Problem 3361: Shift Distance Between Two Strings

Problem Information

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

Problem Description

You are given two strings

s

and

t

of the same length, and two integer arrays

nextCost

and

previousCost

.

In one operation, you can pick any index

i

of

s

, and perform

either one

of the following actions:

Shift

s[i]

to the next letter in the alphabet. If

s[i] == 'z'

, you should replace it with

'a'

. This operation costs

nextCost[j]

where

j

is the index of

s[i]

in the alphabet.

Shift

s[i]

to the previous letter in the alphabet. If

s[i] == 'a'

, you should replace it with

'z'

. This operation costs

previousCost[j]

where

j

is the index of

s[i]

in the alphabet.

The

shift distance

is the

minimum

total cost of operations required to transform

s

into

t

.

Return the

shift distance

from

s

to

t

.

Example 1:

Input:

s = "abab", t = "baba", nextCost = [100,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], previousCost = [1,100,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

Output:

2

Explanation:

We choose index

i = 0

and shift

s[0]

25 times to the previous character for a total cost of 1.

We choose index

i = 1

and shift

s[1]

25 times to the next character for a total cost of 0.

We choose index

i = 2

and shift

s[2]

25 times to the previous character for a total cost of 1.

We choose index

i = 3

and shift

s[3]

25 times to the next character for a total cost of 0.

Example 2:

Input:

s = "leet", t = "code", nextCost = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1], previousCost = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]

Output:

31

Explanation:

We choose index

i = 0

and shift

s[0]

9 times to the previous character for a total cost of 9.

We choose index

i = 1

and shift

s[1]

10 times to the next character for a total cost of 10.

We choose index

i = 2

and shift

s[2]

1 time to the previous character for a total cost of 1.

We choose index

i = 3

and shift

s[3]

11 times to the next character for a total cost of 11.

Constraints:

1 <= s.length == t.length <= 10

5

s

and

t

consist only of lowercase English letters.

nextCost.length == previousCost.length == 26

0 <= nextCost[i], previousCost[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long shiftDistance(string s, string t, vector<int>& nextCost,
vector<int>& previousCost) {

}
};
```

Java:

```
class Solution {
public long shiftDistance(String s, String t, int[] nextCost, int[]
previousCost) {
```

```
}
}
```

Python3:

```
class Solution:
def shiftDistance(self, s: str, t: str, nextCost: List[int], previousCost:
List[int]) -> int:
```

Python:

```
class Solution(object):
def shiftDistance(self, s, t, nextCost, previousCost):
"""
:type s: str
:type t: str
:type nextCost: List[int]
:type previousCost: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {string} t
* @param {number[]} nextCost
* @param {number[]} previousCost
* @return {number}
*/
var shiftDistance = function(s, t, nextCost, previousCost) {

};
```

TypeScript:

```
function shiftDistance(s: string, t: string, nextCost: number[],
previousCost: number[]): number {

};
```

C#:

```
public class Solution {
public long ShiftDistance(string s, string t, int[] nextCost, int[]
previousCost) {

}
}
```

C:

```
long long shiftDistance(char* s, char* t, int* nextCost, int nextCostSize,
int* previousCost, int previousCostSize) {

}
```

Go:

```
func shiftDistance(s string, t string, nextCost []int, previousCost []int)
int64 {

}
```

Kotlin:

```
class Solution {
fun shiftDistance(s: String, t: String, nextCost: IntArray, previousCost:
IntArray): Long {

}
}
```

Swift:

```
class Solution {
func shiftDistance(_ s: String, _ t: String, _ nextCost: [Int], _
previousCost: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn shift_distance(s: String, t: String, next_cost: Vec<i32>,
previous_cost: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} t
# @param {Integer[]} next_cost
# @param {Integer[]} previous_cost
# @return {Integer}
def shift_distance(s, t, next_cost, previous_cost)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $t
* @param Integer[] $nextCost
* @param Integer[] $previousCost
* @return Integer
*/
function shiftDistance($s, $t, $nextCost, $previousCost) {

