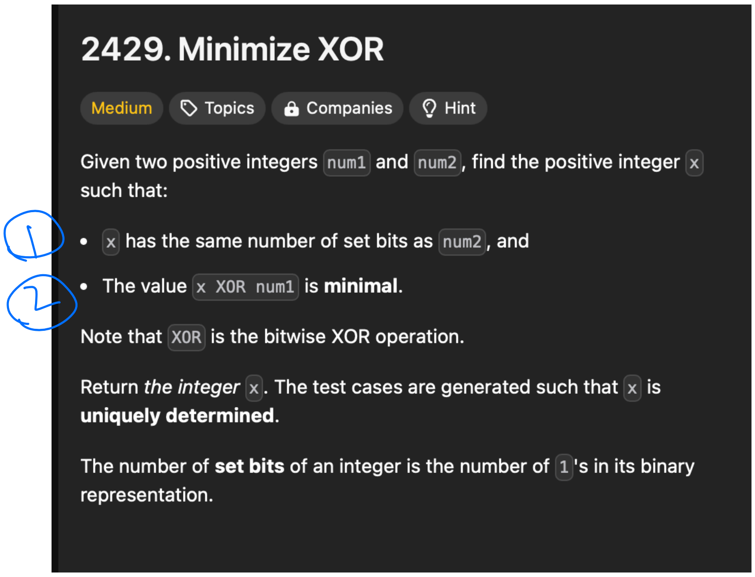


2429. Minimize XOR

Given two positive integers `num1` and `num2`, find the positive integer `x` such that:

- ① `x` has the same number of set bits as `num2`, and
- ② The value `x XOR num1` is **minimal**.

Note that `XOR` is the bitwise XOR operation.

Return *the integer* `x`. The test cases are generated such that `x` is **uniquely determined**.

The number of **set bits** of an integer is the number of `1`'s in its binary representation.

① : if num2 = 1011

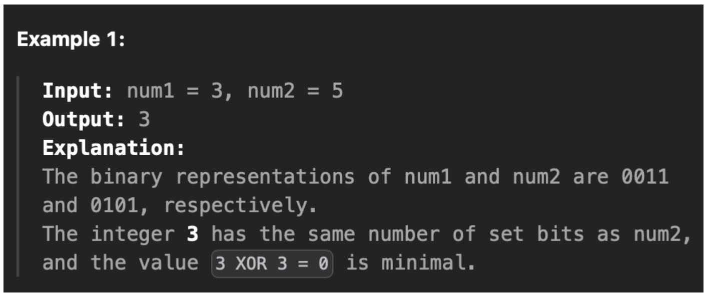
then x must be composed of 3 1s and 1 0s

② This step is about how to put the bits

for example: 0111, 1011, 1101, 1110

XOR → ^

A	B	A XOR B
0	0	0
1	1	0
0	1	1
1	0	1



Example 1:

```
Input: num1 = 3, num2 = 5
Output: 3
Explanation:
The binary representations of num1 and num2 are 0011
and 0101, respectively.
The integer 3 has the same number of set bits as num2,
and the value 3 XOR 3 = 0 is minimal.
```

$num1 = 3_{10} = 0011_2$

$num2 = 5_{10} = 0101_2$

→ $x = 2 \times 1s + 2 \times 0s$

$x \in \{1100, 1010, 1001, 0110, 0101, 0011\}$

x ^ num1

1100	0011	1111
1010		1001
1001		1010
0110		0101
0101		0110
0011		0000 ✓

→ $x = 0011_2$

$= 3_{10}$

Step 1 : count # of 1s and 0s in num2.

Step 2 : being greedy, most significant bits try to be the same as num1.

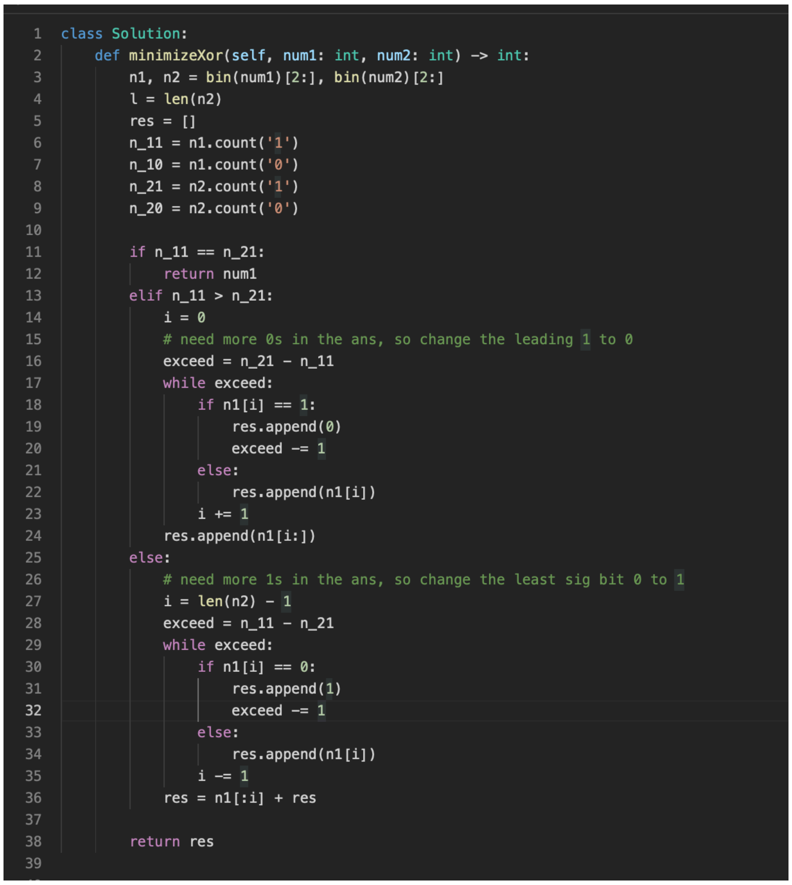
if x could be equal to num1, x^num1 will be the smallest. check the component of num1 and num2.

