Problem 1208: Get Equal Substrings Within Budget

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 an integer

maxCost

.

You want to change

s

to

t

. Changing the

i

th

character of

s

to

i

th

character of

t

costs

|s[i] - t[i]|

(i.e., the absolute difference between the ASCII values of the characters).

Return

the maximum length of a substring of

s

that can be changed to be the same as the corresponding substring of

t

with a cost less than or equal to

maxCost

. If there is no substring from

s

that can be changed to its corresponding substring from

t

, return

0

.

Example 1:

Input:

s = "abcd", t = "bcdf", maxCost = 3

Output:

3

Explanation:

"abc" of s can change to "bcd". That costs 3, so the maximum length is 3.

Example 2:

Input:

s = "abcd", t = "cdef", maxCost = 3

Output:

1

Explanation:

Each character in s costs 2 to change to character in t, so the maximum length is 1.

Example 3:

Input:

s = "abcd", t = "acde", maxCost = 0

Output:

1

Explanation:

You cannot make any change, so the maximum length is 1.

Constraints:

1 <= s.length <= 10

5

t.length == s.length

0 <= maxCost <= 10

6

s

and

t

consist of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int equalSubstring(string s, string t, int maxCost) {

}
};
```

Java:

```
class Solution {
public int equalSubstring(String s, String t, int maxCost) {

}
}
```

Python3:

```
class Solution:
def equalSubstring(self, s: str, t: str, maxCost: int) -> int:
```

Python:

```
class Solution(object):
def equalSubstring(self, s, t, maxCost):
"""
:type s: str
:type t: str
:type maxCost: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {string} t
* @param {number} maxCost
* @return {number}
*/
var equalSubstring = function(s, t, maxCost) {

};
```

TypeScript:

```
function equalSubstring(s: string, t: string, maxCost: number): number {

};
```

C#:

```
public class Solution {
public int EqualSubstring(string s, string t, int maxCost) {

}
}
```

C:

```
int equalSubstring(char* s, char* t, int maxCost) {

}
```

Go:

```
func equalSubstring(s string, t string, maxCost int) int {

}
```

Kotlin:

```
class Solution {
fun equalSubstring(s: String, t: String, maxCost: Int): Int {

}
}
```

Swift:

```
class Solution {
func equalSubstring(_ s: String, _ t: String, _ maxCost: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn equal_substring(s: String, t: String, max_cost: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} t
# @param {Integer} max_cost
# @return {Integer}
def equal_substring(s, t, max_cost)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $t
* @param Integer $maxCost
* @return Integer
*/
function equalSubstring($s, $t, $maxCost) {

}
}
```

Dart:

```
class Solution {
int equalSubstring(String s, String t, int maxCost) {

}
}
```

Scala:

```
object Solution {
def equalSubstring(s: String, t: String, maxCost: Int): Int = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec equal_substring(s :: String.t, t :: String.t, max_cost :: integer) ::
integer
def equal_substring(s, t, max_cost) do

end
end
```

Erlang:

```
-spec equal_substring(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary(), MaxCost :: integer()) -> integer().
equal_substring(S, T, MaxCost) ->
.
```

Racket:

```
(define/contract (equal-substring s t maxCost)
(-> string? string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Get Equal Substrings Within Budget
* Difficulty: Medium
* Tags: array, string, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
class Solution {
public:
int equalSubstring(string s, string t, int maxCost) {

}
};
```

Java Solution:

```
/**
* Problem: Get Equal Substrings Within Budget
* Difficulty: Medium
* Tags: array, string, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int equalSubstring(String s, String t, int maxCost) {

}
}
```

Python3 Solution:

```
"""
Problem: Get Equal Substrings Within Budget
Difficulty: Medium
Tags: array, string, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def equalSubstring(self, s: str, t: str, maxCost: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def equalSubstring(self, s, t, maxCost):
"""
:type s: str
:type t: str
:type maxCost: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Get Equal Substrings Within Budget
* Difficulty: Medium
* Tags: array, string, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} s
* @param {string} t
* @param {number} maxCost
* @return {number}
*/
var equalSubstring = function(s, t, maxCost) {

};
```

TypeScript Solution:

```
/**
* Problem: Get Equal Substrings Within Budget
* Difficulty: Medium
* Tags: array, string, tree, search
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function equalSubstring(s: string, t: string, maxCost: number): number {

};
```

C# Solution:

```
/*
* Problem: Get Equal Substrings Within Budget
* Difficulty: Medium
* Tags: array, string, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int EqualSubstring(string s, string t, int maxCost) {

}
}
```

C Solution:

```
/*
* Problem: Get Equal Substrings Within Budget
* Difficulty: Medium
* Tags: array, string, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int equalSubstring(char* s, char* t, int maxCost) {
```

```
}
```

Go Solution:

```
// Problem: Get Equal Substrings Within Budget
// Difficulty: Medium
// Tags: array, string, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func equalSubstring(s string, t string, maxCost int) int {

}
```

Kotlin Solution:

```
class Solution {
fun equalSubstring(s: String, t: String, maxCost: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func equalSubstring(_ s: String, _ t: String, _ maxCost: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Get Equal Substrings Within Budget
// Difficulty: Medium
// Tags: array, string, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn equal_substring(s: String, t: String, max_cost: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} t
# @param {Integer} max_cost
# @return {Integer}
def equal_substring(s, t, max_cost)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $t
* @param Integer $maxCost
* @return Integer
*/
function equalSubstring($s, $t, $maxCost) {

}
}
```

Dart Solution:

```
class Solution {
int equalSubstring(String s, String t, int maxCost) {

}
}
```

Scala Solution:

```
object Solution {
def equalSubstring(s: String, t: String, maxCost: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec equal_substring(s :: String.t, t :: String.t, max_cost :: integer) ::
integer
def equal_substring(s, t, max_cost) do

end
end
```

Erlang Solution:

```
-spec equal_substring(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary(), MaxCost :: integer()) -> integer().
equal_substring(S, T, MaxCost) ->
.
```

Racket Solution:

```
(define/contract (equal-substring s t maxCost)
(-> string? string? exact-integer? exact-integer?)
)
```