}
}
```

Dart:

```
class Solution {
int shiftDistance(String s, String t, List<int> nextCost, List<int>
previousCost) {

}
}
```

Scala:

```
object Solution {
def shiftDistance(s: String, t: String, nextCost: Array[Int], previousCost:
Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec shift_distance(s :: String.t, t :: String.t, next_cost :: [integer],
previous_cost :: [integer]) :: integer
def shift_distance(s, t, next_cost, previous_cost) do

end
end
```

Erlang:

```
-spec shift_distance(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary(), NextCost :: [integer()], PreviousCost ::
[integer()]) -> integer().
shift_distance(S, T, NextCost, PreviousCost) ->
.
```

Racket:

```
(define/contract (shift-distance s t nextCost previousCost)
(-> string? string? (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shift Distance Between Two Strings
* 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:
long long shiftDistance(string s, string t, vector<int>& nextCost,
vector<int>& previousCost) {

}
};
```

Java Solution:

```
/**
* Problem: Shift Distance Between Two Strings
* 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 long shiftDistance(String s, String t, int[] nextCost, int[]
previousCost) {

}
}
```

Python3 Solution:

```
"""
Problem: Shift Distance Between Two Strings
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 shiftDistance(self, s: str, t: str, nextCost: List[int], previousCost:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def shiftDistance(self, s, t, nextCost, previousCost):
"""
:type s: str
:type t: str
:type nextCost: List[int]
:type previousCost: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Shift Distance Between Two Strings
* 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 {string} t
* @param {number[]} nextCost
* @param {number[]} previousCost
* @return {number}
*/
```

```
var shiftDistance = function(s, t, nextCost, previousCost) {

};
```

TypeScript Solution:

```
/**
* Problem: Shift Distance Between Two Strings
* 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 shiftDistance(s: string, t: string, nextCost: number[],
previousCost: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Shift Distance Between Two Strings
* 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 long ShiftDistance(string s, string t, int[] nextCost, int[]
previousCost) {

}
}
```

C Solution:

```
/*
* Problem: Shift Distance Between Two Strings
* 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
*/

long long shiftDistance(char* s, char* t, int* nextCost, int nextCostSize,
int* previousCost, int previousCostSize) {

}
```

Go Solution:

```
// Problem: Shift Distance Between Two Strings
// 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 shiftDistance(s string, t string, nextCost []int, previousCost []int)
int64 {

}
```

Kotlin Solution:

```
class Solution {
fun shiftDistance(s: String, t: String, nextCost: IntArray, previousCost:
IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func shiftDistance(_ s: String, _ t: String, _ nextCost: [Int], _
previousCost: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Shift Distance Between Two Strings
// 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 shift_distance(s: String, t: String, next_cost: Vec<i32>,
previous_cost: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} t
# @param {Integer[]} next_cost
# @param {Integer[]} previous_cost
# @return {Integer}
def shift_distance(s, t, next_cost, previous_cost)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $t
```

```
* @param Integer[] $nextCost
* @param Integer[] $previousCost
* @return Integer
*/
function shiftDistance($s, $t, $nextCost, $previousCost) {

}
}
```

Dart Solution:

```
class Solution {
int shiftDistance(String s, String t, List<int> nextCost, List<int>
previousCost) {

}
}
```

Scala Solution:

```
object Solution {
def shiftDistance(s: String, t: String, nextCost: Array[Int], previousCost:
Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec shift_distance(s :: String.t, t :: String.t, next_cost :: [integer],
previous_cost :: [integer]) :: integer
def shift_distance(s, t, next_cost, previous_cost) do

end
end
```

Erlang Solution:

```
-spec shift_distance(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary(), NextCost :: [integer()], PreviousCost ::
```

```
[integer()]) -> integer().
shift_distance(S, T, NextCost, PreviousCost) ->
.
```

Racket Solution:

```
(define/contract (shift-distance s t nextCost previousCost)
(-> string? string? (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```