Java 2ms DP solution with detailed explanation and inline comments

6

bdwalker
Reputation: ★ 171

This solution certainly isn't brand new, see similar concepts here, here, here, and probably a few other top rated. Mine also isn't the shortest. However, that is kind of the point. Dynamic Programming is one of my weaknesses and making the solution shorter usually makes it more abstract and difficult to understand. My solution below is a bit more drawn out with more descriptive variable names to help make it a bit easier to understand.

The basic concept is to build up the number of ways to get to state `i` from all the previous states less than `i`. We can do this by initializing a cache with a size of `s.length() + 1`. We always set `waysToDecode[0]` to 1 because there is only 1 way to decode an empty string. We can then build up the number of ways to decode starting from the first value and work our way to the end.

We only ever need to look at the character at `i - 1` because we can't have values greater than 26, so three digits is never possible. There are four possibilities to consider: 1) The previous value is 0 and the current value is 0, we can't make progress, return 0. 2) The current value is 0, we have to use the previous value, if it is greater than 2, we can't make progress, return 0, otherwise we have to transition to this state

from `waysToDecode[i - 1]`. 3) The previous value is 0, we can't use the previous, so the only way to transition to the current state is from the previous, so use `waysToDecode[i]`. 4) lastly, both previous and `curr` can be used so there are two ways to transition to the current state, thus at `waysToDecode[i + 1]` we can get here by using all the ways we can get to `waysToDecode[i]` + all the ways to get to `waysToDecode[i - 1]`.

Keep in mind that the indices are adjusted for the cache because its size differs from the string size.

```
public class Solution {
public int numDecodings(String s) {
    if (s.isEmpty() || s.charAt(0) - '0' == 0)
    {
        return 0;
    }

    int[] waysToDecode = new int[s.length() + 1];
    waysToDecode[0] = 1;
    waysToDecode[1] = 1;
    for (int i = 1; i < s.length(); i++)
    {
        int curr = s.charAt(i) - '0';
        int prev = s.charAt(i - 1) - '0';

        // can't make progress, return 0
        if (prev == 0 && curr == 0 || (curr == 0 && (prev * 10 + curr > 26)))
        {
            return 0;
        }
        // can't use the previous value, so can only get to this state from the previous
        else if (prev == 0 || (prev * 10 + curr) > 26)
        {
            waysToDecode[i + 1] = waysToDecode[i];
        }
```

```
            // can't use current state, can only get to this state from i - 1 state
            else if (curr == 0)
            {
                waysToDecode[i + 1] = waysToDecode[i - 1];
            }
            // can get to this state from the previous two states
            else
            {
                waysToDecode[i + 1] = waysToDecode[i] + waysToDecode[i - 1];
            }
        }
```

}

1 POSTS

520 VIEWS

Log in to reply