Problem 471: Encode String with Shortest Length

Problem Information

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

Problem Description

Given a string

s

, encode the string such that its encoded length is the shortest.

The encoding rule is:

k[encoded_string]

, where the

encoded_string

inside the square brackets is being repeated exactly

k

times.

k

should be a positive integer.

If an encoding process does not make the string shorter, then do not encode it. If there are several solutions, return

any of them

.

Example 1:

Input:

s = "aaa"

Output:

"aaa"

Explanation:

There is no way to encode it such that it is shorter than the input string, so we do not encode it.

Example 2:

Input:

s = "aaaaa"

Output:

"5[a]"

Explanation:

"5[a]" is shorter than "aaaaa" by 1 character.

Example 3:

Input:

s = "aaaaaaaaaa"

Output:

"10[a]"

Explanation:

"a9[a]" or "9[a]a" are also valid solutions, both of them have the same length = 5, which is the same as "10[a]".

Constraints:

1 <= s.length <= 150

s

consists of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string encode(string s) {

}
};
```

Java:

```
class Solution {
public String encode(String s) {

}
}
```

Python3:

```
class Solution:
def encode(self, s: str) -> str:
```

Python:

```
class Solution(object):
def encode(self, s):
"""
:type s: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {string}
*/
var encode = function(s) {

};
```

TypeScript:

```
function encode(s: string): string {

};
```

C#:

```
public class Solution {
public string Encode(string s) {

}
}
```

C:

```
char* encode(char* s) {

}
```

Go:

```
func encode(s string) string {

}
```

Kotlin:

```
class Solution {
fun encode(s: String): String {

}
}
```

Swift:

```
class Solution {
func encode(_ s: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn encode(s: String) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @return {String}
def encode(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return String
```

```
*/
function encode($s) {

}
}
```

Dart:

```
class Solution {
String encode(String s) {

}
}
```

Scala:

```
object Solution {
def encode(s: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec encode(s :: String.t) :: String.t
def encode(s) do

end
end
```

Erlang:

```
-spec encode(S :: unicode:unicode_binary()) -> unicode:unicode_binary().
encode(S) ->
.
```

Racket:

```
(define/contract (encode s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Encode String with Shortest Length
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
string encode(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Encode String with Shortest Length
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public String encode(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Encode String with Shortest Length
Difficulty: Hard
Tags: string, dp

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

class Solution:
def encode(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def encode(self, s):
"""
:type s: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Encode String with Shortest Length
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} s
* @return {string}
*/
var encode = function(s) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Encode String with Shortest Length
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function encode(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Encode String with Shortest Length
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public string Encode(string s) {

}
}
```

C Solution:

```
/*
* Problem: Encode String with Shortest Length
* Difficulty: Hard
```

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

char* encode(char* s) {

}
```

Go Solution:

```
// Problem: Encode String with Shortest Length
// Difficulty: Hard
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func encode(s string) string {

}
```

Kotlin Solution:

```
class Solution {
fun encode(s: String): String {

}
}
```

Swift Solution:

```
class Solution {
func encode(_ s: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Encode String with Shortest Length
// Difficulty: Hard
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn encode(s: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {String}
def encode(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return String
*/
function encode($s) {

}
}
```

Dart Solution:

```
class Solution {
String encode(String s) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def encode(s: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec encode(s :: String.t) :: String.t
def encode(s) do

end
end
```

Erlang Solution:

```
-spec encode(S :: unicode:unicode_binary()) -> unicode:unicode_binary().
encode(S) ->
.
```

Racket Solution:

```
(define/contract (encode s)
(-> string? string?)
)
```