Problem 1598: Crawler Log Folder

Problem Information

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

Problem Description

The Leetcode file system keeps a log each time some user performs a

change folder

operation.

The operations are described below:

"../"

: Move to the parent folder of the current folder. (If you are already in the main folder,

remain in the same folder

).

"./"

: Remain in the same folder.

"x/"

: Move to the child folder named

x

(This folder is

guaranteed to always exist

).

You are given a list of strings

logs

where

logs[i]

is the operation performed by the user at the

i

th

step.

The file system starts in the main folder, then the operations in

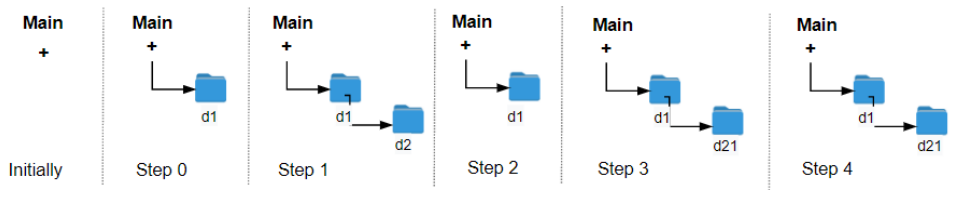
logs

are performed.

Return

the minimum number of operations needed to go back to the main folder after the change folder operations.

Example 1:



Input:

logs = ["d1/","d2/","../","d21/","./"]

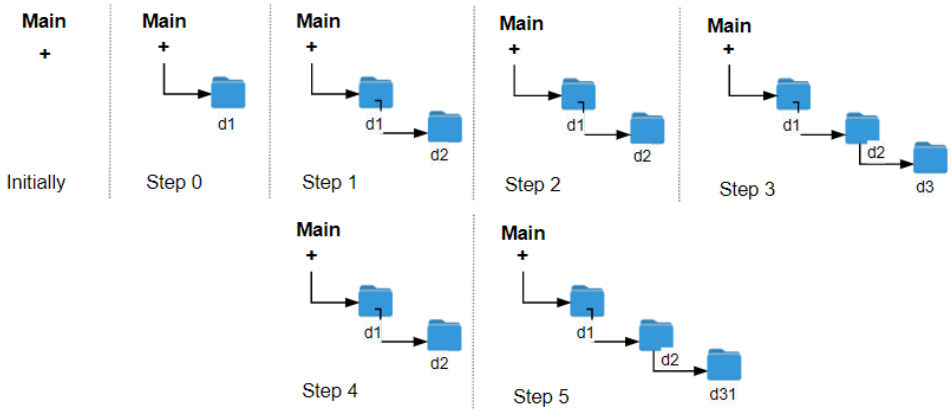
Output:

2

Explanation:

Use this change folder operation "../" 2 times and go back to the main folder.

Example 2:



Input:

logs = ["d1/","d2/","./","d3/","../","d31/"]

Output:

3

Example 3:

Input:

logs = ["d1/","../","../","../"]

Output:

0

Constraints:

1 <= logs.length <= 10

3

2 <= logs[i].length <= 10

logs[i]

contains lowercase English letters, digits,

'.'

, and

'/'

.

logs[i]

follows the format described in the statement.

Folder names consist of lowercase English letters and digits.

Code Snippets

C++:

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

```
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minOperations(logs: string[]): number {

};
```

C#:

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

}
}
```

C:

```
int minOperations(char** logs, int logsSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun minOperations(logs: Array<String>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Crawler Log Folder
* Difficulty: Easy
* Tags: array, string, stack
*
* 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(vector<string>& logs) {

}
};
```

Java Solution:

```
/**
* Problem: Crawler Log Folder
* Difficulty: Easy
* Tags: array, string, stack
*
* 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[] logs) {

}
}
```

Python3 Solution:

```
"""
Problem: Crawler Log Folder
Difficulty: Easy
Tags: array, string, stack

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, logs: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Crawler Log Folder
* Difficulty: Easy
```

```
* Tags: array, string, stack
*
* 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[]} logs
* @return {number}
*/
var minOperations = function(logs) {

};
```

TypeScript Solution:

```
/**
* Problem: Crawler Log Folder
* Difficulty: Easy
* Tags: array, string, stack
*
* 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(logs: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Crawler Log Folder
* Difficulty: Easy
* Tags: array, string, stack
*
* 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[] logs) {

}
}
```

C Solution:

```
/*
* Problem: Crawler Log Folder
* Difficulty: Easy
* Tags: array, string, stack
*
* 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 minOperations(char** logs, int logsSize) {

}
```

Go Solution:

```
// Problem: Crawler Log Folder
// Difficulty: Easy
// Tags: array, string, stack
//
// 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(logs []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minOperations(logs: Array<String>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Crawler Log Folder
// Difficulty: Easy
// Tags: array, string, stack
//
// 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(logs: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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