Previous Next

338. Counting Bits

Dec. 13, 2016 | 37.4K views

Given a non negative integer number **num**. For every numbers **i** in the range **0 ≤ i ≤ num** calculate the number of 1's in their binary representation and return them as an array.

Example 1:

```
Input: 2
Output: [0,1,1]
```

Example 2:

```
Input: 5
Output: [0,1,1,2,1,2]
```

Follow up:

- It is very easy to come up with a solution with run time **O(n*sizeof(integer))**. But can you do it in linear time **O(n)** /possibly in a single pass?
- Space complexity should be **O(n)**.
- Can you do it like a boss? Do it without using any builtin function like **__builtin_popcount** in c++ or in any other language.

Summary

This article is for intermediate readers. It relates to the following ideas: Pop Count, Most Significant Bit, Least Significant Bit, Last Set Bit and Dynamic Programming.

Solutions

Approach #1 Pop Count [Accepted]

Intuition

Solve the problem for one number and applies that for all.

Algorithm

This problem can be seen as a follow-up of the Problem 191 The number of 1 bits. It counts the bits for an unsigned integer. The number is often called pop count or Hamming weight. See the editorial of Problem 191 The number of 1 bits for a detailed explanation of different approaches.

Now we just take that for granted. And suppose we have the function `int popcount(int x)` which will return the count of the bits for a given non-negative integer. We just loop through the numbers in range `[0, num]` and put the results in a list.

Java Copy

```java
public class Solution {
    public int[] countBits(int num) {
        int[] ans = new int[num + 1];
        for (int i = 0; i <= num; ++i)
            ans[i] = popcount(i);
        return ans;
    }
    private int popcount(int x) {
        int count;
        for (count = 0; x != 0; ++count)
          x &= x - 1; //zeroing out the least significant nonzero bit
        return count;
    }
}
```

Complexity Analysis

- Time complexity : $O(nk)$. For each integer x, we need $O(k)$ operations where k is the number of bits in x.
- Space complexity : $O(n)$. We need $O(n)$ space to store the count results. If we exclude that, it costs only constant space.

Approach #2 DP + Most Significant Bit [Accepted]

Intuition

Use previous count results to generate the count for a new integer.

Algorithm

Suppose we have an integer:

$$x = (1001011101)_2 = (605)_{10}$$

and we already calculated and stored all the results of 0 to $x-1$.

Then we know that x is differ by one bit with a number we already calculated:

$$x' = (1011101)_2 = (93)_{10}$$

They are different only in the most significant bit.

Let's exam the range $[0, 3]$ in the binary form:

$$(0) = (0)_2$$
$$(1) = (1)_2$$
$$(2) = (10)_2$$
$$(3) = (11)_2$$

One can see that the binary form of 2 and 3 can be generated by adding 1 bit in front of 0 and 1. Thus, they are different only by 1 regarding pop count.

Similarly, we can generate the results for $[4, 7]$ using $[0, 3]$ as blueprints.

In general, we have the following transition function for popcount $P(x)$:

$$P(x+b) = P(x) + 1, b = 2^m > x$$

With this transition function, we can then apply Dynamic Programming to generate all the pop counts starting from 0.

```java
public class Solution {
    public int[] countBits(int num) {
        int[] ans = new int[num + 1];
        int i = 0, b = 1;
        // [0, b) is calculated
        while (b <= num) {
            // generate [b, 2b) or [b, num) from [0, b)
            while(i < b && i + b <= num){
                ans[i + b] = ans[i] + 1;
                ++i;
            }
            i = 0;   // reset i
            b <<= 1; // b = 2b
        }
        return ans;
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$. For each integer x we need constant operations which do not depend on the number of bits in x.
- Space complexity : $O(n)$. We need $O(n)$ space to store the count results. If we exclude that, it costs only constant space.

Approach #3 DP + Least Significant Bit [Accepted]

Intuition

We can have different transition functions, as long as x' is smaller than x and their pop counts have a function.

Algorithm

Following the same principle of the previous approach, we can also have a transition function by playing with the least significant bit.

Let look at the relation between x and $x' = x/2$

$$x = (1001011101)_2 = (605)_{10}$$
$$x' = (100101110)_2 = (302)_{10}$$

We can see that x' is differ than x by one bit, because x' can be considered as the result of removing the least significant bit of x.

Thus, we have the following transition function of pop count $P(x)$:

$$P(x) = P(x/2) + (x \mod 2)$$

Java Copy

