Code Jam 2009 - Round 3

Analysis: Interesting Ranges

1. How big can the answer be?

First, let's get a crude upper bound on the answer. More specifically, if L=1 and R=N, there are N*(N+1)/2 subsegments of [L,R]. Not all of them will contain an even number of palindromes, but a significant part (roughly half) will.

So even for the small input we might get more than 10^25 segments in the answer; it's obvious we can't enumerate them one by one. We have to figure out a way to process them in bulks.

2. First optimization

The first idea required to solve the small dataset helps us reduce the number of segments to consider from O(N^2) to O(N). To achieve that, we rely on the following observation: [L,R] can be represented as [0,R] minus [0,L-1]; thus it contains an even number of palindromes if and only if [0,L-1] and [0,R] both contain even or both contain odd number of palindromes.

But how exactly does that help us? Suppose we know which of [0,X] segments contain even number of palindromes (we'll call them just "even 0-segments" further on) and which contain odd number of palindromes ("odd 0-segments"). We know that each interesting segment corresponds to exactly one pair of even 0-segments or to exactly one pair of odd 0-segments. But this is true the other way around as well: each pair of distinct even 0-segments corresponds to exactly one interesting segment, and so does each pair of distinct odd 0-segments! (when X is between L-1 and R, inclusive).

That means that if there are A even 0-segments and B odd 0-segments, the answer is A*(A-1)/2+B*(B-1)/2.

2. Second optimization

But we can't even afford O(N) running time since N is 10^13 even in the small dataset! So we have to do another optimization.

Let's write out an infinite string of zeroes and ones, with X-th symbol (0-based) being equal to 0 if [0,X] contains an even number of palindromes, and equal to 1 if [0,X] contains an odd number of palindromes. What we need to find in this problem is how many zeroes (number A above) and ones (number B above) are there in the substring starting with (L-1)-th character and ending with R-th character of this string.

This string looks like this: 101010101001111111111000000000001111111111100...

Now we can spot the second optimization: zeroes and ones tend to go in big blocks in this string. More specifically, one changes to zero or vice versa only when we pass a palindrome. And there are only O(sqrt(N)) palindromes up to N - so the number of groups of consecutive zeros or ones in the first N characters is O(sqrt(N)).

All groups except maybe two boundary ones fit into [L-1,R] segment entirely. So we just need to sum them all and handle the boundary ones carefully, and we get an O(sqrt(N)) algorithm that is sufficient to solve the small dataset.

We can also reduce the number of boundary groups to consider from two to one relying on the fact that the number of zeroes/ones in [L-1,R] is the number of zeroes/ones in [0,R] minus the number of zeroes/ones in [0,L-2].

3. Third optimization

So what about the large one? Sqrt(10^100)=10^50, so we're definitely not there yet.

The final optimization idea still relies on the above infinite string of zeroes and ones. You might have noticed already that many blocks of ones and zeroes have the same length. For example, block of ones from 11 to 21 has length 11, and so is the block of zeroes from 22 to 32, ones from 33 to 43, zeroes from 44 to 54, and so on.

This is because of the fact that a palindrome number is uniquely determined by its first half. For example, consider 6-digit palindrome number 127721. What is the next 6-digit palindrome number? 128821. It's followed by 129921, 130031, and so on. As you can see, in most cases the difference between two consecutive 6-digit palindrome numbers is 1100 (change of +1 in two middle digits). And the difference between two consecutive palindrome numbers is exactly the length of the block of ones/zeroes!

But there are also some blocks which have length different from usual. For example, the block of zeroes from 88 to 98 still has 11 numbers, but the block of ones from 99 to 100 has just two numbers; then follow several blocks of 10 (zeroes from 101 to 110; ones from 111 to 120; ...; zeroes from 181 to 190), then we have a block of 11 ones from 191 to 201.

The above explanation about consecutive palindromes allows us to understand why there are such unusual-length blocks. It happens in two cases: when the amount of digits in the palindrome changes, and when the middle digit of the starting palindrome of the block is 9. The first several unusual-length blocks are: zeroes from 9 to 10, ones from 99 to 100, ones from 191 to 201, ones from 292 to 302, ..., ones from 898 to 908, ones from 999 to 1000, ones from 1991 to 2001, ones from 2992 to 3002, and so on.

And here comes the final step. All such unusual-length blocks except the one from 9 to 10 consist of ones! Or, in other words, all blocks of zeroes have the same length within palindromes of the same amount of digits, except the block from 9 to 10.

Why? Because there's ten blocks between two consecutive palindromes with a 9 in the middle (with the only exception being 9 and 99), and ten is an even number. That's why every time we have a 9 in the middle, we start a block of ones.

And that allows us to calculate in one step the total amount of zeroes in the part of our infinite string that corresponds to one amount of digits in the palindrome. That means we can calculate the overall amount of zeroes in any segment in O(number of digits in the maximal palindrome) steps. Of course, we have to be careful near L and R.

And what about the amount of ones? It's equal to the total length minus the amount of zeroes :)

4. Modulo calculations

With the numbers in the input being quite big, the above calculations have to be performed carefully.

Some programming languages allow easy calculations with arbitrarily long numbers. But what if we use a language without this feature? Luckily, we're asked to find the answer modulo a (relatively) small number M.

That allows us to perform most calculations modulo M, and thus use only small numbers in those calculations. The calculations we need to find the answer are: addition, multiplication, subtraction and division by 2. The first three are performed in a standard way (first calculate the answer in integers, then take it modulo M). The fourth one is made possible by the fact that the modulo used in the problem is odd. When we divide a number X by 2 modulo M, we get just X/2 if X is even, and (X+M)/2 when x is odd. One can easily check that multiplying by 2 gets X back in both those cases.