Week 3&4 Interviewing Challenge Solution

Problem Statement

A message containing letters from A-Z can be encoded into numbers using the following mapping -
A -> 1
B -> 2
.
.
Z -> 26

To decode a message, all the digits must be grouped and then mapped back into letters using the reverse of the mapping above.
For example, 11106 can be mapped into -
- "AAJF" with the grouping (1 1 10 6)
- "KJF" with the grouping (11 10 6)

So, given a string s, containing only digits from 0 to 9, write a function getNumDecodings(string s) that returns the number of ways to decode the given string.

Solution

We can use dynamic programming to solve this problem. Here we'll be discussing an **iterative**, **bottom-up** approach.
The idea is to maintain an array called dp to store the results of our subproblems. Basically, dp[i] will represent the total number of decodings assuming index i represents the string's starting position.
Assuming we've computed dp[i+1] to dp[s.length - 1], the question is how do we use these results to find dp[i]? What should our recurrence relation look like?
Well it turns out that there are many cases we have to take into account.
But before we discuss these cases, just note that to compute dp[i], we only need dp[i+1] and dp[i+2].

Case 1 : s[i+1] = '0'
If s[i+1] is '0', then s[i] must be '1' or '2' to get a valid decoding. If not, then there will be no valid decodings possible at all!
For example, look at the string "2703". It is not possible to decode this string at all since '7' and '0' are consecutive and "70" cannot be decoded.
However, if we had "2103", we would have a valid decoding since "10" can be interpreted as 'J'.
So if s[i] == '1' or '2', **dp[i] = dp[i + 1]**
Or else, no valid decodings (return 0)

Case 2 : s[i] = '0'
This case is pretty simple. If our current character is '0', it is not possible to produce any decodings assuming this is the starting point. However, this doesn't necessarily mean that there are no valid decodings for the whole string.
We can simply say **dp[i] = 0**

Case 3 : All other scenarios (so s[i] & s[i+1] are between '1' & '9')
If s[i] and s[i+1] together produce a number between 11 and 26,
then **dp[i] = dp[i+1] + dp[i+2]**
Or else simply, **dp[i] = dp[i+1]**

Computing this for all characters in the string, our desired result is ultimately what we get in dp[0]. So our asymptotic time and space complexities for this implementation are both O(n). We could actually further reduce the space needed to O(1) since as we mentioned, we only needed dp[i+1] & dp[i+2] to compute dp[i].

You can find the C++ implementation with O(n) time and space here.
If you have any questions, feel free to message **@Abhik Mazumder** on Slack.