Code Jam 2017 - Qualification Round

Analysis: Tidy Numbers

Tidy Numbers: Analysis

Small dataset

The Small dataset can be solved by a simulation, going backwards through the numbers that Tatiana counted. We start with **N** and check whether it is tidy. If it is, we are done; if not, we check **N**-1, and then **N**-2 if **N**-1 is not tidy, and so on.

Checking a number for tidiness takes a single pass over its digits. We can bound the number of digits of numbers no greater than **N** by $\log_{10}$**N**. Since we iterate through at most O(**N**) numbers, the overall time complexity of this approach is O(**N** log **N**). This is easily fast enough to solve the Small dataset, in which **N** ≤ 1000, but it will not hold up for the Large dataset, which can have **N** as large as 10^{18}. If you start at the non-tidy number 11111111111111110, you have a *very* long way down to count before reaching the answer, 99999999999999999!

To iterate through the digits of an integer, you can use a language-provided utility to convert it to a string, or repeatedly take the value modulo 10 to obtain the last digit and divide by 10 to remove the last digit.

Large dataset: A greedy approach

There is an efficient greedy approach for the Large, but it has some tricky cases.

Let us name the digits of **N** N_1, N_2, ..., N_L from most to least significant (i.e., left to right, the way in which we usually write digits). Let A be our desired answer: the most recently counted tidy number. We want A to have as many digits from the left as possible in common with **N**, because that minimizes the value **N** - A.

Find the first "inversion" in **N** — that is, the minimum i such that $N_i > N_{i+1}$. If there is no such i, then **N** is already tidy and A = **N**. Otherwise: no number starting with the sequence N_1, N_2, ..., N_i can be both smaller than **N** and tidy. So, we can try making our answer start with $A_1 = N_1$, $A_2 = N_2$, ..., $A_{i-1} = N_{i-1}$. The next digit A_i has to be less than N_i, so we can try using $A_i = N_i$-1. If A_i is greater than or equal to A_{i-1}, we can make the remaining digits from A_{i+1} onward into 9s; this makes A as large as possible while ensuring that it remains tidy. However, if A_i is less than A_{i-1}, we would just be creating another inversion, and so we should instead try starting A with the digits up to N_{i-2}. If that doesn't work, we can try the digits up to N_{i-3}, and so on. We may even start with zero digits of **N**; for **N** = 211, we get A = 199. Even starting with zero digits might not work: for **N** = 100, for example, the answer is 99.

Let's condense these observations into a final algorithm. If there is no inversion, we are done. Otherwise, if the first digit of the inversion is i, find the maximum j < i such that $N_j < N_{j+1}$. If there is no such j, set j = 0. Then A starts with N_1, N_2, ..., N_j, N_{j+1}-1, and then ends with enough 9s to make it length L. The only exception is if j = 0 and N_1 is 1, in which case the answer is L-1 9s.

Since this strategy requires only one pass forward through the digits of **N** and then another pass backward, it is O(the number of digits in **N**), which, as we argued above, is O(log **N**). In this case, we can even get away with not converting the input data strings into integers!

To bypass some of the complexity above, we could have simply tried all combinations of the form SD999...99, where S is each prefix of **N** (including the empty prefix), D is each possible digit from 0 to 9, and the number of 9s is such that the total length is L. L-1 9s must also be included as a special case. Then the answer must be among those options, and it is the largest tidy number among them that does not exceed **N**. This is a common trick to simplify the code of greedy solutions: try more options than are needed, as long as the number of options is still tractable. It is often easier to implement finding the best solution among several options than to implement the checks necessary to avoid options we know we do not need.

Large dataset: A combinatoric approach

Another way to solve this with a little bit more math is to notice that there are few tidy numbers. For a fixed length L, the number of tidy numbers is the number of ways to put 8 balls in L+1 bins. Each bin represents a position at the start or end or in between two digits, and each ball represents "increase the number by 1". So, for instance, tidy number 2455 is represented by 1 ball in the first bin (skipping 1), 2 balls in the second bin (moving from 2 to 4), 1 ball in the third (4 to 5), no ball in the fourth (5 is repeated), and the rest of the balls in the last bin (moving from 5 up to 9, but there are no digits left to write). 8 balls in L+1 bins is equal to (L+8 choose 8) which, for the maximum L=18, is less than 2 million. So, we can enumerate all of the tidy numbers, skipping the rest and just return the largest we find which is no greater than **N**. To enumerate tidy numbers, we can use recursion like in this pseudocode:

```
best = 1
enum(current_string, current_digit, digits_left):
  if digits_left > 0
    enum(current_string + current_digit, current_digit, digits_left - 1)
    enum(current_string + (current_digit + 1), current_digit + 1, digits_left - 1)
  else
    if best ≤ string_to_int(current_string) ≤ N
      best = string_to_int(current_string)
```

We can also define a function that finds the next tidy number greedily and use it. We present such a function in the next subsection.

Large dataset: A binary search approach

Our original problem is: given an integer **N**, find the largest Y ≤ **N** such that Y is a tidy number. Let's consider a related problem: given an integer X, find the smallest integer Y ≥ X such that Y is a tidy number. That problem turns out to have a significantly simpler solution. Suppose that X is a sequence of digits X_1, X_2, ..., X_L, enumerated from most to least significant, as above. Then Y can be formed by finding the first inversion in the sequence (i.e., the minimum i such that $X_i > X_{i+1}$) and creating a new number that is X_1, X_2, ..., X_i plus enough additional copies of X_i to make Y as long as X. For example, if we start with the number 13254, then the first inversion is at 32, and we replace everything after the 3 with more 3s, forming 13333. If there are no inversions, then X is already tidy and Y = X. This algorithm is O(L) for an X of L digits.

With this related problem solved, we can solve the original problem by binary searching for the smallest range [X, **N**] such that the Y (as defined above) that corresponds to X is not larger than **N**. This approach takes $O(\log^2 \mathbf{N})$ time, because the binary search takes log **N** steps, and each step requires running the O(L) greedy algorithm above. As argued above, L can be bounded by log **N**. This approach is less efficient than the greedy approach above, but it is still easily fast enough to pass the Large dataset.