Number of Digit One

Given an integer n, count the total number of digit 1 appearing in all non-negative integers less than or equal to n.

For example:
Given n = 13,
Return 6, because digit 1 occurred in the following numbers: 1, 10, 11, 12, 13.

1. Beware of overflow.

Solution 1

Go through the digit positions one at a time, find out how often a "1" appears at each position, and sum those up.

C++ solution

```
int countDigitOne(int n) {
    int ones = 0;
    for (long long m = 1; m <= n; m *= 10)
        ones += (n/m + 8) / 10 * m + (n/m % 10 == 1) * (n%m + 1);
    return ones;
}
```

Explanation

Let me use variables `a` and `b` to make the explanation a bit nicer.

```
int countDigitOne(int n) {
    int ones = 0;
    for (long long m = 1; m <= n; m *= 10) {
        int a = n/m, b = n%m;
        ones += (a + 8) / 10 * m + (a % 10 == 1) * (b + 1);
    }
    return ones;
}
```

Go through the digit positions by using position multiplier `m` with values 1, 10, 100, 1000, etc.

For each position, split the decimal representation into two parts, for example split n=3141592 into a=31415 and b=92 when we're at m=100 for analyzing the hundreds-digit. And then we know that the hundreds-digit of n is 1 for prefixes "" to "3141", i.e., 3142 times. Each of those times is a streak, though. Because it's the hundreds-digit, each streak is 100 long. So `(a / 10 + 1) * 100` times, the hundreds-digit is 1.

Consider the thousands-digit, i.e., when m=1000. Then a=3141 and b=592. The thousands-digit is 1 for prefixes "" to "314", so 315 times. And each time is a streak of 1000 numbers. However, since the thousands-digit is a 1, the very last streak isn't 1000 numbers but only 593 numbers, for the suffixes "000" to "592". So `(a / 10 * 1000) + (b + 1)` times, the thousands-digit is 1.

The case distincton between the current digit/position being 0, 1 and >=2 can easily be done in one expression. With `(a + 8) / 10` you get the number of full streaks, and `a % 10 == 1` tells you whether to add a partial streak.

Java version

```
public int countDigitOne(int n) {
    int ones = 0;
    for (long m = 1; m <= n; m *= 10)
        ones += (n/m + 8) / 10 * m + (n/m % 10 == 1 ? n%m + 1 : 0);
    return ones;
}
```

Python version

```
def countDigitOne(self, n):
    ones, m = 0, 1
    while m <= n:
        ones += (n/m + 8) / 10 * m + (n/m % 10 == 1) * (n%m + 1)
        m *= 10
    return ones
```

Using `sum` or recursion it can also be a one-liner.

Old solution

Go through the digit positions from back to front. I found it ugly to explain, so I made up that above new solution instead. The `n` here is the new solution's `a`, and the `r` here is the new solution's `b+1`.

Python

```
def countDigitOne(self, n):
    ones = 0
    m = r = 1
    while n > 0:
        ones += (n + 8) / 10 * m + (n % 10 == 1) * r
        r += n % 10 * m
        m *= 10
        n /= 10
    return ones
```

Java

```
public int countDigitOne(int n) {
    int ones = 0, m = 1, r = 1;
    while (n > 0) {
        ones += (n + 8) / 10 * m + (n % 10 == 1 ? r : 0);
        r += n % 10 * m;
        m *= 10;
        n /= 10;
    }
    return ones;
}
```

C++

```
int countDigitOne(int n) {
    int ones = 0, m = 1, r = 1;
    while (n > 0) {
        ones += (n + 8) / 10 * m + (n % 10 == 1) * r;
        r += n % 10 * m;
        m *= 10;
        n /= 10;
    }
    return ones;
}
```

written by StefanPochmann original link here

Solution 2

```java
public int countDigitOne(int n) {
  int count = 0;

  for (long k = 1; k <= n; k *= 10) {
    long r = n / k, m = n % k;
    // sum up the count of ones on every place k
    count += (r + 8) / 10 * k + (r % 10 == 1 ? m + 1 : 0);
  }

  return count;
}
```

Solution explanation:

Let's start by counting the ones for every 10 numbers...

