Problem 2014: Longest Subsequence Repeated k Times

Problem Information

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

Problem Description

You are given a string

s

of length

n

, and an integer

k

. You are tasked to find the

longest subsequence repeated

k

times in string

s

.

A

subsequence

is a string that can be derived from another string by deleting some or no characters without changing the order of the remaining characters.

A subsequence

seq

is

repeated

k

times in the string

s

if

seq * k

is a subsequence of

s

, where

seq * k

represents a string constructed by concatenating

seq

k

times.

For example,

"bba"

is repeated

2

times in the string

"bababcba"

, because the string

"bbabba"

, constructed by concatenating

"bba"

2

times, is a subsequence of the string

"

b

a

bab

c

ba

"

.

Return

the

longest subsequence repeated

k

times in string

s

. If multiple such subsequences are found, return the

lexicographically largest

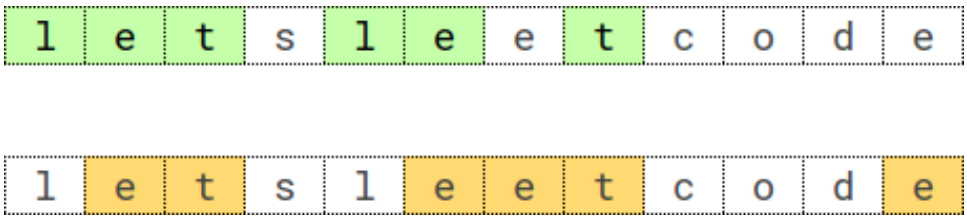
one. If there is no such subsequence, return an

empty

string

.

Example 1:



Input:

s = "letsleetcode", k = 2

Output:

"let"

Explanation:

There are two longest subsequences repeated 2 times: "let" and "ete". "let" is the lexicographically largest one.

Example 2:

Input:

s = "bb", k = 2

Output:

"b"

Explanation:

The longest subsequence repeated 2 times is "b".

Example 3:

Input:

s = "ab", k = 2

Output:

""

Explanation:

There is no subsequence repeated 2 times. Empty string is returned.

Constraints:

n == s.length

2 <= k <= 2000

2 <= n < min(2001, k * 8)

s

consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (longest-subsequence-repeated-k s k)
(-> string? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Subsequence Repeated k Times
* Difficulty: Hard
* Tags: string, graph, greedy
*
* 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 longestSubsequenceRepeatedK(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Subsequence Repeated k Times
* Difficulty: Hard
* Tags: string, graph, greedy
*
* 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 longestSubsequenceRepeatedK(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Subsequence Repeated k Times
```

```
Difficulty: Hard
Tags: string, graph, greedy

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Subsequence Repeated k Times
* Difficulty: Hard
* Tags: string, graph, greedy
*
* 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 longestSubsequenceRepeatedK = function(s, k) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Longest Subsequence Repeated k Times
* Difficulty: Hard
* Tags: string, graph, greedy
*
* 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 longestSubsequenceRepeatedK(s: string, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Longest Subsequence Repeated k Times
* Difficulty: Hard
* Tags: string, graph, greedy
*
* 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 LongestSubsequenceRepeatedK(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Longest Subsequence Repeated k Times
```

```
* Difficulty: Hard
* Tags: string, graph, greedy
*
* 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* longestSubsequenceRepeatedK(char* s, int k) {

}
```

Go Solution:

```
// Problem: Longest Subsequence Repeated k Times
// Difficulty: Hard
// Tags: string, graph, greedy
//
// 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 longestSubsequenceRepeatedK(s string, k int) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Subsequence Repeated k Times
// Difficulty: Hard
// Tags: string, graph, greedy
//
// 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 longest_subsequence_repeated_k(s: String, k: i32) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (longest-subsequence-repeated-k s k)
(-> string? exact-integer? string?)
)
```