SPALNUM - Editorial

0

PROBLEM LINK:

Practice
Contest

Author: Sergey Kulik
Tester: Yanpei Liu
Editorialist: Pawel Kacprzak

DIFFICULTY:

SIMPLE

PREREQUISITES:

Ad hoc, Palindrome

PROBLEM:

Let palindromic number be a number whose decimal digits form a palindrome. For example, $1, 22, 414, 5335$ are palindromic numbers, while 13 and 453 are not. Your task is to find the sum of all palindromic numbers in a range $[L, R]$ inclusive. In one test file, you have to solve this task for at most 100 test cases.

QUICK EXPLANATION:

Precompute the sum of palindromic number not greater than K, for $1 \leq K \leq 10^5$, and store these values in an array. Provide an answer for a single test case $[L, R]$ using precomputed sums for R and $L - 1$.

EXPLANATION:

Let's first consider solving the problem for a single test cases. We are given two number L and R and we have to compute the sum of palindromic numbers from L to R inclusive. If we can check if a number N is palindromic, then we can iterate over all numbers N in a range $[L, R]$ and add N to the result if and only if N is palindromic. How to check if a number N is palindromic? Well, it is pretty straightforward, we can list the sequence of digits of N from right to left, and check if that sequence is a palindrome comparing corresponding digits. A pseudocode of that method can look like that:

```
// we assume that N > 0
bool is_palindromic(N):
    digits = [ ]
    while N > 0:
        digits.append(N % 10)
        N /= 10
    i = 0
    j = digits.size() - 1
    while i < j:
        if digits[i] != digits[j]:
            return False
        i += 1
        j -= 1
    return True
```

This check runs in $O(\log(N))$ time, because the decimal representation of N has $O(\log(N))$ digits.

Being able to perform the palindromic check, we can accumulate the result iterating over all integers in range $[L, R]$. A pseudocode for it might look like this:

```
res = 0
for N = L to R:
    if is_palindromic(N):
        res += N
```

This method works in $O((R - L) \cdot \log(R))$ time for a single test case, but since we have to handle at most 100 of them and a range $[L, R]$ can have up to 10^5 elements, this method will pass only the first subtask and will timeout on the second.

How to speed it up?

The crucial observation here is that, during the whole computation described above, we might check in a number N is palindromic many times! This is not good, but fortunately, there is a common technique to avoid that.

Often when we are asked many times to compute some result for objects in some range $[A, B]$, we can do the following:

Let $F[N] := \texttt{the result for a range } [0, N]$

If we are able to compute $F[N]$ for all possible N, then the answer for a single query $[A, B]$ equals $F[B] - F[A - 1]$, because $F[B]$ contains the result for all numbers not greater than B, so if we subtract $F[A - 1]$, i.e the result for all number smaller than A, from it, we will get the result for all numbers in range $[A, B]$.

If you did not know this technique, please remember it, because it is very useful.

Using the above method, we can precompute:

$S[N] := \texttt{sum of palindromic number not greater than } N$

in the following way:

```
S[0] = 1
for N = 1 to 100000:
    S[N] = S[N - 1]
    if is_palindromic(N):
        S[N] += N
```

The above method runs in $O(10^5 \cdot \log(10^5))$ time, and we can use the S table to answer any single query $[L, R]$ in a constant time.