Problem 2030: Smallest K-Length Subsequence With Occurrences of a Letter

Problem Information

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

Problem Description

You are given a string

s

, an integer

k

, a letter

letter

, and an integer

repetition

.

Return

the

lexicographically smallest

subsequence of

s

of length

k

that has the letter

letter

appear

at least

repetition

times

. The test cases are generated so that the

letter

appears in

s

at least

repetition

times.

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 string

a

is

lexicographically smaller

than a string

b

if in the first position where

a

and

b

differ, string

a

has a letter that appears earlier in the alphabet than the corresponding letter in

b

.

Example 1:

Input:

s = "leet", k = 3, letter = "e", repetition = 1

Output:

"eet"

Explanation:

There are four subsequences of length 3 that have the letter 'e' appear at least 1 time: - "lee" (from "

lee

t") - "let" (from "

le

e

t

") - "let" (from "

l

e

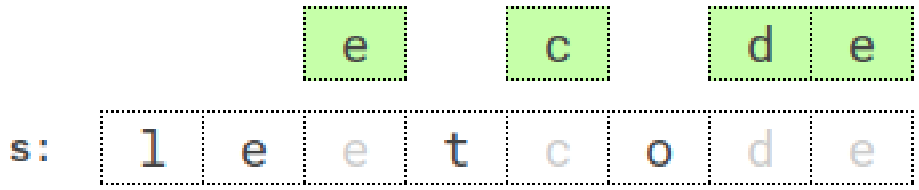
et

") - "eet" (from "l

eet

") The lexicographically smallest subsequence among them is "eet".

Example 2:



Input:

s = "leetcode", k = 4, letter = "e", repetition = 2

Output:

"ecde"

Explanation:

"ecde" is the lexicographically smallest subsequence of length 4 that has the letter "e" appear at least 2 times.

Example 3:

Input:

s = "bb", k = 2, letter = "b", repetition = 2

Output:

"bb"

Explanation:

"bb" is the only subsequence of length 2 that has the letter "b" appear at least 2 times.

Constraints:

1 <= repetition <= k <= s.length <= 5 * 10

4

s

consists of lowercase English letters.

letter

is a lowercase English letter, and appears in s

at least

repetition

times.

Code Snippets

C++:

```
class Solution {
public:
string smallestSubsequence(string s, int k, char letter, int repetition) {

}
};
```

Java:

```
class Solution {
public String smallestSubsequence(String s, int k, char letter, int
repetition) {

}
}
```

Python3:

```
class Solution:
def smallestSubsequence(self, s: str, k: int, letter: str, repetition: int)
-> str:
```

Python:

```
class Solution(object):
def smallestSubsequence(self, s, k, letter, repetition):
"""
:type s: str
```

```
:type k: int
:type letter: str
:type repetition: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number} k
* @param {character} letter
* @param {number} repetition
* @return {string}
*/
var smallestSubsequence = function(s, k, letter, repetition) {

};
```

TypeScript:

```
function smallestSubsequence(s: string, k: number, letter: string,
repetition: number): string {

};
```

C#:

```
public class Solution {
public string SmallestSubsequence(string s, int k, char letter, int
repetition) {

}
}
```

C:

```
char* smallestSubsequence(char* s, int k, char letter, int repetition) {

}
```

Go:

```
func smallestSubsequence(s string, k int, letter byte, repetition int) string
{

}
```

Kotlin:

```
class Solution {
fun smallestSubsequence(s: String, k: Int, letter: Char, repetition: Int):
String {

}
}
```

Swift:

```
class Solution {
func smallestSubsequence(_ s: String, _ k: Int, _ letter: Character, _
repetition: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn smallest_subsequence(s: String, k: i32, letter: char, repetition: i32)
-> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
# @param {Character} letter
# @param {Integer} repetition
# @return {String}
def smallest_subsequence(s, k, letter, repetition)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @param String $letter
* @param Integer $repetition
* @return String
*/
function smallestSubsequence($s, $k, $letter, $repetition) {

}
}
```

Dart:

```
class Solution {
String smallestSubsequence(String s, int k, String letter, int repetition) {

}
}
```

