Daily Coding Problem #140

Problem

This problem was asked by Facebook.

Given an array of integers in which two elements appear exactly once and all other elements appear exactly twice, find the two elements that appear only once.

For example, given the array `[2, 4, 6, 8, 10, 2, 6, 10]`, return 4 and 8. The order does not matter.

Follow-up: Can you do this in linear time and constant space?

Solution

The naive solution here would be to use a dictionary to keep track of the counts of all the numbers, and then to iterate over the dictionary and just return the entries with count 1. That would look like this:

```python
from collections import defaultdict

def array_two_elements_naive(arr):
    d = defaultdict(int)
    for num in arr:
        d[num] += 1

    result = []
    for num, count in d.items():
        if count == 1:
            result.append(num)
    return result
```

However, this takes a proportional amount of space as the number of elements in the array, which means that it's O(n) space as well as O(n) time. We can use the following trick to use only constant space:

- Recall from Daily Coding Problem #40 that we can find the single unique number from a list of duplicate numbers by XORing all the numbers together.
- We can XOR all the numbers in our input array. Since all duplicates will cancel out, we'll end up getting the XOR of the two numbers we're looking for.
- However, we need to untangle the actual real numbers from this XOR'd result. The key here is to find a set bit the XOR'd result.
- Since the two numbers are unique and thus distinct, there must be a set bit. Let's call it `i`. Since this bit ended up set, that also means that the two numbers we're looking for differ at this bit. Let's call the unique number with `i` set x and the unique number with `i` not set y.
- Now we go through the array again. For each number, there are two cases:

    - Bit `i` is set on that number. If so, then it might be x, so let's group it in a bucket.
    - Bit `i` is not set on that number. If so, then it might be y, so let's group it in another bucket.

- Then we can XOR all the numbers in each bucket (like in Problem #40) to get the two numbers we're looking for.

```
def array_two_elements(arr):
    xor = 0
    for num in arr:
        xor = xor ^ num

    # Get rightmost set bit
    xor = xor & -xor

    rets = [0, 0]
    for num in arr:
        if num & xor:
            rets[0] = rets[0] ^ num
        else:
            rets[1] = rets[1] ^ num
    return rets
```

Addendum: Why does `xor & -xor` get the rightmost set bit? This is a neat bit hack that uses Two's complement to isolate the rightmost set bit. In two's complement, `-x = ~x + 1`. `~x` inverts all the bits in a number, and adding one to it will carry all the ones up until the first rightmost set bit in the original number x. Then, since we AND it with the original number, all the bits end up zero except for the rightmost set bit, which is what we're looking for!