Universidad de Talca
Facultad de Ingeniería
Departamento de Ciencias de la Computación

Algoritmos y estructuras de datos

Tarea 1 - Informe modelo

Erik Regla
eregla09@alumnos.utalca.cl

1 de Mayo del 2014

1. Introducción

Un número triangular es aquel que puede recomponerse en la forma de un triángulo equilátero (por convención, el primer número triangular es el 1). Los números triangulares, junto con otros números figurados, fueron objeto de estudio por Pitágoras y los Pitagóricos, quienes consideraban sagrado el 10 escrito en forma triangular, y al que llamaban Tetraktys.

En 1796, el matemático y científico alemán Carl Friedrich Gauss descubrió que todo entero positivo puede representarse como la suma de un máximo de tres números triangulares, hecho que describió en su diario con la misma palabra que usara Arquímedes en su famoso descubrimiento: "¡Eureka! num $= \triangle + \triangle + \triangle$."

Nótese que este teorema no implica que los números triangulares son diferentes (como ocurre en el caso de $20 = 10 + 10$), ni tampoco que debe haber una solución con tres números triangulares que sean diferentes de cero. Se trata de un caso especial del teorema del número poligonal de Fermat.[1]

Para esta tarea se pide verificar la conjetura mencionada anteriormente, para el rango $[1 \ldots 10^9]$ como a su vez para cada número de forma individual.

2. Análisis del Problema

La conjetura dice que un número natural puede descomponerse en la suma de a lo más tres numeros triangulares por lo tanto, podríamos decir que cualquier número natural n sigue la Propiedad 3.

$$n = \triangle_3 + \triangle_3 + \triangle_3 \tag{1}$$

donde n es un número natural.

[1] Extraido de wikipedia: http://es.wikipedia.org/wiki/Número_triangular

El n-esimo número triangular viene dado por la Ecuación 2.

$$\frac{n(n+1)}{2} \tag{2}$$

En donde n es el índice del número triangular pedido.

Puede que se de el caso donde $\triangle_2$ y $\triangle_3$ sean iguales a 0 (lo cual ocurre cuando el número efectivamente es un triangular), lo que nos deja el primer caso:

$$n = \triangle_1 \tag{3}$$

Lo cual nos lleva a pensar, que la primera verificación para ver si el numero cumple, es verificar primero si es triangular. Para eso, almacenamos en una lista los numeros triangulares generados por (2), a modo de poder tenerlos a mano fácilmente.

Algoritmo 1 Generador de números triangulares

Precondición: Lista L vacía, S cantidad de números a generar.

Postcondición: L contiene numeros triangulares ordenados crecientemente

```
para i ← 0...S hacer
    L[i] ←= n(n+1)/2
fin para
devolver L
```

Entonces, habiendo planteado el Algoritmo 1 y la Ecuación 2, podemos decir que un número natural se compone de a lo más tres números triangulares, lo cual puede ser reescrito como:

$$n = L[a_1] + L[a_2] + L[a_3] \tag{4}$$

donde $L[a]$ es el a-ésimo numero triangular y $L[a_1] \leq L[a_2] \leq L[a_3]$.[2]

Si no se cumple lo planteado en 3, estamos frente a un número que bien podría estar compuesto por dos o bien tres numeros triangulares.

Por ende, si tenemos un número $\triangle_1$ que es menor o igual a un n buscado, entonces, podríamos decir que este es el triangular mayor asociado a n. En caso de que este sea igual, se cumple 3, en caso de que sea estrictamente menor estamos frente al caso planteado en la Ecuación 5 o a la Ecuación 6.

$$n = \triangle_1 + \triangle_2 \tag{5}$$
$$n = \triangle_1 + \triangle_2 + \triangle_3 \tag{6}$$

