Problem 848: Shifting Letters

Problem Information

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

Problem Description

You are given a string

s

of lowercase English letters and an integer array

shifts

of the same length.

Call the

shift()

of a letter, the next letter in the alphabet, (wrapping around so that

'z'

becomes

'a'

).

For example,

shift('a') = 'b'

,

shift('t') = 'u'

, and

shift('z') = 'a'

.

Now for each

shifts[i] = x

, we want to shift the first

i + 1

letters of

s

,

x

times.

Return

the final string after all such shifts to s are applied

.

Example 1:

Input:

s = "abc", shifts = [3,5,9]

Output:

"rpl"

Explanation:

We start with "abc". After shifting the first 1 letters of s by 3, we have "dbc". After shifting the first 2 letters of s by 5, we have "igc". After shifting the first 3 letters of s by 9, we have "rpl", the answer.

Example 2:

Input:

s = "aaa", shifts = [1,2,3]

Output:

"gfd"

Constraints:

1 <= s.length <= 10

5

s

consists of lowercase English letters.

shifts.length == s.length

0 <= shifts[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def shiftingLetters(self, s, shifts):
"""
:type s: str
:type shifts: 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) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Scala:

```
object Solution {
def shiftingLetters(s: String, shifts: 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 exact-integer?) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shifting Letters
* 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<int>& shifts) {

}
};
```

Java Solution:

```
/**
* Problem: Shifting Letters
* 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
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[int]) -> str:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shifting Letters
* 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
* 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
* 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
* 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) {

}
```

Go Solution:

```
// Problem: Shifting Letters
// 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: IntArray): String {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shifting Letters
// 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<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<int> shifts) {

}
}
```

Scala Solution:

```
object Solution {
def shiftingLetters(s: String, shifts: 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 exact-integer?) string?)
)
```