Problem 3088: Make String Anti-palindrome

Problem Information

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

Problem Description

We call a string

s

of

even

length

n

an

anti-palindrome

if for each index

0 <= i < n

,

s[i] != s[n - i - 1]

.

Given a string

s

, your task is to make

s

an

anti-palindrome

by doing

any

number of operations (including zero).

In one operation, you can select two characters from

s

and swap them.

Return

the resulting string. If multiple strings meet the conditions, return the

lexicographically smallest

one. If it can't be made into an anti-palindrome, return

"-1"

.

Example 1:

Input:

s = "abca"

Output:

"aabc"

Explanation:

"aabc"

is an anti-palindrome string since

s[0] != s[3]

and

s[1] != s[2]

. Also, it is a rearrangement of

"abca"

.

Example 2:

Input:

s = "abba"

Output:

"aabb"

Explanation:

"aabb"

is an anti-palindrome string since

s[0] != s[3]

and

s[1] != s[2]

. Also, it is a rearrangement of

"abba"

.

Example 3:

Input:

s = "cccd"

Output:

"-1"

Explanation:

You can see that no matter how you rearrange the characters of

"cccd"

, either

s[0] == s[3]

or

s[1] == s[2]

. So it can not form an anti-palindrome string.

Constraints:

2 <= s.length <= 10

5

s.length % 2 == 0

s

consists only of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* makeAntiPalindrome(char* s) {

}
```

Go:

```
func makeAntiPalindrome(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (make-anti-palindrome s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Make String Anti-palindrome
* Difficulty: Hard
* Tags: string, graph, greedy, sort
*
* 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 makeAntiPalindrome(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Make String Anti-palindrome
* Difficulty: Hard
* Tags: string, graph, greedy, sort
*
* 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 makeAntiPalindrome(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Make String Anti-palindrome
Difficulty: Hard
Tags: string, graph, greedy, sort
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Make String Anti-palindrome
* Difficulty: Hard
* Tags: string, graph, greedy, sort
*
* 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 makeAntiPalindrome = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Make String Anti-palindrome
* Difficulty: Hard
* Tags: string, graph, greedy, sort
*
* 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 makeAntiPalindrome(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Make String Anti-palindrome
* Difficulty: Hard
* Tags: string, graph, greedy, sort
*
* 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 MakeAntiPalindrome(string s) {

}
}
```

C Solution:

```
/*
* Problem: Make String Anti-palindrome
* Difficulty: Hard
* Tags: string, graph, greedy, sort
*
* 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* makeAntiPalindrome(char* s) {

}
```

Go Solution:

```
// Problem: Make String Anti-palindrome
// Difficulty: Hard
// Tags: string, graph, greedy, sort
//
// 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 makeAntiPalindrome(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Make String Anti-palindrome
// Difficulty: Hard
// Tags: string, graph, greedy, sort
```

```
//
// 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 make_anti_palindrome(s: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (make-anti-palindrome s)
(-> string? string?)
)
```