Despejando $\triangle_3$ de la Ecuación 6 tenemos:

$$\triangle_3 = n - (\triangle_1 + \triangle_2) \tag{7}$$

[2] Obviamente $a_1 \leq a_2 \leq a_3$

Dada la naturaleza de los números y las ecuaciones anteriormente descritas, además podemos afirmar algunas propiedades:

$$\triangle_1 + \triangle_2 + \triangle_3 = n \rightarrow \triangle_1, \triangle_2, \triangle_3 \geq \frac{n}{3} \tag{8}$$

$$\triangle_1 \geq \frac{2n}{3} \rightarrow \triangle_3 \leq \triangle_2 \leq \frac{n}{3} \tag{9}$$

Es obvio que gracias a la Ecuación 9 podemos ya descartar $\frac{2n}{3}$ de los casos en el caso de que el número a buscar sea mayor o igual a $\frac{2n}{3}$. Para el resto de los casos, solo hay que cambiar el valor de $\triangle_1$ y automáticamente el rango es ajustado.

Por lo cual podemos afirmar que si tenemos un potencial $\triangle_2$, automáticamente tenemos el valor de $\triangle_3$ sin necesidad de mayores cálculos de nuestra parte. Además, aplicando el principio de localidad de memoria tenemos:

Algoritmo 2 Búsqueda de triangular relativo o menor

Precondición: N un número natural, I es el índice desde el cual se comienza a buscar, T es un arreglo de números triangulares

Postcondición: I contiene el índice del triangular relativo directamente menor o igual

si $T[I] \leq N$ y $T[I+1] > N$ **entonces**
 devolver I
si no
 mientras $T[I] > N$ **hacer**
  $I \leftarrow I - 1$
 fin mientras
 mientras $N < T[I+1]$ **hacer**
  $I \leftarrow I + 1$
 fin mientras
fin si

Este algoritmo si bien es $O(n)$, gracias al principio de localidad y a las propiedades de los números triangulares, en promedio solo le toma tres operaciones llegar al índice buscado, salvo algunos casos especificos, los cuales por no representar una cantidad importante serán despreciados.

Aplicando algo de matemática al Algoritmo 3, es facil observar que el algoritmo anteriormente descrito es $O(n^2)$.[3], lo cual tiene sentido, dado que n en este caso no representa el número de entradas, sinó, el número a calcular. Usualmente, la verificación no toma más de 3 pasos, llegando experimentalmente a un máximo de 20.

Experimentalmente, resultó no ser una buena idea implementar la verificación de la conjetura, en especial para 10^9 dada la cantidad de verificaciones que hay que realizar. Pero, si queremos ver si efectivamente n es un numero que cumple con esta. Para esto usamos:

[3]Sin embargo, experimentalmente, la curva de rendimiento es relativamente cercana a $n^{\approx 0,2}$

Algoritmo 3 Descomposción de números naturales

Precondición: N un número natural

Postcondición: $T1$, $T2$ y $T3$ son los índices de la descomposción de N dada la conjetura 1; Arroja un error en caso de fallar.

$T_1 \leftarrow$ indice triangular relativo a N
$\Delta T_1, \Delta T_2, \Delta T_3 \leftarrow 0$
si $L[T_1] = N$ **entonces**
devolver T_1
si no
mientras $L[T_1 - \Delta T_1] \geq \frac{n}{3}$ **hacer**
$T_2 \leftarrow$ indice triangular relativo a $(N - L[T_1 - \Delta T_1])$
si $L[T_1 - \Delta T_1] + L[T_2] = N$ **entonces**
devolver $T_1 - \Delta T_1, T_2$
si no
mientras $L[T_1 - \Delta T_1] + L[T_2 - \Delta T_2] \geq \frac{2n}{3}$ **hacer**
$T_3 \leftarrow$ indice triangular relativo a $(N - (L[T_1 - \Delta T_1] + L[T_2 - \Delta T_2]))$
si $L[T_1 - \Delta T_1] + L[T_2 - \Delta T_2] + L[t_3] = N$ **entonces**
devolver T_1, T_2, T_3
fin si
$\Delta T_2 \leftarrow \Delta T_2 + 1$
fin mientras
fin si
$\Delta T_1 \leftarrow \Delta T_1 + 1$
fin mientras
fin si
devolver ERROR

$$n = a_i \tag{10}$$
$$b = a_i + a_b \tag{11}$$
$$c = b_i + a_i \tag{12}$$

Sean a, b y c dos números naturales que cumplen con la conjetura utilizando 1, 2 y 3 términos respectivamente.

