Problem 3106: Lexicographically Smallest String After Operations With Constraint

Problem Information

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

Problem Description

You are given a string

s

and an integer

k

.

Define a function

distance(s

1

, s

2

)

between two strings

s

1

and

s

2

of the same length

n

as:

The

sum

of the

minimum distance

between

s

1

[i]

and

s

2

[i]

when the characters from

'a'

to

'z'

are placed in a

cyclic

order, for all

i

in the range

[0, n - 1]

.

For example,

distance("ab", "cd") == 4

, and

distance("a", "z") == 1

.

You can

change

any letter of

s

to

any

other lowercase English letter,

any

number of times.

Return a string denoting the

lexicographically smallest

string

t

you can get after some changes, such that

distance(s, t) <= k

.

Example 1:

Input:

s = "zbbz", k = 3

Output:

"aaaz"

Explanation:

Change

s

to

"aaaz"

. The distance between

"zbbz"

and

"aaaz"

is equal to

k = 3

.

Example 2:

Input:

s = "xaxcd", k = 4

Output:

"aawcd"

Explanation:

The distance between "xaxcd" and "aawcd" is equal to k = 4.

Example 3:

Input:

s = "lol", k = 0

Output:

"lol"

Explanation:

It's impossible to change any character as

k = 0

.

Constraints:

1 <= s.length <= 100

0 <= k <= 2000

s

consists only of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```
end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Lexicographically Smallest String After Operations With Constraint
* Difficulty: Medium
* 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 getSmallestString(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Lexicographically Smallest String After Operations With Constraint
* Difficulty: Medium
* 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 getSmallestString(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Lexicographically Smallest String After Operations With Constraint
Difficulty: Medium
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 getSmallestString(self, s: str, k: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Lexicographically Smallest String After Operations With Constraint
* Difficulty: Medium
```

```
* 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 getSmallestString = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Lexicographically Smallest String After Operations With Constraint
* Difficulty: Medium
* 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 getSmallestString(s: string, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Lexicographically Smallest String After Operations With Constraint
* Difficulty: Medium
* 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 GetSmallestString(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Lexicographically Smallest String After Operations With Constraint
* Difficulty: Medium
* 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* getSmallestString(char* s, int k) {

}
```

Go Solution:

```
// Problem: Lexicographically Smallest String After Operations With
Constraint
// Difficulty: Medium
// 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 getSmallestString(s string, k int) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Lexicographically Smallest String After Operations With
Constraint
// Difficulty: Medium
// 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 get_smallest_string(s: String, k: i32) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

```
.
```

Racket Solution:

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