Problem 2904: Shortest and Lexicographically Smallest Beautiful String

Problem Information

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

Problem Description

You are given a binary string

s

and a positive integer

k

.

A substring of

s

is

beautiful

if the number of

1

's in it is exactly

k

.

Let

len

be the length of the

shortest

beautiful substring.

Return

the lexicographically

smallest

beautiful substring of string

s

with length equal to

len

. If

s

doesn't contain a beautiful substring, 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 = "100011001", k = 3

Output:

"11001"

Explanation:

There are 7 beautiful substrings in this example: 1. The substring "

100011

001". 2. The substring "

1000110

01". 3. The substring "

10001100

1". 4. The substring "1

00011001

". 5. The substring "10

0011001

". 6. The substring "100

011001

". 7. The substring "1000

11001

". The length of the shortest beautiful substring is 5. The lexicographically smallest beautiful substring with length 5 is the substring "11001".

Example 2:

Input:

s = "1011", k = 2

Output:

"11"

Explanation:

There are 3 beautiful substrings in this example: 1. The substring "

101

1". 2. The substring "1

011

". 3. The substring "10

11

". The length of the shortest beautiful substring is 2. The lexicographically smallest beautiful substring with length 2 is the substring "11".

Example 3:

Input:

s = "000", k = 1

Output:

""

Explanation:

There are no beautiful substrings in this example.

Constraints:

1 <= s.length <= 100

1 <= k <= s.length

Code Snippets

C++:

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

}
};
```

Java:

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

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```
end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Shortest and Lexicographically Smallest Beautiful String
* Difficulty: Medium
* Tags: array, string, tree, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Shortest and Lexicographically Smallest Beautiful String
* Difficulty: Medium
* Tags: array, string, tree, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public String shortestBeautifulSubstring(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest and Lexicographically Smallest Beautiful String
Difficulty: Medium
Tags: array, string, tree, graph

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def shortestBeautifulSubstring(self, s: str, k: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest and Lexicographically Smallest Beautiful String
* Difficulty: Medium
```

```
* Tags: array, string, tree, graph
*
* Approach: Use two pointers or sliding window technique
* 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 shortestBeautifulSubstring = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest and Lexicographically Smallest Beautiful String
* Difficulty: Medium
* Tags: array, string, tree, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function shortestBeautifulSubstring(s: string, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Shortest and Lexicographically Smallest Beautiful String
* Difficulty: Medium
* Tags: array, string, tree, graph
*
* Approach: Use two pointers or sliding window technique
* 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 ShortestBeautifulSubstring(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Shortest and Lexicographically Smallest Beautiful String
* Difficulty: Medium
* Tags: array, string, tree, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

char* shortestBeautifulSubstring(char* s, int k) {

}
```

Go Solution:

```
// Problem: Shortest and Lexicographically Smallest Beautiful String
// Difficulty: Medium
// Tags: array, string, tree, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func shortestBeautifulSubstring(s string, k int) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shortest and Lexicographically Smallest Beautiful String
// Difficulty: Medium
// Tags: array, string, tree, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

```
.
```

Racket Solution:

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