Problem 3692: Majority Frequency Characters

Problem Information

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

Problem Description

You are given a string

s

consisting of lowercase English letters.

The

frequency group

for a value

k

is the set of characters that appear exactly

k

times in s.

The

majority frequency group

is the frequency group that contains the largest number of distinct

characters.

Return a string containing all characters in the majority frequency group, in

any

order. If two or more frequency groups tie for that largest size, pick the group whose frequency

k

is

larger

.

Example 1:

Input:

s = "aaabbbccdddde"

Output:

"ab"

Explanation:

Frequency (k)

Distinct characters in group

Group size

Majority?

4

{d}

1

No

3

{a, b}

2

Yes

2

{c}

1

No

1

{e}

1

No

Both characters

'a'

and

'b'

share the same frequency 3, they are in the majority frequency group.

"ba"

is also a valid answer.

Example 2:

Input:

s = "abcd"

Output:

"abcd"

Explanation:

Frequency (k)

Distinct characters in group

Group size

Majority?

1

{a, b, c, d}

4

Yes

All characters share the same frequency 1, they are all in the majority frequency group.

Example 3:

Input:

s = "pfpfgi"

Output:

"fp"

Explanation:

Frequency (k)

Distinct characters in group

Group size

Majority?

2

{p, f}

2

Yes

1

{g, i}

2

No (tied size, lower frequency)

Both characters

'p'

and

'f'

share the same frequency 2, they are in the majority frequency group. There is a tie in group size with frequency 1, but we pick the higher frequency: 2.

Constraints:

1 <= s.length <= 100

s

consists only of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* majorityFrequencyGroup(char* s) {

}
```

Go:

```
func majorityFrequencyGroup(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (majority-frequency-group s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Majority Frequency Characters
* 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 majorityFrequencyGroup(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Majority Frequency Characters
* 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 majorityFrequencyGroup(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Majority Frequency Characters
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Majority Frequency Characters
* 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 majorityFrequencyGroup = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Majority Frequency Characters
* 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 majorityFrequencyGroup(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Majority Frequency Characters
* 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 MajorityFrequencyGroup(string s) {

}
}
```

C Solution:

```
/*
* Problem: Majority Frequency Characters
* 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* majorityFrequencyGroup(char* s) {

}
```

Go Solution:

```
// Problem: Majority Frequency Characters
// 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 majorityFrequencyGroup(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Majority Frequency Characters
// 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 majority_frequency_group(s: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (majority-frequency-group s)
(-> string? string?)
)
```