91. Decode Ways 🗹

March 1, 2020 | 23.4K views

Average Rating: 4.96 (24 votes)

'A' -> 1 'B' -> 2

A message containing letters from A-Z is being encoded to numbers using the following mapping:

```
'Z' -> 26
Given a non-empty string containing only digits, determine the total number of ways to decode it.
```

```
Example 1:
```

Input: "12" Output: 2

```
Example 2:
 Input: "226"
 Output: 3
 Explanation: It could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).
```

The most important point to understand in this problem is that at any given step when you are trying to

25 to Y. As long as it's a valid decoding we move ahead to decode the rest of the string.

The subproblem could be thought of as number of ways decoding a substring.

decode a string of numbers it can either be a single digit decode e.g. 1 to A or a double digit decode e.g.

"326" - String to be decoded!

Yes, that's one of the ways. **NOTE:** We did a single digit

decode for all the digits.
Digits 1-9 can be mapped to from A-I. Also keep in my There is no mapping for "32", since double digit mapping zero 'O' is also a single digit and it doesn't have any are only between 11 - 26 mapping by itself.

```
The above diagram shows string "326" could be decoded in two ways.
Approach 1: Recursive Approach with Memoization
Intuition
The problem deals with finding number of ways of decoding a string. What helps to crack the problem is to
think why there would be many ways to decode a string. The reason is simple since at any given point we
```

either decode using two digits or single digit. This choice while decoding can lead to different

■ String to be decoded!

# decode then there is only one choice to make at each step. Hence, there has to be only If no double digit decoding is

one way of decoding any

given string. Assuming there is no zero in the string.

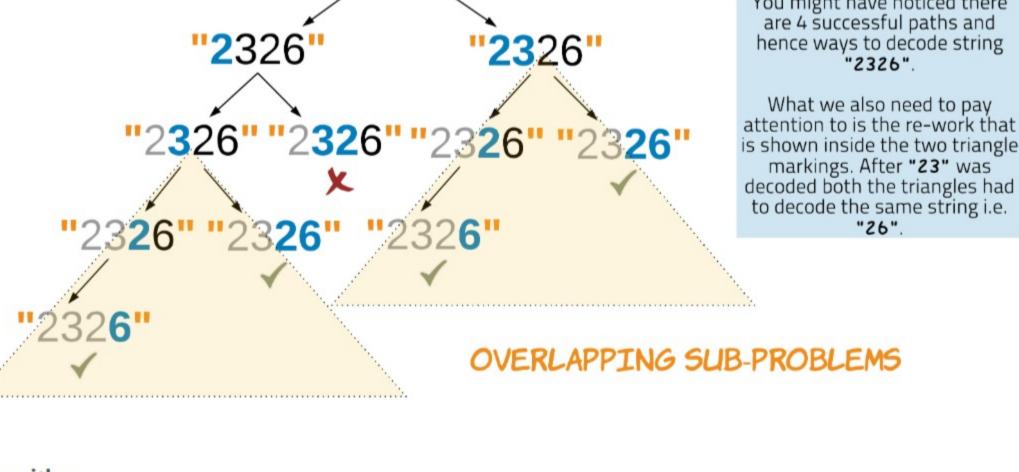
"3**26**" "3**26**"

combinations.

IF ONLY DOUBLE DIGIT DECODING IS ALSO ALLOWED. This problem allows both single and **"326"** two digit decode. Thus, at every step we have **TWO** choices to make. Single digit decode
 Two digit decode

allowed

Thus at any given time for a string we enter a recursion after successfully decoding two digits to a single character or a single digit to a character. This leads to multiple paths to decoding the entire string. If a given path leads to the end of the string this means we could successfully decode the string. If at any point in the traversal we encounter digits which cannot be decoded, we backtrack from that path. <u>"326"</u>



to decode the same string i.e. "26". OVERLAPPING SUB-PROBLEMS **Algorithm** 1. Enter recursion with the given string i.e. start with index 0. 2. For the terminating case of the recursion we check for the end of the string. If we have reached the end of the string we return 1. 3. Every time we enter recursion it's for a substring of the original string. For any recursion if the first

of ways.

