Code Jam 2017 - Round 1B

Analysis: Stable Neigh-bors

Stable Neigh-bors: Analysis

Small dataset

In the Small dataset, all of the unicorns have manes with only a single primary color: red, yellow, or blue. None of these manes share any primary colors, so our only restriction is that two unicorns with the same mane color cannot be next to each other. However, one color might be so common that it is impossible to avoid putting two unicorns with that color next to each other. If we space unicorns with the most common color out, with some other unicorn between each pair of them, then we can accommodate up to floor(**N**/2) of the most common color. If we have more than that number with any color, the case is impossible.

Otherwise, we can extend this "every other stall" strategy to work for any case. Start at some arbitrary position on the ring of stalls, and place unicorns in the first, third, fifth, etc. stalls, all the way around the ring. When you reach the starting point again, continue to place unicorns in the second, fourth, sixth, etc. stalls. As you do this, place all of the unicorns with the most common mane color, and then all the unicorns with the next most common mane color, and then all the rest. Because all of these colors have a frequency no greater than floor(**N**/2), it is not possible for this placement strategy to put two unicorns with the same color next to each other.

Large dataset

The Large dataset introduces more complications. Unicorns with a secondary-colored mane (orange, green, or violet) can only have one type of neighbor. An orange-maned unicorn must be next to two blue-maned unicorns; similarly, a yellow must be next to purples, and a red must be next to greens.

How many instances of a primary color do we need to "surround" all unicorns with the corresponding secondary-colored mane? If the secondary color and its primary neighbor are the only two colors available, then there must be equal numbers of those two colors. Otherwise, if there are *S* instances of the secondary color, we need at least *S* + 1 instances of the primary color.

Moreover, we might as well put all secondary colors of the same type together (separated by the primary color, of course) in a single chain. Any valid arrangement without that property can be rearranged to have that property. For instance, suppose that we have two separate R-G-R chains separated by two other valid chains Z and Z': R-G-R-Z-R-G-R-Z'-. We can just rearrange this to R-G-R-G-R-Z-R-Z'-, which must also be valid.

A final useful insight is that an R-G-R-G-R chain, for example, acts just like a single R in terms of what can neighbor it on either side. So one strategy for the Large is: first, check for the case mentioned above in which only one secondary color and its neighboring primary color are present. Otherwise, for each secondary color, check that there are enough primary-color unicorns to surround the secondary-color unicorns in a chain. Then pretend that each of these chains is just a single instance of the primary-colored mane from that chain; this reduces the problem to primary colors, and the algorithm for the Small dataset works. You can substitute each chain back in (for some arbitrary instance of the appropriate primary color) once that algorithm has finished.

These Small and Large solutions run in O(**N**) time, and they are bound by reading the input, checking color frequencies, and printing the output. Other, more complex solutions (e.g., dynamic programming) also exist.