Problem 411: Minimum Unique Word Abbreviation

Problem Information

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

Problem Description

A string can be

abbreviated

by replacing any number of

non-adjacent

substrings with their lengths. For example, a string such as

"substitution"

could be abbreviated as (but not limited to):

"s10n"

(

"s

ubstitutio

n"

)

"sub4u4"

(

"sub

stit

u

tion

"

)

"12"

(

"

substitution

"

)

"su3i1u2on"

(

"su

bst

i

t

u

ti

on"

)

"substitution"

(no substrings replaced)

Note that

"s55n"

(

"s

ubsti

tutio

n"

) is not a valid abbreviation of

"substitution"

because the replaced substrings are adjacent.

The

length

of an abbreviation is the number of letters that were not replaced plus the number of substrings that were replaced. For example, the abbreviation

"s10n"

has a length of

3

(

2

letters +

1

substring) and

"su3i1u2on"

has a length of

9

(

6

letters +

3

substrings).

Given a target string

target

and an array of strings

dictionary

, return

an

abbreviation

of

target

with the

shortest possible length

such that it is

not an abbreviation

of

any

string in

dictionary

. If there are multiple shortest abbreviations, return any of them

.

Example 1:

Input:

target = "apple", dictionary = ["blade"]

Output:

"a4"

Explanation:

The shortest abbreviation of "apple" is "5", but this is also an abbreviation of "blade". The next shortest abbreviations are "a4" and "4e". "4e" is an abbreviation of blade while "a4" is not. Hence, return "a4".

Example 2:

Input:

target = "apple", dictionary = ["blade","plain","amber"]

Output:

"1p3"

Explanation:

"5" is an abbreviation of both "apple" but also every word in the dictionary. "a4" is an abbreviation of "apple" but also "amber". "4e" is an abbreviation of "apple" but also "blade". "1p3", "2p2", and "3l1" are the next shortest abbreviations of "apple". Since none of them are abbreviations of words in the dictionary, returning any of them is correct.

Constraints:

m == target.length

n == dictionary.length

1 <= m <= 21

0 <= n <= 1000

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

log

2

(n) + m <= 21

if

n > 0

target

and

dictionary[i]

consist of lowercase English letters.

dictionary

does not contain

target

.

Code Snippets

C++:

```
class Solution {
public:
string minAbbreviation(string target, vector<string>& dictionary) {

}
};
```

Java:

```
class Solution {
public String minAbbreviation(String target, String[] dictionary) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minAbbreviation(target: string, dictionary: string[]): string {

};
```

C#:

```
public class Solution {
public string MinAbbreviation(string target, string[] dictionary) {
```

```
}
}
```

C:

```
char* minAbbreviation(char* target, char** dictionary, int dictionarySize) {

}
```

Go:

```
func minAbbreviation(target string, dictionary []string) string {

}
```

Kotlin:

```
class Solution {
fun minAbbreviation(target: String, dictionary: Array<String>): String {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {String} target
# @param {String[]} dictionary
# @return {String}
def min_abbreviation(target, dictionary)

end
```

PHP:

```
class Solution {

/**
* @param String $target
* @param String[] $dictionary
* @return String
*/
function minAbbreviation($target, $dictionary) {

}
}
```

Dart:

```
class Solution {
String minAbbreviation(String target, List<String> dictionary) {

}
}
```

Scala:

```
object Solution {
def minAbbreviation(target: String, dictionary: Array[String]): String = {

}
}
```

Elixir:

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

```
end
end
```

Erlang:

```
-spec min_abbreviation(Target :: unicode:unicode_binary(), Dictionary ::
[unicode:unicode_binary()]) -> unicode:unicode_binary().
min_abbreviation(Target, Dictionary) ->
.
```

Racket:

```
(define/contract (min-abbreviation target dictionary)
(-> string? (listof string?) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Unique Word Abbreviation
* Difficulty: Hard
* Tags: array, string, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
string minAbbreviation(string target, vector<string>& dictionary) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Unique Word Abbreviation
* Difficulty: Hard
* Tags: array, string, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public String minAbbreviation(String target, String[] dictionary) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Unique Word Abbreviation
Difficulty: Hard
Tags: array, string, tree

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Unique Word Abbreviation
* Difficulty: Hard
* Tags: array, string, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Unique Word Abbreviation
* Difficulty: Hard
* Tags: array, string, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function minAbbreviation(target: string, dictionary: string[]): string {

};
```

C# Solution:

```
/*
* Problem: Minimum Unique Word Abbreviation
* Difficulty: Hard
```

```
* Tags: array, string, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public string MinAbbreviation(string target, string[] dictionary) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Unique Word Abbreviation
* Difficulty: Hard
* Tags: array, string, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

char* minAbbreviation(char* target, char** dictionary, int dictionarySize) {

}
```

Go Solution:

```
// Problem: Minimum Unique Word Abbreviation
// Difficulty: Hard
// Tags: array, string, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func minAbbreviation(target string, dictionary []string) string {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun minAbbreviation(target: String, dictionary: Array<String>): String {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Unique Word Abbreviation
// Difficulty: Hard
// Tags: array, string, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn min_abbreviation(target: String, dictionary: Vec<String>) -> String {

}
}
```

Ruby Solution:

```
# @param {String} target
# @param {String[]} dictionary
# @return {String}
def min_abbreviation(target, dictionary)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $target
* @param String[] $dictionary
* @return String
*/
function minAbbreviation($target, $dictionary) {

}
}
```

Dart Solution:

```
class Solution {
String minAbbreviation(String target, List<String> dictionary) {

}
}
```

Scala Solution:

```
object Solution {
def minAbbreviation(target: String, dictionary: Array[String]): String = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec min_abbreviation(Target :: unicode:unicode_binary(), Dictionary ::
[unicode:unicode_binary()]) -> unicode:unicode_binary().
min_abbreviation(Target, Dictionary) ->
.
```

Racket Solution:

```
(define/contract (min-abbreviation target dictionary)
(-> string? (listof string?) string?)
)
```