Problem 394: Decode String

Problem Information

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

Problem Description

Given an encoded string, return its decoded string.

The encoding rule is:

k[encoded_string]

, where the

encoded_string

inside the square brackets is being repeated exactly

k

times. Note that

k

is guaranteed to be a positive integer.

You may assume that the input string is always valid; there are no extra white spaces, square brackets are well-formed, etc. Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers,

k

. For example, there will not be input like

3a

or

2[4]

.

The test cases are generated so that the length of the output will never exceed

10

5

.

Example 1:

Input:

s = "3[a]2[bc]"

Output:

"aaabcbc"

Example 2:

Input:

s = "3[a2[c]]"

Output:

"accaccacc"

Example 3:

Input:

s = "2[abc]3[cd]ef"

Output:

"abcabccdcdcdef"

Constraints:

1 <= s.length <= 30

s

consists of lowercase English letters, digits, and square brackets

'[]'

.

s

is guaranteed to be

a valid

input.

All the integers in

s

are in the range

[1, 300]

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* decodeString(char* s) {

}
```

Go:

```
func decodeString(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Decode String
* Difficulty: Medium
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
```

```
string decodeString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Decode String
* Difficulty: Medium
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String decodeString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Decode String
Difficulty: Medium
Tags: string, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Decode String
* Difficulty: Medium
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Decode String
* Difficulty: Medium
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function decodeString(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Decode String
* Difficulty: Medium
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

C Solution:

```
/*
* Problem: Decode String
* Difficulty: Medium
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* decodeString(char* s) {

}
```

Go Solution:

```
// Problem: Decode String
// Difficulty: Medium
// Tags: string, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func decodeString(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Decode String
// Difficulty: Medium
// Tags: string, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

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