1318. Minimum Flips to Make a OR b Equal to c

Medium | Bit Manipulation

Problem Description

In this problem, we are given three positive integers `a`, `b`, and `c`. We need to determine the minimum number of flips required to make the bitwise OR of `a` and `b` equal to `c`. A flip means changing a single bit from 1 to 0 or from 0 to 1 in the binary representation of `a` or `b`.

To understand the problem better, let's consider what a bitwise OR operation does. For each bit position in two numbers, the OR operation yields 1 if either of the corresponding bits in the two numbers is 1; otherwise, it yields 0. Therefore, for `a | b` to become `c`, each bit position in `a` or `b` must be manipulated so that this rule holds true.

Intuition

The intuition behind the solution is to compare the bits of `a`, `b`, and `c` in each corresponding position and determine if a flip is needed. To do this, we iterate through each bit position of the given numbers from the least significant bit to the most significant bit.

For each bit position `i`, if the ith bit of `a` OR `b` does not match the ith bit of `c`, we must flip bits in `a` or `b` to match `c`. There are a couple of scenarios to consider:

1. If both `a` and `b` have 1 at the ith bit, and `c` has 0, we must flip both bits in `a` and `b`, resulting in two flips.
2. In any other case where `a | b` and `c` do not match (for example, one of `a` or `b` is 1 and `c` is 0, or both `a` and `b` are 0 and `c` is 1), a single flip is sufficient.

Solution Approach

The algorithm uses a straightforward approach to solve the problem by performing bit manipulation to tally the flips required to make `a | b == c`. The solution does not require any complex data structures or design patterns, rather it capitalizes on bitwise operations and logical reasoning. Here's the breakdown of the solution approach:

1. Initialize a variable `ans` to keep track of the number of flips needed.
2. Iterate over each bit position (from 0 to 29) to examine individual bits of `a`, `b`, and `c`. The range is up to 30 to cover typical 32-bit integers without the sign bit.
3. For each iteration, use the right-shift operator `>>` to move the ith bit to the least significant bit position, then use the bitwise AND operator `&` with 1 to extract the value of that bit for `a`, `b`, and `c`. This results in three variables `x`, `y`, and `z` representing the ith bits of `a`, `b`, and `c`, respectively.
4. Use the bitwise OR operator `|` to determine the result of `x | y` and compare it to `z`. If they are not equal, flips are required:
   - If both `x` and `y` are 1 and `z` is 0, it means both bits in `a` and `b` must be flipped, hence `ans` is incremented by 2.
   - In any other case where a flip is needed (such as when one of `a` or `b` has a 1 and `c` has a 0, or both `a` and `b` have a 0 and `c` has a 1), only 1 flip is necessary, so `ans` is incremented by 1.
5. The loop continues until all bits have been considered.
6. The `ans` variable, which now holds the total number of flips needed, is returned.

Here is the critical part of the code that encapsulates the described solution approach:

```
for i in range(30):
    x, y, z = a >> i & 1, b >> i & 1, c >> i & 1
    if x | y != z:
        ans += 2 if x == 1 and y == 1 else 1
```

This solution effectively uses the fundamental principles of bitwise operations to solve the problem in an efficient and direct manner.

Example Walkthrough

Let's walk through an example to illustrate the solution approach using the given content.

Assume the input for the problem is `a = 5`, `b = 3`, and `c = 2`. In binary, these numbers are represented as follows:

```
a = 5 = 101 (binary)
b = 3 = 011 (binary)
c = 2 = 010 (binary)
```

We need to determine the minimum number of flips required to make `a | b` equal to `c`.

The bitwise OR of `a` and `b` (`a | b`) is `101 | 011 = 111 (binary)`, which equals `7` in decimal. This does not match `c = 2 (010 in binary)`, so we need flips to make it equal.

Now, let's apply the solution step by step:

1. We initialize a variable `ans` to 0 to keep count of the number of flips needed.
2. We start iterating through each bit position from the 0th to the 29th (we won't actually go up to 29 since our numbers are small).
3. For each iteration (for each bit), we right-shift `a`, `b`, and `c` by `i` positions and then `AND` them with `1` to extract the ith bit.

For example, at `i = 0` (least significant bit):

- `x = a >> 0 & 1` is `1` (from `101` last bit)
- `y = b >> 0 & 1` is `1` (from `011` last bit)
- `z = c >> 0 & 1` is `0` (from `010` last bit)

4. We then check if `x | y` equals `z`. If it does not, we find the number of flips required.

Continuing our example:

- At `i = 0`: `x | y` is `1` which doesn't equal `z (0)`. Both `x` and `y` are `1` and `z` is `0`, so we must flip both bits from `1` to `0`. Thus, we add `2` to `ans`.

We carry forward with the process:

- At `i = 1`: `x = 0`, `y = 1`, `z = 1`. Here, `x | y` equals `z`, so no flip is needed.
- At `i = 2`: `x = 1`, `y = 0`, `z = 0`. Here, `x | y` does not equal `z`, and since only `x` is `1`, we add `1` flip to `ans`.

5. Since the example uses small numbers, we don't need to go up to `i = 29`.

After checking all the bits, we find that the `ans` became `3` (two flips for `i = 0` and one flip for `i = 2`).

Thus, to make `a | b` equal to `c`, a minimum of 3 flips are needed.

This step-by-step walkthrough shows how bitwise operations are used to find the number of flips required to match the bitwise OR of two numbers to a third number. The process is simple yet effective, leveraging basic operations to arrive at the solution.

