Divide Two Integers

Divide two integers without using multiplication, division and mod operator.

If it is overflow, return MAX_INT.

Solution 1

In this problem, we are asked to divide two integers. However, we are not allowed to use division, multiplication and mod operations. So, what else can we use? Yeah, bit manipulations.

Let's do an example and see how bit manipulations work.

Suppose we want to divide `15` by `3`, so `15` is `dividend` and `3` is `divisor`. Well, division simply requires us to find how many times we can subtract the `divisor` from the the `dividend` without making the `dividend` negative.

Let's get started. We subtract `3` from `15` and we get `12`, which is positive. Let's try to subtract more. Well, we **shift** `3` to the left by `1` bit and we get `6`. Subtracting `6` from `15` still gives a positive result. Well, we shift again and get `12`. We subtract `12` from `15` and it is still positive. We shift again, obtaining `24` and we know we can at most subtract `12`. Well, since `12` is obtained by shifting `3` to left twice, we know it is `4` times of `3`. How do we obtain this `4`? Well, we start from `1` and shift it to left twice at the same time. We add `4` to an answer (initialized to be `0`). In fact, the above process is like `15 = 3 * 4 + 3`. We now get part of the quotient (`4`), with a remainder `3`.

Then we repeat the above process again. We subtract `divisor = 3` from the remaining `dividend = 3` and obtain `0`. We know we are done. No shift happens, so we simply add `1 << 0` to the answer.

Now we have the full algorithm to perform division.

According to the problem statement, we need to handle some exceptions, such as overflow.

Well, two cases may cause overflow:

1. `divisor = 0`;
2. `dividend = INT_MIN` and `divisor = -1` (because `abs(INT_MIN) = INT_MAX + 1`).

Of course, we also need to take the sign into considerations, which is relatively easy.

Putting all these together, we have the following code.

```
class Solution {
public:
    int divide(int dividend, int divisor) {
        if (!divisor || (dividend == INT_MIN && divisor == -1))
            return INT_MAX;
        int sign = ((dividend < 0) ^ (divisor < 0)) ? -1 : 1;
        long long dvd = labs(dividend);
        long long dvs = labs(divisor);
        int res = 0;
        while (dvd >= dvs) {
            long long temp = dvs, multiple = 1;
            while (dvd >= (temp << 1)) {
                temp <<= 1;
                multiple <<= 1;
            }
            dvd -= temp;
            res += multiple;
        }
        return sign == 1 ? res : -res;
    }
};
```

written by jianchao.li.fighter original link here

Solution 2

Long division in binary: The outer loop reduces n by at least half each iteration. So It has O(log N) iterations. The inner loop has at most log N iterations. So the overall complexity is O((log N)^2)

```
typedef long long ll;

int divide(int n_, int d_) {
    ll ans=0;
    ll n=abs((ll)n_);
    ll d=abs((ll)d_);
    while(n>=d){
        ll a=d;
        ll m=1;
        while((a<<1) < n){a<<=1;m<<=1;}
        ans+=m;
        n-=a;
    }
    if((n_<0&&d_>=0)||(n_>=0&&d_<0))
        return -ans;
    return ans;
}
```

written by lucastan original link here

Solution 3

```
class Solution:
# @return an integer
def divide(self, dividend, divisor):
    positive = (dividend < 0) is (divisor < 0)
    dividend, divisor = abs(dividend), abs(divisor)
    res = 0
    while dividend >= divisor:
        temp, i = divisor, 1
        while dividend >= temp:
            dividend -= temp
            res += i
            i <<= 1
            temp <<= 1
    if not positive:
        res = -res
    return min(max(-2147483648, res), 2147483647)
```

written by tusizi original link here

From Leetcoder.