Scala:

```
object Solution {
def smallestSubsequence(s: String, k: Int, letter: Char, repetition: Int):
String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec smallest_subsequence(s :: String.t, k :: integer, letter :: char,
repetition :: integer) :: String.t
def smallest_subsequence(s, k, letter, repetition) do

end
end
```

Erlang:

```
-spec smallest_subsequence(S :: unicode:unicode_binary(), K :: integer(),
Letter :: char(), Repetition :: integer()) -> unicode:unicode_binary().
smallest_subsequence(S, K, Letter, Repetition) ->
.
```

Racket:

```
(define/contract (smallest-subsequence s k letter repetition)
(-> string? exact-integer? char? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest K-Length Subsequence With Occurrences of a Letter
* Difficulty: Hard
* Tags: string, graph, greedy, stack
*
* 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 smallestSubsequence(string s, int k, char letter, int repetition) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest K-Length Subsequence With Occurrences of a Letter
* Difficulty: Hard
* Tags: string, graph, greedy, stack
*
```

```
* 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 smallestSubsequence(String s, int k, char letter, int
repetition) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest K-Length Subsequence With Occurrences of a Letter
Difficulty: Hard
Tags: string, graph, greedy, stack

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Smallest K-Length Subsequence With Occurrences of a Letter
* Difficulty: Hard
* Tags: string, graph, greedy, stack
*
* 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
* @param {character} letter
* @param {number} repetition
* @return {string}
*/
var smallestSubsequence = function(s, k, letter, repetition) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest K-Length Subsequence With Occurrences of a Letter
* Difficulty: Hard
* Tags: string, graph, greedy, stack
*
* 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 smallestSubsequence(s: string, k: number, letter: string,
repetition: number): string {

};
```

C# Solution:

```
/*
* Problem: Smallest K-Length Subsequence With Occurrences of a Letter
* Difficulty: Hard
* Tags: string, graph, greedy, stack
*
* 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 SmallestSubsequence(string s, int k, char letter, int
repetition) {

}
}
```

C Solution:

```
/*
* Problem: Smallest K-Length Subsequence With Occurrences of a Letter
* Difficulty: Hard
* Tags: string, graph, greedy, stack
*
* 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* smallestSubsequence(char* s, int k, char letter, int repetition) {

}
```

Go Solution:

```
// Problem: Smallest K-Length Subsequence With Occurrences of a Letter
// Difficulty: Hard
// Tags: string, graph, greedy, stack
//
// 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 smallestSubsequence(s string, k int, letter byte, repetition int) string
{

}
```

Kotlin Solution:

```
class Solution {
fun smallestSubsequence(s: String, k: Int, letter: Char, repetition: Int):
String {

}
}
```

Swift Solution:

```
class Solution {
func smallestSubsequence(_ s: String, _ k: Int, _ letter: Character, _
repetition: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Smallest K-Length Subsequence With Occurrences of a Letter
// Difficulty: Hard
// Tags: string, graph, greedy, stack
//
// 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 smallest_subsequence(s: String, k: i32, letter: char, repetition: i32)
-> String {

}
}
```

Ruby Solution:

```ruby
# @param {String} s
# @param {Integer} k
# @param {Character} letter
# @param {Integer} repetition
# @return {String}
def smallest_subsequence(s, k, letter, repetition)

end
```

PHP Solution:

```php
class Solution {

/**
* @param String $s
* @param Integer $k
* @param String $letter
* @param Integer $repetition
* @return String
*/
function smallestSubsequence($s, $k, $letter, $repetition) {

}
}
```

Dart Solution:

```dart
class Solution {
String smallestSubsequence(String s, int k, String letter, int repetition) {

}
}
```

Scala Solution:

```scala
object Solution {
def smallestSubsequence(s: String, k: Int, letter: Char, repetition: Int):
String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec smallest_subsequence(s :: String.t, k :: integer, letter :: char,
repetition :: integer) :: String.t
def smallest_subsequence(s, k, letter, repetition) do

end
end
```

Erlang Solution:

```
-spec smallest_subsequence(S :: unicode:unicode_binary(), K :: integer(),
Letter :: char(), Repetition :: integer()) -> unicode:unicode_binary().
smallest_subsequence(S, K, Letter, Repetition) ->
.
```

Racket Solution:

```
(define/contract (smallest-subsequence s k letter repetition)
(-> string? exact-integer? char? exact-integer? string?)
)
```