

Problem 3735: Lexicographically Smallest String After Reverse II

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a string

s

of length

n

consisting of lowercase English letters.

You must perform

exactly

one operation by choosing any integer

k

such that

$1 \leq k \leq n$

and either:

reverse the

first

k

characters of

s

, or

reverse the

last

k

characters of

s

.

Return the

lexicographically smallest

string that can be obtained after

exactly

one such operation.

Example 1:

Input:

s = "dcab"

Output:

"acdb"

Explanation:

Choose

$k = 3$

, reverse the first 3 characters.

Reverse

"dca"

to

"acd"

, resulting string

$s = "acdb"$

, which is the lexicographically smallest string achievable.

Example 2:

Input:

$s = "abba"$

Output:

"aabb"

Explanation:

Choose

$k = 3$

, reverse the last 3 characters.

Reverse

"bba"

to

"abb"

, so the resulting string is

"aabb"

, which is the lexicographically smallest string achievable.

Example 3:

Input:

$s = "zxy"$

Output:

"xzy"

Explanation:

Choose

$k = 2$

, reverse the first 2 characters.

Reverse

"zx"

to

"xz"

, so the resulting string is

"xzy"

, which is the lexicographically smallest string achievable.

Constraints:

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

5

s

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
    string lexSmallest(string s) {
        }
};
```

Java:

```
class Solution {
public String lexSmallest(String s) {
    }
```

```
}
```

Python3:

```
class Solution:  
    def lexSmallest(self, s: str) -> str:
```

Python:

```
class Solution(object):  
    def lexSmallest(self, s):  
        """  
        :type s: str  
        :rtype: str  
        """
```

JavaScript:

```
/**  
 * @param {string} s  
 * @return {string}  
 */  
var lexSmallest = function(s) {  
  
};
```

TypeScript:

```
function lexSmallest(s: string): string {  
  
};
```

C#:

```
public class Solution {  
    public string LexSmallest(string s) {  
  
    }  
}
```

C:

```
char* lexSmallest(char* s) {  
}  
}
```

Go:

```
func lexSmallest(s string) string {  
  
}  
}
```

Kotlin:

```
class Solution {  
    fun lexSmallest(s: String): String {  
  
    }  
}
```

Swift:

```
class Solution {  
    func lexSmallest(_ s: String) -> String {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn lex_smallest(s: String) -> String {  
  
    }  
}
```

Ruby:

```
# @param {String} s  
# @return {String}  
def lex_smallest(s)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param String $s  
     * @return String  
     */  
    function lexSmallest($s) {  
  
    }  
}
```

Dart:

```
class Solution {  
String lexSmallest(String s) {  
  
}  
}
```

Scala:

```
object Solution {  
def lexSmallest(s: String): String = {  
  
}  
}
```

Elixir:

```
defmodule Solution do  
@spec lex_smallest(s :: String.t) :: String.t  
def lex_smallest(s) do  
  
end  
end
```

Erlang:

```
-spec lex_smallest(S :: unicode:unicode_binary()) ->  
unicode:unicode_binary().  
lex_smallest(S) ->  
.
```

Racket:

```
(define/contract (lex-smallest s)
  (-> string? string?))
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Lexicographically Smallest String After Reverse II
 * Difficulty: Hard
 * Tags: array, string, graph, hash, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
public:
    string lexSmallest(string s) {

    }
};
```

Java Solution:

```
/**
 * Problem: Lexicographically Smallest String After Reverse II
 * Difficulty: Hard
 * Tags: array, string, graph, hash, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
    public String lexSmallest(String s) {
```

```
}
```

```
}
```

Python3 Solution:

```
"""
Problem: Lexicographically Smallest String After Reverse II
Difficulty: Hard
Tags: array, string, graph, hash, search
```

```
Approach: Use two pointers or sliding window technique
```

```
Time Complexity: O(n) or O(n log n)
```

```
Space Complexity: O(n) for hash map
```

```
"""
```

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

Python Solution:

```
class Solution(object):
    def lexSmallest(self, s):
        """
        :type s: str
        :rtype: str
        """
```

JavaScript Solution:

```
/**
 * Problem: Lexicographically Smallest String After Reverse II
 * Difficulty: Hard
 * Tags: array, string, graph, hash, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */
```

```

/**
 * @param {string} s
 * @return {string}
 */
var lexSmallest = function(s) {

};


```

TypeScript Solution:

```

/**
 * Problem: Lexicographically Smallest String After Reverse II
 * Difficulty: Hard
 * Tags: array, string, graph, hash, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

function lexSmallest(s: string): string {

};


```

C# Solution:

```

/*
 * Problem: Lexicographically Smallest String After Reverse II
 * Difficulty: Hard
 * Tags: array, string, graph, hash, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

public class Solution {
    public string LexSmallest(string s) {
        }


```

```
}
```

C Solution:

```
/*
 * Problem: Lexicographically Smallest String After Reverse II
 * Difficulty: Hard
 * Tags: array, string, graph, hash, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

char* lexSmallest(char* s) {

}
```

Go Solution:

```
// Problem: Lexicographically Smallest String After Reverse II
// Difficulty: Hard
// Tags: array, string, graph, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func lexSmallest(s string) string {

}
```

Kotlin Solution:

```
class Solution {
    fun lexSmallest(s: String): String {
        }

    }
}
```

Swift Solution:

```
class Solution {  
    func lexSmallest(_ s: String) -> String {  
        }  
    }  
}
```

Rust Solution:

```
// Problem: Lexicographically Smallest String After Reverse II  
// Difficulty: Hard  
// Tags: array, string, graph, hash, search  
//  
// Approach: Use two pointers or sliding window technique  
// Time Complexity: O(n) or O(n log n)  
// Space Complexity: O(n) for hash map  
  
impl Solution {  
    pub fn lex_smallest(s: String) -> String {  
        }  
    }  
}
```

Ruby Solution:

```
# @param {String} s  
# @return {String}  
def lex_smallest(s)  
  
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param String $s  
     * @return String  
     */  
    function lexSmallest($s) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
    String lexSmallest(String s) {  
  
    }  
}
```

Scala Solution:

```
object Solution {  
    def lexSmallest(s: String): String = {  
  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
    @spec lex_smallest(s :: String.t) :: String.t  
    def lex_smallest(s) do  
  
    end  
end
```

Erlang Solution:

```
-spec lex_smallest(S :: unicode:unicode_binary()) ->  
    unicode:unicode_binary().  
lex_smallest(S) ->  
.
```

Racket Solution:

```
(define/contract (lex-smallest s)  
  (-> string? string?)  
)
```