1018. Binary Prefix Divisible By 5

Easy Array

Leetcode Link

Problem Description

In this problem, we are given an array of binary digits, `nums`, which is an array of 0s and 1s. Our task is to iterate through this array to form a series of numbers `x_i` such that `x_i` is the binary number formed by the digits from `nums[0]` to `nums[i]`, with `nums[0]` as the most significant bit. For each `x_i`, we need to determine whether it is divisible by 5. The result should be an array of boolean values (`true` or `false`) where each boolean value represents whether the corresponding `x_i` is divisible by 5.

For example, given `nums = [1, 0, 1]`, we have:

- `x_0` = 1 (1 in binary), and since 1 is not divisible by 5, the result is `false`.
- `x_1` = 2 (10 in binary), and since 2 is not divisible by 5, the result is `false`.
- `x_2` = 5 (101 in binary), and since 5 is divisible by 5, the result is `true`.

So the output for this input would be `[false, false, true]`.

Intuition

The key to solving this problem is to iteratively build the number `x_i` by appending each digit from the array `nums`. Since the problem only asks for the divisibility by 5, we don't actually need to store the entire number `x_i`, which could become very large and might cause issues with integer overflow. Instead, we can take advantage of mathematical properties of divisibility by 5 and binary numbers.

The approach is to use modular arithmetic. Since we're interested in divisibility by 5, we'll use `% 5` to keep track of the remainder of `x_i`. In each iteration, we calculate the new `x` by shifting the current value of `x` one position to the left (equivalent to multiplying by 2) and adding the current digit `v`. This is equivalent to converting from binary to decimal incrementally. Then we immediately apply `% 5` to get the remainder.

If at any step, `x` is `0` after the `% 5` operation, this means the current `x_i` is divisible by 5 and we add `true` to the resulting list, `false` otherwise.

By doing so for every digit in the input array, we efficiently and correctly fill the answer list with the divisibility results for every `x_i`.

Solution Approach

The solution for this problem involves the use of a loop to process each digit in the binary array. The code uses a single integer variable `x` to keep track of the decimal value of the current binary number being evaluated. The answer is built up in the list `ans` with a corresponding boolean value indicating the divisibility of each number by 5.

Here is a step-by-step breakdown of the implementation:

1. Initialize `x` to 0, which will represent the current value of the decimal number as we scan through the binary digits.
2. Create an empty list `ans` to store the results (boolean values).

To process the digits and to keep the implementation efficient, the solution uses the following pattern:

- For each digit `v` in the input array `nums`:
  - Shift the previous value of `x` to the left by one position. This is done by using the bitwise left shift operator "`<<`". In binary terms, this operation is equivalent to appending a zero to the binary number (or multiplying by 2 in decimal terms).
  - Combine the shifted value of `x` with the current digit `v` using the bitwise OR operator "`|`". This is equivalent to appending the current binary digit to the right-most end of the existing binary number.
  - Apply a modulo operation "`%`" with 5 to keep only the remainder of `x` divided by 5. This step ensures that the value of `x` stays within a manageable range and directly gives us the information we need regarding divisibility by 5.
  - Check if `x` is equal to 0, if so, it means that the current number is divisible by 5; hence append `True` to `ans`. Otherwise, append `False`.

Finally, the list `ans` is returned, which contains the boolean divisibility result for each position in the binary array `nums`.

The key algorithmic concepts involved include bitwise operations (to manipulate the binary representation directly), modular arithmetic (to manage the size of numbers and determine divisibility), and loop iteration (to traverse the array).

Example Walkthrough

Let us illustrate the solution approach using a small example. Consider the binary array `nums = [1, 1, 1, 0, 1]`.

We are tasked with evaluating the divisibility of the corresponding binary number at each position in the array by 5. Let's walk through the problem step by step as per our solution approach:

1. Initialize `x` to 0. This will hold the decimal value represented by the binary digits considered so far.
2. Create an empty list `ans` to store the results as boolean values.

Next, we iterate over the digits in `nums` and perform bitwise operations and modular arithmetic:

- Start with the first digit `1`:
  - Shift `x` left (x becomes 0 << 1 = 0).
  - Add the current digit (x becomes 0 | 1 = 1).
  - Apply modulo 5 (x becomes 1 % 5 = 1).
  - Since x is not 0, append `False` to `ans`.
- Next digit `1`:
  - Shift `x` left (x becomes 1 << 1 = 2).
  - Add the current digit (x becomes 2 | 1 = 3).
  - Apply modulo 5 (x becomes 3 % 5 = 3).
  - Since x is not 0, append `False` to `ans`.
- Next digit `1`:
  - Shift `x` left (x becomes 3 << 1 = 6).
  - Add the current digit (x becomes 6 | 1 = 7).
  - Apply modulo 5 (x becomes 7 % 5 = 2).
  - Since x is not 0, append `False` to `ans`.
- Next digit `0`:
  - Shift `x` left (x becomes 2 << 1 = 4).
  - Add the current digit (x becomes 4 | 0 = 4).
  - Apply modulo 5 (x becomes 4 % 5 = 4).
  - Since x is not 0, append `False` to `ans`.
- Last digit `1`:
  - Shift `x` left (x becomes 4 << 1 = 8).
  - Add the current digit (x becomes 8 | 1 = 9).
  - Apply modulo 5 (x becomes 9 % 5 = 4).
  - Since x is not 0, append `False` to `ans`.

