Problem 2483: Minimum Penalty for a Shop

Problem Information

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

Problem Description

You are given the customer visit log of a shop represented by a

0-indexed

string

customers

consisting only of characters

'N'

and

'Y'

:

if the

i

th

character is

'Y'

, it means that customers come at the

i

th

hour

whereas

'N'

indicates that no customers come at the

i

th

hour.

If the shop closes at the

j

th

hour (

0 <= j <= n

), the

penalty

is calculated as follows:

For every hour when the shop is open and no customers come, the penalty increases by

1

.

For every hour when the shop is closed and customers come, the penalty increases by

1

.

Return

the

earliest

hour at which the shop must be closed to incur a

minimum

penalty.

Note

that if a shop closes at the

j

th

hour, it means the shop is closed at the hour

j

.

Example 1:

Input:

customers = "YYNY"

Output:

2

Explanation:

- Closing the shop at the 0

th

hour incurs in 1+1+0+1 = 3 penalty. - Closing the shop at the 1

st

hour incurs in 0+1+0+1 = 2 penalty. - Closing the shop at the 2

nd

hour incurs in 0+0+0+1 = 1 penalty. - Closing the shop at the 3

rd

hour incurs in 0+0+1+1 = 2 penalty. - Closing the shop at the 4

th

hour incurs in 0+0+1+0 = 1 penalty. Closing the shop at 2

nd

or 4

th

hour gives a minimum penalty. Since 2 is earlier, the optimal closing time is 2.

Example 2:

Input:

customers = "NNNNN"

Output:

0

Explanation:

It is best to close the shop at the 0

th

hour as no customers arrive.

Example 3:

Input:

customers = "YYYY"

Output:

4

Explanation:

It is best to close the shop at the 4

th

hour as customers arrive at each hour.

Constraints:

1 <= customers.length <= 10

5

customers

consists only of characters

'Y'

and

'N'

.

Code Snippets

C++:

```
class Solution {
public:
int bestClosingTime(string customers) {

}
};
```

Java:

```
class Solution {
public int bestClosingTime(String customers) {

}
}
```

Python3:

```
class Solution:
def bestClosingTime(self, customers: str) -> int:
```

Python:

```
class Solution(object):
def bestClosingTime(self, customers):
"""
:type customers: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} customers
* @return {number}
*/
var bestClosingTime = function(customers) {

};
```

TypeScript:

```
function bestClosingTime(customers: string): number {

};
```

C#:

```
public class Solution {
public int BestClosingTime(string customers) {

}
}
```

C:

```
int bestClosingTime(char* customers) {

}
```

Go:

```
func bestClosingTime(customers string) int {
```

```
}
```

Kotlin:

```
class Solution {
fun bestClosingTime(customers: String): Int {

}
}
```

Swift:

```
class Solution {
func bestClosingTime(_ customers: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn best_closing_time(customers: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} customers
# @return {Integer}
def best_closing_time(customers)

end
```

PHP:

```
class Solution {

/**
* @param String $customers
* @return Integer
*/
```

```
function bestClosingTime($customers) {

}
}
```

Dart:

```
class Solution {
int bestClosingTime(String customers) {

}
}
```

Scala:

```
object Solution {
def bestClosingTime(customers: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec best_closing_time(customers :: String.t) :: integer
def best_closing_time(customers) do

end
end
```

Erlang:

```
-spec best_closing_time(Customers :: unicode:unicode_binary()) -> integer().
best_closing_time(Customers) ->
.
```

Racket:

```
(define/contract (best-closing-time customers)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Penalty for a Shop
* 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 bestClosingTime(string customers) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Penalty for a Shop
* 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 bestClosingTime(String customers) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Penalty for a Shop
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 bestClosingTime(self, customers: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def bestClosingTime(self, customers):
"""
:type customers: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Penalty for a Shop
* 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} customers
* @return {number}
*/
var bestClosingTime = function(customers) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Minimum Penalty for a Shop
* 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 bestClosingTime(customers: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Penalty for a Shop
* 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 BestClosingTime(string customers) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Penalty for a Shop
* 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
*/

int bestClosingTime(char* customers) {

}
```

Go Solution:

```
// Problem: Minimum Penalty for a Shop
// 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 bestClosingTime(customers string) int {

}
```

Kotlin Solution:

```
class Solution {
fun bestClosingTime(customers: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func bestClosingTime(_ customers: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Penalty for a Shop
// 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 best_closing_time(customers: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} customers
# @return {Integer}
def best_closing_time(customers)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $customers
* @return Integer
*/
function bestClosingTime($customers) {

}
}
```

Dart Solution:

```
class Solution {
int bestClosingTime(String customers) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def bestClosingTime(customers: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec best_closing_time(customers :: String.t) :: integer
def best_closing_time(customers) do

end
end
```

Erlang Solution:

```
-spec best_closing_time(Customers :: unicode:unicode_binary()) -> integer().
best_closing_time(Customers) ->
.
```

Racket Solution:

```
(define/contract (best-closing-time customers)
(-> string? exact-integer?)
)
```