Number of Zeros

Another month has passed and it is time to reveal how a programming challenge, codenamed Mu, can be solved. You can still give it a try, although no certificate will be granted.

The task was to count the total number of zeros in a decimal representation of numbers $0, 1, 2, \ldots, N$. The important aspect is that N could be as large as $10^{10\,000}$. Not all programming languages support such large integers. Therefore, two technical simplifications have been applied:

- number N is given as a string containing its decimal representation,
- it suffices to calculate the result modulo 1 410 000 017.

1 410 000 017 has appeared in Codility Challenges before, and there is a good reason: it is a prime number which is relatively large but which fits easily into 32-bit signed integers. Therefore, all arithmetical operations can be performed modulo 1 410 000 017 and intermediate values fit into 64-bit arithmetic. That simplifies the technical details a lot.

Brute-force solution

The simplest, and the least efficient, solution enumerates all integers from 0 to N and counts all the zeros that appear in their representations. Here is its implementation in Python:

1: Brute-force solution — $O(N \cdot \log N)$

```
    number_of_zeros(S):
        N = int(S)
        a = 0
        while N >= 0:
            a = a + str(N).count('0')
            N = N - 1
        return a % 1410000017
```

Obviously, such a solution works only for small values of N, since it runs in $O(N \cdot \log N)$ time. Even if it ran for a year (on a typical PC), it wouldn't calculate the result for $N = 10^{16}$. Instead of counting zeros one by one, we must count many of them at once.

Optimal solution

Let us assume that we have calculated $F =$ `number_of_zeros`(N). What is `number_of_zeros`$(10 \cdot N + 9)$? Each counted zero (except the one in number 0) is repeated ten times. Additionally, there are $N+1$ zeros at the least significant positions. Hence:

$$\texttt{number\_of\_zeros}(10 \cdot N + 9) = 10 \cdot (\texttt{number\_of\_zeros}(N) - 1) + N + 1$$

How about $\texttt{number\_of\_zeros}(10 \cdot N + C)$ for $0 \leqslant C \leqslant 9$? If Z is the number of zeros in a decimal representation of N, then there are $(9 - C) \cdot Z$ fewer zeros. Therefore:

$$\texttt{number\_of\_zeros}(10 \cdot N + C) = 10 \cdot (\texttt{number\_of\_zeros}(N) - 1) + N - (9 - C) \cdot Z + 1$$

Additionally, for $N \leqslant 9$ we have $\texttt{number\_of\_zeros}(N) = 1$. This leads to the following algorithm:

2: Optimal solution — $O(\log N)$

```
def number_of_zeros(S):
    P = 1410000017
    Z = 0
    N = 0
    F = 0
    for j in xrange(len(S)):
        F = (10 * F + N + P - Z * (9 - int(S[j])) ) % P
        if S[j] == '0':
            Z += 1
        N = (10 * N + int(S[j])) % P
    return ((1 + F) % P)
```

The invariant of the `for`-loop is crucial to show that this algorithm really implements the above recursive equations. The invariant consists of the following conditions:

- `Z` = number of zeros in `S[:j]`,
- `N` = number represented by `S[:j]` (modulo `P`),
- `F` = `number_of_zeros(S[:j]) - 1` (modulo `P`).

Clearly, this algorithm runs in $O(\log N)$ time. It is optimal because reading the data itself takes $O(\log N)$ time.

It is important that we process the digits of N from most to least significant. This way, we can maintain variables `N` and `Z`, and at each step of the iteration update their values in constant time. Otherwise, we would have to calculate their values from the beginning at each step of the iteration. Since that requires scanning `S[:j]`, it yields $O((\log N)^2 <)$ overall time complexity. For $N = 10^{10\,000}$ it would mean a minute or two instead of a fraction of a second.