Daily Coding Problem #205

Problem

This problem was asked by IBM.

Given an integer, find the next permutation of it in absolute order. For example, given 48975, the next permutation would be 49578.

Solution

This is a problem that may easier to solve in one's head than to code. Nevertheless, by examining our thought process we can figure out a solution.

In the problem above, how do we know the next permutation of 48975 is 49578? Intuitively, we know that smaller changes will occur towards the end of the number. For example, if we can generate a larger permutation by only changing the ones' and tens' place digits, this will be better than a permutation that changes the hundreds' place digit.

Looking at our number, we see that there is no way to only change the ones' and tens' digits to create a higher permutation, because the tens' digit is already higher. Similarly, there is no higher permutation we can make that only changes the last three digits, because 9 > 7 > 5. Generalizing this, to find the first digit to modify, we must find the digit just before the "tail" of descending final digits.

Once we have found the number we need to change, we must decide what to put in its place. This part is straightforward: we should replace it with the next biggest number in the tail! (We know such a number exists, or else the tail would not have been broken.) So in our example, we would swap 8 with 9.

Finally, we must ensure that after our swap, the tail is sorted in ascending order. If we think about it, since the number before the tail was just slightly smaller than the number it was swapped with, the tail will *still* be in descending order. Therefore, all we need to do is reverse the elements in the tail.

```python
def get_next_permutation(num):
    num, n = list(num), len(num)

    # Get the first digit of the "tail".
    tail_start = n - 1
    while tail_start >= 0 and num[tail_start - 1] > num[tail_start]:
        tail_start -= 1

    # If the entire list is sorted in descending order, there is no larger permutation.
    if tail_start == 0:
        return None

    # Find the smallest digit in the tail that is greater than the element we need to swap.
    swap = tail_start
    while swap < n and num[tail_start - 1] < num[swap]:
        swap += 1
    swap -= 1

    # Perform the swap.
    num[tail_start - 1], num[swap] = num[swap], num[tail_start - 1]

    # Reverse the tail elements.
    start, end = tail_start, len(num) - 1
    while start < end:
        num[start], num[end] = num[end], num[start]
        start += 1; end -= 1

    return num
```

The worst-case time complexity for this algorithm is O(N). But interestingly, the average time complexity for this algorithm, averaged over all possible "next" permutations, is constant! We omit the full proof here, but as a hint, note that after using this algorithm once, the last two elements will be in sorted order, so the next run of the algorithm can be done in a single swap.

Fun fact: this method is quite old: it was discovered by Narayana Pandita in 14th century India!

Press