Tags: permutations, dynamic programming, fast calculation, cycle by module

The task has a pretty straightforward statement – find the number of permutations of $\boldsymbol{N}$ numbers which are *pretty* i.e. having at least one pair of adjacent numbers in the form $(x, x+1)$.

The first subtask is for 9 points. It is for the naïve idea – we generate all permutations of $\boldsymbol{N}$ numbers and check each of them for *prettiness*. The only tricky moment is that the module can be very small, so we still have to calculate the answer modulo $\boldsymbol{M}$. The complexity is $O(\boldsymbol{N}! * \boldsymbol{N})$.

The second subtask is for 14 points. Here we will optimize the brute-force solution. Let's think that we generate the permutations using the standard recursion. A standard approach for optimization, when we have this situation, is to see that when we are at a certain number, we don't care what was the exact order of the previous numbers, only what they were. So, we care only about the set of the numbers that we used and the current number. We can use this to make memoization of this recursion: $\mathrm{dp}[\mathrm{mask}][\mathrm{last}]$, with *mask* showing what set of numbers we have already used and *last* being the last used number. This *dp* will tell us what is the number of *pretty* permutations that have a certain set of numbers in the first k positions, at the k-th position the number is *last* and with the property that there wasn't a position from 1 to (k-1) in the form $(x, x+1)$. To calculate certain *dp,* we have two cases. The first is that we use the number *(last+1)* (if we can) in the next position. This will make the permutation *pretty* no matter what are the numbers after that. So, we just have to put them in some order. This means that we just have to calculate some factorial accounting the other numbers in this case. The other cases are easier. If we use some other number, the property that there wasn't position in the form $(x, x+1)$ remains, so we need the value of some other state of the *dp*. In this solution all calculations have to be made modulo $\boldsymbol{M}$. Here the complexity is slightly better - $O(2^{\boldsymbol{N}} * \boldsymbol{N}^2)$.

The third subtask is for 11 points. We use another standard approach. When we have only one number at input and there are subtasks with small constraints, we can precompute some of the answers. But in this task the answer has to be calculated by some module known at input. This is no problem because the constraints for this subtask are rather small. One simple calculation can show us that 20! is small enough to be in *long long*. So, we can use the previous optimized brute-force solution to calculate the answers up to $\boldsymbol{N}$=20 without using module. The author has further optimized the precompute. We can notice that when we calculate the answer for certain $\boldsymbol{N}$, the answers for all smaller $\boldsymbol{N}$s are some states of the *dp*. So, we can run the previous solution only once to know all the answers we need. In this way, the precompute should run less than a minute on a modern computer. The complexity of the precompute is $O(2^{\boldsymbol{N}} * \boldsymbol{N}^2)$ and the solution is only to output the already calculated answer by the inputted modulo.