Como antecedente adicional, se precalculó el número de números triangulares para 10^9, los cuales son 44720 numeros triangulares, siendo $\triangle_{44720} = 999961560$. Númericamente, es correcto afirmar que para 10^9 se cumple:

$$\#\triangle_{10^9} \leq \sqrt{10^9} \tag{13}$$

Lo cual nos garantiza que cualquier operación efectuada sobre la cantidad misma de triangulares tendrá un costo de $O(n)$.[4]

Entonces, una de las operaciones que se pueden ejecutar a un costo relativamente aceptable, es la obtención de los números que pertenecen a la familia descrita en la Equación 11.

Algoritmo 4 Obtención de triangulares mediante suma

Precondición: T es un arreglo que contiene números triangulares, A es un arreglo de bits inicializado en 0.

Postcondición: A contiene marcados con 1 los indices que refieren a un numero del cual se ha comprobado su conjetura y pertenecen al tipo indicado en la Ecuación 11

para $i \leftarrow 0 \dots T_{size}$ **hacer**
  para $j \leftarrow i \dots T_{size}$ **hacer**
    si $i \times j \geq T_{size}$ **entonces**
      break
    fin si
    $A[i \times j] \leftarrow 1$
  fin para
fin para

Lo cual nos da aproximadamente $\frac{(n^2)-n}{2}$ combinaciones, de las cuales un número indeterminado ya están repetidas posiblemente.[5] Después de este punto, otra operación que sería relativamente económica sería la obtención de los números que cumplen con el criterio de la Ecuación 12 la cual consiste en copiar sobre el mismo resultado, los resultados anteriores desplazados en un triangular.

Como es posible observar, el Algoritmo 5 tiene un costo de $O(mn)$, siendo m la cantidad de desplzamientos a aplicar y n el tamaño del arreglo, que en este caso es 10^9. Experientalmente, luego

[4] Para 10^9 el costo teórico esperado es de $\frac{\sqrt{10^9}}{44720} = \approx 0{,}7071 \leq 1$.

[5] Experimentalmente, después de esta etapa, para 10^9 quedan 671058847 numeros por verificar que realmente cumplen la conjetura.

Algoritmo 5 Desplazamiento de resultados para cálculo de triangulares

Precondición: T es una lista de números triangulares ordenados crecientemente, A_t es un arreglo de bits que contiene el resultado del Algoritmo 2, A_r es un arreglo de bits que contiene el resultado del Algoritmo 2 junto con la iteración producida. D es la cantidad de desplazamientos a ejecutar.

Postcondición: A_r contiene los resultados de A_t desplzadados, cumpliendo el criterio de la Ecuación 12

para $j \leftarrow 0 \ldots D$ **hacer**
    para $i \leftarrow 0 \ldots A_t t_{size}$ **hacer**
        si $A_r[i] + T[j]$ **entonces**
            break
        fin si
        $A_r[i + T[j]] \leftarrow 1$
    fin para
fin para

de 22 iteraciones, para 10^9 términos la cantidad a verificar de estos se reduce de 671058847 términos a 192436, cantidad a la cual usar el algoritmo de descomposición se vuelve feasible.[6]

A este punto, ya es posible visualizar la ejecución del programa para verificar la conjetura en el Algoritmo 6.