	num1		num2
1	1	3	2
0	3	1	2

0010 → 0011

1000 → 1001

1101 → 0101



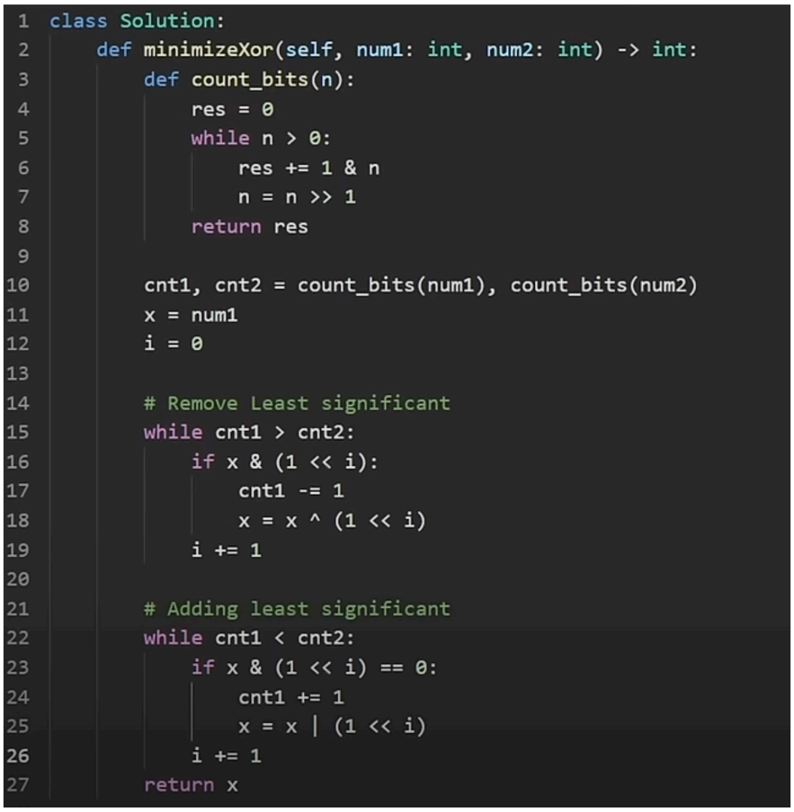
```python
class Solution:
    def minimizeXor(self, num1: int, num2: int) -> int:
        n1, n2 = bin(num1)[2:], bin(num2)[2:]
        l = len(n2)
        res = []
        n_11 = n1.count('1')
        n_10 = n1.count('0')
        n_21 = n2.count('1')
        n_20 = n2.count('0')

        if n_11 == n_21:
            return num1
        elif n_11 > n_21:
            i = 0
            # need more 0s in the ans, so change the leading 1 to 0
            exceed = n_21 - n_11
            while exceed:
                if n1[i] == 1:
                    res.append(0)
                    exceed -= 1
                else:
                    res.append(n1[i])
                i += 1
            res.append(n1[i:])
        else:
            # need more 1s in the ans, so change the least sig bit 0 to 1
            i = len(n2) - 1
            exceed = n_11 - n_21
            while exceed:
                if n1[i] == 0:
                    res.append(1)
                    exceed -= 1
                else:
                    res.append(n1[i])
                i -= 1
            res = n1[:i] + res

        return res
```

NOT working

should use bit operation



```python
class Solution:
    def minimizeXor(self, num1: int, num2: int) -> int:
        def count_bits(n):
            res = 0
            while n > 0:
                res += 1 & n
                n = n >> 1
            return res

        cnt1, cnt2 = count_bits(num1), count_bits(num2)
        x = num1
        i = 0

        # Remove Least significant
        while cnt1 > cnt2:
            if x & (1 << i):
                cnt1 -= 1
                x = x ^ (1 << i)
            i += 1

        # Adding least significant
        while cnt1 < cnt2:
            if x & (1 << i) == 0:
                cnt1 += 1
                x = x | (1 << i)
            i += 1
        return x
```

1° only 1s matters.

2°