```java
public class Solution {
  public int[] countBits(int num) {
      int[] ans = new int[num + 1];
      for (int i = 1; i <= num; ++i)
        ans[i] = ans[i >> 1] + (i & 1); // x / 2 is x >> 1 and x % 2 is x & 1
      return ans;
  }
}
```

Complexity Analysis

- Time complexity : $O(n)$. For each integer x we need constant operations which do not depend on the number of bits in x.
- Space complexity : $O(n)$. Same as approach #2.

Approach #4 DP + Last Set Bit [Accepted]

Algorithm

With the same logic as previous approaches, we can also manipulate the last set bit.

Last set bit is the rightmost set bit. Setting that bit to zero with the bit trick, `x &= x - 1`, leads to the following transition function:

$$P(x) = P(x \& (x-1)) + 1;$$

Java Copy

```java
public class Solution {
  public int[] countBits(int num) {
      int[] ans = new int[num + 1];
      for (int i = 1; i <= num; ++i)
        ans[i] = ans[i & (i - 1)] + 1;
      return ans;
  }
}
```

Complexity Analysis

- Time complexity : $O(n)$. Same as approach #3.
- Space complexity : $O(n)$. Same as approach #3.

Rate this article: ★★★★★

Previous Next

deepak74 ★ 46 August 27, 2018 12:19 PM

4th approach is one of the most beautiful solutions I have ever seen. Thank you.

48 Share Reply

janani2 ★ 12 January 27, 2017 2:41 PM

i&i-1 clears the last bit set in i. The number of bits set in i would therefore be, the number of bits set in i&i-1 plus the bit that the AND operation cleared. For eg, take i = 3 (11). i-1 = 2 (10). 3&2 = 2 (10). The answer would be, the number of bits set in 2(=1), plus 1(from the cleared bit) = 2.

10 Share Reply

SHOW 1 REPLY

lsheng_mel ★ 175 October 22, 2019 6:44 PM

for anyone cannot make sense out of approach 2:
P(x+b)=P(x)+1

One can understand that b is the closet power of 2 that (x+b) is larger than, for example:
if x+b=6 => the closet power of 2 is 2^2=4, so b=4 => P(x+b)=P(2+4)=P(2)+P(4)=P(2)+1

Read More

5 Share Reply

kapania ★ 8 June 10, 2019 1:23 AM

Here's the python version of approach 4.

```python
class Solution:
    def countBits(self, num: int) -> List[int]:
        result = [0]*(num+1)
```

Read More

3 Share Reply

NideeshT ★ 590 May 14, 2019 7:17 AM

Java Code + Youtube Video Explanation accepted -https://www.youtube.com/watch?v=QjEyO1137cM(clickable link)

Leetcode 338 - Dyna...

Read More

2 Share Reply

rmadilao ★ 0 December 21, 2018 9:09 AM

Although the 4th solution is beautiful, implementing it in C is actually 28ms vs 16ms for 1st solution.

0 Share Reply

wnmmxy ★ 106 August 7, 2018 10:42 AM

Great article!

0 Share Reply

chrislzm ★ 1461 April 1, 2018 6:08 AM

Wow. Awesome solutions. Great article.

0 Share Reply

zpng ★ 6 December 28, 2017 4:05 PM

can you add all the solutions to python solution?

0 Share Reply

xiaojing1989 ★ 0 July 4, 2017 2:15 AM

The last solution is a revised version of the first solution. Very nice!

0 Share Reply

‹ 1 2 ›