3. Detalles de implementación

3.1. Memoria

El consumo de memoria siempre es un problema. Considerando el peso en memoria de un tipo de dato *Long*[7], para un computador normal, está fuera de lo que uno puede manejar en memoria. Además, usar la memoria virtual no es una opción, dada lo baja de la velocidad de esta. Para estos fines, se recomienda la aplicación de estructuras adecuadas, como por ejemplo *mapas de bits*.[8]

3.2. Lenguaje

Parte de ir tras un problema es elegir el arma adecuada. Sin mucha discusión, elegimos *C++* como lenguaje objetivo para poder implementar la solución. Dada la libertad que proveen las

[6]La cantidad de iteraciones es directamente proporcional de términos que se descartan, dada la distribución de los numeros triangulares. La cantidad de iteraciones de por si, dependerá de la naturaleza del problema.

[7]Se refiere al tipo de dato *Long Integer* dado que es imposible almacenar en un tipo de dato mayor los números del rango solicitado. Un *Long Integer* pesa 32 bits.

[8]Un mapa de bits es una estructura que solo almacena bits en un arreglo. Dados que estos pueden ser interpretados como *verdadero* o *falso*, proveen de una solución simple para poder comprimir los datos a utilizar. Sin embargo, implementarlos supone un costo adicional en operaciones de comparación y asignación, pero en este caso, dada la naturaleza de nuestro algoritmo, podemos sobrellevar el peso que esta técnica implica. http://en.wikipedia.org/wiki/Bit_array

Algoritmo 6 Desplazamiento de resultados para cálculo de triangulares

Precondición: N es el rango de números a verificar, A_t y A_r son arreglos de bits inicializads en 0. T es un arreglo de números inicializado en 0. D es la cantidad de desplazamientos a ejecutar.

Postcondición: retorna **verdadero** si la conjetura es verificada para todo número en el rango

```
T ← generarNmerosTriangulares(N)
copiarresultadosdeTaA_t
A_r ← obtenerTriangularesMedianteSuma(A_t, D, N)
para i ← 0 ... A_r A_r_size hacer
   si A_r[i] ≠ 1 entonces
      A_r[i] ← verificarConjetura(i)
   fin si
fin para
para i ← 0 ... A_r A_r_size hacer
   si A_r[i] ≠ 1 entonces
      devolver falso
   si no
      devolver verdadero
   fin si
fin para
```

librerías de entrada y salida estándar, junto con la flexibilidad provista para el manejo de memoria, es la opción obvia para este caso.

3.3. Tiempos de ejecución

Sin mucho que agregar, teóricamente sin mucho esfuerzo, el tiempo de ejecución debería ser $O(n^3)$, sin embargo, dado que este algoritmo fue desarrollado para un caso muy especifico, se habla de que es en realidad $O(kn)$ con una k constante la cual consideramos que si bien es mayor que 1 nunca es mayor o igual que n, la cual está en función de las operaciones de comparación implementadas y de gestión de memoria.

3.4. Compilación y Ejecución

Se recomienda usar *Linux G++ 11* como compilador. Compilación para versión a entregar:

```
g++ -Wall -std=c++0x t1.cpp -o t1.out
```

Compilación incluyendo salidas para *debug*:

```
g++ -Wall -std=c++0x -DDEBUG t1.cpp -o t1.out
```

Ejecución:

```
t1.out < input.in > output.out
```

4. Notas

No se han incluido como sección separada los gráficos ni los diagramas de estado, dado que este es un informe modelo con la finalidad de explicar el funcionamiento de un algoritmo implementado. Dada la profundidad de la explicación y las aclaraciónes experimentales hechas durante el desarrollo de este, agregar los diagramas de estado y resultados de pruebas no suponen complemento alguno a la comprensión del problema.

Cabe destacar que esta es solo una de las soluciones para este problema. Por ende, podría no ser eficiente y no ser la mejor, pese a eso, para efectos del problema planteado, lo resuelve en un tiempo aceptable.