K. The World of Trains

Editorial

In this problem, given N, L, T and K, your task is to count the number of arrays of size N, with integer values in a range $[1, K]$, containing exactly T consecutive subarrays of length L with **equal** elements. From now, we assume that any array described below is an array with integer values in a range $[1, K]$. Moreover, we call a consecutive subarray **good**, if it has length L and all elements equal.

The problem can be defined as a dynamic programming problem. However, the crucial thing here is to capture only required variables in a dynamic programming state. For example, perhaps the most straightforward, but a naive and slow, is use the following method. Let $g_{i,t,k,l}$ be the number of arrays of length i, containing exactly t good subarrays, with the suffix of value k occurring l times. Having g defined, we can easily compute its value for any state base on the previously computed values. However, since L and K are quite large here, we cannot use this method. It would be perfect if we only can avoid capturing the last two variables of g in a dynamic programming state.

Based on the above observation, we can use the following idea: we are going to extend any array of size i containing exactly t good subarrays, by appending a consecutive block of same values at its end. We do this, assuring that the appended value is different that the value of the last element in an array we extend, without capturing this information in a dynamic programming state.

In more details, let $f_{i,t}$ be the number of arrays of length i, containing exactly t good subarrays. If we are only able to compute $f_{i,t}$ efficiently using dynamic programming, $f_{N,T}$ will be the final result, and the problem is solved. Based on the above idea, we can define dynamic programming transitions as follows:

$$f_{i,0} = \begin{cases} K^i & \text{if } i < L \\ (K-1) \cdot \sum_{j=1}^{L-1} f_{i-j,0} & \text{if } i \geq L \end{cases}$$

$$f_{i,t} = (K-1) \cdot \left(\sum_{j=1}^{L-1} f_{i-j,t} \right) + (K-1) \cdot \left(\sum_{j=1}^{\min(t,i-L+1)} f_{i-L+1-j,t-j} \right) \text{ for } t > 0$$

When $t = 0$, so there are no good subarrays, we have two cases, In the first case, when $i < L$, each possible subarray is valid, because there is no way to form any good subarray. In the second case, we can append $K-1$ different blocks of size $j = 1, 2, \ldots, L-1$ consisting of one value different that the last element of any array taken into account to $f_{i-j,0}$. By doing this, we do not create any new good subarray, and moreover, we count take into account every valid subarray here.

On the other hand, when $t > 0$, we can either extend any valid array not shorter than $i-L+1$, with t good subarrays, kepping the number of good subarrays the same, by applying the method used in the first case, or we can create exactly $j = 1, 2, \ldots, \min(t, i-L+1)$ new good subarrays at the end of a resulting array of length i, by appending a block of $L+j-1$ same elements at the end.

Notice that, we can update our dynamic programming state as long as sums used in the above equations in order to avoid computing them from the scratch for each entry we compute. Thus, the total time complexity of this method is $O(N \cdot T)$.