1 的数目

给定一个十进制正整数 N，写下从 1 开始，到 N 的所有整数，然后数一下其中出现的所有“1”的个数。

例如：

N= 2，写下 1，2。这样只出现了 1 个“1”。

N= 12，我们会写下 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12。这样，1 的个数是 5。

问题是：

1. 写一个函数 $f(N)$，返回 1 到 N 之间出现的“1”的个数，比如 $f(12)=5$。
2. 在 32 位整数范围内，满足条件“$f(N)=N$”的最大的 N 是多少？

分析与解法

【问题 1 的解法一】

这个问题看上去并不是一个困难的问题，因为不需要太多的思考，我想大家都能找到一个最简单的方法来计算 $f(N)$，那就是从 1 开始遍历到 N，将其中每一个数中含有“1”的个数加起来，自然就得到了从 1 到 N 所有“1”的个数的和。写成程序如下：

代码清单 2-9

```
ULONGLONG Count1InAInteger(ULONGLONG n)
{
    ULONGLONG iNum = 0;
    while(n != 0)
    {
        iNum += (n % 10 == 1) ? 1 : 0;
        n /= 10;
    }

    return iNum;
    }

ULONGLONG f(ULONGLONG n)
{
    ULONGLONG iCount = 0;
    for (ULONGLONG i = 1; i <= n; i++)
    {
        iCount += Count1InAInteger(i);
    }
```

```
    return iCount;
}
```

这个方法很简单，只要学过一点编程知识的人都能想到，实现也很简单，容易理解。但是这个算法的致命问题是效率，它的时间复杂度是

$O(N)$×计算一个整数数字里面“1”的个数的复杂度 $= O(N * \log_2 N)$

如果给定的 N 比较大，则需要很长的运算时间才能得到计算结果。比如在笔者的机器上，如果给定 N=100 000 000，则算出 $f(N)$ 大概需要 40 秒的时间，计算时间会随着 N 的增大而线性增长。

看起来要计算从 1 到 N 的数字中所有 1 的和，至少也得遍历 1 到 N 之间所有的数字才能得到。那么能不能找到快一点的方法来解决这个问题呢？要提高效率，必须摈弃这种遍历 1 到 N 所有数字来计算 $f(N)$ 的方法，而应采用另外的思路来解决这个问题。

【问题 1 的解法二】

仔细分析这个问题，给定了 N，似乎就可以通过分析“小于 N 的数在每一位上可能出现 1 的次数”之和来得到这个结果。让我们来分析一下对于一个特定的 N，如何得到一个规律来分析在每一位上所有出现 1 的可能性，并求和得到最后的 $f(N)$。

先从一些简单的情况开始观察，看看能不能总结出什么规律。

先看 1 位数的情况。

如果 $N = 3$，那么从 1 到 3 的所有数字：1、2、3，只有个位数字上可能出现 1，而且只出现 1 次，进一步可以发现如果 N 是个位数，如果 $N>=1$，那么 $f(N)$ 都等于 1，如果 $N=0$，则 $f(N)$ 为 0。

再看 2 位数的情况。

如果 $N=13$，那么从 1 到 13 的所有数字：1、2、3、4、5、6、7、8、9、10、11、12、13，个位和十位的数字上都可能有 1，我们可以将它们分开来考虑，个位出现 1 的次数有两次：1 和 11，十位出现 1 的次数有 4 次：10、11、12 和 13，所以 $f(N)=2+4=6$。要注意的是 11 这个数字在十位和个位都出现了 1，但是 11 恰好在个位为 1 和十位为 1 中被计算了两次，所以不用特殊处理，是对的。再考虑 $N=23$ 的情况，它和 $N=13$ 有点不同，十位出现 1 的次数为 10 次，从 10 到 19，个位出现 1 的次数为 1、11 和 21，所以 $f(N)=3+10=13$。通过对两位数进行分析，我们发现，个位数出现 1 的次数不仅和个位数字有关，还和十位数有关：如果 N 的个位数大于等于 1，则个位出现 1 的次数为十位数的数字加 1；如果 N 的个位数为 0，则个位出现 1 的次数等于十位数的数字。而十位数上出现 1 的次数不仅和十位数有关，还和个位数有关：如果十位数字等于 1，则十位数上出现 1 的次数为个位数的数字加 1；如果十位数大于 1，则十位数上出现 1 的次数为 10。

```
f(13) = 个位出现1的个数 + 十位出现1的个数 = 2 + 4 = 6;
f(23) = 个位出现1的个数 + 十位出现1的个数 = 3 + 10 = 13;
f(33) = 个位出现1的个数 + 十位出现1的个数 = 4 + 10 = 14;
…
f(93) = 个位出现1的个数 + 十位出现1的个数 = 10 + 10 = 20;
```

