Problem 3438: Find Valid Pair of Adjacent Digits in String

Problem Information

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

Problem Description

You are given a string

s

consisting only of digits. A

valid pair

is defined as two

adjacent

digits in

s

such that:

The first digit is

not equal

to the second.

Each digit in the pair appears in

s

exactly

as many times as its numeric value.

Return the first

valid pair

found in the string

s

when traversing from left to right. If no valid pair exists, return an empty string.

Example 1:

Input:

s = "2523533"

Output:

"23"

Explanation:

Digit

'2'

appears 2 times and digit

'3'

appears 3 times. Each digit in the pair

"23"

appears in

s

exactly as many times as its numeric value. Hence, the output is

"23"

.

Example 2:

Input:

s = "221"

Output:

"21"

Explanation:

Digit

'2'

appears 2 times and digit

'1'

appears 1 time. Hence, the output is

"21"

.

Example 3:

Input:

s = "22"

Output:

""

Explanation:

There are no valid adjacent pairs.

Constraints:

2 <= s.length <= 100

s

only consists of digits from

'1'

to

'9'

.

Code Snippets

C++:

```
class Solution {
public:
string findValidPair(string s) {

}
};
```

Java:

```
class Solution {
public String findValidPair(String s) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findValidPair(s: string): string {

};
```

C#:

```
public class Solution {
public string FindValidPair(string s) {
```

```
}
}
```

C:

```
char* findValidPair(char* s) {

}
```

Go:

```
func findValidPair(s string) string {

}
```

Kotlin:

```
class Solution {
fun findValidPair(s: String): String {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
String findValidPair(String s) {

}
}
```

Scala:

```
object Solution {
def findValidPair(s: String): String = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec find_valid_pair(S :: unicode:unicode_binary()) ->
unicode:unicode_binary().
find_valid_pair(S) ->
.
```

Racket:

```
(define/contract (find-valid-pair s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Valid Pair of Adjacent Digits in String
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
string findValidPair(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Find Valid Pair of Adjacent Digits in String
* Difficulty: Easy
* Tags: string, hash
*
```

```
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String findValidPair(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Valid Pair of Adjacent Digits in String
Difficulty: Easy
Tags: string, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def findValidPair(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Valid Pair of Adjacent Digits in String
```

```
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} s
* @return {string}
*/
var findValidPair = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Valid Pair of Adjacent Digits in String
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function findValidPair(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Find Valid Pair of Adjacent Digits in String
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string FindValidPair(string s) {

}
}
```

C Solution:

```
/*
* Problem: Find Valid Pair of Adjacent Digits in String
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* findValidPair(char* s) {

}
```

Go Solution:

```
// Problem: Find Valid Pair of Adjacent Digits in String
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func findValidPair(s string) string {

}
```

Kotlin Solution:

```
class Solution {
fun findValidPair(s: String): String {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find Valid Pair of Adjacent Digits in String
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn find_valid_pair(s: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param String $s
* @return String
*/
function findValidPair($s) {

}
}
```

Dart Solution:

```
class Solution {
String findValidPair(String s) {

}
}
```

Scala Solution:

```
object Solution {
def findValidPair(s: String): String = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec find_valid_pair(S :: unicode:unicode_binary()) ->
unicode:unicode_binary().
find_valid_pair(S) ->
.
```

Racket Solution:

```
(define/contract (find-valid-pair s)
(-> string? string?)
)
```