After processing all digits, our `ans` list stands as `[False, False, False, False, False]` which correctly represents that none of the binary numbers formed at each index is divisible by 5.

Hence, the final output for input `nums = [1, 1, 1, 0, 1]` is `[False, False, False, False, False]`. This walkthrough clearly demonstrates how each step of the solution approach is executed to solve the problem efficiently.

Python Solution

```python
class Solution:
    def prefixesDivBy5(self, nums: List[int]) -> List[bool]:
        # Initialize an empty list to store the results
        is_divisible = []

        # Initialize a variable to represent the current prefix number
        prefix_number = 0

        # Iterate over the binary digits in the input list
        for digit in nums:
            # Shift the current prefix number left by 1 bit and add the new digit
            # Then take the modulo by 5 to keep the number manageable and check divisibility
            prefix_number = ((prefix_number << 1) | digit) % 5

            # Append True to the result list if the current prefix number is divisible by 5, otherwise append False
            is_divisible.append(prefix_number == 0)

        # Return the list containing divisibility status of prefixes
        return is_divisible
```

Java Solution

```java
class Solution {
    // This method takes an array of binary digits (0s and 1s) and checks if the
    // binary number formed by the prefix of the array is divisible by 5.
    public List<Boolean> prefixesDivBy5(int[] binaryArray) {
        // This list will store the results as Booleans corresponding to each prefix.
        List<Boolean> results = new ArrayList<>();

        // This variable will hold the current value of the binary number.
        int currentValue = 0;

        // Iterate over the binary digits in the array.
        for (int digit : binaryArray) {
            // Left shift the current value to make room for the new digit,
            // and then OR it with the new digit. After that,
            // we take modulo 5 to keep the currentValue within range and
            // also to check divisibility by 5.
            currentValue = ((currentValue << 1) | digit) % 5;

            // If currentValue is 0, that means the binary number
            // formed by the current prefix is divisible by 5.
            results.add(currentValue == 0);
        }
        // Return the list of Booleans representing divisibility by 5.
        return results;
    }
}
```

C++ Solution

```cpp
class Solution {
public:
    // Function to return a vector of boolean values indicating
    // whether the binary number formed by the given prefix of
    // bits is divisible by 5
    vector<bool> prefixesDivBy5(vector<int>& nums) {
        vector<bool> results; // Vector to store the results
        int currentValue = 0; // Variable to keep track of the binary value converted to int

        // Iterate over every bit in the given vector
        for (int bit : nums) {
            // Shift currentValue one bit to the left and add the current bit.
            // The `% 5` operation is used to avoid an overflow by keeping only
            // the remainder of the current value when divided by 5, which is sufficient
            // to determine divisibility by 5 for any subsequent steps.
            currentValue = ((currentValue << 1) | bit) % 5;

            // Check if currentValue is divisible by 5 and add the result to the results vector
            results.push_back(currentValue == 0);
        }
        // Return the filled results vector
        return results;
    }
};
```

Typescript Solution

```typescript
function prefixesDivBy5(nums: number[]): boolean[] {
    // Initialize the result array to hold boolean values
    const results: boolean[] = [];

    // Initialize a variable to hold the binary number being formed
    let currentPrefix = 0;

    // Loop through each number in the input array
    for (const num of nums) {
        // Left shift the current prefix by 1 and add the current number
        // Use modulo 5 to avoid large integer issues
        currentPrefix = ((currentPrefix << 1) | num) % 5;

        // If the current prefix is divisible by 5, add true to the results, otherwise false
        results.push(currentPrefix === 0);
    }

    // Return the array of boolean values indicating divisibility by 5
    return results;
}
```

Time and Space Complexity

The time complexity of the given code is `O(N)`, where `N` is the length of the input list `nums`. This is because the code iterates through each element of the list exactly once, performing a constant amount of work for each element by shifting the number `x`, performing a bitwise OR operation with the current value `v`, and then taking the modulo with 5.

The space complexity of the given code is `O(N)`, since it maintains a list `ans` that contains one boolean for each element in the input list. Thus, the size of the auxiliary space used by the program scales linearly with the input size.