Problem 3720: Lexicographically Smallest Permutation Greater Than Target

Problem Information

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

Problem Description

You are given two strings

s

and

target

, both having length

n

, consisting of lowercase English letters.

Return the

lexicographically smallest

permutation

of

s

that is

strictly

greater than

target

. If no permutation of

s

is lexicographically strictly greater than

target

, return an empty string.

A string

a

is

lexicographically strictly greater

than a string

b

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

a

and

b

differ, string

a

has a letter that appears later in the alphabet than the corresponding letter in

b

.

Example 1:

Input:

s = "abc", target = "bba"

Output:

"bca"

Explanation:

The permutations of

s

(in lexicographical order) are

"abc"

,

"acb"

,

"bac"

,

"bca"

,

"cab"

, and

"cba"

.

The lexicographically smallest permutation that is strictly greater than

target

is

"bca"

.

Example 2:

Input:

s = "leet", target = "code"

Output:

"eelt"

Explanation:

The permutations of

s

(in lexicographical order) are

"eelt"

,

"eetl"

,

"elet"

,

"elte"

,

"etel"

,

"etle"

,

"leet"

,

"lete"

,

"ltee"

,

"teel"

,

"tele"

, and

"tlee"

.

The lexicographically smallest permutation that is strictly greater than

target

is

"eelt"

.

Example 3:

Input:

s = "baba", target = "bbaa"

Output:

""

Explanation:

The permutations of

s

(in lexicographical order) are

"aabb"

,

"abab"

,

"abba"

,

"baab"

,

"baba"

, and

"bbaa"

.

None of them is lexicographically strictly greater than

target

. Therefore, the answer is

""

.

Constraints:

1 <= s.length == target.length <= 300

s

and

target

consist of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string lexGreaterPermutation(string s, string target) {

}
};
```

Java:

```
class Solution {
public String lexGreaterPermutation(String s, String target) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function lexGreaterPermutation(s: string, target: string): string {

};
```

C#:

```
public class Solution {
public string LexGreaterPermutation(string s, string target) {

}
}
```

C:

```
char* lexGreaterPermutation(char* s, char* target) {

}
```

Go:

```
func lexGreaterPermutation(s string, target string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
String lexGreaterPermutation(String s, String target) {
```

```
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Lexicographically Smallest Permutation Greater Than Target
```

```
* Difficulty: Medium
* Tags: string, graph, greedy, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Lexicographically Smallest Permutation Greater Than Target
* Difficulty: Medium
* Tags: string, graph, greedy, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Lexicographically Smallest Permutation Greater Than Target
Difficulty: Medium
Tags: string, graph, greedy, hash

Approach: String manipulation with hash map or two pointers
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Lexicographically Smallest Permutation Greater Than Target
* Difficulty: Medium
* Tags: string, graph, greedy, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Lexicographically Smallest Permutation Greater Than Target
* Difficulty: Medium
* Tags: string, graph, greedy, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function lexGreaterPermutation(s: string, target: string): string {

};
```

C# Solution:

```
/*
* Problem: Lexicographically Smallest Permutation Greater Than Target
* Difficulty: Medium
* Tags: string, graph, greedy, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string LexGreaterPermutation(string s, string target) {

}
}
```

C Solution:

```
/*
* Problem: Lexicographically Smallest Permutation Greater Than Target
* Difficulty: Medium
* Tags: string, graph, greedy, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

char* lexGreaterPermutation(char* s, char* target) {

}
```

Go Solution:

```go
// Problem: Lexicographically Smallest Permutation Greater Than Target
// Difficulty: Medium
// Tags: string, graph, greedy, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func lexGreaterPermutation(s string, target string) string {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun lexGreaterPermutation(s: String, target: String): String {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```rust
// Problem: Lexicographically Smallest Permutation Greater Than Target
// Difficulty: Medium
// Tags: string, graph, greedy, hash
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn lex_greater_permutation(s: String, target: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
String lexGreaterPermutation(String s, String target) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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