Problem 1910: Remove All Occurrences of a Substring

Problem Information

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

Problem Description

Given two strings

s

and

part

, perform the following operation on

s

until

all

occurrences of the substring

part

are removed:

Find the

leftmost

occurrence of the substring

part

and

remove

it from

s

.

Return

s

after removing all occurrences of

part

.

A

substring

is a contiguous sequence of characters in a string.

Example 1:

Input:

s = "daabcbaabcbc", part = "abc"

Output:

"dab"

Explanation

: The following operations are done: - s = "da

abc

baabcbc", remove "abc" starting at index 2, so s = "dabaabcbc". - s = "daba

abc

bc", remove "abc" starting at index 4, so s = "dababc". - s = "dab

abc

", remove "abc" starting at index 3, so s = "dab". Now s has no occurrences of "abc".

Example 2:

Input:

s = "axxxxyyyyb", part = "xy"

Output:

"ab"

Explanation

: The following operations are done: - s = "axxx

xy

yyyb", remove "xy" starting at index 4 so s = "axxxyyyb". - s = "axx

xy

yyb", remove "xy" starting at index 3 so s = "axxyyb". - s = "ax xy

yb", remove "xy" starting at index 2 so s = "axyb". - s = "a

xy

b", remove "xy" starting at index 1 so s = "ab". Now s has no occurrences of "xy".

Constraints:

1 <= s.length <= 1000

1 <= part.length <= 1000

s

and

part

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string removeOccurrences(string s, string part) {

}
};
```

Java:

```
class Solution {
public String removeOccurrences(String s, String part) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} s
* @param {string} part
* @return {string}
*/
var removeOccurrences = function(s, part) {

};
```

TypeScript:

```
function removeOccurrences(s: string, part: string): string {

};
```

C#:

```
public class Solution {
public string RemoveOccurrences(string s, string part) {

}
}
```

C:

```
char* removeOccurrences(char* s, char* part) {

}
```

Go:

```
func removeOccurrences(s string, part string) string {

}
```

Kotlin:

```
class Solution {
fun removeOccurrences(s: String, part: String): String {

}
}
```

Swift:

```
class Solution {
func removeOccurrences(_ s: String, _ part: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn remove_occurrences(s: String, part: String) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} part
# @return {String}
def remove_occurrences(s, part)
```

```
end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $part
* @return String
*/
function removeOccurrences($s, $part) {

}
}
```

Dart:

```
class Solution {
String removeOccurrences(String s, String part) {

}
}
```

Scala:

```
object Solution {
def removeOccurrences(s: String, part: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec remove_occurrences(s :: String.t, part :: String.t) :: String.t
def remove_occurrences(s, part) do

end
end
```

Erlang:

```
-spec remove_occurrences(S :: unicode:unicode_binary(), Part ::
unicode:unicode_binary()) -> unicode:unicode_binary().
remove_occurrences(S, Part) ->
.
```

Racket:

```
(define/contract (remove-occurrences s part)
(-> string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove All Occurrences of a Substring
* 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 removeOccurrences(string s, string part) {

}
};
```

Java Solution:

```
/**
* Problem: Remove All Occurrences of a Substring
* 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 removeOccurrences(String s, String part) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove All Occurrences of a Substring
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 removeOccurrences(self, s: str, part: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove All Occurrences of a Substring
* 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 {string} part
* @return {string}
*/
var removeOccurrences = function(s, part) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove All Occurrences of a Substring
* 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 removeOccurrences(s: string, part: string): string {

};
```

C# Solution:

```
/*
* Problem: Remove All Occurrences of a Substring
* 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 RemoveOccurrences(string s, string part) {

}
}
```

C Solution:

```
/*
* Problem: Remove All Occurrences of a Substring
* 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* removeOccurrences(char* s, char* part) {

}
```

Go Solution:

```
// Problem: Remove All Occurrences of a Substring
// 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 removeOccurrences(s string, part string) string {

}
```

Kotlin Solution:

```
class Solution {
fun removeOccurrences(s: String, part: String): String {

}
}
```

Swift Solution:

```
class Solution {
func removeOccurrences(_ s: String, _ part: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Remove All Occurrences of a Substring
// 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_occurrences(s: String, part: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} part
# @return {String}
def remove_occurrences(s, part)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $part
* @return String
*/
function removeOccurrences($s, $part) {

}
}
```

Dart Solution:

```
class Solution {
String removeOccurrences(String s, String part) {

}
}
```

Scala Solution:

```
object Solution {
def removeOccurrences(s: String, part: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec remove_occurrences(s :: String.t, part :: String.t) :: String.t
def remove_occurrences(s, part) do

end
end
```

Erlang Solution:

```
-spec remove_occurrences(S :: unicode:unicode_binary(), Part ::
unicode:unicode_binary()) -> unicode:unicode_binary().
remove_occurrences(S, Part) ->
```

```
.
```

Racket Solution:

```
(define/contract (remove-occurrences s part)
(-> string? string? string?)
)
```