Third grader's task

Let K be the size of the alphabet, that is, the number of the maximum letter that occurs in it.

First, let's calculate how many different strings can be composed if we have c_1 letters of the 1th type, c_2 letters of the 2th type, ..., c_K letters of the K type. This is the school formula:

$P(c_1, c_2, \ldots, c_K) = \frac{(c_1+c_2+\ldots+c_K)!}{c_1!\cdot c_2!\cdot\ldots\cdot c_K!}$

to quickly calculate it for different $c_1, c_2, \ldots, c_k$ pre-calculate all factorials and their reciprocals modulo $C = 998244353$ in $O(n \cdot \log C)$

In order for the string x to be less than the string t, they must have the same prefix. Let's iterate over the length of this matching prefix from 0 to $\min(n, m)$. If strings x and t have the same first i characters, then we know exactly how many letters we have left. To support this, let's create an array cnt, at the i-th position of which there will be the number of remaining letters of type i.

Let's iterate over the letter that will appear immediately after the matching prefix. For the resulting string to be less than t, this letter must be strictly less than the corresponding letter in t, and all subsequent letters can be arranged in any order. Let's calculate the number of rows x considered in this way according to the formula above.

The only case where the resulting string x can be lexicographically less than t, which we will not count, is when it is a prefix of the string t, but has a shorter length. We will separately check whether we can get such a string, and if so, add 1 to the answer.

Since at each of at most $\min(n, m)$ steps we need to go through at most K options for the next letter, and we calculate each option in $O(K)$ - we get the asymptotics $O(\min(n, m) \cdot K^2 + n \cdot \log C)$

To speed up the resulting solution, let's create an array add, in the i-th cell of which we store how many ways it will be possible to arrange the remaining letters if the letter i is put in the current position. In fact $add_i = \frac{(cnt_1+cnt_2+\ldots+cnt_K-1)!}{cnt_1!\cdot cnt_2!\cdot\ldots\cdot(cnt_i-1)!\cdot\ldots\cdot cnt_K!}$

If we learn how to maintain this array, then at each step we only need to take the sum of the elements at some of its prefix. Let's see how it changes if the next letter in the string t is i, i.e. cnt_i should decrease by 1.

For all cells $j \neq i$ add_j is replaced by $add_j \cdot \frac{cnt_i}{cnt_1+cnt_2+\ldots+cnt_K-1}$. To apply modifications to the entire array, let's create a separate variable $modify$, by which we need to multiply the value in the cell to get the value that should be there.

For cell i, add_i will be replaced by $add_i \cdot \frac{cnt_i-1}{cnt_1+cnt_2+\ldots+cnt_K-1}$. And taking into account the fact that we applied a modifier to all cells, it is enough to multiply the value of add_i by $\frac{cnt_i-1}{cnt_i}$

With this optimization, we now spend only O(K) actions at each step to calculate the prefix sum, and $O(\log(C))$ to calculate what to multiply the array cells by. We get the asymptotics $O(\min(n, m) \cdot (K + \log(C)))$

To get rid of K asymptotically, note that the only thing we want to do with the add array is take the sum at the prefix and change the value at the point. This can be done in $O(log(K)$ using the Fenwick Tree or the Segment Tree. Applying them, we get the final asymptotic $O(\min(n, m) \cdot (\log(K) + \log(C)))$.

In fact, $\log(C)$ in the asymptotics can be eliminated by precalculating modulo reciprocals for all numbers from 1 to n faster than $O(n \cdot \log(C))$, but in this task was not required.