Word Break Problem Documentation

Problem Statement

Given a string s and a dictionary wordDict of strings, determine if the string s can be segmented into a sequence of one or more dictionary words. The dictionary words can be reused multiple times.

Function Signature:

```
def wordBreak(s: str, wordDict: List[str]) -> bool:
```

Inputs

- **s (str):** A string that needs to be segmented.
- **wordDict (List[str]):** A list of words that form the dictionary. All words in the dictionary are unique and consist of lowercase English letters only.

Outputs

- Returns True if the string s can be segmented into a sequence of one or more words from the dictionary. Otherwise, returns False.

Example 1:

- **Input:** s = "leetcode", wordDict = ["leet", "code"]
- **Output:** True
- **Explanation:** The string "leetcode" can be segmented into "leet" and "code" which are both present in the dictionary.

Example 2:

- **Input:** s = "applepenapple", wordDict = ["apple", "pen"]
- **Output:** True
- **Explanation:** The string "applepenapple" can be segmented into "apple", "pen", and "apple", all of which are present in the dictionary. The same word can be reused.

Example 3:

- **Input:** s = "catsandog", wordDict = ["cats", "dog", "sand", "and", "cat"]
- **Output:** False
- **Explanation:** The string "catsandog" cannot be segmented into words from the dictionary.

Constraints

- 1 <= s.length <= 300
- 1 <= wordDict.length <= 1000
- 1 <= wordDict[i].length <= 20
- s and wordDict[i] consist of only lowercase English letters.
- All strings in wordDict are unique.

Solution Approach

The solution employs dynamic programming to determine if the string s can be segmented using the dictionary wordDict. Here's the detailed explanation of the approach:

1. **Initialization:**

- Convert the list wordDict to a set wordSet for O(1) average time complexity on lookups.
- Initialize a list dp of size n + 1 (where n is the length of s), with all values set to False. The value dp[i] represents whether the substring s[0:i] can be segmented using the dictionary. Set dp[0] to True because an empty string can always be segmented.

2. **Dynamic Programming Transition:**

- Iterate through each index i from 1 to n.
- For each i, check all substrings s[j:i] where 0 \leq j < i.
- If dp[j] is True (indicating that s[0:j] can be segmented) and s[j:i] is present in wordSet, set dp[i] to True.
- If dp[i] is set to True, break the inner loop as there's no need to check further substrings.

3. **Result:**

- Return dp[n], which will be True if the entire string s can be segmented into dictionary words, otherwise False.

This solution efficiently checks whether the string s can be segmented into a sequence of words from wordDict using dynamic programming, with a time complexity of O(n2)O(n^2)O(n2) and space complexity of O(n)O(n)O(n).