Problem 2977: Minimum Cost to Convert String II

Problem Information

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

Problem Description

You are given two

0-indexed

strings

source

and

target

, both of length

n

and consisting of

lowercase

English characters. You are also given two

0-indexed

string arrays

original

and

changed

, and an integer array

cost

, where

cost[i]

represents the cost of converting the string

original[i]

to the string

changed[i]

.

You start with the string

source

. In one operation, you can pick a

substring

x

from the string, and change it to

y

at a cost of

z

if

there exists

any

index

j

such that

cost[j] == z

,

original[j] == x

, and

changed[j] == y

. You are allowed to do

any

number of operations, but any pair of operations must satisfy

either

of these two conditions:

The substrings picked in the operations are

source[a..b]

and

source[c..d]

with either

b < c

or

d < a

. In other words, the indices picked in both operations are

disjoint

.

The substrings picked in the operations are

source[a..b]

and

source[c..d]

with

a == c

and

b == d

. In other words, the indices picked in both operations are

identical

.

Return

the

minimum

cost to convert the string

source

to the string

target

using

any

number of operations

.

If it is impossible to convert

source

to

target

,

return

-1

.

Note

that there may exist indices

i

,

j

such that

original[j] == original[i]

and

changed[j] == changed[i]

.

Example 1:

Input:

source = "abcd", target = "acbe", original = ["a","b","c","c","e","d"], changed = ["b","c","b","e","b","e"], cost = [2,5,5,1,2,20]

Output:

28

Explanation:

To convert "abcd" to "acbe", do the following operations: - Change substring source[1..1] from "b" to "c" at a cost of 5. - Change substring source[2..2] from "c" to "e" at a cost of 1. - Change substring source[2..2] from "e" to "b" at a cost of 2. - Change substring source[3..3] from "d" to "e" at a cost of 20. The total cost incurred is 5 + 1 + 2 + 20 = 28. It can be shown that this is the minimum possible cost.

Example 2:

Input:

source = "abcdefgh", target = "acdeeghh", original = ["bcd","fgh","thh"], changed = ["cde","thh","ghh"], cost = [1,3,5]

Output:

9

Explanation:

To convert "abcdefgh" to "acdeeghh", do the following operations: - Change substring source[1..3] from "bcd" to "cde" at a cost of 1. - Change substring source[5..7] from "fgh" to "thh" at a cost of 3. We can do this operation because indices [5,7] are disjoint with indices picked in the first operation. - Change substring source[5..7] from "thh" to "ghh" at a cost of 5. We can do this operation because indices [5,7] are disjoint with indices picked in the first operation, and identical with indices picked in the second operation. The total cost incurred is 1 + 3 + 5 = 9. It can be shown that this is the minimum possible cost.

Example 3:

Input:

source = "abcdefgh", target = "addddddd", original = ["bcd","defgh"], changed = ["ddd","ddddd"], cost = [100,1578]

Output:

-1

Explanation:

It is impossible to convert "abcdefgh" to "addddddd". If you select substring source[1..3] as the first operation to change "abcdefgh" to "adddefgh", you cannot select substring source[3..7] as the second operation because it has a common index, 3, with the first operation. If you select substring source[3..7] as the first operation to change "abcdefgh" to "abcddddd", you cannot select substring source[1..3] as the second operation because it has a common index, 3, with the first operation.

Constraints:

1 <= source.length == target.length <= 1000

source

,

target

consist only of lowercase English characters.

1 <= cost.length == original.length == changed.length <= 100

1 <= original[i].length == changed[i].length <= source.length

original[i]

,

changed[i]

consist only of lowercase English characters.

original[i] != changed[i]

1 <= cost[i] <= 10

6

Code Snippets

C++:

```
class Solution {
public:
long long minimumCost(string source, string target, vector<string>& original,
vector<string>& changed, vector<int>& cost) {
```

```
}
};
```

Java:

```
class Solution {
public long minimumCost(String source, String target, String[] original,
String[] changed, int[] cost) {

}
}
```

Python3:

```
class Solution:
def minimumCost(self, source: str, target: str, original: List[str], changed:
List[str], cost: List[int]) -> int:
```

Python:

```
class Solution(object):
def minimumCost(self, source, target, original, changed, cost):
"""
:type source: str
:type target: str
:type original: List[str]
:type changed: List[str]
:type cost: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} source
* @param {string} target
* @param {string[]} original
* @param {string[]} changed
* @param {number[]} cost
* @return {number}
*/
```

```
var minimumCost = function(source, target, original, changed, cost) {

};
```

TypeScript:

```
function minimumCost(source: string, target: string, original: string[],
changed: string[], cost: number[]): number {

};
```

C#:

```
public class Solution {
public long MinimumCost(string source, string target, string[] original,
string[] changed, int[] cost) {

}
}
```

C:

```
long long minimumCost(char* source, char* target, char** original, int
originalSize, char** changed, int changedSize, int* cost, int costSize) {

}
```

Go:

```
func minimumCost(source string, target string, original []string, changed
[]string, cost []int) int64 {

}
```

