Problem 3170: Lexicographically Minimum String After Removing Stars

Problem Information

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

Problem Description

You are given a string

s

. It may contain any number of

'*'

characters. Your task is to remove all

'*'

characters.

While there is a

'*'

, do the following operation:

Delete the leftmost

'*'

and the

smallest

non-

'*'

character to its

left

. If there are several smallest characters, you can delete any of them.

Return the

lexicographically smallest

resulting string after removing all

'*'

characters.

Example 1:

Input:

s = "aaba*"

Output:

"aab"

Explanation:

We should delete one of the

'a'

characters with

'*'

. If we choose

s[3]

,

s

becomes the lexicographically smallest.

Example 2:

Input:

s = "abc"

Output:

"abc"

Explanation:

There is no

'*'

in the string.

Constraints:

1 <= s.length <= 10

5

s

consists only of lowercase English letters and

'*'

.

The input is generated such that it is possible to delete all

'*'

characters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def clearStars(self, s):
```

```
"""
:type s: str
:rtype: str
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* clearStars(char* s) {

}
```

Go:

```
func clearStars(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (clear-stars s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Lexicographically Minimum String After Removing Stars
* Difficulty: Medium
* Tags: string, graph, greedy, hash, stack, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Lexicographically Minimum String After Removing Stars
* Difficulty: Medium
* Tags: string, graph, greedy, hash, stack, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String clearStars(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Lexicographically Minimum String After Removing Stars
Difficulty: Medium
Tags: string, graph, greedy, hash, stack, queue, heap
```

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

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Lexicographically Minimum String After Removing Stars
* Difficulty: Medium
* Tags: string, graph, greedy, hash, stack, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Lexicographically Minimum String After Removing Stars
* Difficulty: Medium
* Tags: string, graph, greedy, hash, stack, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function clearStars(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Lexicographically Minimum String After Removing Stars
* Difficulty: Medium
* Tags: string, graph, greedy, hash, stack, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
}
```

C Solution:

```
/*
* Problem: Lexicographically Minimum String After Removing Stars
* Difficulty: Medium
* Tags: string, graph, greedy, hash, stack, queue, heap
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

char* clearStars(char* s) {

}
```

Go Solution:

```
// Problem: Lexicographically Minimum String After Removing Stars
// Difficulty: Medium
// Tags: string, graph, greedy, hash, stack, queue, heap
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func clearStars(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Lexicographically Minimum String After Removing Stars
// Difficulty: Medium
// Tags: string, graph, greedy, hash, stack, queue, heap
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (clear-stars s)
(-> string? string?)
)
```