🎉 Master algorithms together on Binary Search! Create a room, invite your friends, and race to finish the problems. ✕

Daily Coding Problem #88

Problem

This question was asked by ContextLogic.

Implement division of two positive integers without using the division, multiplication, or modulus operators. Return the quotient as an integer, ignoring the remainder.

Solution

We can start by trying the simplest solution. Define x as the dividend and y as the divisor. To get the quotient, we need to ask how many times we can subtract y from x until the remainder is less than y. The number of times we subtract is the resulting quotient x/y. The time complexity of this brute force approach is on the order of `x / y`, which can be very high, for example if x is `2^31 - 1` and y is `1`.

Let's instead think about how to perform division on paper. Recall grade-school long division, where we take consider the left-most digit that can be divided by the divisor. At each step, the quotient becomes the first digit of the result, and we subtract the product from the dividend to get the remainder. The remainder is initially the value x. We can abstract this process into subtracting the largest multiple of `y * 10^d` from the remainder, where `d` is the place of the digit (`d=0` for the zeros place). Then we add the multiple times `10^d` to our result.

This process would be straightforward if we had the modulus or multiplication operators. However, we instead can take advantage of the bit shift operators in order to multiply by powers of two, since `a<<z` results in a multiplied by `2^z` (e.g. `3<<2 = 12`. Now, we can find the largest `y * 2^d` that fits within the remainder. As we do in long division, we decrease the possible value of d in each iteration. We start by finding the largest value of `y * 2^d <= x`, then test `y * 2^d, y * 2^(d-1), ...` until the remainder is less than y.

For example:

```
x = 31, y = 3 => x = 1001, y = 0011
11111 - 0011 <<3 = 0111, quotient = 1<<3
0111 - 0011<<1 = 0001, quotient = 1<<3 + 1<<1
1<<3 + 1<<1 = 1010 = 10
```

Here is the Python implementation:

```
def divide(x, y):
    if y == 0:
        raise ZeroDivisionError('division by zero')

    quotient = 0
    power = 32           # Assume 32-bit integer
    yPower = y << power  # Initial y^d value is y^32
    remainder = x        # Initial remainder is x
    while remainder >= y:
        while yPower > remainder:
            yPower >>= 1
            power -= 1
        quotient += 1 << power
        remainder -= yPower

    return quotient
```

The time complexity of this solution is `O(N)`, where `N` is the number of bits used to represent `x/y`, assuming shift and add operations take `O(1)` time.

© Daily Coding Problem 2019 Privacy Policy Terms of Service Press