

Problem 2663: Lexicographically Smallest Beautiful String

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

A string is

beautiful

if:

It consists of the first

k

letters of the English lowercase alphabet.

It does not contain any substring of length

2

or more which is a palindrome.

You are given a beautiful string

s

of length

n

and a positive integer

k

.

Return

the lexicographically smallest string of length

n

, which is larger than

s

and is

beautiful

. If there is no such string, return an empty string.

A string

a

is lexicographically larger than a string

b

(of the same length) if in the first position where

a

and

b

differ,

a

has a character strictly larger than the corresponding character in

b

.

For example,

"abcd"

is lexicographically larger than

"abcc"

because the first position they differ is at the fourth character, and

d

is greater than

c

.

Example 1:

Input:

s = "abcz", k = 26

Output:

"abda"

Explanation:

The string "abda" is beautiful and lexicographically larger than the string "abcz". It can be proven that there is no string that is lexicographically larger than the string "abcz", beautiful, and lexicographically smaller than the string "abda".

Example 2:

Input:

s = "dc", k = 4

Output:

""

Explanation:

It can be proven that there is no string that is lexicographically larger than the string "dc" and is beautiful.

Constraints:

$1 \leq n == s.length \leq 10$

5

$4 \leq k \leq 26$

s

is a beautiful string.

Code Snippets

C++:

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

```
}
```

```
};
```

Java:

```
class Solution {
    public String smallestBeautifulString(String s, int k) {
        ...
    }
}
```

Python3:

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

Python:

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

```

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

C:

```
char* smallestBeautifulString(char* s, int k) {  
    }
```

Go:

```
func smallestBeautifulString(s string, k int) string {  
    }
```

Kotlin:

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

Swift:

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

Rust:

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

```
}
```

```
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

    }
}
```

Dart:

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

}
```

Scala:

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

}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
 * Problem: Lexicographically Smallest Beautiful String
 * Difficulty: Hard
 * Tags: string, tree, graph, greedy
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

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

```
};
```

Java Solution:

```
/**  
 * Problem: Lexicographically Smallest Beautiful String  
 * Difficulty: Hard  
 * Tags: string, tree, graph, greedy  
 *  
 * Approach: String manipulation with hash map or two pointers  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(h) for recursion stack where h is height  
 */  
  
class Solution {  
    public String smallestBeautifulString(String s, int k) {  
        // Implementation  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Lexicographically Smallest Beautiful String  
Difficulty: Hard  
Tags: string, tree, graph, greedy  
  
Approach: String manipulation with hash map or two pointers  
Time Complexity: O(n) or O(n log n)  
Space Complexity: O(h) for recursion stack where h is height  
"""  
  
class Solution:  
    def smallestBeautifulString(self, s: str, k: int) -> str:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

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

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

JavaScript Solution:

```
/**
 * Problem: Lexicographically Smallest Beautiful String
 * Difficulty: Hard
 * Tags: string, tree, graph, greedy
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

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

};
```

TypeScript Solution:

```
/**
 * Problem: Lexicographically Smallest Beautiful String
 * Difficulty: Hard
 * Tags: string, tree, graph, greedy
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

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

```
};
```

C# Solution:

```
/*
 * Problem: Lexicographically Smallest Beautiful String
 * Difficulty: Hard
 * Tags: string, tree, graph, greedy
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

public class Solution {
    public string SmallestBeautifulString(string s, int k) {
        return s;
    }
}
```

C Solution:

```
/*
 * Problem: Lexicographically Smallest Beautiful String
 * Difficulty: Hard
 * Tags: string, tree, graph, greedy
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

char* smallestBeautifulString(char* s, int k) {
    return s;
}
```

Go Solution:

```
// Problem: Lexicographically Smallest Beautiful String
// Difficulty: Hard
// Tags: string, tree, graph, greedy
```

```

// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func smallestBeautifulString(s string, k int) string {
}

```

Kotlin Solution:

```

class Solution {
    fun smallestBeautifulString(s: String, k: Int): String {
        return s
    }
}

```

Swift Solution:

```

class Solution {
    func smallestBeautifulString(_ s: String, _ k: Int) -> String {
        return s
    }
}

```

Rust Solution:

```

// Problem: Lexicographically Smallest Beautiful String
// Difficulty: Hard
// Tags: string, tree, graph, greedy
// 
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
    pub fn smallest_beautiful_string(s: String, k: i32) -> String {
        return s
    }
}

```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }
}
```

Dart Solution:

```
class Solution {
  String smallestBeautifulString(String s, int k) {
    }
}
```

Scala Solution:

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

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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