Daily Coding Problem #40

Problem

This problem was asked by Google.

Given an array of integers where every integer occurs three times except for one integer, which only occurs once, find and return the non-duplicated integer.

For example, given [6, 1, 3, 3, 3, 6, 6], return 1. Given [13, 19, 13, 13], return 19.

Do this in O(N) time and O(1) space.

Solution

We can find the unique number in an array of *two* duplicates by XORing all the numbers in the array. What this does is cancel out all the bits that have an even number of 1s, leaving only the unique (odd) bits out.

Let's try to extend this technique to three duplicates. Instead of cancelling out all the bits with an even number of bits, we want to cancel those out that have a number of bits that are multiple of three.

Let's assume all integers fit in 32 bits. Then let's create an array 32 zeroes long, and when iterating over each number in our array, we can add up all the bits to its proper spot in the array. Finally, we'll go over each bit in the array and make it equal to itself modulo 3. This means that any bit that has been set some multiple of 3 times will effectively be cleared, leaving only the bit from the unique number.

```
def find_unique(arr):
    result_arr = [0] * 32
```

```
    for num in arr:
        for i in range(32):
            bit = num >> i & 1
            result_arr[i] = (result_arr[i] + bit) % 3

    result = 0
    for i, bit in enumerate(result_arr):
        if bit:
            result += 2 ** i

    return result
```

This runs in linear time, since we iterate over the array once, and in constant space, since we initialize an array of constant size.