接着分析 3 位数。

如果 $N = 123$：

个位出现 1 的个数为 13：1, 11, 21, …, 91, 101, 111, 121

十位出现 1 的个数为 20：10～19, 110～119

百位出现 1 的个数为 24：100～123

$f(23)$ = 个位出现 1 的个数 + 十位出现 1 的个数 + 百位出现 1 的次数 = 13 + 20 + 24 = 57；

同理我们可以再分析 4 位数、5 位数。读者朋友们可以写一写，总结一下各种情况有什么不同。

根据上面的一些尝试，下面我们推导出一般情况下，从 N 得到 $f(N)$ 的计算方法：

假设 $N=abcde$，这里 a、b、c、d、e 分别是十进制数 N 的各个数位上的数字。如果要计算百位上出现 1 的次数，它将会受到三个因素的影响：百位上的数字，百位以下（低位）的数字，百位（更高位）以上的数字。

如果百位上的数字为 0，则可以知道，百位上可能出现 1 的次数由更高位决定，比如 12 013，则可以知道百位出现 1 的情况可能是 100～199，1 100～1 199，2 100～2 199，…，11 100~11 199，一共有 1 200 个。也就是由更高位数字（12）决定，并且等于更高位数字（12）×当前位数（100）。

如果百位上的数字为 1，则可以知道，百位上可能出现 1 的次数不仅受更高位影响，还受低位影响，也就是由更高位和低位共同决定。例如对于 12 113，受更高位影响，百位出现 1 的情况是 100～199，1 100～1 199，2 100～2 199，…，11 100~11 199，一共 1 200 个，和上面第一种情况一样，等于更高位数字（12）×当前位数（100）。但是它还受低位影响，百位出现 1 的情况是 12 100～12 113，一共 114 个，等于低位数字（123）+1。

如果百位上数字大于 1（即为 2~9），则百位上可能出现 1 的次数也仅由更高位决定，比如 12 213，则百位出现 1 的可能性为：100～199，1 100～1 199，2 100～2 199，…，11 100～11 199，12 100～12 199，一共有 1 300 个，并且等于更高位数字+1（12+1）×当前位数（100）。

通过上面的归纳和总结，我们可以写出如下的更高效算法来计算 $f(N)$：

代码清单 2-10

```
LONGLONG Sum1s(ULONGLONG n)
{
  ULONGLONG iCount = 0;

  ULONGLONG iFactor = 1;

  ULONGLONG iLowerNum = 0;
  ULONGLONG iCurrNum = 0;
```

```
    ULONGLONG iHigherNum = 0;

    while(n / iFactor != 0)
    {
        iLowerNum = n - (n / iFactor) * iFactor;
        iCurrNum = (n / iFactor) % 10;
        iHigherNum = n / (iFactor * 10);

        switch(iCurrNum)
        {
        case 0:
            iCount += iHigherNum * iFactor;
            break;
        case 1:
            iCount += iHigherNum * iFactor + iLowerNum
+ 1;
            break;
        default:
            iCount += (iHigherNum + 1) * iFactor;
            break;
        }

        iFactor *= 10;
    }

    return iCount;
}
```

这个方法只要分析 N 就可以得到 $f(N)$，避开了从 1 到 N 的遍历，输入长度为 Len 的数字 N 的时间复杂度为 $O(Len)$，即为 $O(\ln(n)/\ln(10)+1)$。在笔者的计算机上，计算 N=100 000 000，

