Code Jam 2017 - Round 3

Analysis: Googlements

Googlements: Analysis

Small dataset

In the Small dataset, the googlements can have a maximum length of 5. We will discuss googlements of that length; the procedure for shorter ones is the same.

Any string of length 5 that uses digits only in the range [0, 5] and is not `00000` is a googlement; there are $6^5 - 1 = 7775$ of these. One workable solution is to iterate through all of these possible googlements and simulate the decay process for each one until it reaches the sole possible looping end state of `10000`. As we do this, for each googlement, we maintain a count of how many decay chains it appears in. When we are finished, we will know how many possible ancestors each googlement has, and so we will have the answer to every possible Small test case!

Later in this analysis, we will show why `10000` is the only googlement of length 5 that is its own descendant, and we will look into how long a decay chain can go on. Even if each chain were somehow as long as the number of googlements, though, with only 7775 googlements, this method would still easily run fast enough to solve the Small.

That approach is "top-down"; a "bottom-up" approach to this tree problem also works. For a given googlement, we can figure out what digits must have been in the googlement that created it (its "direct ancestor"). For example, if we are given `12000`, we know that any direct ancestor must have one `1`, two `2`s, and two `0`s to bring the total number of digits to five. Then we can create all such googlements (e.g., `12020`) and recursively count *their* direct ancestors (e.g., `42412`) in the same way. This process does not go on forever, because (as we noted above) there are at most 7775 googlements in the tree. The worst-case scenario is `10000`, which is a descendant of every googlement.

After thinking through the bottom-up approach, only implementation details remain. The toughest part is generating direct ancestors with particular counts of particular digits. A permutation library can help, or we can write code to generate permutations in a way that avoids repeatedly considering duplicates. We can save some time across test cases by memoizing results for particular nodes of the tree, since the decay behavior of a given googlement is always exactly the same, and some of the same googlements could (and probably will) show up many times in different cases.

Large dataset

In the bottom-up Small approach outlined above, we spent a lot of time generating new direct ancestors to check. For ancestors such as `12020` that themselves had direct ancestors, this was necessary. However, it was a waste of time to construct and enumerate ancestors such as `42412`, which have no ancestors of their own. More generally, a googlement of length L with a digit sum of more than L cannot have any ancestors, since it is impossible to fit more than L digits into a length of L.

It turns out that avoiding enumerating these ancestor-less ancestors saves enough time to turn that bottom-up Small solution into a Large solution. We can do that with some help from combinatorics and the multinomial theorem.

Let's think again about the direct ancestors of `12020`. Each must have one `1`, two `2`s, and two `4`s. How many ways are there to construct such a string? Starting with a blank set of 5 digits, there are (5 choose 2) = 10 ways to place the `2`s, then (3 choose 2) = 3 ways to place the `4`s into two of the three remaining slots, then (1 choose 1) = 1 ways to place the leftover `1`. These terms multiply to give us a total of 30 ways, so `12020` has 30 direct ancestors. Since each has a digit sum of 13, which is greater than 5, none of them have their own ancestors. So we do not care what they are; we can just add 30 to our total.

At this point, we can either test this improvement on the worst-case googlement `100000000` before downloading the Large dataset, or we can reassure ourselves via more combinatorics. The googlements of length 9 with ancestors are the ones with a digit sum less than or equal to 9; we can use a balls-in-boxes argument to find that that number is (10 + 9 - 1)! / (10! * (9-1)!), which is 92377. This is a tiny fraction of the 999999999 possible googlements of length 9; we have avoided enumerating the other 999900000 or so! As long as our method for generating ancestors doesn't impose too much overhead, this is easily fast enough to solve 100 Large test cases in time, even if most or all of them explore most or all of the tree.

Appendix: Some justifications

Let's prove that there is only one looping state for a given googlement length, and that there are no other loops in the graph. We will start with some observations.

- When a googlement decays, the number of non-`0` digits in the googlement equals the sum of the digits in the googlement it decays into.
- Because of this, any googlement that has a digit other than `0`s or `1`s will decay into a googlement with a smaller digit sum.
- Any googlement consisting only of `0`s and `1`s will decay into a googlement with the same digit sum. If the googlement is `1` followed by zero or more `0`s, it will decay into itself. If the googlement has a single `1` at another position, it will decay into a `1` followed by `0`s. Otherwise, it must have at least two `1`s, so it will decay into a googlement with the same digit sum but with at least one digit other than `0` or `1`.

We can draw some useful conclusions from the above observations:

- The *only* googlement of length L that can decay into itself is a `1` followed by L-1 `0`s. For any other googlement, decay would either reduce its digit sum or create a new googlement with a digit other than `0` or `1`, which would itself decay into a googlement with a smaller digit sum.
- It can take at most two steps for a chain of googlements to lose at least one point of digit sum. This puts a bound on the height of the decay tree; it can be at most 2 times the maximum possible digit sum. (Moreover, it is even less than this; for example, a googlement of `999999999` will immediately decay into `000000009`, losing a large amount of its digit sum.