The fourth subtask is for 43 points. This is the most important subtask in which we will describe linear solution for the problem. We will think about similar task – we will count the number of permutations not being *pretty*. Let's call them *ugly* for simplicity. Obviously if we know the number of *ugly* permutations of $\boldsymbol{N}$ numbers, then the *pretty* permutations can be calculated by subtracting from $\boldsymbol{N}$!. Let's denote with $dp[\boldsymbol{N}]$ – the number of *ugly* permutations of $\boldsymbol{N}$ numbers. There are two cases how we can make *ugly* permutations of $\boldsymbol{N}$ numbers using permutations of ($\boldsymbol{N}$-1) numbers. Let's think of some random permutation p of ($\boldsymbol{N}$-1) numbers. If this permutation is *ugly*, then we have to put the number $\boldsymbol{N}$ in such a way that the permutation stays *ugly*. There are $\boldsymbol{N}$ possible positions in general but one of them is impossible in our case – the position after the number ($\boldsymbol{N}$-1). So, the *ugly* permutations of $\boldsymbol{N}$ numbers obtained in this way are $(\boldsymbol{N}-1) * dp[\boldsymbol{N}-1]$. Now we will consider the harder case – let p is a *pretty* permutation. Obviously, we have to put $\boldsymbol{N}$ in such a way so that we break the pairs in the form $(x, x+1)$. But if there are two or more such pairs, it is impossible to break them with only putting one number. So, we need p to be *pretty* with only one pair in the form $(x, x+1)$. Let's call such permutation *super pretty*. If we have a *super pretty* permutation, then we have only one place where we should place $\boldsymbol{N}$ – between x and $(x+1)$. We can note that x can be any of the numbers from 1 to ($\boldsymbol{N}$-2) (it cannot be ($\boldsymbol{N}$-1)). So, when putting $\boldsymbol{N}$ and making $(x, \boldsymbol{N}, x+1)$ we will not have pair in the form $(y, y+1)$ and that means that the *super pretty* permutation p can become *ugly* in one and only one way. Lastly, we have to find what is the number of *super pretty* permutations of ($\boldsymbol{N}$-1) numbers. One approach, which we won't discuss, is to make another *dp* for this number, which will be mutually recurrent with the first *dp*. Let we calculate the number of *super pretty* permutations of ($\boldsymbol{N}$-1) numbers, which have the pair (1, 2). This can be done in the following way. We can put 2 in the position right after 1 in some *ugly* permutation of the numbers in $S = \{1, 3, 4, \dots, \boldsymbol{N}-2, \boldsymbol{N}-1\}$ (S has ($\boldsymbol{N}$-2) numbers). Also is we have a permutation q of the numbers in S, to make it *super pretty* with the pair (1, 2) the only way is to put 2 after 1 in q. After that we have to make sure that there aren't other pairs in the form $(x, x+1)$. When we put 2 after 1, we will have $(1, 2, y)$ (we may not have third number y, if 1 was at the end). If y is 3, then the new permutation won't be *super pretty*. So, we want q to not have any pairs from the set $\{(1,3), (3,4), (4,5), \dots, (\boldsymbol{N}-2, \boldsymbol{N}-1)\}$. This is equivalent to the number of *ugly* permutations of ($\boldsymbol{N}$-2) numbers. Let's summarize. The number of *super pretty* permutations of ($\boldsymbol{N}$-1) numbers, which have the pair $(x, x+1)$ for a fixed x is $dp[\boldsymbol{N}-2]$ for all x in $\{1, 2, 3, \dots, \boldsymbol{N}-2\}$. This means that the number of *super pretty* permutations of ($\boldsymbol{N}$-1) numbers is $(\boldsymbol{N}-2) * dp[\boldsymbol{N}-2]$. In the end, the number of *ugly* permutations of $\boldsymbol{N}$ numbers becomes: $dp[\boldsymbol{N}] = (\boldsymbol{N}-1) * dp[\boldsymbol{N}-1] + (\boldsymbol{N}-2) * dp[\boldsymbol{N}-2]$, just a simple recurrent relation. It is easy to see that $dp[1] = dp[2] = 1$. This means that we can calculate $dp[\boldsymbol{N}]$ modulo $\boldsymbol{M}$ with no problem simultaneously with $\boldsymbol{N}$!. The final answer will be $\boldsymbol{N}! - dp[\boldsymbol{N}]$ modulo $\boldsymbol{M}$. The complexity is linear - $O(\boldsymbol{N})$.

The last subtask is for 23 points. Because the coefficients in the recurrent relation are dependent on $\boldsymbol{N}$, most of the standard approaches for fast calculation are not working – finding formula, matrix multiplication and so on. Here is the moment, to use that we are only calculating the answer modulo $\boldsymbol{M}$. We can make this observation – because the coefficients depend on $\boldsymbol{N}$, they repeat periodically with period $\boldsymbol{M}$. Let's look at the moment, when they start to repeat. We have:

$$\begin{aligned} dp[\boldsymbol{M}+1] &= (\boldsymbol{M} * dp[\boldsymbol{M}] + (\boldsymbol{M}-1) * dp[\boldsymbol{M}-1]) \,\%\, \boldsymbol{M} = (0 * dp[\boldsymbol{M}] - 1 * dp[\boldsymbol{M}-1]) \,\%\, \boldsymbol{M} \\ &= 1 * (-dp[\boldsymbol{M}-1]) \,\%\, \boldsymbol{M} = dp[1] * (-dp[\boldsymbol{M}-1]) \,\%\, M \end{aligned}$$

