Números Triangulares

|  |  |
| --- | --- |
| Fecha de Elaboración: | Octubre 20, 2013 |
| Autores: | Diego Serrano |
| Fuente: | Insilico – UTP Judge |

## Problema

Los números triangulares son números enteros positivos tales que representan una cantidad de "puntos" con la que se puede formar un triángulo equilátero de puntos.

Los cinco primeros números triangulares son:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **3** | **6** | **10** | **15** |

Para este problema, debe crear un programa que determine si un número *n* dado es triangular o no.

**Entrada**

La entrada puede contener varios casos de prueba. Cada caso de prueba se da en una línea propia, y contiene un entero n (1 ≤ n ≤ 16 × 1018). La entrada termina con un caso de prueba en el que *n* es *cero*, y no debe ser procesada.

**Salida**

Para cada caso de prueba, el programa debe imprimir *YES* o *NO* para indicar si *n* es un número triangular o no. Debe haber una sola línea de salida para cada caso de prueba.

**Ejemplo de Entrada**

1

15

16

101

0

**Ejemplo de Salida**

YES

YES

NO

NO

## Planteamiento de la Solución BÁSICA

Como probablemente lo haya visto en sus primeros cursos de matemáticas, la fórmula para calcular un número triangular está definida por:

Aunque el título de esta sección diga que es una solución básica, por ningún motivo es ingenua. Despejando la anterior ecuación, tenemos:

La cual podemos resolver aplicando la formula general para ecuaciones cuadráticas:

Entonces,

De esta manera, resolviendo la ecuación e inspeccionando que la solución sea un número entero, habremos estado seguros que el número *Tn* es un número triangular. De hecho, solo basta con que la raíz sea un número entero.

Sin embargo, para números grandes, como 10237812502262500000, que es el numero triangular T4525000000, nuestro algoritmo lo detecta correctamente, pero el siguiente número, el cual no debería ser triangular 10237812502262500001, también es detectado como triangular. Lastimosamente, las operaciones requeridas, en especial la raíz cuadrada, no son muy precisas con números tan grandes.

## Código Fuente

import java.io.BufferedReader;

import java.io.InputStream;

import java.io.InputStreamReader;

public class Naive {

public static boolean isPerfectSquare(double n) {

double sqrt = Math.sqrt(n);

sqrt = sqrt - (sqrt % 1);

if ((sqrt \* sqrt) == n) {

return true;

}

return false;

}

public static void main(String[] args) {

try {

InputStream input = System.in;

BufferedReader reader = new BufferedReader(new

InputStreamReader(input));

String params;

while (!(params = reader.readLine()).equals("0")) {

double n = Double.parseDouble(params);

if (isPerfectSquare(8\*n + 1)) {

System.out.println("YES");

} else {

System.out.println("NO");

}

}

reader.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

## Planteamiento de la Solución ÓPTIMA

De las propiedades de los números triangulares, podemos deducir que es una secuencia monofónicamente creciente, es decir, ordenada de forma ascendente, donde

Haciendo cálculos rápidos, vemos que tenemos que chequear números triangulares entre *T0= 1* y *T6000000000=18000000003000000000*. Así que podemos hacer una simple búsqueda binaria entre *0* y *6000000000*, tratando de encontrar nuestro número, usando el tipo de datos *BigInteger* con el cual preservaremos la precisión en los números. La complejidad para resolver cada caso de prueba es claramente *O(log N)*, debido al algoritmo de búsqueda usado.

La solución fue enviada al sistema Insilico de la Universidad Tecnología de Pereira, el 20 de Octubre de 2013, siendo aceptada con un tiempo de 0.432026 segundos.

## Código Fuente

import java.io.BufferedReader;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.math.BigInteger;

public class Main {

public static BigInteger TWO = new BigInteger("2");

public static void main(String[] args) {

try {

InputStream input = System.in;

BufferedReader reader = new BufferedReader(

new InputStreamReader(input));

String params;

while (!(params = reader.readLine()).equals("0")) {

BigInteger n = new BigInteger(params);

long end = 6000000000l;

long start = 0;

long mid;

while (start <= end) {

mid = (start + end) / 2;

BigInteger midBig = new BigInteger(String.valueOf(mid));

BigInteger midNumber = (midBig.multiply(midBig).add(midBig))

.divide(TWO);

if (midNumber.equals(n)) {

System.out.println("YES");

break;

} else if (midNumber.compareTo(n) < 0) {

start = mid + 1;

} else {

end = mid - 1;

}

}

if (start > end)

System.out.println("NO");

}

reader.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}