Non-negative Integers without Consecutive Ones

Given a positive integer n, find the number of**non-negative** integers less than or equal to n, whose binary representations do NOT contain **consecutive ones**.

Example 1:

Input: 5
Output: 5
Explanation:
Here are the non-negative integers

Note:
1 9

Solution 1

Reference: http://www.geeksforgeeks.org/count-number-binary-strings-without-consecutive-1s/

```
public class Solution {
    public int findIntegers(int num) {
        StringBuilder sb = new StringBuilder(Integer.toBinaryString(num)).reverse();
        int n = sb.length();

        int a[] = new int[n];
        int b[] = new int[n];
        a[0] = b[0] = 1;
        for (int i = 1; i < n; i++) {
            a[i] = a[i - 1] + b[i - 1];
            b[i] = a[i - 1];
        }

        int result = a[n - 1] + b[n - 1];
        for (int i = n - 2; i >= 0; i--) {
            if (sb.charAt(i) == '1' && sb.charAt(i + 1) == '1') break;
            if (sb.charAt(i) == '0' && sb.charAt(i + 1) == '0') result -= b[i];
        }

        return result;
    }
}
```

written by shawngao original link here

Solution 2

If we have n bits, the number of integers without consecutive ones f(n) = f(n - 1) + f (n - 2). f(n - 1) is for the case when first bit is zero, and f(n - 2) is when then the first bit is one and second bit is zero (as we cannot have consecutive ones):

- f(n) = "0" f(n - 1) + "10" f(n - 2).

These are Fibonacci numbers, and we can have them in a static array for [0..31] bits.

First, we find *n*, which is the position of the highest set bit in our number.

Now, if the binary representation of our number starts with "11", then all integers will be smaller than our number, and we can just return a Fibonacci number for *n* bits.

- For example, if n == 12 (binary 0b1100), highest bit at the position 4, next bit is set, so the result is 8 (fb[4]).

If the binary representation of our number starts with "10", then all integers with*n - 1* bits will be smaller than our number. So, we will grab a Fibonacci number for*n - 1* bits. That's "0" + f(n - 1) case. Plus, we need to add "10.." case, so we remove the highest bit from our number and recursively call our function.

- For example, if n == 18 (binary 0b10010), the highest bit at the position 5, next bit is unset, so taking 8 (fb[4] for 0x01111) and adding 3 (fb[2] for 0x00010).

```
int findIntegers(int num) {
    static int fb[31] = { 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,
1597, 2584, 4181, 6765, 10946,
        17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269
, 2178309, 3524578 };
    if (num < 3) return num + 1;
    for (int bt = 29; bt >= 0; --bt) // bt should start from 30, but OJ accepts it f
rom 29.
        if (num & (1 << bt)) return num & (1 << (bt - 1)) ? fb[bt] : fb[bt - 1] + f
indIntegers((num & ~(1 << bt)));
}
```

written by votrubac original link here

Solution 3

The solution is based on 2 facts:

1. the number of length k string without consecutive 1 is Fibonacci sequence f(k); For example, if k = 5, the range is 00000-11111. We can consider it as two ranges, which are 00000-01111 and 10000-10111. Any number >= 11000 is not allowed due to consecutive 1. The first case is actually f(4), and the second case is f(3), so f(5)= f(4)+f(3).
2. Scan the number from most significant digit, i.e. left to right, in binary format. If we find a '1' with k digits to the right, count increases by f(k) because we can put a '0' at this digit and any valid length k string behind; After that, we continue the loop to consider the remaining cases, i.e., we put a '1' at this digit. If consecutive 1s are found, we exit the loop and return the answer. By the end of the loop, we return count+1 to include the number n itself.
   For example, if n is 10010110,
   we find first '1' at 7 digits to the right, we add range 00000000-01111111, which is f(7);
   second '1' at 4 digits to the right, add range 10000000-10001111, f(4);
   third '1' at 2 digits to the right, add range 10010000-10010011, f(2);
   fourth '1' at 1 digits to the right, add range 10010100-10010101, f(1);
   Those ranges are continuous from 00000000 to 10010101. And any greater number <= n will have consecutive 1.

```
class Solution {
public:
    int findIntegers(int num) {
        int f[32];
        f[0] = 1;
        f[1] = 2;
        for (int i = 2; i < 32; ++i)
            f[i] = f[i-1]+f[i-2];
        int ans = 0, k = 30, pre_bit = 0;
        while (k >= 0) {
            if (num&(1<<k)) {
                ans += f[k];
                if (pre_bit) return ans;
                pre_bit = 1;
            }
            else
                pre_bit = 0;
            --k;
        }
        return ans+1;
    }
};
```

written by zestypanda original link here