

91. Decode Ways

Medium

A message containing letters from A-Z can be encoded into numbers using the following mapping:

'A' -> "1"

'B' -> "2"

...

'Z' -> "26"

To decode an encoded message, all the digits must be grouped then mapped back into letters using the reverse of the mapping above (there may be multiple ways). For example, "11106" can be mapped into:

"AAJF" with the grouping (1 1 10 6)

"KJF" with the grouping (11 10 6)

Note that the grouping (1 11 06) is invalid because "06" cannot be mapped into 'F' since "6" is different from "06".

Given a string *s* containing only digits, return the number of ways to decode it.

The test cases are generated so that the answer fits in a 32-bit integer.

Example 1:

Input: *s* = "12"

Output: 2

Explanation: "12" could be decoded as "AB" (1 2) or "L" (12).

Example 2:

Input: *s* = "226"

Output: 3

Explanation: "226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

Example 3:

Input: *s* = "06"

Output: 0

Explanation: "06" cannot be mapped to "F" because of the leading zero ("6" is different from "06").

```
class Solution:
    def numDecodings(self, s: str) -> int:
        dp_memo = {len(s): 1}

        for i in range(len(s) - 1, -1, -1):
            if s[i] == "0":
                dp_memo[i] = 0
            else:
                dp_memo[i] = dp_memo[i + 1]

            if i + 1 < len(s) and (
                s[i] == "1" or s[i] == "2" and s[i + 1] in "0123456"
            ):
                dp_memo[i] += dp_memo[i + 2]

        return dp_memo[0]
```