Question 5.4 from CTCI: Next Number

September 13, 2017

Question

Given a positive integer, print the next largest number that have the same number of 1 bits in their binary representation.

Explanation and Algorithm

Input = x

The brute force solution is easy so we'll start there. Run through the binary representation of x as if it were an array, count the number of 1s and store the count in a tmp variable. Now, to find the next number increment the decimal representation by 1 and each time check the amount of 1s in the new numbers binary representation. If it is equal to tmp, stop because you have found the next largest otherwise keep incrementing x.

The key to this problem is to think through carefully how changing digits from 0 to 1 effects the value of the number, the ways in which 0s and 1s can be exchanged to ensure you have raised or lowered the number by exactly 1 binary value up or down and how to compensate for that through the rest of the binary number. For get next you want to identify the right most non-trailing 0 and flip the 1 the value directly to the right of it (will always be 1) with itself. Then you want to talk all 1s after that slot and make them 0 bits. Then you want to populate, starting from the least sig. digit, 1s from right to left equal to the amount of 1s you dropped -1. That will give you the next largest.

Hints

1. Start with a brute force solution.

2. Picture a binary number-something with a bunch of 1s and 0s spread out throughout the number. Suppose you flip a 1 to a 0 and a 0 to a 1. ln what case will the number get bigger? In what case will it get smaller?

3. If you flip a 1 to a 0 and a 0 to a 1, it will get bigger if the 0-¿ 1 bit is more significant than the 1-¿0 bit. How can you use this to create the next biggest number (with the same number of 1s)?

4. Can you flip a 0 to a 1 to create the next biggest number?

5. Flipping a 0 to a 1 will create a bigger number. The farther right the index is the smaller the bigger number is. If we have a number like 1001 , we want to flip the rightmost 0 (to create 1011). But if we have a number like 1010, we should not flip the rightmost 1.

6. We should flip the rightmost non-trailing O. The number 1010 would become 1110. Once we've done that, we need to flip a 1 to a 0 to make the number as small as possible, but bigger than the original number (1010) . What do we do? How can we shrink the number?

7. We can shrink the number by moving all the 1s to the right of the flipped bit as far right as possible (removing a 1 in the process).

Code

```
/*Get Next */

int getNext (int n){
   int c = n;
   int c0 = 0;
   int c1 = 0;

   while ( ( (c & 1) == 0) && ( c != 0)){
      c0++;
      c >>= 1;
   }

   while( (c & 1) == 1){
      c1++;
      c>>= 1;
   }

   if(c0 + c1  == 31 || c0 + c1 == 0){
      return -1;
   }

   int p = c0 + c1;
```

```
    n |= (1 << p);
    n &= ((1 <<p) - 1);
    n |= (1 << (c1 - 1)) - 1;

    return n;
}
```

Run Time Analysis

The runtime is O(n) where n is the length of the binary representation of the number passed into the function.

Sources

Question, answer and other material taken from Cracking the Coding Interview 6th edition by Gayle Laakmann McDowell.