Problem 2697: Lexicographically Smallest Palindrome

Problem Information

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

Problem Description

You are given a string

s

consisting of

lowercase English letters

, and you are allowed to perform operations on it. In one operation, you can

replace

a character in

s

with another lowercase English letter.

Your task is to make

s

a

palindrome

with the

minimum

number

of operations

possible. If there are

multiple palindromes

that can be

made using the

minimum

number of operations,

make the

lexicographically smallest

one.

A string

a

is lexicographically smaller than a string

b

(of the same length) if in the first position where

a

and

b

differ, string

a

has a letter that appears earlier in the alphabet than the corresponding letter in

b

.

Return

the resulting palindrome string.

Example 1:

Input:

s = "egcfe"

Output:

"efcfe"

Explanation:

The minimum number of operations to make "egcfe" a palindrome is 1, and the lexicographically smallest palindrome string we can get by modifying one character is "efcfe", by changing 'g'.

Example 2:

Input:

s = "abcd"

Output:

"abba"

Explanation:

The minimum number of operations to make "abcd" a palindrome is 2, and the lexicographically smallest palindrome string we can get by modifying two characters is "abba".

Example 3:

Input:

s = "seven"

Output:

"neven"

Explanation:

The minimum number of operations to make "seven" a palindrome is 1, and the lexicographically smallest palindrome string we can get by modifying one character is "neven".

Constraints:

1 <= s.length <= 1000

s

consists of only lowercase English letters

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

}
}
```

C:

```
char* makeSmallestPalindrome(char* s) {

}
```

Go:

```
func makeSmallestPalindrome(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Lexicographically Smallest Palindrome
* Difficulty: Easy
* Tags: array, string, graph, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Lexicographically Smallest Palindrome
* Difficulty: Easy
* Tags: array, string, graph, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String makeSmallestPalindrome(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Lexicographically Smallest Palindrome
Difficulty: Easy
Tags: array, string, graph, greedy

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Lexicographically Smallest Palindrome
* Difficulty: Easy
* Tags: array, string, graph, greedy
*
* Approach: Use two pointers or sliding window technique
* 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 makeSmallestPalindrome = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Lexicographically Smallest Palindrome
* Difficulty: Easy
* Tags: array, string, graph, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function makeSmallestPalindrome(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Lexicographically Smallest Palindrome
* Difficulty: Easy
* Tags: array, string, graph, greedy
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

C Solution:

```
/*
* Problem: Lexicographically Smallest Palindrome
* Difficulty: Easy
* Tags: array, string, graph, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* makeSmallestPalindrome(char* s) {

}
```

Go Solution:

```
// Problem: Lexicographically Smallest Palindrome
// Difficulty: Easy
// Tags: array, string, graph, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func makeSmallestPalindrome(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Lexicographically Smallest Palindrome
// Difficulty: Easy
// Tags: array, string, graph, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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