Codility_

CodeCheck Report: training4JFNKR-7UC

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
CountConformingBitmasks ⚠ Python	35 min	100%

Total score

100%

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 $0 \leq B < 2^{30}$.

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:

```
def solution(A, B, 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 the given integers.

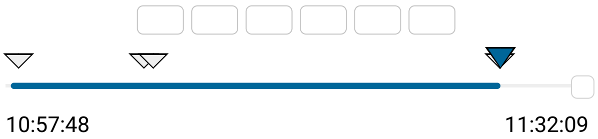
For example, for integers:

Solution

Programming language used:	Python	
Total time used:	35 minutes	?
Effective time used:	35 minutes	?
Notes:	*not defined yet*	

Task timeline

?



Code: 11:32:08 UTC, py, final, score: **100**

show code in pop-up

```
# you can write to stdout for debugging purpose
# print("this is a debug message")

def solution(A, B, C):
```

- 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].

```
    # Implement your solution here
    # n stand for the number of counted occuren

    nA = 0
    nB = 0
    nC = 0

    nAorB = 0
    nAorC = 0
    nBorC = 0
    nAorBorC = 0

    AorB = A | B
    AorC = A | C
    BorC = B | C
    AorBorC = A | B | C

    # For each bit position i, the code checks
    # operations and bitwise AND (&).
    # If the bit at position i in A is 0, nA is
    # If the bit at position i in B is 0, nB is
    # If the bit at position i in C is 0, nC is
    # The same process is followed for the comb

    for i in range(30):
        if ((A >> i) & 0x01) == 0:
            nA += 1
        if ((B >> i) & 0x01) == 0:
            nB += 1
        if ((C >> i) & 0x01) == 0:
            nC += 1

        if ((AorB >> i) & 0x01) == 0:
            nAorB += 1
        if ((AorC >> i) & 0x01) == 0:
            nAorC += 1
        if ((BorC >> i) & 0x01) == 0:
            nBorC += 1

        if ((AorBorC >> i) & 0x01) == 0:
            nAorBorC += 1

    # The result is obtained by adding the numb
    # -> 1 shifted left by the corresponding co

    # Subtracting the numbers of possibilities
    # -> 1 shifted left by the corresponding co

    # Adding the number of possibilities for A
    # 1 shifted left by nAorBorC: 1 << nAorBorC

    result = (1 << nA) - (1 << nAorB) - (1 << n

    return result

```

Analysis summary

The solution obtained perfect score.

Analysis

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

expand all **Example tests**

▶ example1 ✔ OK
example test

expand all **Correctness tests**

▶ simple ✔ OK
simple test

▶ disjoint_bits ✔ OK

simple test	
▶ chain simple test	✔ OK
▶ incl_excl_rule1	✔ OK
▶ incl_excl_rule2	✔ OK
▶ extreme_min_result	✔ OK
expand all	**Performance tests**
▶ low_stairs	✔ OK
▶ high_stairs	✔ OK
▶ large_result_a	✔ OK
▶ large_result_b	✔ OK
▶ random	✔ OK
▶ max_result1	✔ OK
▶ max_result2	✔ OK