Problem 420: Strong Password Checker

Problem Information

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

Problem Description

A password is considered strong if the below conditions are all met:

It has at least

6

characters and at most

20

characters.

It contains at least

one lowercase

letter, at least

one uppercase

letter, and at least

one digit

.

It does not contain three repeating characters in a row (i.e.,

"B

aaa

bb0"

is weak, but

"B

aa

b

a

0"

is strong).

Given a string

password

, return

the minimum number of steps required to make

password

strong. if

password

is already strong, return

0

.

In one step, you can:

Insert one character to

password

,

Delete one character from

password

, or

Replace one character of

password

with another character.

Example 1:

Input:

password = "a"

Output:

5

Example 2:

Input:

password = "aA1"

Output:

3

Example 3:

Input:

password = "1337C0d3"

Output:

0

Constraints:

1 <= password.length <= 50

password

consists of letters, digits, dot

'.'

or exclamation mark

'!'

.

Code Snippets

C++:

```
class Solution {
public:
int strongPasswordChecker(string password) {
```

```
}
};
```

Java:

```
class Solution {
public int strongPasswordChecker(String password) {

}
}
```

Python3:

```
class Solution:
def strongPasswordChecker(self, password: str) -> int:
```

Python:

```
class Solution(object):
def strongPasswordChecker(self, password):
"""
:type password: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} password
* @return {number}
*/
var strongPasswordChecker = function(password) {

};
```

TypeScript:

```
function strongPasswordChecker(password: string): number {

};
```

C#:

```
public class Solution {
public int StrongPasswordChecker(string password) {

}
}
```

C:

```
int strongPasswordChecker(char* password) {

}
```

Go:

```
func strongPasswordChecker(password string) int {

}
```

Kotlin:

```
class Solution {
fun strongPasswordChecker(password: String): Int {

}
}
```

Swift:

```
class Solution {
func strongPasswordChecker(_ password: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn strong_password_checker(password: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} password
# @return {Integer}
def strong_password_checker(password)

end
```

PHP:

```
class Solution {

/**
* @param String $password
* @return Integer
*/
function strongPasswordChecker($password) {

}
}
```

Dart:

```
class Solution {
int strongPasswordChecker(String password) {

}
}
```

Scala:

```
object Solution {
def strongPasswordChecker(password: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec strong_password_checker(password :: String.t) :: integer
def strong_password_checker(password) do

end
end
```

Erlang:

```
-spec strong_password_checker(Password :: unicode:unicode_binary()) ->
integer().
strong_password_checker(Password) ->
.
```

Racket:

```
(define/contract (strong-password-checker password)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Strong Password Checker
* Difficulty: Hard
* Tags: string, greedy, queue, heap
*
* 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 strongPasswordChecker(string password) {

}
};
```

Java Solution:

```
/**
* Problem: Strong Password Checker
* Difficulty: Hard
* Tags: string, greedy, queue, heap
*
```

```
* 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 strongPasswordChecker(String password) {

}
}
```

Python3 Solution:

```
"""
Problem: Strong Password Checker
Difficulty: Hard
Tags: string, greedy, queue, heap

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 strongPasswordChecker(self, password: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def strongPasswordChecker(self, password):
"""
:type password: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Strong Password Checker
```

```
* Difficulty: Hard
* Tags: string, greedy, queue, heap
*
* 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} password
* @return {number}
*/
var strongPasswordChecker = function(password) {

};
```

TypeScript Solution:

```
/**
* Problem: Strong Password Checker
* Difficulty: Hard
* Tags: string, greedy, queue, heap
*
* 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 strongPasswordChecker(password: string): number {

};
```

C# Solution:

```
/*
* Problem: Strong Password Checker
* Difficulty: Hard
* Tags: string, greedy, queue, heap
*
* 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 StrongPasswordChecker(string password) {

}
}
```

C Solution:

```
/*
* Problem: Strong Password Checker
* Difficulty: Hard
* Tags: string, greedy, queue, heap
*
* 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 strongPasswordChecker(char* password) {

}
```

Go Solution:

```
// Problem: Strong Password Checker
// Difficulty: Hard
// Tags: string, greedy, queue, heap
//
// 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 strongPasswordChecker(password string) int {

}
```

Kotlin Solution:

```
class Solution {
fun strongPasswordChecker(password: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func strongPasswordChecker(_ password: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Strong Password Checker
// Difficulty: Hard
// Tags: string, greedy, queue, heap
//
// 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 strong_password_checker(password: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} password
# @return {Integer}
def strong_password_checker(password)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param String $password
* @return Integer
*/
function strongPasswordChecker($password) {

}
}
```

Dart Solution:

```
class Solution {
int strongPasswordChecker(String password) {

}
}
```

Scala Solution:

```
object Solution {
def strongPasswordChecker(password: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec strong_password_checker(password :: String.t) :: integer
def strong_password_checker(password) do

end
end
```

Erlang Solution:

```
-spec strong_password_checker(Password :: unicode:unicode_binary()) ->
integer().
strong_password_checker(Password) ->
.
```

Racket Solution:

```
(define/contract (strong-password-checker password)
(-> string? exact-integer?)
)
```