Same as DP, my explanation and Python sample code

1

orbuluh
Reputation: ★ 353

Think of the case: "leetcode" with dictionary {"leet", "ode", "lee", "tcode"}

The simple logic is: `if it can't be constructed from the beginning, it can't form the whole string`

Take "ode" case in the example, since we can't form the string "leetc" from the dictionary, even though "ode" does in the target string, we still can't finish the whole target string.

This observation gives us an insight: we need to track, from the beginning, what parts have we solved, use variable `beforeHasSolved`.

So, initially, `beforeHasSolved = [0]`, which says we have solved the substring before index `0`, namely, the empty substring.

Then if we start to traverse the string: "l"... not find, "le"...not find, "lee" ... find! Then we can make `beforeHasSolved = [0, 3]`

Then for further traversing the string at index `e`, we can check whether substring([0, e]) or substring([3, e]) is

in the dictionary. This ensures us to say: "we've found a way to form the substring to index `e` from the beginning"

and so on...

PS.
`for b in beforeHasSolved[::-1]:`, would speed up a little bit than using `for b in beforeHasSolved:`, which means that there might be more short terms in the test cases. But both of them could solved the problem.

```
def wordBreak(self, s, wordDict):
    beforeHasSolved = [0]
    for e in range(len(s)):
        for b in beforeHasSolved[::-1]:
            if s[b:e+1] in wordDict:
                beforeHasSolved.append(e+1)
                break
    return len(s) in beforeHasSolved
```

DYNAMIC-PROGRAMMING 1260 PYTHON 4178 SOLUTION-SHARING 15815

1 POSTS

286 VIEWS

Log in to reply