cantidad de números aleatorios almacenados en ellos. (Cantidad/Archivos)

- 1Knum.log > 317 Tripletes (148 diferentes)
- 2Knum.log > 650 Tripletes (165 diferentes)
- 5Knum.log > 1614 Tripletes (167 diferentes)
- 10Knum.log > 3232 Tripletes (179 diferentes)
- 100Knum.log > 32865 Tripletes (192 diferentes)
- 1Mnum.log > 326240 Tripletes (189 diferentes)

Así mismo, mostramos a continuación el tiempo de ejecución del algoritmo, al leer los datos de los respectivos archivos. Tiempo que le tomo procesar los tripletes (suma es cero) por cada archivo.

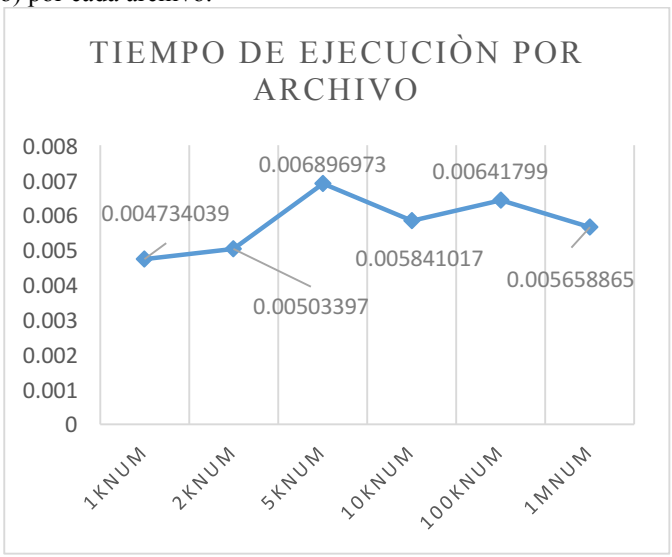


Fig. 1.2 Tiempo (segundos) que el algoritmo proceso los archivos, encontrando la cantidad de tripletes (suma sea 0). (Segundos/Archivos)

- 1Knum.log > 0.00473403930664s
- 2Knum.log > 0.00503396987915s
- 5Knum.log > 0.00689697265625s
- 10Knum.log > 0.00584101676941s
- 100Knum.log > 0.00641798973083s
- 1Mnum.log > 0.00565886497498s

Donde en la suma total del tiempo de procesamiento que le tomo al algoritmo generar, guardar en archivo, leer e identificar los tripletes donde su suma sea cero fue **2.4001310 segundos.** A su vez, como objetivo de prueba hacia el algoritmo, se realizaron 100 corridas, generando 100,000 números aleatorios, guardándolos en 100 archivos diferentes, para leerlos y procesar estos números e identificar la tripleta donde su suma sea cero. Obteniendo los siguientes resultados.

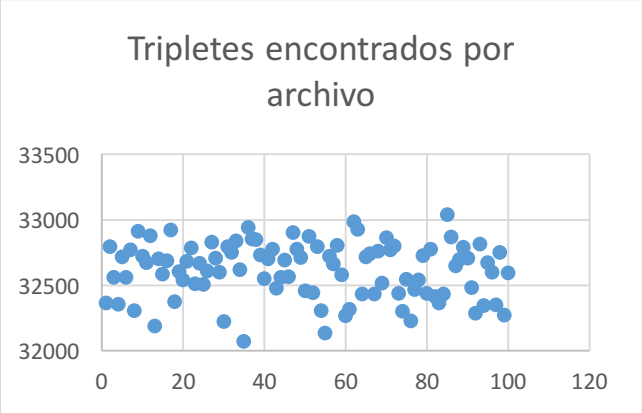


Figura 1.3 Tripletes encontrados por archivo. (No. Tripletes/No. Archivo)

El resultado muestra que el promedio de tripletes encontrados por archivo en 100 corridas es de **32617.** El mayor número de tripletes encontrados fue en el archivo 85_K100nums.log con **33044**, así mismo el menor fue el archivo 35_K100nums.log con **32073.**

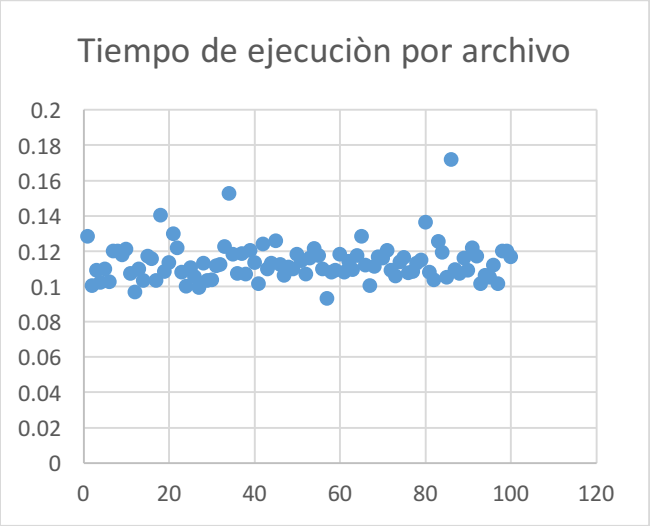


Figura 1.4. Tiempo de ejecución por archivo (seg/no. Archivos).

Con la figura 1.4 podemos deducir que el promedio de tiempo en generar, escribir, leer y procesar haciendo la búsqueda de tripletes es de **0.11** segundos por archivo. Siendo el archivo 57_K100nums.log el que tardo menos con **0.093 segundos**, y el archivo 86_K100nums.log con mayor tiempo con **0.1716 segundos**.

Haciendo una fusión de los archivos obtenidos en las 100 corridas, por obtener como simple dato el triplete más y menos frecuente, se obtuvo la siguiente información.

Triplete	Ocurrencias
(18, -37, 19):	1
(23, -40, 17):	1
(-30, 27, 3):	1
(30, -46, 16):	1
(32, -63, 31):	1
(-33, 21, 12):	1
(-33, 24, 9):	1
(-33, 27, 6):	1
(-33, 28, 5):	1
(-33, 30, 3):	1

Tabla 1.1 Tripletes con menos ocurrencias como resultado de la fusión.

Triplete	Ocurrencias
(-82, 91, -9):	42424
(-84, 92, -8):	42858
(-86, 93, -7):	43259
(-88, 94, -6):	43716
(-90, 95, -5):	44124
(-92, 96, -4):	44636
(-94, 97, -3):	44905
(-96, 98, -2):	45710
(-98, 99, -1):	46319
(-100, 100, 0):	48038

Tabla 1.2 Tripletes con mayor ocurrencias como resultado de la fusión.

IV. Conclusiones

A medida que estudiamos la relación que tiene el orden de las ecuaciones descritas nos hizo optar por la solución donde podemos manejar los tripletes de forma constante. Ya que no aumenta el tiempo de proceso ayudándonos de diccionarios en Python, ya que de esa manera accedemos al dato de una vez.

Agradecimientos

Se agradece a Miguel Ochoa Hernández que ya que fue concebida esta idea por él, también a Eduardo Armando Villarreal y Cesar Fernando Laguna Ambriz por colaborar con ello y obtener los datos de este paper.

Referencias

[1] Villalpando B. J., "Análisis asintótico de aplicación de funciones de Landau", 2003, e-Gnosis [online], Vol.1.

[2] Cárdenas, "La notación científica en el computo científico" (Abril 2013), eumed.net. Universidad Autónoma de Baja California. http://www.eumed.net/rev/tlatemoani/12/algoritmo.html