Problem 767: Reorganize String

Problem Information

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

Problem Description

Given a string

s

, rearrange the characters of

s

so that any two adjacent characters are not the same.

Return

any possible rearrangement of

s

or return

""

if not possible

.

Example 1:

Input:

s = "aab"

Output:

"aba"

Example 2:

Input:

s = "aaab"

Output:

""

Constraints:

1 <= s.length <= 500

s

consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* reorganizeString(char* s) {

}
```

Go:

```
func reorganizeString(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

```
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Reorganize String
* Difficulty: Medium
* Tags: string, greedy, hash, sort, queue, heap
*
* 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 reorganizeString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Reorganize String
* Difficulty: Medium
* Tags: string, greedy, hash, sort, queue, heap
*
* 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 reorganizeString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Reorganize String
Difficulty: Medium
Tags: string, greedy, hash, sort, queue, heap

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 reorganizeString(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Reorganize String
* Difficulty: Medium
* Tags: string, greedy, hash, sort, queue, heap
*
* 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 reorganizeString = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Reorganize String
* Difficulty: Medium
* Tags: string, greedy, hash, sort, queue, heap
*
* 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 reorganizeString(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Reorganize String
* Difficulty: Medium
* Tags: string, greedy, hash, sort, queue, heap
*
* 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 ReorganizeString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Reorganize String
* Difficulty: Medium
* Tags: string, greedy, hash, sort, queue, heap
*
* 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* reorganizeString(char* s) {

}
```

Go Solution:

```
// Problem: Reorganize String
// Difficulty: Medium
// Tags: string, greedy, hash, sort, queue, heap
//
// 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 reorganizeString(s string) string {

}
```

Kotlin Solution:

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

}
```

```
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Reorganize String
// Difficulty: Medium
// Tags: string, greedy, hash, sort, queue, heap
//
// 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 reorganize_string(s: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return String
```

```
*/
function reorganizeString($s) {

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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