Problem 3703: Remove K-Balanced Substrings

Problem Information

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

Problem Description

You are given a string

s

consisting of

'('

and

')'

, and an integer

k

.

A

string

is

k-balanced

if it is

exactly

k

consecutive

'('

followed by

k

consecutive

')'

, i.e.,

'(' * k + ')' * k

.

For example, if

k = 3

, k-balanced is

"((()))"

.

You must

repeatedly

remove all

non-overlapping k-balanced

substrings

from

s

, and then join the remaining parts. Continue this process until no k-balanced

substring

exists.

Return the final string after all possible removals.

Example 1:

Input:

s = "(())", k = 1

Output:

""

Explanation:

k-balanced substring is

"()"

Step

Current

s

k-balanced

Result

s

1

(())

(

()

)

()

2

()

()

Empty

Thus, the final string is

""

.

Example 2:

Input:

s = "(()(", k = 1

Output:

"(("

Explanation:

k-balanced substring is

"()"

Step

Current

s

k-balanced

Result

s

1

(()(

(

()

(

((

2

((

-

((

Thus, the final string is

"(("

.

Example 3:

Input:

s = "(((()))()()()", k = 3

Output:

"()()()"

Explanation:

k-balanced substring is

"((()))"

Step

Current

s

k-balanced

Result

s

1

((()))()()()

((()))

()()()

()()()

2

()()()

-

()()()

Thus, the final string is

"()()()"

.

Constraints:

2 <= s.length <= 10

5

s

consists only of

'('

and

')'

.

1 <= k <= s.length / 2

Code Snippets

C++:

```
class Solution {
public:
string removeSubstring(string s, int k) {

}
};
```

Java:

```
class Solution {
public String removeSubstring(String s, int k) {

}
}
```

Python3:

```
class Solution:
def removeSubstring(self, s: str, k: int) -> str:
```

Python:

```
class Solution(object):
def removeSubstring(self, s, k):
"""
:type s: str
:type k: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number} k
```

```
* @return {string}
*/
var removeSubstring = function(s, k) {

};
```

TypeScript:

```
function removeSubstring(s: string, k: number): string {

};
```

C#:

```
public class Solution {
public string RemoveSubstring(string s, int k) {

}
}
```

C:

```
char* removeSubstring(char* s, int k) {

}
```

Go:

```
func removeSubstring(s string, k int) string {

}
```

Kotlin:

```
class Solution {
fun removeSubstring(s: String, k: Int): String {

}
}
```

Swift:

```
class Solution {
func removeSubstring(_ s: String, _ k: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn remove_substring(s: String, k: i32) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
# @return {String}
def remove_substring(s, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return String
*/
function removeSubstring($s, $k) {

}
}
```

Dart:

```
class Solution {
String removeSubstring(String s, int k) {

}
```

```
}
```

Scala:

```
object Solution {
def removeSubstring(s: String, k: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec remove_substring(s :: String.t, k :: integer) :: String.t
def remove_substring(s, k) do

end
end
```

Erlang:

```
-spec remove_substring(S :: unicode:unicode_binary(), K :: integer()) ->
unicode:unicode_binary().
remove_substring(S, K) ->
.
```

Racket:

```
(define/contract (remove-substring s k)
(-> string? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove K-Balanced Substrings
* Difficulty: Medium
* Tags: string, tree, stack
```

```
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
string removeSubstring(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Remove K-Balanced Substrings
* Difficulty: Medium
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public String removeSubstring(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove K-Balanced Substrings
Difficulty: Medium
Tags: string, tree, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
```

```
"""

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

Python Solution:

```
class Solution(object):
def removeSubstring(self, s, k):
"""
:type s: str
:type k: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Remove K-Balanced Substrings
* Difficulty: Medium
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} s
* @param {number} k
* @return {string}
*/
var removeSubstring = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove K-Balanced Substrings
* Difficulty: Medium
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function removeSubstring(s: string, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Remove K-Balanced Substrings
* Difficulty: Medium
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* 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 RemoveSubstring(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Remove K-Balanced Substrings
* Difficulty: Medium
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

char* removeSubstring(char* s, int k) {

}
```

Go Solution:

```
// Problem: Remove K-Balanced Substrings
// Difficulty: Medium
// Tags: string, tree, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func removeSubstring(s string, k int) string {

}
```

Kotlin Solution:

```
class Solution {
fun removeSubstring(s: String, k: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func removeSubstring(_ s: String, _ k: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Remove K-Balanced Substrings
// Difficulty: Medium
// Tags: string, tree, stack
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn remove_substring(s: String, k: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @return {String}
def remove_substring(s, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return String
*/
function removeSubstring($s, $k) {

}
}
```

Dart Solution:

```
class Solution {
String removeSubstring(String s, int k) {

}
}
```

Scala Solution:

```
object Solution {
def removeSubstring(s: String, k: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec remove_substring(s :: String.t, k :: integer) :: String.t
def remove_substring(s, k) do

end
end
```

Erlang Solution:

```
-spec remove_substring(S :: unicode:unicode_binary(), K :: integer()) ->
unicode:unicode_binary().
remove_substring(S, K) ->
.
```

Racket Solution:

```
(define/contract (remove-substring s k)
(-> string? exact-integer? string?)
)
```