2's complement

by HackerRank

Editorial by Tanzir5

Let's first solve the problem for only the positive integers.

For Positive Numbers:

We will try to solve the problem recursively. Let $F(n)$ be the number of $1s$ written down if we write the numbers from 0 to n.

For even numbers,

$$\text{F(n)} = \text{number of 1s in binary representation of n} + \text{F(n-1)}$$

This is pretty simple. But for the odd numbers, we can find an elegant recurrence. Say you have the number 101001(Binary) . Let's at first see how many $1s$ will be needed by the leftmost 5 bits if we write down all the numbers from 0 to 101001. The leftmost five bits can be anything ranging from 0 to 10100. With each of them, the least significant bit can be either 1 or 0. So how many $1s$ will be needed by the leftmost five bits ? The answer is twice the number of ones we would need to write down for all the numbers from 0 to 10100 i.e.

$$\text{f(10100)*2}$$

For the least significant bit (LSB), half of the numbers will have the LSB on. So this is our recurrence relation:

```
if(n is odd)
    F(n) = F(n>>1)*2 + ((n+1)/2)
else
    F(n) = number of 1s in binary representation of n + F(n-1)
```

So to calculate the number of $1s$ written down if we write down every number from a to b, the solution will be:

$$\text{F(b) - F(a-1)}$$

General solution:

So, now let's see how we can solve the problem for negative numbers. Let's first understand how 2's complement works for negative numbers. According to Wikipedia,

> The two's complement of an N-bit number is defined as the complement with respect to 2^N; in other words, it is the result of subtracting the number from 2^N.

The total number of distinct numbers that can be represented by 32 bits is 2^{32}. Say, we have written down every number from 0 to $(2^{32}-1)$ in binary. Unsigned integer data type uses the first 2^{31} numbers for representing the positive numbers. Now we need to represent the negative numbers using the next 2^{31} numbers. According to the definition, two's complement of $-a$ should be equal to $(2^{32}-a)$. The numbers from -2^{31} to -1 are represented by the numbers 2^{31} to $(2^{32}-1)$ sequentially. So when you need to know the number of $1s$ written down when every number from $-a$ to $-b$ is written, you actually need to know the solution for $(2^{32}-a)$ to $(2^{32}-b)$. So the answer to the problem is:

$$\text{F}(2^{32}-a) \text{ - F}(2^{32}-b-1).$$

But what if you need to know the number of 1s written down when every number from $-a$ to $+b$ is written down, i.e. when the lower limit is negative and the upper limit is positive ?

You simply calculate the solution for $-a$ to -1 and the solution for 0 to b and add them.

Time complexity:

In each step we divide the number by 2 if the number is odd or subtract 1 from it if the number is even. So we will need $O(log(n))$ steps to come down to the base case. So the time complexity is $O(log(n))$.

Editorialist's solution

C++

Statistics

Difficulty: Advanced
Time Complexity: O(log(n))
Required Knowledge: Bit Manipulation, Recursion
Publish Date: Sep 11 2016

This is a Practice Challenge

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int LL;

LL find_number_of_ones(LL n)
{
    if(n == 0)
        return 0;
    else if(n%2 == 0)
        return find_number_of_ones(n-1) + __builtin_popcount(n);
    else
        return find_number_of_ones(n >> 1) * 2 + (n+1)/2;
}

LL solve(int a, int b)
{
    LL ret;
    if( b >= 0 && a >= 0)
        ret = find_number_of_ones(b) - find_number_of_ones(max(0, a-1));
    else if( b >= 0 && a < 0)
    {
        ret = find_number_of_ones(b);
        ret += (find_number_of_ones((1LL << 32) - 1) - find_number_of_ones((1LL << 32)
+ a - 1)) ;
    }
    else
        ret = find_number_of_ones((1LL << 32) + b) - find_number_of_ones((1LL << 32) +
a - 1);
    return ret;
}

int main()
{
    int cs, t;
    int A, B;
    cin >> t;
    for(cs = 1; cs<=t; cs++)
    {
        cin >> A >> B;
        cout << solve(A, B) << '\n';
    }
}
```