Problem 761: Special Binary String

Problem Information

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

Problem Description

Special binary strings

are binary strings with the following two properties:

The number of

0

's is equal to the number of

1

's.

Every prefix of the binary string has at least as many

1

's as

0

's.

You are given a

special binary

string

s

.

A move consists of choosing two consecutive, non-empty, special substrings of

s

, and swapping them. Two strings are consecutive if the last character of the first string is exactly one index before the first character of the second string.

Return

the lexicographically largest resulting string possible after applying the mentioned operations on the string

.

Example 1:

Input:

s = "11011000"

Output:

"11100100"

Explanation:

The strings "10" [occuring at s[1]] and "1100" [at s[3]] are swapped. This is the lexicographically largest string possible after some number of swaps.

Example 2:

Input:

s = "10"

Output:

"10"

Constraints:

1 <= s.length <= 50

s[i]

is either

'0'

or

'1'

.

s

is a special binary string.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public string MakeLargestSpecial(string s) {
}
}
```

C:

```
char* makeLargestSpecial(char* s) {

}
```

Go:

```
func makeLargestSpecial(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (make-largest-special s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Special Binary String
* Difficulty: Hard
* Tags: string, tree, graph
*
* 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 makeLargestSpecial(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Special Binary String
* Difficulty: Hard
* Tags: string, tree, graph
*
```

```
* 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 makeLargestSpecial(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Special Binary String
Difficulty: Hard
Tags: string, tree, graph

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 makeLargestSpecial(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Special Binary String
* Difficulty: Hard
* Tags: string, tree, graph
*
* 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
* @return {string}
*/
var makeLargestSpecial = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Special Binary String
* Difficulty: Hard
* Tags: string, tree, graph
*
* 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 makeLargestSpecial(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Special Binary String
* Difficulty: Hard
* Tags: string, tree, graph
*
* 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 MakeLargestSpecial(string s) {

}
}
```

C Solution:

```
/*
* Problem: Special Binary String
* Difficulty: Hard
* Tags: string, tree, graph
*
* 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* makeLargestSpecial(char* s) {

}
```

Go Solution:

```
// Problem: Special Binary String
// Difficulty: Hard
// Tags: string, tree, graph
//
// 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 makeLargestSpecial(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Special Binary String
// Difficulty: Hard
// Tags: string, tree, graph
//
// 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 make_largest_special(s: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (make-largest-special s)
(-> string? string?)
)
```