Problem 358: Rearrange String k Distance Apart

Problem Information

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

Problem Description

Given a string

s

and an integer

k

, rearrange

s

such that the same characters are

at least

distance

k

from each other. If it is not possible to rearrange the string, return an empty string

""

.

Example 1:

Input:

s = "aabbcc", k = 3

Output:

"abcabc"

Explanation:

The same letters are at least a distance of 3 from each other.

Example 2:

Input:

s = "aaabc", k = 3

Output:

""

Explanation:

It is not possible to rearrange the string.

Example 3:

Input:

s = "aaadbbcc", k = 2

Output:

"abacabcd"

Explanation:

The same letters are at least a distance of 2 from each other.

Constraints:

1 <= s.length <= 3 * 10

5

s

consists of only lowercase English letters.

0 <= k <= s.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

```
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
```

```
* @return String
*/
function rearrangeString($s, $k) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Rearrange String k Distance Apart
* Difficulty: Hard
* 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 rearrangeString(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Rearrange String k Distance Apart
* Difficulty: Hard
* 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 rearrangeString(String s, int k) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Rearrange String k Distance Apart
Difficulty: Hard
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 rearrangeString(self, s: str, k: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Rearrange String k Distance Apart
* Difficulty: Hard
* 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
* @param {number} k
* @return {string}
*/
var rearrangeString = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Rearrange String k Distance Apart
* Difficulty: Hard
* 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 rearrangeString(s: string, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Rearrange String k Distance Apart
* Difficulty: Hard
* 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 RearrangeString(string s, int k) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Rearrange String k Distance Apart
* Difficulty: Hard
* 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* rearrangeString(char* s, int k) {

}
```

Go Solution:

```
// Problem: Rearrange String k Distance Apart
// Difficulty: Hard
// 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 rearrangeString(s string, k int) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Rearrange String k Distance Apart
// Difficulty: Hard
// 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 rearrange_string(s: String, k: i32) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

```
}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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