Solution Implementation

Python

```python
class Solution:
    def minFlips(self, num_a: int, num_b: int, num_c: int) -> int:
        # Initialize answer to count the minimum number of flips
        min_flips = 0
        # Iterate through each bit position (0 to 29) since the problem states 32-bit integers
        for i in range(30):
            # Extract the i-th bit of num_a, num_b, and num_c
            bit_a = num_a >> i & 1
            bit_b = num_b >> i & 1
            bit_c = num_c >> i & 1

            # If the OR of bit_a and bit_b is not equal to bit_c, we need to flip bits
            if bit_a | bit_b != bit_c:
                # If both bit_a and bit_b are set (1), we need to flip both to make the OR result match bit_c
                # which is 0 in this case. This counts as two flips.
                if bit_a == 1 and bit_b == 1:
                    min_flips += 2
                # Otherwise, we only need to flip one bit to match the OR result to bit_c.
                else:
                    min_flips += 1
        # Return the total number of flips required
        return min_flips
```

Java

```java
class Solution {

    // This method calculates the minimum number of flips required to make
    // bitwise OR of 'a' and 'b' equal to 'c'.
    public int minFlips(int a, int b, int c) {
        // Initialize the variable to store the count of minimum flips.
        int minFlipsCount = 0;

        // Iterate over each bit position from 0 to 29 (30 bits in total assuming 32-bit integers)
        // to compare the bits in 'a', 'b', and 'c'.
        for (int i = 0; i < 30; ++i) {
            // Extract the i-th bit from 'a', 'b', and 'c'.
            int bitA = (a >> i) & 1;
            int bitB = (b >> i) & 1;
            int bitC = (c >> i) & 1;

            // Check if the result of 'bitA | bitB' doesn't match 'bitC'.
            if ((bitA | bitB) != bitC) {
                // If both bits in 'a' and 'b' are 1, and 'c' is 0, we need 2 flips.
                // Otherwise, we need just 1 flip (either from 'a' or 'b', or to match a 1 in 'c').
                minFlipsCount += (bitA == 1 && bitB == 1) ? 2 : 1;
            }
        }

        // Return the total count of flips required.
        return minFlipsCount;
    }
}
```

C++

```cpp
class Solution {
public:
    // Method to calculate the minimum number of flips to make a | b equal to c
    int minFlips(int a, int b, int c) {
        int flipsCount = 0; // Variable to hold the total number of bit flips required

        // Loop through each bit position (from 0 to 29)
        for (int i = 0; i < 30; ++i) {
            // Extract the i-th bits from a, b, and c
            int bitA = (a >> i) & 1;
            int bitB = (b >> i) & 1;
            int bitC = (c >> i) & 1;

            // Check if the current bits of a OR b differ from the corresponding bit of c
            if ((bitA | bitB) != bitC) {
                // If both bits in a and b are 1 then we need to flip both (2 flips),
                // otherwise, we need to flip only one bit
                flipsCount += (bitA == 1 && bitB == 1) ? 2 : 1;
            }
        }

        // Return the total number of flips required
        return flipsCount;
    }
};
```

TypeScript

```typescript
// Function to calculate the minimum number of flips to make a | b equal to c
function minFlips(a: number, b: number, c: number): number {
    let flipsCount: number = 0; // Variable to hold the total number of bit flips required

    // Loop through each bit position (from 0 to 29)
    for (let i = 0; i < 30; ++i) {
        // Extract the i-th bits from a, b, and c
        const bitA: number = (a >> i) & 1;
        const bitB: number = (b >> i) & 1;
        const bitC: number = (c >> i) & 1;

        // Check if the current bits of a OR b differ from the corresponding bit of c
        if ((bitA | bitB) !== bitC) {
            // If both bits in a and b are 1 then we need to flip both (2 flips),
            // otherwise, we flip only one bit
            flipsCount += (bitA === 1 && bitB === 1) ? 2 : 1;
        }
    }

    // Return the total number of flips required
    return flipsCount;
}
```

```python
class Solution:
    def minFlips(self, num_a: int, num_b: int, num_c: int) -> int:
        # Initialize answer to count the minimum number of flips
        min_flips = 0
        # Iterate through each bit position (0 to 29) since the problem states 32-bit integers
        for i in range(30):
            # Extract the i-th bit of num_a, num_b, and num_c
            bit_a = num_a >> i & 1
            bit_b = num_b >> i & 1
            bit_c = num_c >> i & 1

            # If the OR of bit_a and bit_b is not equal to bit_c, we need to flip bits
            if bit_a | bit_b != bit_c:
                # If both bit_a and bit_b are set (1), we need to flip both to make the OR result match bit_c
                # which is 0 in this case. This counts as two flips.
                if bit_a == 1 and bit_b == 1:
                    min_flips += 2
                # Otherwise, we only need to flip one bit to match the OR result to bit_c.
                else:
                    min_flips += 1
        # Return the total number of flips required
        return min_flips
```

Time and Space Complexity

The time complexity of the provided code is `O(1)`. While there is a loop that iterates up to 30 times, this is a constant factor because the size of an integer in most languages is fixed (here it's assumed to be 32 bits, and the loop runs for up to 30 significant bits). Thus, the number of iterations does not depend on the size of the input, making it constant-time complexity.

The space complexity of the code is also `O(1)`. No additional space is used that grows with the size of the input. The variables `x`, `y`, `z`, and `ans` use a fixed amount of space.