Kotlin:

```
class Solution {
fun minimumCost(source: String, target: String, original: Array<String>,
changed: Array<String>, cost: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func minimumCost(_ source: String, _ target: String, _ original: [String], _
changed: [String], _ cost: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_cost(source: String, target: String, original: Vec<String>,
changed: Vec<String>, cost: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {String} source
# @param {String} target
# @param {String[]} original
# @param {String[]} changed
# @param {Integer[]} cost
# @return {Integer}
def minimum_cost(source, target, original, changed, cost)

end
```

PHP:

```
class Solution {

/**
* @param String $source
* @param String $target
* @param String[] $original
* @param String[] $changed
* @param Integer[] $cost
* @return Integer
*/
function minimumCost($source, $target, $original, $changed, $cost) {
```

```
}
}
```

Dart:

```
class Solution {
int minimumCost(String source, String target, List<String> original,
List<String> changed, List<int> cost) {

}
}
```

Scala:

```
object Solution {
def minimumCost(source: String, target: String, original: Array[String],
changed: Array[String], cost: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_cost(source :: String.t, target :: String.t, original ::
[String.t], changed :: [String.t], cost :: [integer]) :: integer
def minimum_cost(source, target, original, changed, cost) do

end
end
```

Erlang:

```
-spec minimum_cost(Source :: unicode:unicode_binary(), Target ::
unicode:unicode_binary(), Original :: [unicode:unicode_binary()], Changed ::
[unicode:unicode_binary()], Cost :: [integer()]) -> integer().
minimum_cost(Source, Target, Original, Changed, Cost) ->
.
```

Racket:

```
(define/contract (minimum-cost source target original changed cost)
(-> string? string? (listof string?) (listof string?) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost to Convert String II
* Difficulty: Hard
* Tags: array, string, tree, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
long long minimumCost(string source, string target, vector<string>& original,
vector<string>& changed, vector<int>& cost) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost to Convert String II
* Difficulty: Hard
* Tags: array, string, tree, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public long minimumCost(String source, String target, String[] original,
```

```
String[] changed, int[] cost) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost to Convert String II
Difficulty: Hard
Tags: array, string, tree, graph, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minimumCost(self, source: str, target: str, original: List[str], changed:
List[str], cost: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumCost(self, source, target, original, changed, cost):
"""
:type source: str
:type target: str
:type original: List[str]
:type changed: List[str]
:type cost: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cost to Convert String II
* Difficulty: Hard
* Tags: array, string, tree, graph, dp
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} source
* @param {string} target
* @param {string[]} original
* @param {string[]} changed
* @param {number[]} cost
* @return {number}
*/
var minimumCost = function(source, target, original, changed, cost) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost to Convert String II
* Difficulty: Hard
* Tags: array, string, tree, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimumCost(source: string, target: string, original: string[],
changed: string[], cost: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost to Convert String II
* Difficulty: Hard
* Tags: array, string, tree, graph, dp
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public long MinimumCost(string source, string target, string[] original,
string[] changed, int[] cost) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost to Convert String II
* Difficulty: Hard
* Tags: array, string, tree, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

long long minimumCost(char* source, char* target, char** original, int
originalSize, char** changed, int changedSize, int* cost, int costSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost to Convert String II
// Difficulty: Hard
// Tags: array, string, tree, graph, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimumCost(source string, target string, original []string, changed
```

```
[]string, cost []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minimumCost(source: String, target: String, original: Array<String>,
changed: Array<String>, cost: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func minimumCost(_ source: String, _ target: String, _ original: [String], _
changed: [String], _ cost: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost to Convert String II
// Difficulty: Hard
// Tags: array, string, tree, graph, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn minimum_cost(source: String, target: String, original: Vec<String>,
changed: Vec<String>, cost: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {String} source
# @param {String} target
# @param {String[]} original
# @param {String[]} changed
# @param {Integer[]} cost
# @return {Integer}
def minimum_cost(source, target, original, changed, cost)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $source
* @param String $target
* @param String[] $original
* @param String[] $changed
* @param Integer[] $cost
* @return Integer
*/
function minimumCost($source, $target, $original, $changed, $cost) {

}
}
```

Dart Solution:

```
class Solution {
int minimumCost(String source, String target, List<String> original,
List<String> changed, List<int> cost) {

}
}
```

Scala Solution:

```
object Solution {
def minimumCost(source: String, target: String, original: Array[String],
changed: Array[String], cost: Array[Int]): Long = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_cost(source :: String.t, target :: String.t, original ::
[String.t], changed :: [String.t], cost :: [integer]) :: integer
def minimum_cost(source, target, original, changed, cost) do

end
end
```

Erlang Solution:

```
-spec minimum_cost(Source :: unicode:unicode_binary(), Target ::
unicode:unicode_binary(), Original :: [unicode:unicode_binary()], Changed ::
[unicode:unicode_binary()], Cost :: [integer()]) -> integer().
minimum_cost(Source, Target, Original, Changed, Cost) ->
.
```

Racket Solution:

```
(define/contract (minimum-cost source target original changed cost)
(-> string? string? (listof string?) (listof string?) (listof exact-integer?)
exact-integer?)
)
```