

Problem 3734: Lexicographically Smallest Palindromic Permutation Greater Than Target

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given two strings

s

and

target

, each of length

n

, consisting of lowercase English letters.

Return the

lexicographically smallest

string

that is

both

a

palindromic

permutation

of

s

and

strictly

greater than

target

. If no such permutation exists, return an empty string.

Example 1:

Input:

s = "baba", target = "abba"

Output:

"baab"

Explanation:

The palindromic permutations of

s

(in lexicographical order) are

"abba"

and

"baab"

The lexicographically smallest permutation that is strictly greater than

target

is

"baab"

Example 2:

Input:

s = "baba", target = "bbaa"

Output:

""

Explanation:

The palindromic permutations of

s

(in lexicographical order) are

"abba"

and

"baab"

None of them is lexicographically strictly greater than

target

. Therefore, the answer is

""

Example 3:

Input:

s = "abc", target = "abb"

Output:

""

Explanation:

s

has no palindromic permutations. Therefore, the answer is

""

Example 4:

Input:

s = "aac", target = "abb"

Output:

"aca"

Explanation:

The only palindromic permutation of

s

is

"aca"

"aca"

is strictly greater than

target

. Therefore, the answer is

"aca"

Constraints:

$1 \leq n \leq s.length \leq target.length \leq 300$

s

and

target

consist of only lowercase English letters.

Code Snippets

C++:

```
class Solution {  
public:  
    string lexPalindromicPermutation(string s, string target) {  
  
    }  
};
```

Java:

```
class Solution {  
public String lexPalindromicPermutation(String s, String target) {  
  
}  
}
```

Python3:

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

Python:

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

JavaScript:

```
/**  
 * @param {string} s  
 * @param {string} target  
 * @return {string}  
 */
```

```
var lexPalindromicPermutation = function(s, target) {  
};
```

TypeScript:

```
function lexPalindromicPermutation(s: string, target: string): string {  
};
```

C#:

```
public class Solution {  
    public string LexPalindromicPermutation(string s, string target) {  
        }  
    }
```

C:

```
char* lexPalindromicPermutation(char* s, char* target) {  
}
```

Go:

```
func lexPalindromicPermutation(s string, target string) string {  
}
```

Kotlin:

```
class Solution {  
    fun lexPalindromicPermutation(s: String, target: String): String {  
        }  
    }
```

Swift:

```
class Solution {  
    func lexPalindromicPermutation(_ s: String, _ target: String) -> String {
```

```
}
```

```
}
```

Rust:

```
impl Solution {
    pub fn lex_palindromic_permutation(s: String, target: String) -> String {
        }
    }
}
```

Ruby:

```
# @param {String} s
# @param {String} target
# @return {String}
def lex_palindromic_permutation(s, target)

end
```

PHP:

```
class Solution {

    /**
     * @param String $s
     * @param String $target
     * @return String
     */
    function lexPalindromicPermutation($s, $target) {

    }
}
```

Dart:

```
class Solution {
    String lexPalindromicPermutation(String s, String target) {
        }
    }
}
```

Scala:

```
object Solution {  
    def lexPalindromicPermutation(s: String, target: String): String = {  
        }  
        }  
}
```

Elixir:

```
defmodule Solution do  
    @spec lex_palindromic_permutation(s :: String.t, target :: String.t) ::  
        String.t  
    def lex_palindromic_permutation(s, target) do  
  
        end  
        end
```

Erlang:

```
-spec lex_palindromic_permutation(S :: unicode:unicode_binary(), Target ::  
    unicode:unicode_binary()) -> unicode:unicode_binary().  
lex_palindromic_permutation(S, Target) ->  
.
```

Racket:

```
(define/contract (lex-palindromic-permutation s target)  
    (-> string? string? string?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Lexicographically Smallest Palindromic Permutation Greater Than  
 * Target  
 * Difficulty: Hard  
 * Tags: array, string, graph  
 */
```

```

* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

```

```

class Solution {
public:
    string lexPalindromicPermutation(string s, string target) {

```

```

    }
};

```

Java Solution:

```

/**
 * Problem: Lexicographically Smallest Palindromic Permutation Greater Than
Target
* Difficulty: Hard
* Tags: array, string, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

```

```

class Solution {
public String lexPalindromicPermutation(String s, String target) {

```

```

}
}

```

Python3 Solution:

```

"""
Problem: Lexicographically Smallest Palindromic Permutation Greater Than
Target
Difficulty: Hard
Tags: array, string, graph

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)

```

```

Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:

def lexPalindromicPermutation(self, s: str, target: str) -> str:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def lexPalindromicPermutation(self, s, target):
"""
:type s: str
:type target: str
:rtype: str
"""

```

JavaScript Solution:

```

/**
 * Problem: Lexicographically Smallest Palindromic Permutation Greater Than
Target
* Difficulty: Hard
* Tags: array, string, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

```

```

/**
* @param {string} s
* @param {string} target
* @return {string}
*/
var lexPalindromicPermutation = function(s, target) {

};

```

TypeScript Solution:

```

/**
 * Problem: Lexicographically Smallest Palindromic Permutation Greater Than
Target
* Difficulty: Hard
* Tags: array, string, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
function lexPalindromicPermutation(s: string, target: string): string {
}

```

C# Solution:

```

/*
 * Problem: Lexicographically Smallest Palindromic Permutation Greater Than
Target
* Difficulty: Hard
* Tags: array, string, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
public class Solution {
    public string LexPalindromicPermutation(string s, string target) {
        }
    }
}

```

C Solution:

```

/*
 * Problem: Lexicographically Smallest Palindromic Permutation Greater Than
Target
* Difficulty: Hard
* Tags: array, string, graph
*

```

```

* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
char* lexPalindromicPermutation(char* s, char* target) {

}

```

Go Solution:

```

// Problem: Lexicographically Smallest Palindromic Permutation Greater Than
Target
// Difficulty: Hard
// Tags: array, string, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func lexPalindromicPermutation(s string, target string) string {
}

```

Kotlin Solution:

```

class Solution {
    fun lexPalindromicPermutation(s: String, target: String): String {
        }
    }
}

```

Swift Solution:

```

class Solution {
    func lexPalindromicPermutation(_ s: String, _ target: String) -> String {
        }
    }
}

```

Rust Solution:

```

// Problem: Lexicographically Smallest Palindromic Permutation Greater Than
Target
// Difficulty: Hard
// Tags: array, string, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
    pub fn lex_palindromic_permutation(s: String, target: String) -> String {
        //
    }
}

```

Ruby Solution:

```

# @param {String} s
# @param {String} target
# @return {String}
def lex_palindromic_permutation(s, target)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param String $s
     * @param String $target
     * @return String
     */
    function lexPalindromicPermutation($s, $target) {

    }
}

```

Dart Solution:

```

class Solution {
    String lexPalindromicPermutation(String s, String target) {

```

```
}
```

```
}
```

Scala Solution:

```
object Solution {  
    def lexPalindromicPermutation(s: String, target: String): String = {  
  
    }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec lex_palindromic_permutation(s :: String.t, target :: String.t) ::  
  String.t  
  def lex_palindromic_permutation(s, target) do  
  
  end  
  end
```

Erlang Solution:

```
-spec lex_palindromic_permutation(S :: unicode:unicode_binary(), Target ::  
  unicode:unicode_binary()) -> unicode:unicode_binary().  
lex_palindromic_permutation(S, Target) ->  
.
```

Racket Solution:

```
(define/contract (lex-palindromic-permutation s target)  
  (-> string? string? string?)  
  )
```