$$\begin{aligned} dp[\boldsymbol{M}+2] &= ((\boldsymbol{M}+1) * dp[\boldsymbol{M}+1] + \boldsymbol{M} * dp[\boldsymbol{M}]) \,\%\, \boldsymbol{M} = dp[\boldsymbol{M}+1] \,\%\, \boldsymbol{M} \\ &= 1 * (-dp[\boldsymbol{M}-1]) \,\%\, \boldsymbol{M} = dp[2] * (-dp[\boldsymbol{M}-1]) \,\%\, \boldsymbol{M} \end{aligned}$$

$$\begin{aligned} dp[\boldsymbol{M}+3] &= ((\boldsymbol{M}+2) * dp[\boldsymbol{M}+2] + (\boldsymbol{M}+1) * dp[\boldsymbol{M}+1]) \,\%\, \boldsymbol{M} \\ &= (2 * dp[\boldsymbol{M}+2] + 1 * dp[\boldsymbol{M}+1]) \,\%\, \boldsymbol{M} \\ &= (2 * dp[2] + 1 * dp[1]) * (-dp[\boldsymbol{M}-1]) \,\%\, \boldsymbol{M} = dp[3] * (-dp[\boldsymbol{M}-1]) \,\%\, \boldsymbol{M} \end{aligned}$$

…

Similarly, we will have:

$$\begin{aligned} dp[2 * \boldsymbol{M}] &= ((2 * \boldsymbol{M}-1) * dp[2 * \boldsymbol{M}-1] + (2 * \boldsymbol{M}-2) * dp[2 * \boldsymbol{M}-2])) \,\%\, \boldsymbol{M} \\ &= ((\boldsymbol{M}-1) * dp[\boldsymbol{M}-1] + (\boldsymbol{M}-2) * dp[\boldsymbol{M}-2]) * (-dp[\boldsymbol{M}-1]) \,\%\, \boldsymbol{M} \\ &= dp[\boldsymbol{M}] * (-dp[\boldsymbol{M}-1]) \,\%\, \boldsymbol{M} \end{aligned}$$

$$dp[2 * \boldsymbol{M}+1] = dp[\boldsymbol{M}+1] * (-dp[\boldsymbol{M}-1]) \,\%\, \boldsymbol{M} = dp[1] * (-dp[\boldsymbol{M}-1])^2 \,\%\, \boldsymbol{M}$$

Now we can clearly see this formula in the general case:

$$dp[\boldsymbol{N}] = \begin{cases} dp[\boldsymbol{M}] * (-dp[\boldsymbol{M}-1])^{[(\boldsymbol{N}-1)/\boldsymbol{M}]}, & for\ M|N \\ dp[\boldsymbol{N}\%\boldsymbol{M}] * (-dp[\boldsymbol{M}-1])^{[(\boldsymbol{N}-1)/\boldsymbol{M}]}, & otherwise \end{cases}$$

That’s the reason the module is not very big – up to 10^7. We can use the previous solution for calculating $dp[1], dp[2], \ldots, dp[\boldsymbol{M}]$ and use the above formula for calculating the dp for larger $\boldsymbol{N}$. We have to note that for calculating the remainder of the exponent, the easiest way is to use fast exponentiation. Also, the factorial for $\boldsymbol{N} \geq \boldsymbol{M}$ modulo $\boldsymbol{M}$ is zero. The final complexity of the solution is $O\left(\boldsymbol{M} + \log \frac{\boldsymbol{N}}{\boldsymbol{M}}\right)$.

An interesting fact is that $dp[\boldsymbol{M} - 1] = \pm 1$ modulo $\boldsymbol{M}$ for all $\boldsymbol{M}$, but this is hard to be proven and of course the competitors aren’t expected to see that.