Codility_

CodeCheck Report: trainingW9PKYU-ECX

Test Name:

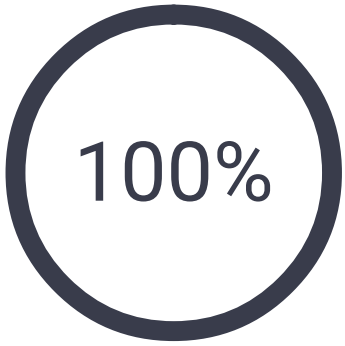
Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
SparseBinaryDecomposition Java 8 ⚠	73 min	100%

Total score



Tasks Details

1. **SparseBinaryDecomposition**
Decompose int into sum of ints having no consecutive 1s in binary form.

Hard

Task Score	Correctness	Performance
100%	100%	100%

Task description

A non-negative integer N is called *sparse* if its binary representation does not contain two consecutive bits set to 1. For example, 41 is sparse, because its binary representation is "101001" and it does not contain two consecutive 1s. On the other hand, 26 is not sparse, because its binary representation is "11010" and it contains two consecutive 1s.

Two non-negative integers P and Q are called a *sparse decomposition* of integer N if P and Q are sparse and N = P + Q.

For example:

- 8 and 18 are a sparse decomposition of 26 (binary representation of 8 is "1000", binary representation of 18 is "10010");
- 9 and 17 are a sparse decomposition of 26 (binary representation of 9 is "1001", binary representation of 17 is "10001");
- 2 and 24 are not a sparse decomposition of 26; though 2 + 24 = 26, the binary representation of 24 is "11000", which is not sparse.

Write a function:

```
class Solution { public int solution(int N); }
```

Solution

Programming language used:	Java 8
Total time used:	73 minutes
Effective time used:	73 minutes
Notes:	*not defined yet*

Task timeline

18:30:42 19:43:15

Code: 19:43:15 UTC, java, final, score: **100** show code in pop-up

```
1  // you can also use imports, for example:
2  import java.util.*;
3
4  // you can write to stdout for debugging purpos
```

that, given a non-negative integer N, returns any integer that is one part of a sparse decomposition of N. The function should return −1 if there is no sparse decomposition of N.

For example, given N = 26 the function may return 8, 9, 17 or 18, as explained in the example above. All other possible results for N = 26 are 5, 10, 16 and 21.

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

- N is an integer within the range [0..1,000,000,000].


```
// System.out.println("this is a debug message"

class Solution {
        public int solution(int N) {
                BitSet bs = BitSet.valueOf(new
                BitSet result = new BitSet(Inte

                int eldestBit = bs.previousSetB
                if (eldestBit < 1) {
                        return 0;
                }

                if (bs.get(eldestBit - 1)) {
                        eldestBit--;
                }

                for (int bit = eldestBit; bit >
                        if (bs.get(bit)) {
                                result.set(bit)
                        }
                }

                return (int) result.toLongArray
        }

        /**
         * This method is used only for unit te
         * @param a
         * @return
         */
        public static boolean isSpareNumber(int
                BitSet bs = BitSet.valueOf(new

                int previous = bs.nextSetBit(0)
                if (previous == -1) {
                        return false; // a == 0
                }

                for (int next; (next = bs.nextS
                        if (next - previous ==
                                return false;
                        }
                }

                return true;
        }}
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(log(N)) or O(1)**

collapse all	Example tests	
▼ example1 example test n=26		✔ OK
1. 0.008 s OK		

collapse all	Correctness tests	
▼ simple1 n=1166		✔ OK
1. 0.004 s OK		
▼		

simple2 ✔ OK
n=561892

1. 0.008 s OK

▼ simple3 ✔ OK
n=1031

1. 0.004 s OK

▼ small_power_of_two_minus_one ✔ OK
n=1023

1. 0.008 s OK

▼ extreme ✔ OK
n <= 5

1. 0.008 s OK
2. 0.004 s OK
3. 0.004 s OK
4. 0.008 s OK
5. 0.004 s OK
6. 0.008 s OK

collapse all Performance tests

▼ medium1 ✔ OK
n=74901729

1. 0.004 s OK

▼ medium2 ✔ OK
n=216188401

1. 0.008 s OK

▼ power_of_two_minus_one ✔ OK
n=536870911

1. 0.008 s OK

▼ big_random ✔ OK
n=~1000000000

1. 0.004 s OK

▼ maximal ✔ OK
n=1000000000

1. 0.004 s OK