Problem 1417: Reformat The String

Problem Information

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

Problem Description

You are given an alphanumeric string

s

. (

Alphanumeric string

is a string consisting of lowercase English letters and digits).

You have to find a permutation of the string where no letter is followed by another letter and no digit is followed by another digit. That is, no two adjacent characters have the same type.

Return

the reformatted string

or return

an empty string

if it is impossible to reformat the string.

Example 1:

Input:

s = "a0b1c2"

Output:

"0a1b2c"

Explanation:

No two adjacent characters have the same type in "0a1b2c". "a0b1c2", "0a1b2c", "0c2a1b" are also valid permutations.

Example 2:

Input:

s = "leetcode"

Output:

""

Explanation:

"leetcode" has only characters so we cannot separate them by digits.

Example 3:

Input:

s = "1229857369"

Output:

""

Explanation:

"1229857369" has only digits so we cannot separate them by characters.

Constraints:

1 <= s.length <= 500

s

consists of only lowercase English letters and/or digits.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* reformat(char* s) {

}
```

Go:

```
func reformat(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (reformat s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reformat The String
* 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:
string reformat(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Reformat The String
* 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 String reformat(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Reformat The String
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 reformat(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Reformat The String
* 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 {string}
*/
var reformat = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Reformat The String
* 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 reformat(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Reformat The String
* 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 string Reformat(string s) {

}
}
```

C Solution:

```
/*
* Problem: Reformat The String
* 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
*/

char* reformat(char* s) {
```

```
}
```

Go Solution:

```
// Problem: Reformat The String
// 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 reformat(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Reformat The String
// 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 reformat(s: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (reformat s)
(-> string? string?)
)
```