Problem 127: Word Ladder

Problem Information

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

Problem Description

A

transformation sequence

from word

beginWord

to word

endWord

using a dictionary

wordList

is a sequence of words

beginWord -> s

1

-> s

2

-> ... -> s

k

such that:

Every adjacent pair of words differs by a single letter.

Every

s

i

for

1 <= i <= k

is in

wordList

. Note that

beginWord

does not need to be in

wordList

.

s

k

== endWord

Given two words,

beginWord

and

endWord

, and a dictionary

wordList

, return

the

number of words

in the

shortest transformation sequence

from

beginWord

to

endWord

, or

0

if no such sequence exists.

Example 1:

Input:

beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]

Output:

5

Explanation:

One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> cog", which is 5 words long.

Example 2:

Input:

beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]

Output:

0

Explanation:

The endWord "cog" is not in wordList, therefore there is no valid transformation sequence.

Constraints:

1 <= beginWord.length <= 10

endWord.length == beginWord.length

1 <= wordList.length <= 5000

wordList[i].length == beginWord.length

beginWord

,

endWord

, and

wordList[i]

consist of lowercase English letters.

beginWord != endWord

All the words in

wordList

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int ladderLength(string beginWord, string endWord, vector<string>& wordList)
{

}
};
```

Java:

```
class Solution {
public int ladderLength(String beginWord, String endWord, List<String>
wordList) {

}
```

```
}
```

Python3:

```
class Solution:
def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) ->
int:
```

Python:

```
class Solution(object):
def ladderLength(self, beginWord, endWord, wordList):
"""
:type beginWord: str
:type endWord: str
:type wordList: List[str]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} beginWord
* @param {string} endWord
* @param {string[]} wordList
* @return {number}
*/
var ladderLength = function(beginWord, endWord, wordList) {

};
```

TypeScript:

```
function ladderLength(beginWord: string, endWord: string, wordList:
string[]): number {

};
```

C#:

```
public class Solution {
public int LadderLength(string beginWord, string endWord, IList<string>
```

```
wordList) {

}
}
```

C:

```
int ladderLength(char* beginWord, char* endWord, char** wordList, int
wordListSize) {

}
```

Go:

```
func ladderLength(beginWord string, endWord string, wordList []string) int {

}
```

Kotlin:

```
class Solution {
fun ladderLength(beginWord: String, endWord: String, wordList: List<String>):
Int {

}
}
```

Swift:

```
class Solution {
func ladderLength(_ beginWord: String, _ endWord: String, _ wordList:
[String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn ladder_length(begin_word: String, end_word: String, word_list:
Vec<String>) -> i32 {

}
```

```
}
```

Ruby:

```
# @param {String} begin_word
# @param {String} end_word
# @param {String[]} word_list
# @return {Integer}
def ladder_length(begin_word, end_word, word_list)

end
```

PHP:

```
class Solution {

/**
* @param String $beginWord
* @param String $endWord
* @param String[] $wordList
* @return Integer
*/
function ladderLength($beginWord, $endWord, $wordList) {

}
}
```

Dart:

```
class Solution {
int ladderLength(String beginWord, String endWord, List<String> wordList) {

}
}
```

Scala:

```
object Solution {
def ladderLength(beginWord: String, endWord: String, wordList: List[String]):
Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec ladder_length(begin_word :: String.t, end_word :: String.t, word_list
:: [String.t]) :: integer
def ladder_length(begin_word, end_word, word_list) do

end
end
```

Erlang:

```
-spec ladder_length(BeginWord :: unicode:unicode_binary(), EndWord ::
unicode:unicode_binary(), WordList :: [unicode:unicode_binary()]) ->
integer().
ladder_length(BeginWord, EndWord, WordList) ->
.
```

Racket:

```
(define/contract (ladder-length beginWord endWord wordList)
(-> string? string? (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Word Ladder
* Difficulty: Hard
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
class Solution {
public:
int ladderLength(string beginWord, string endWord, vector<string>& wordList)
{

}
};
```

Java Solution:

```
/**
* Problem: Word Ladder
* Difficulty: Hard
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int ladderLength(String beginWord, String endWord, List<String>
wordList) {

}
}
```

Python3 Solution:

```
"""
Problem: Word Ladder
Difficulty: Hard
Tags: string, hash, search

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) ->
```

```
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def ladderLength(self, beginWord, endWord, wordList):
"""
:type beginWord: str
:type endWord: str
:type wordList: List[str]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Word Ladder
* Difficulty: Hard
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} beginWord
* @param {string} endWord
* @param {string[]} wordList
* @return {number}
*/
var ladderLength = function(beginWord, endWord, wordList) {

};
```

TypeScript Solution:

```
/**
* Problem: Word Ladder
* Difficulty: Hard
```

```
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function ladderLength(beginWord: string, endWord: string, wordList:
string[]): number {

};
```

C# Solution:

```
/*
* Problem: Word Ladder
* Difficulty: Hard
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int LadderLength(string beginWord, string endWord, IList<string>
wordList) {

}
}
```

C Solution:

```
/*
* Problem: Word Ladder
* Difficulty: Hard
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

int ladderLength(char* beginWord, char* endWord, char** wordList, int
wordListSize) {

}
```

Go Solution:

```
// Problem: Word Ladder
// Difficulty: Hard
// Tags: string, hash, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func ladderLength(beginWord string, endWord string, wordList []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun ladderLength(beginWord: String, endWord: String, wordList: List<String>):
Int {

}
}
```

Swift Solution:

```
class Solution {
func ladderLength(_ beginWord: String, _ endWord: String, _ wordList:
[String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Word Ladder
// Difficulty: Hard
// Tags: string, hash, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn ladder_length(begin_word: String, end_word: String, word_list:
Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} begin_word
# @param {String} end_word
# @param {String[]} word_list
# @return {Integer}
def ladder_length(begin_word, end_word, word_list)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $beginWord
* @param String $endWord
* @param String[] $wordList
* @return Integer
*/
function ladderLength($beginWord, $endWord, $wordList) {

}
}
```

Dart Solution:

```
class Solution {
int ladderLength(String beginWord, String endWord, List<String> wordList) {

}
}
```

Scala Solution:

```
object Solution {
def ladderLength(beginWord: String, endWord: String, wordList: List[String]):
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec ladder_length(begin_word :: String.t, end_word :: String.t, word_list
:: [String.t]) :: integer
def ladder_length(begin_word, end_word, word_list) do

end
end
```

Erlang Solution:

```
-spec ladder_length(BeginWord :: unicode:unicode_binary(), EndWord ::
unicode:unicode_binary(), WordList :: [unicode:unicode_binary()]) ->
integer().
ladder_length(BeginWord, EndWord, WordList) ->
.
```

Racket Solution:

```
(define/contract (ladder-length beginWord endWord wordList)
(-> string? string? (listof string?) exact-integer?)
)
```