codility

Candidate Report: Anonymous

Test Name:

Summary   Timeline

Test Score

100 out of 100 points

100%

Tasks in Test

	Time Spent ⓘ	Task Score
BinaryGap Submitted in: Scala	3 min	100%

TASKS DETAILS

1.

EASY

BinaryGap
Find longest sequence of zeros in binary representation of an integer.

Task Score	Correctness	Performance
100%	100%	Not assessed

Task description

A *binary gap* within a positive integer N is any maximal sequence of consecutive zeros that is surrounded by ones at both ends in the binary representation of N.

For example, number 9 has binary representation 1001 and contains a binary gap of length 2. The number 529 has binary representation 1000010001 and contains two binary gaps: one of length 4 and one of length 3. The number 20 has binary representation 10100 and contains one binary gap of length 1. The number 15 has binary representation 1111 and has no binary gaps. The number 32 has binary representation 100000 and has no binary gaps.

Write a function:

Solution

Programming language used:	Scala
Total time used:	3 minutes ❓
Effective time used:	3 minutes ❓
Notes:	*not defined yet*

Task timeline ❓

object Solution { def solution(n: Int): Int }

that, given a positive integer N, returns the length of its longest binary gap. The function should return 0 if N doesn't contain a binary gap.

For example, given N = 1041 the function should return 5, because N has binary representation 10000010001 and so its longest binary gap is of length 5. Given N = 32 the function should return 0, because N has binary representation '100000' and thus no binary gaps.

Write an **efficient** algorithm for the following assumptions:

- N is an integer within the range [1..2,147,483,647].

11:40:25   11:42:50

Code: 11:42:49 UTC,   show code in pop-up
scala, final, score: **100**

```scala
1   import scala.collection.JavaConverters._
2
3   // you can write to stdout for debugging purposes, e
4   // println("this is a debug message")
5
6   object Solution {
7     def solution(n: Int): Int = {
8       val idxOfOne = n.toBinaryString.zipWithIndex
9         .filter { case (c, i) => c == '1' }
10        .map { case (c, i) => i }
11        .toSeq
12
13      idxOfOne.length match {
14        case i if i <= 1 => 0
15        case _ => {
16          idxOfOne.sliding(2).map(slide => slide(1) - slide(
17        }
18      }
19    }
20  }
```

Analysis summary

The solution obtained perfect score.

Analysis ❓

expand all	Example tests	
▶ example1		✔ OK
example test		
n=1041=10000010001_2		
▶ example2		✔ OK
example test n="15=1111_2		
▶ example3		✔ OK
example test n=32=100000_2		
expand all	Correctness tests	
▶ extremes		✔ OK
n=1, n=5=101_2 and		
n=2147483647=2**31-1		
▶ trailing_zeroes		✔ OK
n=6=110_2 and		
n=328=101001000_2		
▶ power_of_2		✔ OK
n=5=101_2, n=16=2**4 and		
n=1024=2**10		

▶	simple1	✔ OK
	n=9=1001_2 and n=11=1011_2	

▶	simple2	✔ OK
	n=19=10011 and n=42=101010_2	

▶	simple3	✔ OK
	n=1162=10010001010_2 and n=5=101_2	

▶	medium1	✔ OK
	n=51712=110010100000000_2 and n=20=10100_2	

▶	medium2	✔ OK
	n=561892=100010010010111001 00_2 and n=9=1001_2	

▶	medium3	✔ OK
	n=66561=10000010000000001_2	

▶	large1	✔ OK
	n=6291457=110000000000000000 00001_2	

▶	large2	✔ OK
	n=74901729=10001110110111010 0011100001	

▶	large3	✔ OK
	n=805306373=1100000000000000 00000000000101_2	

▶	large4	✔ OK
	n=1376796946=101001000010000 0100000100010010_2	

▶	large5	✔ OK
	n=1073741825=100000000000000 0000000000000001_2	

▶	large6	✔ OK
	n=1610612737=110000000000000 0000000000000001_2	