Problem 2515: Shortest Distance to Target String in a Circular Array

Problem Information

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

Problem Description

You are given a

0-indexed

circular

string array

words

and a string

target

. A

circular array

means that the array's end connects to the array's beginning.

Formally, the next element of

words[i]

is

words[(i + 1) % n]

and the previous element of

words[i]

is

words[(i - 1 + n) % n]

, where

n

is the length of

words

.

Starting from

startIndex

, you can move to either the next word or the previous word with

1

step at a time.

Return

the

shortest

distance needed to reach the string

target

. If the string

target

does not exist in

words

, return

-1

.

Example 1:

Input:

words = ["hello","i","am","leetcode","hello"], target = "hello", startIndex = 1

Output:

1

Explanation:

We start from index 1 and can reach "hello" by - moving 3 units to the right to reach index 4. - moving 2 units to the left to reach index 4. - moving 4 units to the right to reach index 0. - moving 1 unit to the left to reach index 0. The shortest distance to reach "hello" is 1.

Example 2:

Input:

words = ["a","b","leetcode"], target = "leetcode", startIndex = 0

Output:

1

Explanation:

We start from index 0 and can reach "leetcode" by - moving 2 units to the right to reach index 3. - moving 1 unit to the left to reach index 3. The shortest distance to reach "leetcode" is 1.

Example 3:

Input:

words = ["i","eat","leetcode"], target = "ate", startIndex = 0

Output:

-1

Explanation:

Since "ate" does not exist in

words

, we return -1.

Constraints:

1 <= words.length <= 100

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

words[i]

and

target

consist of only lowercase English letters.

0 <= startIndex < words.length

Code Snippets

C++:

```
class Solution {
public:
int closestTarget(vector<string>& words, string target, int startIndex) {

}
};
```

Java:

```
class Solution {
public int closestTarget(String[] words, String target, int startIndex) {

}
}
```

Python3:

```
class Solution:
def closestTarget(self, words: List[str], target: str, startIndex: int) ->
int:
```

Python:

```
class Solution(object):
def closestTarget(self, words, target, startIndex):
"""
:type words: List[str]
:type target: str
:type startIndex: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} words
* @param {string} target
* @param {number} startIndex
* @return {number}
*/
var closestTarget = function(words, target, startIndex) {

};
```

TypeScript:

```
function closestTarget(words: string[], target: string, startIndex: number):
number {

};
```

C#:

```
public class Solution {
public int ClosestTarget(string[] words, string target, int startIndex) {

}
}
```

C:

```
int closestTarget(char** words, int wordsSize, char* target, int startIndex)
{

}
```

Go:

```
func closestTarget(words []string, target string, startIndex int) int {

}
```

Kotlin:

```
class Solution {
fun closestTarget(words: Array<String>, target: String, startIndex: Int): Int
{
```

```
}
}
```

Swift:

```
class Solution {
func closestTarget(_ words: [String], _ target: String, _ startIndex: Int) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn closest_target(words: Vec<String>, target: String, start_index: i32)
-> i32 {

}
}
```

Ruby:

```
# @param {String[]} words
# @param {String} target
# @param {Integer} start_index
# @return {Integer}
def closest_target(words, target, start_index)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @param String $target
* @param Integer $startIndex
* @return Integer
*/
```

```
function closestTarget($words, $target, $startIndex) {

}
}
```

Dart:

```
class Solution {
int closestTarget(List<String> words, String target, int startIndex) {

}
}
```

Scala:

```
object Solution {
def closestTarget(words: Array[String], target: String, startIndex: Int): Int
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec closest_target(words :: [String.t], target :: String.t, start_index ::
integer) :: integer
def closest_target(words, target, start_index) do

end
end
```

Erlang:

```
-spec closest_target(Words :: [unicode:unicode_binary()], Target ::
unicode:unicode_binary(), StartIndex :: integer()) -> integer().
closest_target(Words, Target, StartIndex) ->
.
```

Racket:

```
(define/contract (closest-target words target startIndex)
(-> (listof string?) string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Distance to Target String in a Circular Array
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int closestTarget(vector<string>& words, string target, int startIndex) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Distance to Target String in a Circular Array
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int closestTarget(String[] words, String target, int startIndex) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Shortest Distance to Target String in a Circular Array
Difficulty: Easy
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def closestTarget(self, words: List[str], target: str, startIndex: int) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def closestTarget(self, words, target, startIndex):
"""
:type words: List[str]
:type target: str
:type startIndex: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Shortest Distance to Target String in a Circular Array
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

/**
* @param {string[]} words
* @param {string} target
* @param {number} startIndex
* @return {number}
*/
var closestTarget = function(words, target, startIndex) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Distance to Target String in a Circular Array
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function closestTarget(words: string[], target: string, startIndex: number):
number {

};
```

C# Solution:

```
/*
* Problem: Shortest Distance to Target String in a Circular Array
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
public class Solution {
public int ClosestTarget(string[] words, string target, int startIndex) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Distance to Target String in a Circular Array
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int closestTarget(char** words, int wordsSize, char* target, int startIndex)
{

}
```

Go Solution:

```
// Problem: Shortest Distance to Target String in a Circular Array
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func closestTarget(words []string, target string, startIndex int) int {

}
```

Kotlin Solution:

```
class Solution {
fun closestTarget(words: Array<String>, target: String, startIndex: Int): Int
```

```
{

}
}
```

Swift Solution:

```
class Solution {
func closestTarget(_ words: [String], _ target: String, _ startIndex: Int) ->
Int {

}
}
```

Rust Solution:

```
// Problem: Shortest Distance to Target String in a Circular Array
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn closest_target(words: Vec<String>, target: String, start_index: i32)
-> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @param {String} target
# @param {Integer} start_index
# @return {Integer}
def closest_target(words, target, start_index)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @param String $target
* @param Integer $startIndex
* @return Integer
*/
function closestTarget($words, $target, $startIndex) {

}
}
```

Dart Solution:

```
class Solution {
int closestTarget(List<String> words, String target, int startIndex) {

}
}
```

Scala Solution:

```
object Solution {
def closestTarget(words: Array[String], target: String, startIndex: Int): Int
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec closest_target(words :: [String.t], target :: String.t, start_index ::
integer) :: integer
def closest_target(words, target, start_index) do

end
end
```

Erlang Solution:

```
-spec closest_target(Words :: [unicode:unicode_binary()], Target ::
unicode:unicode_binary(), StartIndex :: integer()) -> integer().
closest_target(Words, Target, StartIndex) ->
.
```

Racket Solution:

```
(define/contract (closest-target words target startIndex)
(-> (listof string?) string? exact-integer? exact-integer?)
)
```