Detailed explanation and generalization of the bitwise operation method for single numbers

fun4LeetCode 10924 Last Edit: October 25, 2018 2:19 PM 80.5K VIEWS

`I -- Statement of our problem`

"Given an array of integers, every element appears `k` (`k > 1`) times except for one, which appears `p` times (`p >= 1, p % k != 0`). Find that single one."

`II -- Special case with 1-bit numbers`

As others pointed out, in order to apply the bitwise operations, we should rethink how integers are represented in computers -- by bits. To start, let's consider only one bit for now. Suppose we have an array of **1-bit** numbers (which can only be `0` or `1`), we'd like to count the number of `1` 's in the array such that whenever the counted number of `1` reaches a certain value, say `k` , the count returns to zero and starts over (in case you are curious, this `k` will be the same as the one in the problem statement above). To keep track of how many `1` 's we have encountered so far, we need a counter. Suppose the counter has `m` bits in binary form: `xm, ..., x1` (from most significant bit to least significant bit). We can conclude at least the following four properties of the counter:

1. There is an initial state of the counter, which for simplicity is zero;
2. For each input from the array, if we hit a `0` , the counter should remain unchanged;
3. For each input from the array, if we hit a `1` , the counter should increase by one;
4. In order to cover `k` counts, we require `2^m >= k` , which implies `m >= logk` .

Here is the key part: how each bit in the counter (`x1` to `xm`) changes as we are scanning the array. Note we are prompted to use bitwise operations. In order to satisfy the second property, recall what bitwise operations will not change the operand if the other operand is `0` ? Yes, you got it: `x = x | 0` and `x = x ^ 0` .

Okay, we have an expression now: `x = x | i` or `x = x ^ i` , where `i` is the scanned element from the array. Which one is better? We don't know yet. So, let's just do the actual counting.