Codility_

CodeCheck Report: trainingTUVSYZ-MXT

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task		Time spent	Score
CountConformingBitmasks Java 8	⚠	1 min	100%

Total score

Tasks Details

Medium

1. CountConformingBitmasks
Count 30-bit bitmasks conforming to at least one of three given 30-bit bitmasks.

Task Score	Correctness	Performance
100%	100%	100%

Task description

In this problem we consider unsigned 30-bit integers, i.e. all integers B such that $0 \le B < 2^{30}$.

We say that integer A *conforms* to integer B if, in all positions where B has bits set to 1, A has corresponding bits set to 1.

For example:

- `00 0000 1111 0111 1101 1110 0000 1111(BIN) = 16,244,239` conforms to `00 0000 1100 0110 1101 1110 0000 0001(BIN) = 13,032,961`, but
- `11 0000 1101 0111 0000 1010 0000 0101(BIN) = 819,399,173` does not conform to `00 0000 1001 0110 0011 0011 0000 1111(BIN) = 9,843,471`.

Write a function:

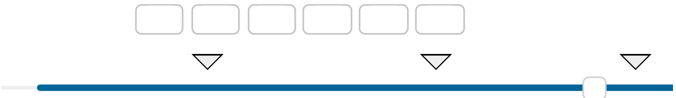
```
class Solution { public int solution(int A,
int B, int C); }
```

that, given three unsigned 30-bit integers A, B and C, returns the number of unsigned 30-bit integers conforming to at least one of

Solution

Programming language used:	Java 8
Total time used:	1 minutes
Effective time used:	1 minutes
Notes:	*not defined yet*

Task timeline



17:52:08 17:52:35

Code: 17:52:35 UTC, java, final, score: **100**

show code in pop-up

```
// you can also use imports, for example:
import java.util.*;

```

the given integers.

For example, for integers:

- A = `11 1111 1111 1111 1111 1111 1001 1111(BIN) = 1,073,741,727`,
- B = `11 1111 1111 1111 1111 1111 0011 1111(BIN) = 1,073,741,631`, and
- C = `11 1111 1111 1111 1111 1111 0110 1111(BIN) = 1,073,741,679`,

the function should return 8, since there are 8 unsigned 30-bit integers conforming to A, B or C, namely:

- `11 1111 1111 1111 1111 1111 0011 1111(BIN) = 1,073,741,631`,
- `11 1111 1111 1111 1111 1111 0110 1111(BIN) = 1,073,741,679`,
- `11 1111 1111 1111 1111 1111 0111 1111(BIN) = 1,073,741,695`,
- `11 1111 1111 1111 1111 1111 1001 1111(BIN) = 1,073,741,727`,
- `11 1111 1111 1111 1111 1111 1011 1111(BIN) = 1,073,741,759`,
- `11 1111 1111 1111 1111 1111 1101 1111(BIN) = 1,073,741,791`,
- `11 1111 1111 1111 1111 1111 1110 1111(BIN) = 1,073,741,807`,
- `11 1111 1111 1111 1111 1111 1111 1111(BIN) = 1,073,741,823`.

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

- A, B and C are integers within the range [0..1,073,741,823].

```
// you can write to stdout for debugging purpo
// System.out.println("this is a debug message'

class Solution {
        public int solution(int A, int B, int
                int a = getCardinality(A);
                int b = getCardinality(B);
                int c = getCardinality(C);
                int ab = getCardinality(A | B)
                int ac = getCardinality(A | C)
                int bc = getCardinality(B | C)
                int abc = getCardinality(A | B

                return (int)((long)a + b + c -
        }

        private static int getCardinality(int
                int numberOfOneBits = BitSet.v
                int numberOfFreeBits = 30 - nu
                return 1 << numberOfFreeBits;
        }
}
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(log(A+B+C))**

collapse all Example tests

▼ example1 ✔ OK

example test

1. 0.004 s OK

collapse all Correctness tests

▼ simple ✔ OK

simple test

1. 0.004 s OK

▼ disjoint_bits ✔ OK

simple test

1. 0.008 s OK

▼ chain ✔ OK

simple test

1. 0.004 s OK

▼ incl_excl_rule1 ✔ OK

1. 0.004 s OK

▼ incl_excl_rule2 ✔ OK

1. 0.004 s OK

▼ extreme_min_result ✔ OK

1. 0.004 s OK

collapse all Performance tests

▼ low_stairs ✔ OK

1. 0.008 s **OK**

▼ high_stairs ✔ **OK**

1. 0.004 s **OK**

▼ large_result_a ✔ **OK**

1. 0.008 s **OK**

2. 0.008 s **OK**

▼ large_result_b ✔ **OK**

1. 0.004 s **OK**

▼ random ✔ **OK**

1. 0.008 s **OK**

2. 0.008 s **OK**

▼ max_result1 ✔ **OK**

1. 0.004 s **OK**

▼ max_result2 ✔ **OK**

1. 0.008 s **OK**