Problem 2086: Minimum Number of Food Buckets to Feed the Hamsters

Problem Information

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

Problem Description

You are given a

0-indexed

string

hamsters

where

hamsters[i]

is either:

'H'

indicating that there is a hamster at index

i

, or

'.'

indicating that index

i

is empty.

You will add some number of food buckets at the empty indices in order to feed the hamsters. A hamster can be fed if there is at least one food bucket to its left or to its right. More formally, a hamster at index

i

can be fed if you place a food bucket at index

i - 1

and/or

at index

i + 1

.

Return

the minimum number of food buckets you should

place at empty indices

to feed all the hamsters or

-1

if it is impossible to feed all of them

.

Example 1:

Input:

hamsters = "H..H"

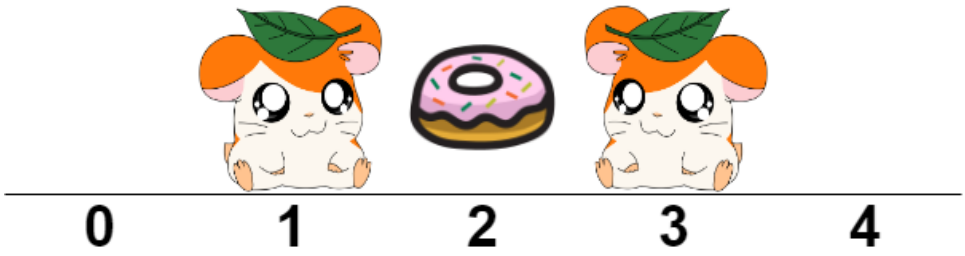
Output:

2

Explanation:

We place two food buckets at indices 1 and 2. It can be shown that if we place only one food bucket, one of the hamsters will not be fed.

Example 2:



Input:

hamsters = ".H.H."

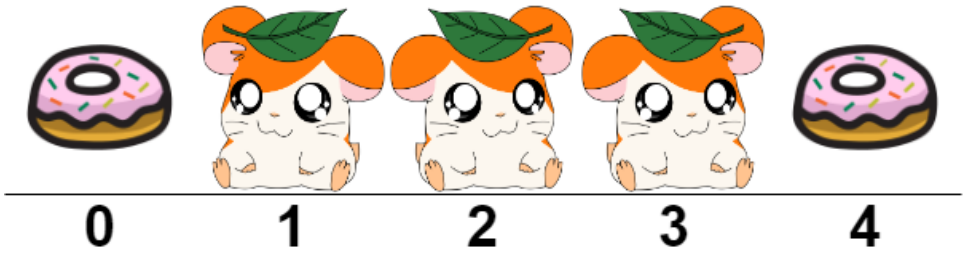
Output:

1

Explanation:

We place one food bucket at index 2.

Example 3:



Input:

hamsters = ".HHH."

Output:

-1

Explanation:

If we place a food bucket at every empty index as shown, the hamster at index 2 will not be able to eat.

Constraints:

1 <= hamsters.length <= 10

5

hamsters[i]

is either

'H'

or

'.'

.

Code Snippets

C++:

```
class Solution {
public:
int minimumBuckets(string hamsters) {

}
};
```

Java:

```
class Solution {
public int minimumBuckets(String hamsters) {

}
}
```

Python3:

```
class Solution:
def minimumBuckets(self, hamsters: str) -> int:
```

Python:

```
class Solution(object):
def minimumBuckets(self, hamsters):
"""
:type hamsters: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} hamsters
* @return {number}
*/
```

```
var minimumBuckets = function(hamsters) {

};
```

TypeScript:

```
function minimumBuckets(hamsters: string): number {

};
```

C#:

```
public class Solution {
public int MinimumBuckets(string hamsters) {

}
}
```

C:

```
int minimumBuckets(char* hamsters) {

}
```

Go:

```
func minimumBuckets(hamsters string) int {

}
```

Kotlin:

```
class Solution {
fun minimumBuckets(hamsters: String): Int {

}
}
```

Swift:

```
class Solution {
func minimumBuckets(_ hamsters: String) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn minimum_buckets(hamsters: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} hamsters
# @return {Integer}
def minimum_buckets(hamsters)

end
```

PHP:

```
class Solution {

/**
* @param String $hamsters
* @return Integer
*/
function minimumBuckets($hamsters) {

}
}
```

Dart:

```
class Solution {
int minimumBuckets(String hamsters) {

}
}
```

Scala:

```
object Solution {
def minimumBuckets(hamsters: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_buckets(hamsters :: String.t) :: integer
def minimum_buckets(hamsters) do

end
end
```

Erlang:

```
-spec minimum_buckets(Hamsters :: unicode:unicode_binary()) -> integer().
minimum_buckets(Hamsters) ->
.
```

Racket:

```
(define/contract (minimum-buckets hamsters)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Food Buckets to Feed the Hamsters
* Difficulty: Medium
* Tags: string, dp, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
class Solution {
public:
int minimumBuckets(string hamsters) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Food Buckets to Feed the Hamsters
* Difficulty: Medium
* Tags: string, dp, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minimumBuckets(String hamsters) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Food Buckets to Feed the Hamsters
Difficulty: Medium
Tags: string, dp, greedy

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minimumBuckets(self, hamsters: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumBuckets(self, hamsters):
"""
:type hamsters: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of Food Buckets to Feed the Hamsters
* Difficulty: Medium
* Tags: string, dp, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} hamsters
* @return {number}
*/
var minimumBuckets = function(hamsters) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Food Buckets to Feed the Hamsters
* Difficulty: Medium
* Tags: string, dp, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimumBuckets(hamsters: string): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Minimum Number of Food Buckets to Feed the Hamsters
* Difficulty: Medium
* Tags: string, dp, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MinimumBuckets(string hamsters) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Food Buckets to Feed the Hamsters
* Difficulty: Medium
* Tags: string, dp, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minimumBuckets(char* hamsters) {

}
```

Go Solution:

```
// Problem: Minimum Number of Food Buckets to Feed the Hamsters
// Difficulty: Medium
```

```
// Tags: string, dp, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimumBuckets(hamsters string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumBuckets(hamsters: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumBuckets(_ hamsters: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Food Buckets to Feed the Hamsters
// Difficulty: Medium
// Tags: string, dp, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn minimum_buckets(hamsters: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} hamsters
# @return {Integer}
def minimum_buckets(hamsters)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $hamsters
* @return Integer
*/
function minimumBuckets($hamsters) {

}
}
```

Dart Solution:

```
class Solution {
int minimumBuckets(String hamsters) {

}
}
```

Scala Solution:

```
object Solution {
def minimumBuckets(hamsters: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_buckets(hamsters :: String.t) :: integer
def minimum_buckets(hamsters) do
```

```
end
end
```

Erlang Solution:

```
-spec minimum_buckets(Hamsters :: unicode:unicode_binary()) -> integer().
minimum_buckets(Hamsters) ->
.
```

Racket Solution:

```
(define/contract (minimum-buckets hamsters)
(-> string? exact-integer?)
)
```