7

8 9

10

11 12

13

14 15

21

22 23

24 25

26

27

**Complexity Analysis** 

# Return 1 for success.

if index == len(s):

if s[index] == '0':

if index == len(s)-1:

# Save for memoization

self.memo[index] = ans

return 0

return 1

return 1 16 17 # Memoization is needed since we might encounter the same sub-string. 18 19 if index in self.memo: 20 return self.memo[index]

+ (self.recursive\_with\_memo(index+2, s) if (int(s[index : index+2]) <= 26) else 0)

• Time Complexity: O(N), where N is length of the string. Memoization helps in pruning the recursion

tree and hence decoding for an index only once. Thus this solution is linear time complexity. ullet Space Complexity: O(N). The dictionary used for memoization would take the space equal to the length of the string. There would be an entry for each index value. The recursion stack would also be equal to the length of the string.

Algorithm

character at index i-1 of s.

Python

class Solution(object):

:type s: str

:rtype: int

if not s:

def numDecodings(self, s):

return 0

# Array to store the subproblem results

Follow up : Optimized Iterative Approach, O(1) Space

Thanks Takkes for suggesting this optimization to second approach.

Java

1

2 3 4

5

6 7

8

9

10

Initialize the DP

array

decode is possible we add the results of the previous indices. Thus, in this race we don't just pass the baton.

5. We check if valid single digit decode is possible. This just means the character at index s[i-1] is non-

zero. Since we do not have a decoding for zero. If the valid single digit decoding is possible then we

add dp[i-1] to dp[i]. Since all the ways up to (i-1)-th character now lead up to i-th character too.

6. We check if valid two digit decode is possible. This means the substring s[i-2]s[i-1] is between 10

7. Once we reach the end of the dp array we would have the number of ways of decoding string s.

to 26. If the valid two digit decoding is possible then we add dp[i-2] to dp[i].

 $dp = [0 \text{ for } \underline{\ } \text{ in } range(len(s) + 1)]$ 11 12 13 dp[0] = 1# Ways to decode a string of size 1 is 1. Unless the string is '0'. 14 15 # '0' doesn't have a single digit decode. dp[1] = 0 if s[0] == '0' else 116 17 18 for i in range(2, len(dp)): 19 20 21 # Check if successful single digit decode is possible. 22 if s[i-1] != '0': 23 dp[i] += dp[i-1]24 # Check if successful two digit decode is possible. 25 26 two\_digit = int(s[i-2 : i]) 27 if two\_digit >= 10 and two\_digit <= 26:

In Approach 2 we are using an array dp to save the results for future. As we move ahead character by character of the given string, we look back only two steps. For calculating dp[i] we need to know dp[i-1] and dp[i-2] only. Thus, we can easily cut down our O(N) space requirement to O(1) by using only two variables to store the last two results.

O Previous

Comments: 22

Rate this article: \* \* \* \* \*

Preview

This problem is all about the edge cases involved with "0" being in the input. Without the "0"s the problem is pretty straightforward, but these solutions choose not to include it? Seems pretty lazy, despite the informative graphics. This solution does not help me understand test cases like "01" -> 0, "10" -> 1, or "909" -> 0.

two int variables instead of an array. Brings down space complexity from O(n) to O(1).

In the second solution we are only ever accessing the last two elements. Therefore we could just use

- it is similar with climbing stairs, right? 9 A V C Share Reply
- frustratedcoder 🛊 3 🗿 April 6, 2020 8:31 AM @godayaldivya - Can you please help me understand why dp[0] = 1 ? Thanks!
- edge cases makes this question more difficult than the average of medium. 2 A V C Share Reply undefitied ★ 92 ② March 3, 2020 10:52 PM

ZoroDuncan ★ 20 ② May 16, 2020 4:01 AM

chaaloftin \* 18 • July 4, 2020 1:17 AM

tpt5cu \* 121 ② June 29, 2020 10:43 AM

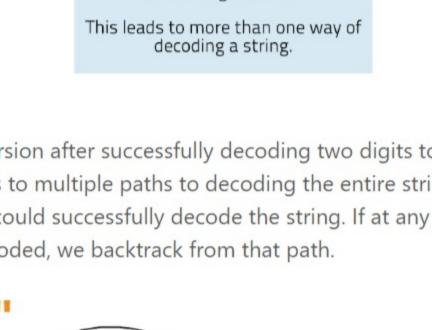
If I could give this a triple downvote I would.

- we return 0 2 A V C Share Reply SHOW 1 REPLY
  - purplebeth **1** 6 **2** June 29, 2020 9:40 PM
    - the substring that starts with i? how do we know if this is a substring of length 1 or 2? **SHOW 1 REPLY**

Solution

IF SINGLE DIGIT DECODING WAS ALLOWED. **"3**26**"** If there was just single digit

# When two digit decode is allowed, after decoding "3" we have two choices. We can either decode "2" to "B" or "26" to "Z"



"32" cannot be mapped to a character. Only digits which lie in the range of 1 to 26 have a mapping.

