Problem 899: Orderly Queue

Problem Information

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

Problem Description

You are given a string

s

and an integer

k

. You can choose one of the first

k

letters of

s

and append it at the end of the string.

Return

the lexicographically smallest string you could have after applying the mentioned step any number of moves

.

Example 1:

Input:

s = "cba", k = 1

Output:

"acb"

Explanation:

In the first move, we move the 1

st

character 'c' to the end, obtaining the string "bac". In the second move, we move the 1

st

character 'b' to the end, obtaining the final result "acb".

Example 2:

Input:

s = "baaca", k = 3

Output:

"aaabc"

Explanation:

In the first move, we move the 1

st

character 'b' to the end, obtaining the string "aacab". In the second move, we move the 3

rd

character 'c' to the end, obtaining the final result "aaabc".

Constraints:

1 <= k <= s.length <= 1000

s

consist of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def orderlyQueue(self, s, k):
```

```
"""
:type s: str
:type k: int
:rtype: str
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

```
}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Orderly Queue
* Difficulty: Hard
* Tags: string, graph, math, sort
*
* 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 orderlyQueue(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Orderly Queue
* Difficulty: Hard
* Tags: string, graph, math, sort
*
* 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 orderlyQueue(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Orderly Queue
Difficulty: Hard
Tags: string, graph, math, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Orderly Queue
* Difficulty: Hard
* Tags: string, graph, math, sort
*
* 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
* @param {number} k
* @return {string}
*/
```

```
var orderlyQueue = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Orderly Queue
* Difficulty: Hard
* Tags: string, graph, math, sort
*
* 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 orderlyQueue(s: string, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Orderly Queue
* Difficulty: Hard
* Tags: string, graph, math, sort
*
* 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 OrderlyQueue(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Orderly Queue
* Difficulty: Hard
* Tags: string, graph, math, sort
*
* 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* orderlyQueue(char* s, int k) {

}
```

Go Solution:

```
// Problem: Orderly Queue
// Difficulty: Hard
// Tags: string, graph, math, sort
//
// 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 orderlyQueue(s string, k int) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
```

```
}
```

Rust Solution:

```
// Problem: Orderly Queue
// Difficulty: Hard
// Tags: string, graph, math, sort
//
// 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 orderly_queue(s: String, k: i32) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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