03/08/2020

Carlos Alberto Cordero Robles

implementación fibonacci funcional + tabla comparativa

Algunas modificaciones tuvieron que ser hechas a la entrega anterior que hice para poder manejar valores long en vez de ints.

La forma funcional de Fibonacci la busque en internet y la tome de [1].

Para esta tarea ejecute las tres maneras de Fibonacci en pasos de 10 en 10 hasta el 100.

A continuación, presento los resultados.

|  |  |  |  |
| --- | --- | --- | --- |
| N | iterativo(s) | recursivo(s) | funcional(s) |
| 10 | 0.1 | 0.1 | 0.15 |
| 20 | 0.1 | 0.12 | 0.14 |
| 30 | 0.11 | 0.13 | 0.15 |
| 40 | 0.11 | 0.74 | 0.15 |
| 50 | 0.1 | 1.34 | 0.16 |
| 60 | 0.1 | inf | 0.14 |
| 70 | 0.12 | inf | 0.17 |
| 80 | 0.1 | inf | 0.18 |
| 90 | 0.1 | inf | 0.18 |
| 100 | 0.09 | inf | 0.15 |

Lo que podemos notar es que tanto el iterativo como el funcional se encuentran siempre cercanos a 0.1 y van creciendo muy gradualmente en cambio el recursivo su crecimiento es exponencial.

El código lo agrego a continuación

Iterativo

|  |
| --- |
| def fibIterative( input : Long ) : Long = {  var firstVal : Long = 0  var secondVal : Long = 1  var i : Long = 0  var addition : Long = 0    while( i < input ) {  addition = firstVal + secondVal  firstVal = secondVal  secondVal = addition  i = i + 1  }  return firstVal  } |

Recursivo

|  |
| --- |
| def fibRecursive( n : Long) : Long = {  if ((0 == n) | (1 == n))  {  return n  }  else  {  return(fibRecursivo( n-1 ) + fibRecursivo( n-2 ))  }  } |

Fincional

|  |
| --- |
| def fibFunctional: Stream[Long] = {  def tail(h: Long, n: Long): Stream[Long] = h #:: tail(n, h + n)  tail(0, 1)  } |

Referencias

[1]<<generate-a-sequence-of-fibonacci-number-in-scala>> [En Línea]. Disponible en: <https://stackoverflow.com/questions/9864497/generate-a-sequence-of-fibonacci-number-in-scala> [Accedido: 08-Mar-2020]