Problem 2193: Minimum Number of Moves to Make Palindrome

Problem Information

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

Problem Description

You are given a string

s

consisting only of lowercase English letters.

In one

move

, you can select any two

adjacent

characters of

s

and swap them.

Return

the

minimum number of moves

needed to make

s

a palindrome

.

Note

that the input will be generated such that

s

can always be converted to a palindrome.

Example 1:

Input:

s = "aabb"

Output:

2

Explanation:

We can obtain two palindromes from s, "abba" and "baab". - We can obtain "abba" from s in 2 moves: "a

ab

b" -> "ab

ab

" -> "abba". - We can obtain "baab" from s in 2 moves: "a ab

b" -> "

ab

ab" -> "baab". Thus, the minimum number of moves needed to make s a palindrome is 2.

Example 2:

Input:

s = "letelt"

Output:

2

Explanation:

One of the palindromes we can obtain from s in 2 moves is "lettel". One of the ways we can obtain it is "lete

lt

" -> "let

et

l" -> "lettel". Other palindromes such as "tleelt" can also be obtained in 2 moves. It can be shown that it is not possible to obtain a palindrome in less than 2 moves.

Constraints:

1 <= s.length <= 2000

s

consists only of lowercase English letters.

```
s
```

can be converted to a palindrome using a finite number of moves.

Code Snippets

C++:

```
class Solution {
public:
int minMovesToMakePalindrome(string s) {

}
};
```

Java:

```
class Solution {
public int minMovesToMakePalindrome(String s) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minMovesToMakePalindrome(s: string): number {

};
```

C#:

```
public class Solution {
public int MinMovesToMakePalindrome(string s) {

}
}
```

C:

```
int minMovesToMakePalindrome(char* s) {

}
```

Go:

```
func minMovesToMakePalindrome(s string) int {

}
```

Kotlin:

```
class Solution {
fun minMovesToMakePalindrome(s: String): Int {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn min_moves_to_make_palindrome(s: String) -> i32 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int minMovesToMakePalindrome(String s) {

}
}
```

Scala:

```
object Solution {
def minMovesToMakePalindrome(s: String): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec min_moves_to_make_palindrome(S :: unicode:unicode_binary()) ->
integer().
min_moves_to_make_palindrome(S) ->
.
```

Racket:

```
(define/contract (min-moves-to-make-palindrome s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Moves to Make Palindrome
* Difficulty: Hard
* Tags: array, string, tree, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

class Solution {
public:
int minMovesToMakePalindrome(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Moves to Make Palindrome
* Difficulty: Hard
* Tags: array, string, tree, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int minMovesToMakePalindrome(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Moves to Make Palindrome
Difficulty: Hard
Tags: array, string, tree, greedy

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Moves to Make Palindrome
* Difficulty: Hard
* Tags: array, string, tree, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Moves to Make Palindrome
* Difficulty: Hard
* Tags: array, string, tree, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function minMovesToMakePalindrome(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Moves to Make Palindrome
* Difficulty: Hard
* Tags: array, string, tree, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int MinMovesToMakePalindrome(string s) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Moves to Make Palindrome
* Difficulty: Hard
* Tags: array, string, tree, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int minMovesToMakePalindrome(char* s) {

}
```

Go Solution:

```
// Problem: Minimum Number of Moves to Make Palindrome
// Difficulty: Hard
// Tags: array, string, tree, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func minMovesToMakePalindrome(s string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minMovesToMakePalindrome(s: String): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Moves to Make Palindrome
// Difficulty: Hard
// Tags: array, string, tree, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn min_moves_to_make_palindrome(s: String) -> i32 {
}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int minMovesToMakePalindrome(String s) {

}
}
```

Scala Solution:

```
object Solution {
def minMovesToMakePalindrome(s: String): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec min_moves_to_make_palindrome(S :: unicode:unicode_binary()) ->
integer().
min_moves_to_make_palindrome(S) ->
.
```

Racket Solution:

```
(define/contract (min-moves-to-make-palindrome s)
(-> string? exact-integer?)
)
```