Problem 3168: Minimum Number of Chairs in a Waiting Room

Problem Information

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

Problem Description

You are given a string

s

. Simulate events at each second

i

:

If

s[i] == 'E'

, a person enters the waiting room and takes one of the chairs in it.

If

s[i] == 'L'

, a person leaves the waiting room, freeing up a chair.

Return the

minimum

number of chairs needed so that a chair is available for every person who enters the waiting room given that it is initially

empty

.

Example 1:

Input:

s = "EEEEEEE"

Output:

7

Explanation:

After each second, a person enters the waiting room and no person leaves it. Therefore, a minimum of 7 chairs is needed.

Example 2:

Input:

s = "ELELEEL"

Output:

2

Explanation:

Let's consider that there are 2 chairs in the waiting room. The table below shows the state of the waiting room at each second.

Second

Event

People in the Waiting Room

Available Chairs

0

Enter

1

1

1

Leave

0

2

2

Enter

1

1

3

Leave

0

2

4

Enter

1

1

5

Enter

2

0

6

Leave

1

1

Example 3:

Input:

s = "ELEELEELLL"

Output:

3

Explanation:

Let's consider that there are 3 chairs in the waiting room. The table below shows the state of the waiting room at each second.

Second

Event

People in the Waiting Room

Available Chairs

0

Enter

1

2

1

Leave

0

3

2

Enter

1

2

3

Enter

2

1

4

Leave

1

2

5

Enter

2

1

6

Enter

3

0

7

Leave

2

1

8

Leave

1

2

9

Leave

0

3

Constraints:

1 <= s.length <= 50

s

consists only of the letters

'E'

and

'L'

.

s

represents a valid sequence of entries and exits.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

```
int minimumChairs(char* s) {

}
```

Go:

```
func minimumChairs(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-chairs s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Chairs in a Waiting Room
* Difficulty: Easy
* Tags: string
*
* 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:
int minimumChairs(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Chairs in a Waiting Room
* Difficulty: Easy
* Tags: string
*
* 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 int minimumChairs(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Chairs in a Waiting Room
Difficulty: Easy
Tags: string

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Chairs in a Waiting Room
* Difficulty: Easy
```

```
* Tags: string
*
* 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 {number}
*/
var minimumChairs = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Chairs in a Waiting Room
* Difficulty: Easy
* Tags: string
*
* 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 minimumChairs(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Chairs in a Waiting Room
* Difficulty: Easy
* Tags: string
*
* 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 int MinimumChairs(string s) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Chairs in a Waiting Room
* Difficulty: Easy
* Tags: string
*
* 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
*/

int minimumChairs(char* s) {

}
```

Go Solution:

```
// Problem: Minimum Number of Chairs in a Waiting Room
// Difficulty: Easy
// Tags: string
//
// 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 minimumChairs(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Chairs in a Waiting Room
// Difficulty: Easy
// Tags: string
//
// 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 minimum_chairs(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-chairs s)
(-> string? exact-integer?)
)
```