Problem 820: Short Encoding of Words

Problem Information

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

Problem Description

A

valid encoding

of an array of

words

is any reference string

s

and array of indices

indices

such that:

words.length == indices.length

The reference string

s

ends with the

'#'

character.

For each index

indices[i]

, the

substring

of

s

starting from

indices[i]

and up to (but not including) the next

'#'

character is equal to

words[i]

.

Given an array of

words

, return

the

length of the shortest reference string

s

possible of any

valid encoding

of

words

.

Example 1:

Input:

words = ["time", "me", "bell"]

Output:

10

Explanation:

A valid encoding would be s =

"time#bell#" and indices = [0, 2, 5

]. words[0] = "time", the substring of s starting from indices[0] = 0 to the next '#' is underlined in "

time

#bell#" words[1] = "me", the substring of s starting from indices[1] = 2 to the next '#' is underlined in "ti

me

#bell#" words[2] = "bell", the substring of s starting from indices[2] = 5 to the next '#' is underlined in "time#

bell

#"

Example 2:

Input:

words = ["t"]

Output:

2

Explanation:

A valid encoding would be s = "t#" and indices = [0].

Constraints:

1 <= words.length <= 2000

1 <= words[i].length <= 7

words[i]

consists of only lowercase letters.

Code Snippets

C++:

```
class Solution {
public:
int minimumLengthEncoding(vector<string>& words) {
```

```
}
};
```

Java:

```
class Solution {
public int minimumLengthEncoding(String[] words) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string[]} words
* @return {number}
*/
var minimumLengthEncoding = function(words) {

};
```

TypeScript:

```
function minimumLengthEncoding(words: string[]): number {

};
```

C#:

```
public class Solution {
public int MinimumLengthEncoding(string[] words) {

}
}
```

C:

```
int minimumLengthEncoding(char** words, int wordsSize) {

}
```

Go:

```
func minimumLengthEncoding(words []string) int {

}
```

Kotlin:

```
class Solution {
fun minimumLengthEncoding(words: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func minimumLengthEncoding(_ words: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_length_encoding(words: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} words
# @return {Integer}
def minimum_length_encoding(words)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @return Integer
*/
function minimumLengthEncoding($words) {

}
}
```

Dart:

```
class Solution {
int minimumLengthEncoding(List<String> words) {

}
}
```

Scala:

```
object Solution {
def minimumLengthEncoding(words: Array[String]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_length_encoding(words :: [String.t]) :: integer
def minimum_length_encoding(words) do
```

```
end
end
```

Erlang:

```
-spec minimum_length_encoding(Words :: [unicode:unicode_binary()]) ->
integer().
minimum_length_encoding(Words) ->
.
```

Racket:

```
(define/contract (minimum-length-encoding words)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Short Encoding of Words
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* 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:
int minimumLengthEncoding(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Short Encoding of Words
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* 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 int minimumLengthEncoding(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Short Encoding of Words
Difficulty: Medium
Tags: array, string, tree, hash

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 minimumLengthEncoding(self, words: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Short Encoding of Words
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* 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[]} words
* @return {number}
*/
var minimumLengthEncoding = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Short Encoding of Words
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* 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 minimumLengthEncoding(words: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Short Encoding of Words
* Difficulty: Medium
* Tags: array, string, tree, hash
*
```

```
* 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 int MinimumLengthEncoding(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Short Encoding of Words
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* 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
*/

int minimumLengthEncoding(char** words, int wordsSize) {

}
```

Go Solution:

```
// Problem: Short Encoding of Words
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// 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 minimumLengthEncoding(words []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumLengthEncoding(words: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumLengthEncoding(_ words: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Short Encoding of Words
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// 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 minimum_length_encoding(words: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @return {Integer}
def minimum_length_encoding(words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @return Integer
*/
function minimumLengthEncoding($words) {

}
}
```

Dart Solution:

```
class Solution {
int minimumLengthEncoding(List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def minimumLengthEncoding(words: Array[String]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_length_encoding(words :: [String.t]) :: integer
def minimum_length_encoding(words) do

end
end
```

Erlang Solution:

```
-spec minimum_length_encoding(Words :: [unicode:unicode_binary()]) ->
integer().
minimum_length_encoding(Words) ->
.
```

Racket Solution:

```
(define/contract (minimum-length-encoding words)
(-> (listof string?) exact-integer?)
)
```