Problem 3662: Filter Characters by Frequency

Problem Information

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

Problem Description

You are given a string

s

consisting of lowercase English letters and an integer

k

.

Your task is to construct a new string that contains only those characters from

s

which appear

fewer

than

k

times in the entire string. The order of characters in the new string must be the

same

as their

order

in

s

.

Return the resulting string. If no characters qualify, return an empty string.

Note:

Every occurrence

of a character that occurs fewer than

k

times is kept.

Example 1:

Input:

s = "aadbbcccca", k = 3

Output:

"dbb"

Explanation:

Character frequencies in

s

:

'a'

appears 3 times

'd'

appears 1 time

'b'

appears 2 times

'c'

appears 4 times

Only

'd'

and

'b'

appear fewer than 3 times. Preserving their order, the result is

"dbb"

.

Example 2:

Input:

s = "xyz", k = 2

Output:

"xyz"

Explanation:

All characters (

'x'

,

'y'

,

'z'

) appear exactly once, which is fewer than 2. Thus the whole string is returned.

Constraints:

1 <= s.length <= 100

s

consists of lowercase English letters.

1 <= k <= s.length

Code Snippets

C++:

```
class Solution {
public:
string filterCharacters(string s, int k) {

}
};
```

Java:

```
class Solution {
public String filterCharacters(String s, int k) {

}
}
```

Python3:

```
class Solution:
def filterCharacters(self, s: str, k: int) -> str:
```

Python:

```
class Solution(object):
def filterCharacters(self, s, k):
"""
:type s: str
:type k: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number} k
* @return {string}
*/
var filterCharacters = function(s, k) {

};
```

TypeScript:

```
function filterCharacters(s: string, k: number): string {

};
```

C#:

```
public class Solution {
public string FilterCharacters(string s, int k) {

}
}
```

C:

```
char* filterCharacters(char* s, int k) {

}
```

Go:

```
func filterCharacters(s string, k int) string {

}
```

Kotlin:

```
class Solution {
fun filterCharacters(s: String, k: Int): String {

}
}
```

Swift:

```
class Solution {
func filterCharacters(_ s: String, _ k: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn filter_characters(s: String, k: i32) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
# @return {String}
def filter_characters(s, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return String
*/
function filterCharacters($s, $k) {

}
}
```

Dart:

```
class Solution {
String filterCharacters(String s, int k) {

}
}
```

Scala:

```
object Solution {
def filterCharacters(s: String, k: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec filter_characters(s :: String.t, k :: integer) :: String.t
def filter_characters(s, k) do
```

```
end
end
```

Erlang:

```
-spec filter_characters(S :: unicode:unicode_binary(), K :: integer()) ->
unicode:unicode_binary().
filter_characters(S, K) ->
.
```

Racket:

```
(define/contract (filter-characters s k)
(-> string? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Filter Characters by Frequency
* 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 filterCharacters(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Filter Characters by Frequency
* 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 filterCharacters(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Filter Characters by Frequency
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 filterCharacters(self, s: str, k: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def filterCharacters(self, s, k):
"""
:type s: str
:type k: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Filter Characters by Frequency
* 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
* @param {number} k
* @return {string}
*/
var filterCharacters = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Filter Characters by Frequency
* 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 filterCharacters(s: string, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Filter Characters by Frequency
* 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 FilterCharacters(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Filter Characters by Frequency
* 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* filterCharacters(char* s, int k) {

}
```

Go Solution:

```
// Problem: Filter Characters by Frequency
// 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 filterCharacters(s string, k int) string {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun filterCharacters(s: String, k: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func filterCharacters(_ s: String, _ k: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Filter Characters by Frequency
// 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 filter_characters(s: String, k: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @return {String}
def filter_characters(s, k)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return String
*/
function filterCharacters($s, $k) {

}
}
```

Dart Solution:

```
class Solution {
String filterCharacters(String s, int k) {

}
}
```

Scala Solution:

```
object Solution {
def filterCharacters(s: String, k: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec filter_characters(s :: String.t, k :: integer) :: String.t
def filter_characters(s, k) do

end
end
```

Erlang Solution:

```
-spec filter_characters(S :: unicode:unicode_binary(), K :: integer()) ->
unicode:unicode_binary().
filter_characters(S, K) ->
.
```

Racket Solution:

```
(define/contract (filter-characters s k)
(-> string? exact-integer? string?)
)
```