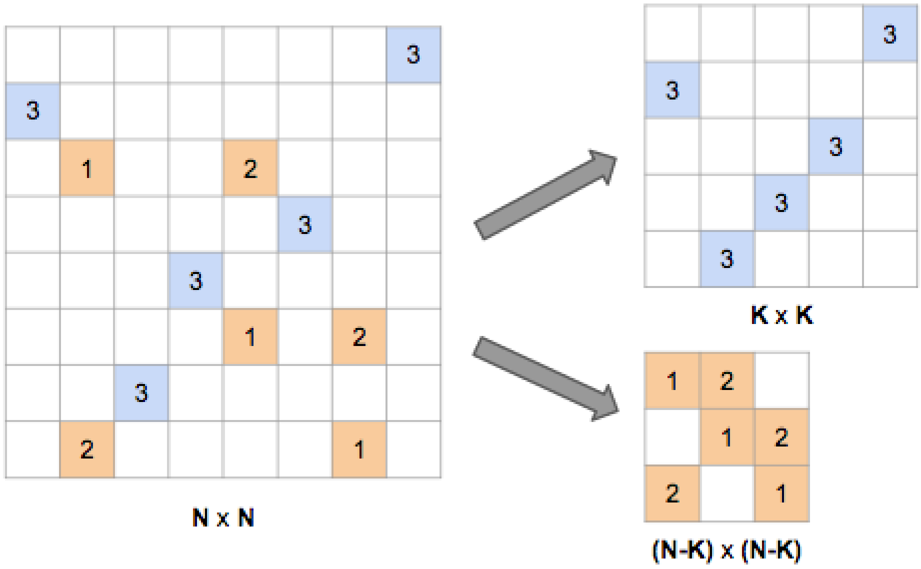
Code Jam 2015 - World Finals

Analysis: Campinatorics



Let **N** be the size of the grid, and K be the number of 3-family tents. We can decompose the problem of counting the number of arrangements for given values of **N** and K by noting that we can first choose a set of K rows and K columns to contain the 3-family tents, and the remaining set of **N**-K rows and **N**-K columns will contain the 2-family and 1-family tents. (See the diagram above.) The number of arrangements is then the product of the following:

- **The number of ways to choose the sets of rows and columns.** There are C(**N**,K)^2 ways of choosing the K rows and columns, where C(**N**,K) is a [binomial coefficient].
- **The number of ways to assign the 3-family tents to (row, column) pairs.** We can do this assignment by taking a permutation of the columns, and assigning the i^{th} column in the permutation to the i^{th} row in the set of K rows. There are [K!] such permutations.
- **The number of ways to assign the 2-family tents to (row, column) pairs.** Similarly to the 3-family tents, there are (**N**-K)! ways to do this.
- **The number of ways to assign the 1-family tents to (row, column) pairs.** Not all of the (**N**-K)! permutations of columns can be used, since we have the additional requirement that we can't use a (row, column) pair in which we've already placed a 2-family tent. To deal with this, we reformulate what we need to do here: for each row X in the set of **N**-K rows, we need to choose a unique row Y from the same set, find the column C such that there is a 2-family tent at (Y,C), and place a 1-family tent at (X,C). The space at row X, column C is guaranteed not to already have a tent. So we need a permutation of the **N**-K rows, with the additional requirement that no row is unchanged by the permutation – that is, for each row X, we choose a row *different to X* as the corresponding row Y in the permutation. Such a permutation is called a [derangement]. The number of derangements of size N-K is written !(N-K).

Now, the product of all these terms is:

C(**N**,K)^2 × K! × (**N**-K)! × !(**N**-K) = **N**!^2 / (K! × (**N**-K)!) × !(**N**-K).

We get the answer to the problem by summing this value (mod 10^9 + 7) for all values of K from X to **N**.

We can do that efficiently by precomputing K!, 1/K!, and !K mod 10^9 + 7 for all values of K up to **N**. Factorials can be computed with the obvious recurrence. 1/K! mod 10^9+7 can be computed from K! with the extended Euclidean algorithm or using Fermat's little theorem. !K can be computed with the recurrence:

!1 = 0, !2 = 1, !X=(X-1)(!(X-1)+!(X-2)).