A succesful path means successful decode. Thus there are two paths

and hence two ways to decode "326"

If no double digit decoding is allowed

2326 You might have noticed there are 4 successful paths and hence ways to decode string "2326". What we also need to pay attention to is the re-work that is shown inside the two triangle

In the following diagram we can see how the paths have to deal with similar subproblems. Overlapping

subproblems means we can reuse the answer. Thus, we do memoization to solve this problem.

# is determined by making a recursive call to the function with index + 1 for next substring string and index + 2 after checking for valid 2-digit decode. The result is also stored in memo with key as current index, for saving for future overlapping subproblems. **С**ору Python Java 1 class Solution: 2 def \_\_init\_\_(self): self.memo = {} 3 4 5 def recursive\_with\_memo(self, index, s) -> int: 6 # If you reach the end of the string

character is 0 then terminate that path by returning 0. Thus this path won't contribute to the number

4. Memoization helps to reduce the complexity which would otherwise be exponential. We check the

5. If the result is already in memo we return the result. Otherwise the number of ways for the given string

dictionary memo to see if the result for the given substring already exists.

# If the string starts with a zero, it can't be decoded

ans = self.recursive\_with\_memo(index+1, s) \

- Approach 2: Iterative Approach The iterative approach might be a little bit less intuitive. Let's try to understand it. We use an array for DP to store the results for subproblems. A cell with index i of the dp array is used to store the number of decode ways for substring of s from index 0 to index i-1. We initialize the starting two indices of the dp array. It's similar to relay race where the first runner is given a baton to be passed to the subsequent runners. The first two indices of the dp array hold a baton. As we iterate the dp array from left to right this baton which signifies the number of ways of decoding is passed to the next index or not depending on whether the decode is possible. dp[i] can get the baton from two other previous indices, either i-1 or i-2. Two previous indices are involved since both single and two digit decodes are possible. Unlike the relay race we don't get only one baton in the end. The batons add up as we pass on. If someone has one baton, they can provide a copy of it to everyone who comes to them with a success. Thus, leading to number of ways of reaching the end. LET'S BREAK IT DOWN i DP ARRAY 1 1 1 1 1 1 1
- just means if there lead to **DP[i]**. This is are N ways of an example of how the reaching **DP[i-1]**. baton at **DP[i-2]** is not Now those N ways passed on to **DP[i]**. also lead to **DP[i]** dp[i] = Number of ways of decoding substring s[:i]. So we might say <math>dp[i] = dp[i-1] + dp[i-2], which is not always true for this decode ways problem. As shown in the above diagram, only when the The baton is passed to the next index or not depending on possibility of the decode. 1. If the string s is empty or null we return the result as 0. 2. Initialize dp array. dp[0] = 1 to provide the baton to be passed. 3. If the first character of the string is zero then no decode is possible hence initialize dp[1] to 0, otherwise the first character is valid to pass on the baton, dp[1] = 1. 4. Iterate the dp array starting at index 2. The index i of dp is the i-th character of the string s, that is

Finding the

number ways of

decoding the

substring "32"

what we would

store in **DP[i]** 

SINGLE DIGIT

DECODE

"2" is a valid single

digit decode. Thus

whatever is in **DP[i-1]** 

we add to **DP[i]**. This

DOUBLE DIGIT

DECODE

"32" is not valid

decode. Thus the

decode paths that

lead to **DP[i-2]** do not

**С**ору

Next 👀

Sort By -

Post

A Report

A Report

A Report

**Complexity Analysis** • Time Complexity: O(N), where N is length of the string. We iterate the length of dp array which is N+1. • Space Complexity: O(N). The length of the DP array.

32 A V C Share Reply **SHOW 4 REPLIES** 

8 A V C Share Reply

SHOW 1 REPLY

**SHOW 2 REPLIES** 

Type comment here... (Markdown is supported)

A Report **SHOW 1 REPLY** windrainer ★ 6 ② March 3, 2020 3:13 AM for the recursive solution, May I know why it returns 1 in the case index = str.length()-1 **SHOW 3 REPLIES** 

There is a typo in If the string s is empty or null we return the result as 1. - actually

I don't understand what the i represents in the memoization dict = is it the substring that ends with i, or

0 ∧ ∨ ♂ Share ¬ Reply (123)

This is a hard problem I think

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