0, 1, 2, 3 ... 9 (1)

10, 11, 12, 13 ... 19 (1) + **10**

20, 21, 22, 23 ... 29 (1)

...

90, 91, 92, 93 ... 99 (1)

-

100, 101, 102, 103 ... 109 (10 + 1)

110, 111, 112, 113 ... 119 (10 + 1) + **10**

120, 121, 122, 123 ... 129 (10 + 1)

...

190, 191, 192, 193 ... 199 (10 + 1)

-

1). If we don't look at those special rows (start with 10, 110 etc), we know that there's a one at ones' place in every 10 numbers, there are 10 ones at tens' place in every 100 numbers, and 100 ones at hundreds' place in every 1000 numbers, so on and so forth.

Ok, let's start with ones' place and count how many ones at this place, set k = 1, as mentioned above, there's a one at ones' place in every 10 numbers, so how many 10 numbers do we have?

The answer is (n / k) / 10.

Now let's count the ones in tens' place, set k = 10, as mentioned above, there are 10 ones at tens' place in every 100 numbers, so how many 100 numbers do we have?

The answer is (n / k) / 10, and the number of ones at ten's place is (n / k) / 10 * k.

Let r = n / k, now we have a formula to count the ones at k's place:**r / 10 * k**

-

2). So far, everything looks good, but we need to fix those special rows, how?

We can use the mod. Take 10, 11, and 12 for example, if n is 10, we get (n / 1) / 10 * 1 = 1 ones at ones's place, perfect, but for tens' place, we get (n / 10) / 10 * 10 = 0, that's not right, there should be a one at tens' place! Calm down, from 10 to 19, we always have a one at tens's place, let m = n % k, the number of ones at this special place is m + 1, so let's fix the formula to be:

r / 10 * k + (r % 10 == 1 ? m + 1 : 0)

-

3). Wait, how about 20, 21 and 22?

Let's say 20, use the above formula we get 0 ones at tens' place, but it should be 10! How to fix it? We know that once the digit is larger than 2, we should add 10 more ones to the tens' place, a clever way to fix is to add 8 to r, so our final formula is:

(r + 8) / 10 * k + (r % 10 == 1 ? m + 1 : 0)

As you can see, it's all about how we fix the formula. Really hope that makes sense to you.

written by jeantimex original link here

Solution 3

For every digit in n (Suppose n = 240315, the digits are 2, 4, 0, 3, 1, 5)， I respectively count the number of digit 1 assuming the position of current digit is 1 and other digits of n is arbitrary.

For example, I select 3 in n as the current digit, and I suppose the position of 3 is 1.

The highn is the number composed with the digits before the current digit. In the example, highn = 240;

The lown is the number composed with the digits after the current digit. In the example, lown = 15.

The lowc = 10 ^ (the number of lower digits). In the example, lowc = 100;

As curn = 3 and curn > 1, (highn * 10 + 1) must be less than (highn * 10 + curn). Then the higher part can be 0 ~ highn, the lower part can be 0 ~ (lowc-1), and the current result = (highn + 1) * lowc.

```
int countDigitOne(int n) {
        long long int res(0);
        int highn(n), lowc(1), lown(0);
        while(highn > 0){
            int curn = highn % 10;
            highn = highn / 10;
            if(1 == curn){
                //higher: 0~(highn-1);  lower:  0 ~ (lowc-1)
                res += highn * lowc;
                //higher: highn ~ highn;     lower:0~lown
                res += lown + 1;
            }else if(0 == curn){
                //curn < 1
               //higher: 0~(highn-1);  lower:  0 ~ (lowc-1)
                res += highn * lowc;
            }else{
                //curn > 1
                res += (highn + 1) * lowc;
            }
            //update lown and lowc
            lown = curn * lowc + lown;
            lowc = lowc * 10;
        }
        return res;
    }
```

written by tju_xu original link here