Problem 2299: Strong Password Checker II

Problem Information

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

Problem Description

A password is said to be

strong

if it satisfies all the following criteria:

It has at least

8

characters.

It contains at least

one lowercase

letter.

It contains at least

one uppercase

letter.

It contains at least

one digit

.

It contains at least

one special character

. The special characters are the characters in the following string:

"!@#$%^&*()-+"

.

It does

not

contain

2

of the same character in adjacent positions (i.e.,

"aab"

violates this condition, but

"aba"

does not).

Given a string

password

, return

true

if it is a

strong

password

. Otherwise, return

false

.

Example 1:

Input:

password = "IloveLe3tcode!"

Output:

true

Explanation:

The password meets all the requirements. Therefore, we return true.

Example 2:

Input:

password = "Me+You--IsMyDream"

Output:

false

Explanation:

The password does not contain a digit and also contains 2 of the same character in adjacent positions. Therefore, we return false.

Example 3:

Input:

password = "1aB!"

Output:

false

Explanation:

The password does not meet the length requirement. Therefore, we return false.

Constraints:

1 <= password.length <= 100

password

consists of letters, digits, and special characters:

"!@#$%^&*()-+"

.

Code Snippets

C++:

```
class Solution {
public:
bool strongPasswordCheckerII(string password) {

}
};
```

Java:

```
class Solution {
public boolean strongPasswordCheckerII(String password) {

}
}
```

Python3:

```
class Solution:
def strongPasswordCheckerII(self, password: str) -> bool:
```

Python:

```
class Solution(object):
def strongPasswordCheckerII(self, password):
"""
:type password: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} password
* @return {boolean}
*/
var strongPasswordCheckerII = function(password) {

};
```

TypeScript:

```
function strongPasswordCheckerII(password: string): boolean {

};
```

C#:

```
public class Solution {
public bool StrongPasswordCheckerII(string password) {
```

```
}
}
```

C:

```
bool strongPasswordCheckerII(char* password) {

}
```

Go:

```
func strongPasswordCheckerII(password string) bool {

}
```

Kotlin:

```
class Solution {
fun strongPasswordCheckerII(password: String): Boolean {

}
}
```

Swift:

```
class Solution {
func strongPasswordCheckerII(_ password: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn strong_password_checker_ii(password: String) -> bool {

}
}
```

Ruby:

```
# @param {String} password
# @return {Boolean}
def strong_password_checker_ii(password)

end
```

PHP:

```
class Solution {

/**
* @param String $password
* @return Boolean
*/
function strongPasswordCheckerII($password) {

}
}
```

Dart:

```
class Solution {
bool strongPasswordCheckerII(String password) {

}
}
```

Scala:

```
object Solution {
def strongPasswordCheckerII(password: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec strong_password_checker_ii(password :: String.t) :: boolean
def strong_password_checker_ii(password) do

end
end
```

Erlang:

```
-spec strong_password_checker_ii(Password :: unicode:unicode_binary()) ->
boolean().
strong_password_checker_ii(Password) ->
.
```

Racket:

```
(define/contract (strong-password-checker-ii password)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Strong Password Checker II
* 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:
bool strongPasswordCheckerII(string password) {

}
};
```

Java Solution:

```
/**
* Problem: Strong Password Checker II
* 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 boolean strongPasswordCheckerII(String password) {

}
}
```

Python3 Solution:

```
"""
Problem: Strong Password Checker II
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 strongPasswordCheckerII(self, password: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def strongPasswordCheckerII(self, password):
"""
:type password: str
:rtype: bool
"""
```

JavaScript Solution:

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

```
* 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} password
* @return {boolean}
*/
var strongPasswordCheckerII = function(password) {

};
```

TypeScript Solution:

```
/**
* Problem: Strong Password Checker II
* 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 strongPasswordCheckerII(password: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Strong Password Checker II
* 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 bool StrongPasswordCheckerII(string password) {

}
}
```

C Solution:

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

bool strongPasswordCheckerII(char* password) {

}
```

Go Solution:

```
// Problem: Strong Password Checker II
// 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 strongPasswordCheckerII(password string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun strongPasswordCheckerII(password: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func strongPasswordCheckerII(_ password: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Strong Password Checker II
// 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 strong_password_checker_ii(password: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} password
# @return {Boolean}
def strong_password_checker_ii(password)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param String $password
* @return Boolean
*/
function strongPasswordCheckerII($password) {

}
}
```

Dart Solution:

```
class Solution {
bool strongPasswordCheckerII(String password) {

}
}
```

Scala Solution:

```
object Solution {
def strongPasswordCheckerII(password: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec strong_password_checker_ii(password :: String.t) :: boolean
def strong_password_checker_ii(password) do

end
end
```

Erlang Solution:

```
-spec strong_password_checker_ii(Password :: unicode:unicode_binary()) ->
boolean().
strong_password_checker_ii(Password) ->
.
```

Racket Solution:

```
(define/contract (strong-password-checker-ii password)
(-> string? boolean?)
)
```