Solution: Complement of Base 10 Number

Let's solve the Complement of Base 10 Number problem using the Bitwise Manipulation pattern.

We'll cover the following

- Statement
- Solution
  - Naive approach
  - Optimized approach using bitwise manipulation
    - Solution summary
    - Time complexity
    - Space complexity

Statement

For any n positive number in base 10, return the complement of its binary representation as an integer in base 10.

Constraints

- $0 \leq n \leq 10^9$

Solution

So far, you've probably brainstormed some approaches and have an idea of how to solve this problem. Let's explore some of these approaches and figure out which one to follow based on considerations such as time complexity and any implementation constraints.

Naive approach

To calculate the complement of any positive integer, we need to perform the following steps:

1. Convert the positive base 10 integer to its binary value.
2. Once we have the binary value, we can use a loop to incrementally convert each 1 to 0 and each 0 to 1.
3. Now that we have the complemented binary number, we can convert it to its respective base 10 value.

The conversion of binary values from 1 to 0 and 0 to 1 can take up to m iterations, where $O(m)$ is the length of binary value of the base 10 number. So, the time complexity of if this approach is $O(m)$.

Optimized approach using bitwise manipulation

This problem is a classic example of when to use the bitwise manipulation pattern. Using the XOR ($\wedge$) properties, we can get the complement of any n number.

The table below represents the complement of 4 possible combinations of 0s and 1s. Let's use this to determine the complements of our decimal values.

A	B	A ^ B

0	0	0
0	1	1
1	0	1
1	1	0

Let's take an input value of 42. Its binary equivalent will be as shown below.

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Input	42					
Binary	1	0	1	0	1	0
	32	16	8	4	2	1

Given the table on the left, we can convert the binary value to find its complement by taking the **XOR** with its all 1-bits set.

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Input	42					
Binary	1	0	1	0	1	0
All bits	1	1	1	1	1	1

The binary complement of 42 is 10101. Let's convert it back to its decimal value.

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Input	42					
Binary	1	0	1	0	1	0
All bits	1	1	1	1	1	1
Complement	0	1	0	1	0	1
	32	16	8	4	2	1

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

We get 21 by taking the complement of the binary value of 42.

Let's see another example for a different value.

Input	42					
Binary	1	0	1	0	1	0
All bits	1	1	1	1	1	1
Complement	0	1	0	1	0	1
Output	21					

The primary goal here is to flip all the bits of a number. Here's how we'll implement this algorithm:

1. Calculate the number of bits required to store a certain positive integer. We can get this by rounding down `(Math.log(num) / Math.log(2)` and adding 1.
2. Create an all 1-bit set against the number of bits of the input value. For example, if we need 4 bits to represent an integer, all its set bits will be `1111`. The decimal value of `1111` $= 2^3 + 2^2 + 2^1 + 2^0 = 15$, which is equal to $2^{bits} - 1 = 2^4 - 1$. So, we can use this formula to get the required number.
3. Flip all occurrences of 1s to 0s and 0s to 1s by computing the XOR operation between the two numbers.

Let's look at the code for this solution below:

Java

```
class ComplementNumber {
    public static int findBitwiseComplement(int num) {
        // if the value of num is 0, return 1
        if (num == 0) {
            return 1;
        }
        // converting the value into its binary representation and
        // counting the number of bits required by this number
        int bitCount = (int) Math.floor((int)(Math.log(num) / Math.log(2))) + 1;

        // computing the all bits set of the number
        int allBitsSet = (int) Math.pow(2, bitCount) - 1;
        // flipping all bits of number by taking xor with allBitsSet
        return num ^ allBitsSet;
    }
    public static void main(String[] args) {
        int[] decimalValues = {42, 233, 100, 999999, 54};
        for (int i = 0; i < decimalValues.length; i++) {
            System.out.println(findBitwiseComplement(decimalValues[i]));
            System.out.println(new String(new char[100]).replace('\0', '-'));
        }

    }

}
```

Complement of Base 10 Number

Solution summary

To recap, the solution to this problem can be divided into the following parts:

1. We calculate the number of bits required to store any positive number.
2. We create an all bits set against it and flip all occurrences of 1s to 0s and 0s to 1s by computing the XOR operation.
3. By converting the binary value back to base-10, we got our final answer.

Time complexity

To take the complement, we first calculate the number of bits required for the binary representation of the given integer and then take XOR, which are constant time operations. So, the time complexity of this solution is $O(1)$.

Space complexity

The space complexity of this solution is $O(1)$ because at any time, we have a positive decimal number to have it converted into its complement.

Back

Next