相对于第一种方法的 40 秒时间，这种算法不到 1 毫秒就可以返回结果，速度至少提高了 40 000 倍。

【问题 2 的解法】

要确定最大的数 N，满足 $f(N)=N$。我们通过简单的分析可以知道（仿照上面给出的方法来分析）：

9 以下为：1 个；

99 以下为：$1\times10+10\times1=20$ 个；

999 以下为：$1\times100+10\times20=300$ 个；

9999 以下为：$1\times1000+10\times300=4000$ 个；

…

999999999 以下为：900000000 个；

9999999999 以下为：10000000000 个。

容易从上面的式子归纳出：$f(10^{n-1}) = n * 10^{n-1}$。通过这个递推式，很容易看到，当 $n = 9$ 时候，$f(n)$ 的开始值大于 n，所以我们可以猜想，当 n 大于某一个数 N 时，$f(n)$ 会始终比 n 大，也就是说，最大满足条件在 $0 \sim N$ 之间，亦即 N 是最大满足条件 $f(n) = n$ 的一个上界。如果能估计出这个 N，那么只要让 n 从 N 往 0 递减，每个分别检查是否有 $f(n) = n$，第一个满足条件的数

就是我们要求的整数。

因此，问题转化为如何证明上界 N 确实存在，并估计出这个上界 N。

证明满足条件 $f(n) = n$ 的数存在一个上界

首先，用类似数学归纳法的思路来推理这个问题。很容易得到下面这些结论（读者朋友可以自己试着列举验证一下）：

当 n 增加 10 时，$f(n)$ 至少增加 1；
当 n 增加 100 时，$f(n)$ 至少增加 20；
当 n 增加 1 000 时，$f(n)$ 至少增加 300；
当 n 增加 10 000 时，$f(n)$ 至少增加 4 000；
……
当 n 增加 10^k 时，$f(n)$ 至少增加 $k*10^{k-1}$。

首先，当 $k>=10$ 时，$k*10^{k-1}> 10^k$，所以 $f(n)$ 的增加量大于 n 的增加量。

其次，$f(10^{10-1})=10^{10}>10^{10-1}$。如果存在 N，当 $n = N$ 时，$f(N)-N>10^{10-1}$ 成立时，此时不管 n 增加多少，$f(n)$ 的值将始终大于 n。

具体来说，设 n 的增加量为 m：当 m 小于 10^{10-1} 时，由于 $f(N)-N>10^{10-1}$，因此有 $f(N + m) > f(N) > N + 10^{10-1} > N + m$，即 $f(n)$ 的值仍然比 n 的值大；当 m 大于等于 10^{10-1} 时，$f(n)$

的增量始终比 n 的增量大，即 $f(N+m)-f(N)>(N+m)-N$，也就是 $f(N+m)>f(N)+m>N+10^{10}-1+m>N+m$，即 $f(n)$ 的值仍然比 n 的值大。

因此，对于满足 $f(N)-N>10^{10}-1$ 成立的 N 一定是所求该数的一个上界。

求出上界 N

又由于 $f(10^{10}-1)=n*10^{10}-1$，不妨设 $N=10^{K}-1$，有 $f(10^{K}-1)-(10^{K}-1)>10^{10}-1$，即 $K*10^{K}-1-(10^{K}-1)>10^{10}-1$，易得 $K>=11$ 时候均满足。所以，当 $K=11$ 时，$N=10^{11}-1$ 即为最小一个上界。

计算这个最大数 n

令 $N=10^{11}-1$=99 999 999 999，让 n 从 N 往 0 递减，每个分别检查是否有 $f(n)=n$，第一满足条件的就是我们要求的整数。很容易解出 n = 1 111 111 110 是满足 $f(n)=n$ 的最大整数。

扩展问题

对于其他进制表达方式，也可以试一试，看看有什么规律。例如二进制：

$f(1)=1$

$f(10)=10$（因为 01, 10 有两个 1）

$f(11)=100$（因为 01, 10, 11 有四个 1）

读者朋友可以模仿我们的分析方法，给出相应的解答。