Problem 3019: Number of Changing Keys

Problem Information

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

Problem Description

You are given a

0-indexed

string

s

typed by a user. Changing a key is defined as using a key different from the last used key. For example,

s = "ab"

has a change of a key while

s = "bBBb"

does not have any.

Return

the number of times the user had to change the key.

Note:

Modifiers like

shift

or

caps lock

won't be counted in changing the key that is if a user typed the letter

'a'

and then the letter

'A'

then it will not be considered as a changing of key.

Example 1:

Input:

s = "aAbBcC"

Output:

2

Explanation:

From s[0] = 'a' to s[1] = 'A', there is no change of key as caps lock or shift is not counted. From s[1] = 'A' to s[2] = 'b', there is a change of key. From s[2] = 'b' to s[3] = 'B', there is no change of key as caps lock or shift is not counted. From s[3] = 'B' to s[4] = 'c', there is a change of key. From s[4] = 'c' to s[5] = 'C', there is no change of key as caps lock or shift is not counted.

Example 2:

Input:

s = "AaAaAaaA"

Output:

0

Explanation:

There is no change of key since only the letters 'a' and 'A' are

pressed which does not require change of key.

Constraints:

1 <= s.length <= 100

s

consists of only upper case and lower case English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int countKeyChanges(char* s) {

}
```

Go:

```
func countKeyChanges(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-key-changes s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Changing Keys
* 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 countKeyChanges(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Changing Keys
* 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 countKeyChanges(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Changing Keys
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 countKeyChanges(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Changing Keys
* 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 countKeyChanges = function(s) {
};
```

TypeScript Solution:

```
/**
* Problem: Number of Changing Keys
* 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 countKeyChanges(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Number of Changing Keys
* 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 CountKeyChanges(string s) {

}
}
```

C Solution:

```
/*
* Problem: Number of Changing Keys
* 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 countKeyChanges(char* s) {

}
```

Go Solution:

```
// Problem: Number of Changing Keys
// 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 countKeyChanges(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Changing Keys
// 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 count_key_changes(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-key-changes s)
(-> string? exact-integer?)
)
```