Problem 648: Replace Words

Problem Information

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

Problem Description

In English, we have a concept called

root

, which can be followed by some other word to form another longer word - let's call this word

derivative

. For example, when the

root

"help"

is followed by the word

"ful"

, we can form a derivative

"helpful"

.

Given a

dictionary

consisting of many

roots

and a

sentence

consisting of words separated by spaces, replace all the derivatives in the sentence with the

root

forming it. If a derivative can be replaced by more than one

root

, replace it with the

root

that has

the shortest length

.

Return

the

sentence

after the replacement.

Example 1:

Input:

dictionary = ["cat","bat","rat"], sentence = "the cattle was rattled by the battery"

Output:

"the cat was rat by the bat"

Example 2:

Input:

dictionary = ["a","b","c"], sentence = "aadsfasf absbs bbab cadsfafs"

Output:

"a a b c"

Constraints:

1 <= dictionary.length <= 1000

1 <= dictionary[i].length <= 100

dictionary[i]

consists of only lower-case letters.

1 <= sentence.length <= 10

6

sentence

consists of only lower-case letters and spaces.

The number of words in

sentence

is in the range

[1, 1000]

The length of each word in

sentence

is in the range

[1, 1000]

Every two consecutive words in

sentence

will be separated by exactly one space.

sentence

does not have leading or trailing spaces.

Code Snippets

C++:

```
class Solution {
public:
string replaceWords(vector<string>& dictionary, string sentence) {

}
};
```

Java:

```
class Solution {
public String replaceWords(List<String> dictionary, String sentence) {

}
```

```
}
```

Python3:

```
class Solution:
def replaceWords(self, dictionary: List[str], sentence: str) -> str:
```

Python:

```
class Solution(object):
def replaceWords(self, dictionary, sentence):
"""
:type dictionary: List[str]
:type sentence: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string[]} dictionary
* @param {string} sentence
* @return {string}
*/
var replaceWords = function(dictionary, sentence) {

};
```

TypeScript:

```
function replaceWords(dictionary: string[], sentence: string): string {

};
```

C#:

```
public class Solution {
public string ReplaceWords(IList<string> dictionary, string sentence) {

}
}
```

C:

```
char* replaceWords(char** dictionary, int dictionarySize, char* sentence) {

}
```

Go:

```
func replaceWords(dictionary []string, sentence string) string {

}
```

Kotlin:

```
class Solution {
fun replaceWords(dictionary: List<String>, sentence: String): String {

}
}
```

Swift:

```
class Solution {
func replaceWords(_ dictionary: [String], _ sentence: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn replace_words(dictionary: Vec<String>, sentence: String) -> String {

}
}
```

Ruby:

```
# @param {String[]} dictionary
# @param {String} sentence
# @return {String}
def replace_words(dictionary, sentence)
```

```
end
```

PHP:

```
class Solution {

/**
* @param String[] $dictionary
* @param String $sentence
* @return String
*/
function replaceWords($dictionary, $sentence) {

}
}
```

Dart:

```
class Solution {
String replaceWords(List<String> dictionary, String sentence) {

}
}
```

Scala:

```
object Solution {
def replaceWords(dictionary: List[String], sentence: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec replace_words(dictionary :: [String.t], sentence :: String.t) ::
String.t
def replace_words(dictionary, sentence) do

end
end
```

Erlang:

```
-spec replace_words(Dictionary :: [unicode:unicode_binary()], Sentence ::
unicode:unicode_binary()) -> unicode:unicode_binary().
replace_words(Dictionary, Sentence) ->
.
```

Racket:

```
(define/contract (replace-words dictionary sentence)
(-> (listof string?) string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Replace Words
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
string replaceWords(vector<string>& dictionary, string sentence) {

}
};
```

Java Solution:

```
/**
* Problem: Replace Words
* Difficulty: Medium
* Tags: array, string, hash
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String replaceWords(List<String> dictionary, String sentence) {

}
}
```

Python3 Solution:

```
"""
Problem: Replace Words
Difficulty: Medium
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def replaceWords(self, dictionary: List[str], sentence: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def replaceWords(self, dictionary, sentence):
"""
:type dictionary: List[str]
:type sentence: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Replace Words
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[]} dictionary
* @param {string} sentence
* @return {string}
*/
var replaceWords = function(dictionary, sentence) {

};
```

TypeScript Solution:

```
/**
* Problem: Replace Words
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function replaceWords(dictionary: string[], sentence: string): string {

};
```

C# Solution:

```
/*
* Problem: Replace Words
* Difficulty: Medium
* Tags: array, string, hash
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string ReplaceWords(IList<string> dictionary, string sentence) {

}
}
```

C Solution:

```
/*
* Problem: Replace Words
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* replaceWords(char** dictionary, int dictionarySize, char* sentence) {

}
```

Go Solution:

```
// Problem: Replace Words
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func replaceWords(dictionary []string, sentence string) string {

}
```

Kotlin Solution:

```
class Solution {
fun replaceWords(dictionary: List<String>, sentence: String): String {

}
}
```

Swift Solution:

```
class Solution {
func replaceWords(_ dictionary: [String], _ sentence: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Replace Words
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn replace_words(dictionary: Vec<String>, sentence: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String[]} dictionary
# @param {String} sentence
# @return {String}
def replace_words(dictionary, sentence)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $dictionary
* @param String $sentence
* @return String
*/
function replaceWords($dictionary, $sentence) {

}
}
```

Dart Solution:

```
class Solution {
String replaceWords(List<String> dictionary, String sentence) {

}
}
```

Scala Solution:

```
object Solution {
def replaceWords(dictionary: List[String], sentence: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec replace_words(dictionary :: [String.t], sentence :: String.t) ::
String.t
def replace_words(dictionary, sentence) do

end
end
```

Erlang Solution:

```
-spec replace_words(Dictionary :: [unicode:unicode_binary()], Sentence ::
unicode:unicode_binary()) -> unicode:unicode_binary().
```

```
replace_words(Dictionary, Sentence) ->
.
```

Racket Solution:

```
(define/contract (replace-words dictionary sentence)
(-> (listof string?) string? string?)
)
```