# Uso de estructura repetitiva *while*

Actividad de Algoritmia y Programación

## Conjetura de Collatz

La **conjetura de Collatz**, conocida también como **conjetura de Ulam** (por el matemático polaco-estadounidense **Stanislaw Marcin Ulam**), **el problema de Kakutani** (por el matemático japonés-estadounidense **Shizuo Kakutani**), **la conjetura de Thwaites** (por el académico británico Sir **Bryan Thwaites**), **el algoritmo de Hasse** (por el matemático alemán **Helmut Hasse**) o **el problema de Siracusa**, fue propuesta por primera vez por el matemático alemán Lothar Collatz en 1937, y a la fecha no se ha demostrado ni refutado. Quizás el nombre más descriptivo sea: **la conjetura de 3n+1**.

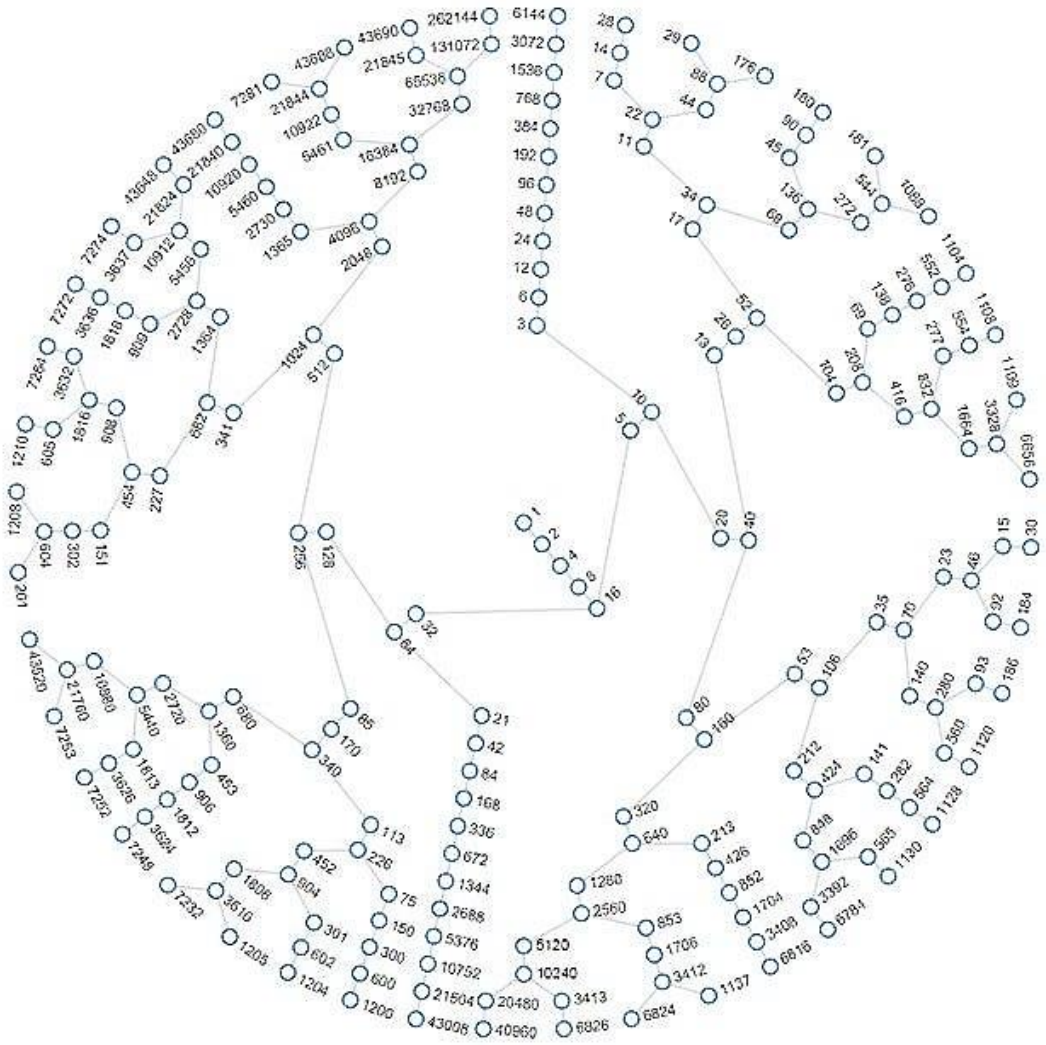
### Enunciado

Sea la siguiente operación, aplicable a cualquier número entero positivo:

- Si el número es par, se divide entre 2.
- Si el número es impar, se multiplica por 3 y se suma 1.

La conjetura dice que siempre alcanzaremos el 1 (y por tanto el ciclo 4, 2, 1) para **cualquier** número con el que comencemos.

A la secuencia de números involucrada se la conoce como **secuencia o números de granizo** o como **números maravillosos**.



Así lo ilustra Jason Davies, un ingeniero de software, en su gráfico Collatz: Todos los números llevan al uno

## Ejemplos

Comenzando en *n* = 6, uno llega a la siguiente sucesión: 6, 3, 10, 5, 16, 8, 4, 2, 1.

Empezando en *n* = 11, la sucesión tarda un poco más en alcanzar el 1: 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

Empezando *n* = 27, la sucesión tiene 111 pasos, llegando hasta 9232 antes de descender a 1: 27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350, 175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502, 251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719, 2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051, 6154, 3077, **9232**, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1.

Supercomputadoras han hecho pruebas con números que van hasta más o menos 5.764.607.500.000.000.000. Todos eventualmente llegan a 2 ÷ 2 = 1. No obstante, como los números son infinitos, eso no prueba que ese sea el caso para todos los números naturales. Pero como no se pudo encontrar una excepción, tampoco hay prueba de que no sea así. El otro interrogante sin resolver es el eterno por qué. **¿Por qué se comportan así los números?**

Como el granizo al caer, los números saltan de un lugar al otro antes de llegar al 4, 2, 1. Unos más y unos menos, sin ton ni son. La mayor cantidad de escalas que hace un número inicial menor de 100 millones para llegar a 4, 2, 1 es **986**. Pero mientras que, por ejemplo, para los múltiplos de 2 el viaje es más corto, otros toman más tiempo.

Un ejemplo citado a menudo es la comparación entre los números 8.192 y 27. Al 8.192 le toma 13 pasos llegar aparentemente ineludible final: 4, 2, 1. El número 27 no sólo toma 111 pasos en llegar, sino que en el camino sube hasta 9.232. La falta de patrones dificulta aún más resolver una conjetura ya tachada de imposible.

# Problema para resolver

Desarrollar un programa que solicite al usuario una secuencia de números enteros positivos o 0 para finalizar, y para cada uno muestre en pantalla su secuencia de granizo, el mayor número de su secuencia, y la cantidad de pasos hasta obtener el 1. Por ejemplo, si el usuario ingresara el número 11, el programa debería mostrar:

```
Secuencia de granizo:
34 17 52 26 13 40 20 10 5 16 8 4 2 1

Mayor número de la secuencia: 52

Pasos: 14
```

# Requerimientos del desarrollo

1. El problema debe solucionarse utilizando **únicamente** los recursos de Python conocidos hasta el momento de su planteo (sin funciones ni listas).
2. El programa debe ser **eficaz** (informar resultado esperado y cumplir con especificación del enunciado), **inteligible** (utilizar el modelo o plantilla y completar todas las secciones que sean pertinentes al problema) y **eficiente** (lograr calidad de diseño).