Problem 2381: Shifting Letters II

Problem Information

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

Problem Description

You are given a string

s

of lowercase English letters and a 2D integer array

shifts

where

shifts[i] = [start

i

, end

i

, direction

i

]

. For every

i

,

shift

the characters in

s

from the index

start

i

to the index

end

i

(

inclusive

) forward if

direction

i

= 1

, or shift the characters backward if

direction

i

= 0

.

Shifting a character

forward

means replacing it with the

next

letter in the alphabet (wrapping around so that

'z'

becomes

'a'

). Similarly, shifting a character

backward

means replacing it with the

previous

letter in the alphabet (wrapping around so that

'a'

becomes

'z'

).

Return

the final string after all such shifts to

s

are applied

.

Example 1:

Input:

s = "abc", shifts = [[0,1,0],[1,2,1],[0,2,1]]

Output:

"ace"

Explanation:

Firstly, shift the characters from index 0 to index 1 backward. Now s = "zac". Secondly, shift the characters from index 1 to index 2 forward. Now s = "zbd". Finally, shift the characters from index 0 to index 2 forward. Now s = "ace".

Example 2:

Input:

s = "dztz", shifts = [[0,0,0],[1,1,1]]

Output:

"catz"

Explanation:

Firstly, shift the characters from index 0 to index 0 backward. Now s = "cztz". Finally, shift the characters from index 1 to index 1 forward. Now s = "catz".

Constraints:

1 <= s.length, shifts.length <= 5 * 10

4

shifts[i].length == 3

0 <= start

i

<= end

i

< s.length

0 <= direction

i

<= 1

s

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string shiftingLetters(string s, vector<vector<int>>& shifts) {
```

```
}
};
```

Java:

```
class Solution {
public String shiftingLetters(String s, int[][] shifts) {

}
}
```

Python3:

```
class Solution:
def shiftingLetters(self, s: str, shifts: List[List[int]]) -> str:
```

Python:

```
class Solution(object):
def shiftingLetters(self, s, shifts):
"""
:type s: str
:type shifts: List[List[int]]
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number[][]} shifts
* @return {string}
*/
var shiftingLetters = function(s, shifts) {

};
```

TypeScript:

```
function shiftingLetters(s: string, shifts: number[][]): string {
```

```
};
```

C#:

```
public class Solution {
public string ShiftingLetters(string s, int[][] shifts) {

}
}
```

C:

```
char* shiftingLetters(char* s, int** shifts, int shiftsSize, int*
shiftsColSize) {

}
```

Go:

```
func shiftingLetters(s string, shifts [][]int) string {

}
```

Kotlin:

```
class Solution {
fun shiftingLetters(s: String, shifts: Array<IntArray>): String {

}
}
```

Swift:

```
class Solution {
func shiftingLetters(_ s: String, _ shifts: [[Int]]) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn shifting_letters(s: String, shifts: Vec<Vec<i32>>) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer[][]} shifts
# @return {String}
def shifting_letters(s, shifts)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer[][] $shifts
* @return String
*/
function shiftingLetters($s, $shifts) {

}
}
```

Dart:

```
class Solution {
String shiftingLetters(String s, List<List<int>> shifts) {

}
}
```

Scala:

```
object Solution {
def shiftingLetters(s: String, shifts: Array[Array[Int]]): String = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec shifting_letters(s :: String.t, shifts :: [[integer]]) :: String.t
def shifting_letters(s, shifts) do

end
end
```

Erlang:

```
-spec shifting_letters(S :: unicode:unicode_binary(), Shifts ::
[[integer()]]) -> unicode:unicode_binary().
shifting_letters(S, Shifts) ->
.
```

Racket:

```
(define/contract (shifting-letters s shifts)
(-> string? (listof (listof exact-integer?)) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shifting Letters II
* Difficulty: Medium
* Tags: array, string
*
* 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 shiftingLetters(string s, vector<vector<int>>& shifts) {

}
};
```

Java Solution:

```
/**
* Problem: Shifting Letters II
* Difficulty: Medium
* Tags: array, string
*
* 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 shiftingLetters(String s, int[][] shifts) {

}
}
```

Python3 Solution:

```
"""
Problem: Shifting Letters II
Difficulty: Medium
Tags: array, string

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 shiftingLetters(self, s: str, shifts: List[List[int]]) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def shiftingLetters(self, s, shifts):
"""
:type s: str
:type shifts: List[List[int]]
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Shifting Letters II
* Difficulty: Medium
* Tags: array, string
*
* 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 {number[][]} shifts
* @return {string}
*/
var shiftingLetters = function(s, shifts) {

};
```

TypeScript Solution:

```
/**
* Problem: Shifting Letters II
* Difficulty: Medium
* Tags: array, string
*
* 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 shiftingLetters(s: string, shifts: number[][]): string {
```

```
};
```

C# Solution:

```
/*
* Problem: Shifting Letters II
* Difficulty: Medium
* Tags: array, string
*
* 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 ShiftingLetters(string s, int[][] shifts) {

}
}
```

C Solution:

```
/*
* Problem: Shifting Letters II
* Difficulty: Medium
* Tags: array, string
*
* 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* shiftingLetters(char* s, int** shifts, int shiftsSize, int*
shiftsColSize) {

}
```

Go Solution:

```
// Problem: Shifting Letters II
// Difficulty: Medium
// Tags: array, string
//
// 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 shiftingLetters(s string, shifts [][]int) string {

}
```

Kotlin Solution:

```
class Solution {
fun shiftingLetters(s: String, shifts: Array<IntArray>): String {

}
}
```

Swift Solution:

```
class Solution {
func shiftingLetters(_ s: String, _ shifts: [[Int]]) -> String {

}
}
```

Rust Solution:

```
// Problem: Shifting Letters II
// Difficulty: Medium
// Tags: array, string
//
// 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 shifting_letters(s: String, shifts: Vec<Vec<i32>>) -> String {

}
```

```
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer[][]} shifts
# @return {String}
def shifting_letters(s, shifts)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer[][] $shifts
* @return String
*/
function shiftingLetters($s, $shifts) {

}
}
```

Dart Solution:

```
class Solution {
String shiftingLetters(String s, List<List<int>> shifts) {

}
}
```

Scala Solution:

```
object Solution {
def shiftingLetters(s: String, shifts: Array[Array[Int]]): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec shifting_letters(s :: String.t, shifts :: [[integer]]) :: String.t
def shifting_letters(s, shifts) do

end
end
```

Erlang Solution:

```
-spec shifting_letters(S :: unicode:unicode_binary(), Shifts ::
[[integer()]]) -> unicode:unicode_binary().
shifting_letters(S, Shifts) ->
.
```

Racket Solution:

```
(define/contract (shifting-letters s shifts)
(-> string? (listof (listof exact-integer?)) string?)
)
```