Problem 139: Word Break

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given a string

s

and a dictionary of strings

wordDict

, return

true

if

s

can be segmented into a space-separated sequence of one or more dictionary words.

Note

that the same word in the dictionary may be reused multiple times in the segmentation.

Example 1:

Input:

s = "leetcode", wordDict = ["leet","code"]

Output:

true

Explanation:

Return true because "leetcode" can be segmented as "leet code".

Example 2:

Input:

s = "applepenapple", wordDict = ["apple","pen"]

Output:

true

Explanation:

Return true because "applepenapple" can be segmented as "apple pen apple". Note that you are allowed to reuse a dictionary word.

Example 3:

Input:

s = "catsandog", wordDict = ["cats","dog","sand","and","cat"]

Output:

false

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 the strings of

wordDict

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
bool wordBreak(string s, vector<string>& wordDict) {

}
};
```

Java:

```
class Solution {
public boolean wordBreak(String s, List<String> wordDict) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def wordBreak(self, s, wordDict):
"""
:type s: str
:type wordDict: List[str]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {string[]} wordDict
* @return {boolean}
*/
var wordBreak = function(s, wordDict) {

};
```

TypeScript:

```
function wordBreak(s: string, wordDict: string[]): boolean {

};
```

C#:

```
public class Solution {
public bool WordBreak(string s, IList<string> wordDict) {

}
}
```

C:

```
bool wordBreak(char* s, char** wordDict, int wordDictSize) {

}
```

Go:

```
func wordBreak(s string, wordDict []string) bool {

}
```

Kotlin:

```
class Solution {
fun wordBreak(s: String, wordDict: List<String>): Boolean {

}
}
```

Swift:

```
class Solution {
func wordBreak(_ s: String, _ wordDict: [String]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn word_break(s: String, word_dict: Vec<String>) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @param {String[]} word_dict
# @return {Boolean}
def word_break(s, word_dict)
```

```
end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String[] $wordDict
* @return Boolean
*/
function wordBreak($s, $wordDict) {

}
}
```

Dart:

```
class Solution {
bool wordBreak(String s, List<String> wordDict) {

}
}
```

Scala:

```
object Solution {
def wordBreak(s: String, wordDict: List[String]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec word_break(s :: String.t, word_dict :: [String.t]) :: boolean
def word_break(s, word_dict) do

end
end
```

Erlang:

```
-spec word_break(S :: unicode:unicode_binary(), WordDict ::
[unicode:unicode_binary()]) -> boolean().
word_break(S, WordDict) ->
.
```

Racket:

```
(define/contract (word-break s wordDict)
(-> string? (listof string?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Word Break
* Difficulty: Medium
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
bool wordBreak(string s, vector<string>& wordDict) {

}
};
```

Java Solution:

```
/**
* Problem: Word Break
* Difficulty: Medium
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public boolean wordBreak(String s, List<String> wordDict) {

}
}
```

Python3 Solution:

```
"""
Problem: Word Break
Difficulty: Medium
Tags: array, string, dp, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def wordBreak(self, s: str, wordDict: List[str]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def wordBreak(self, s, wordDict):
"""
:type s: str
:type wordDict: List[str]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Word Break
* Difficulty: Medium
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} s
* @param {string[]} wordDict
* @return {boolean}
*/
var wordBreak = function(s, wordDict) {

};
```

TypeScript Solution:

```
/**
* Problem: Word Break
* Difficulty: Medium
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function wordBreak(s: string, wordDict: string[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Word Break
* Difficulty: Medium
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public bool WordBreak(string s, IList<string> wordDict) {

}
}
```

C Solution:

```
/*
* Problem: Word Break
* Difficulty: Medium
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

bool wordBreak(char* s, char** wordDict, int wordDictSize) {

}
```

Go Solution:

```
// Problem: Word Break
// Difficulty: Medium
// Tags: array, string, dp, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func wordBreak(s string, wordDict []string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun wordBreak(s: String, wordDict: List<String>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func wordBreak(_ s: String, _ wordDict: [String]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Word Break
// Difficulty: Medium
// Tags: array, string, dp, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn word_break(s: String, word_dict: Vec<String>) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String[]} word_dict
# @return {Boolean}
def word_break(s, word_dict)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String[] $wordDict
* @return Boolean
*/
function wordBreak($s, $wordDict) {

}
}
```

Dart Solution:

```
class Solution {
bool wordBreak(String s, List<String> wordDict) {

}
}
```

Scala Solution:

```
object Solution {
def wordBreak(s: String, wordDict: List[String]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec word_break(s :: String.t, word_dict :: [String.t]) :: boolean
def word_break(s, word_dict) do

end
end
```

Erlang Solution:

```
-spec word_break(S :: unicode:unicode_binary(), WordDict ::
[unicode:unicode_binary()]) -> boolean().
word_break(S, WordDict) ->
```

```
.
```

Racket Solution:

```
(define/contract (word-break s wordDict)
(-> string? (listof string?) boolean?)
)
```