Problem 1769: Minimum Number of Operations to Move All Balls to Each Box

Problem Information

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

Problem Description

You have

n

boxes. You are given a binary string

boxes

of length

n

, where

boxes[i]

is

'0'

if the

i

th

box is

empty

, and

'1'

if it contains

one

ball.

In one operation, you can move

one

ball from a box to an adjacent box. Box

i

is adjacent to box

j

if

abs(i - j) == 1

. Note that after doing so, there may be more than one ball in some boxes.

Return an array

answer

of size

n

, where

answer[i]

is the

minimum

number of operations needed to move all the balls to the

i

th

box.

Each

answer[i]

is calculated considering the

initial

state of the boxes.

Example 1:

Input:

boxes = "110"

Output:

[1,1,3]

Explanation:

The answer for each box is as follows: 1) First box: you will have to move one ball from the second box to the first box in one operation. 2) Second box: you will have to move one ball from the first box to the second box in one operation. 3) Third box: you will have to move one ball from the first box to the third box in two operations, and move one ball from the second box to the third box in one operation.

Example 2:

Input:

boxes = "001011"

Output:

[11,8,5,4,3,4]

Constraints:

n == boxes.length

1 <= n <= 2000

boxes[i]

is either

'0'

or

'1'

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> minOperations(string boxes) {

}
};
```

Java:

```
class Solution {
public int[] minOperations(String boxes) {

}
}
```

Python3:

```
class Solution:
def minOperations(self, boxes: str) -> List[int]:
```

Python:

```
class Solution(object):
def minOperations(self, boxes):
"""
:type boxes: str
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {string} boxes
* @return {number[]}
*/
var minOperations = function(boxes) {

};
```

TypeScript:

```
function minOperations(boxes: string): number[] {
};
```

C#:

```
public class Solution {
public int[] MinOperations(string boxes) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* minOperations(char* boxes, int* returnSize) {

}
```

Go:

```
func minOperations(boxes string) []int {

}
```

Kotlin:

```
class Solution {
fun minOperations(boxes: String): IntArray {

}
}
```

Swift:

```
class Solution {
func minOperations(_ boxes: String) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn min_operations(boxes: String) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {String} boxes
# @return {Integer[]}
def min_operations(boxes)

end
```

PHP:

```
class Solution {

/**
* @param String $boxes
* @return Integer[]
*/
function minOperations($boxes) {

}
}
```

Dart:

```
class Solution {
List<int> minOperations(String boxes) {

}
}
```

Scala:

```
object Solution {
def minOperations(boxes: String): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations(boxes :: String.t) :: [integer]
def min_operations(boxes) do

end
end
```

Erlang:

```
-spec min_operations(Boxes :: unicode:unicode_binary()) -> [integer()].
min_operations(Boxes) ->
.
```

Racket:

```
(define/contract (min-operations boxes)
(-> string? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Operations to Move All Balls to Each Box
* Difficulty: Medium
* Tags: array, string
*
* 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:
vector<int> minOperations(string boxes) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Operations to Move All Balls to Each Box
* Difficulty: Medium
* Tags: array, string
*
* 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 int[] minOperations(String boxes) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Operations to Move All Balls to Each Box
Difficulty: Medium
Tags: array, string

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 minOperations(self, boxes: str) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minOperations(self, boxes):
"""
:type boxes: str
:rtype: List[int]
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of Operations to Move All Balls to Each Box
* Difficulty: Medium
* Tags: array, string
*
* 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} boxes
* @return {number[]}
*/
var minOperations = function(boxes) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Operations to Move All Balls to Each Box
* Difficulty: Medium
* Tags: array, string
*
* 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 minOperations(boxes: string): number[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Operations to Move All Balls to Each Box
* Difficulty: Medium
* Tags: array, string
*
* 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 int[] MinOperations(string boxes) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Operations to Move All Balls to Each Box
* Difficulty: Medium
* Tags: array, string
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* minOperations(char* boxes, int* returnSize) {

}
```

Go Solution:

```
// Problem: Minimum Number of Operations to Move All Balls to Each Box
// Difficulty: Medium
// Tags: array, string
//
// 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 minOperations(boxes string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun minOperations(boxes: String): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func minOperations(_ boxes: String) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Operations to Move All Balls to Each Box
// Difficulty: Medium
// Tags: array, string
//
// 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 min_operations(boxes: String) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String} boxes
# @return {Integer[]}
def min_operations(boxes)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $boxes
* @return Integer[]
*/
function minOperations($boxes) {

}
}
```

Dart Solution:

```
class Solution {
List<int> minOperations(String boxes) {

}
}
```

Scala Solution:

```
object Solution {
def minOperations(boxes: String): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations(boxes :: String.t) :: [integer]
def min_operations(boxes) do

end
```

```
end
```

Erlang Solution:

```
-spec min_operations(Boxes :: unicode:unicode_binary()) -> [integer()].
min_operations(Boxes) ->
  .
```

Racket Solution:

```
(define/contract (min-operations boxes)
  (-> string? (listof exact-integer?))
  )
```