Problem 936: Stamping The Sequence

Problem Information

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

Problem Description

You are given two strings

stamp

and

target

. Initially, there is a string

s

of length

target.length

with all

s[i] == '?'

.

In one turn, you can place

stamp

over

s

and replace every letter in the

s

with the corresponding letter from

stamp

.

For example, if

stamp = "abc"

and

target = "abcba"

, then

s

is

"?????"

initially. In one turn you can:

place

stamp

at index

0

of

s

to obtain

"abc??"

,

place

stamp

at index

1

of

s

to obtain

"?abc?"

, or

place

stamp

at index

2

of

s

to obtain

"??abc"

.

Note that

stamp

must be fully contained in the boundaries of

s

in order to stamp (i.e., you cannot place

stamp

at index

3

of

s

).

We want to convert

s

to

target

using

at most

10 * target.length

turns.

Return

an array of the index of the left-most letter being stamped at each turn

. If we cannot obtain

target

from

s

within

10 * target.length

turns, return an empty array.

Example 1:

Input:

stamp = "abc", target = "ababc"

Output:

[0,2]

Explanation:

Initially s = "?????". - Place stamp at index 0 to get "abc??". - Place stamp at index 2 to get "ababc". [1,0,2] would also be accepted as an answer, as well as some other answers.

Example 2:

Input:

stamp = "abca", target = "aabcaca"

Output:

[3,0,1]

Explanation:

Initially s = "???????". - Place stamp at index 3 to get "???abca". - Place stamp at index 0 to get "abcabca". - Place stamp at index 1 to get "aabcaca".

Constraints:

1 <= stamp.length <= target.length <= 1000

stamp

and

target

consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<int> movesToStamp(string stamp, string target) {

}
};
```

Java:

```
class Solution {
public int[] movesToStamp(String stamp, String target) {

}
}
```

Python3:

```
class Solution:
def movesToStamp(self, stamp: str, target: str) -> List[int]:
```

Python:

```
class Solution(object):
def movesToStamp(self, stamp, target):
"""
:type stamp: str
:type target: str
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {string} stamp
* @param {string} target
* @return {number[]}
*/
var movesToStamp = function(stamp, target) {

};
```

TypeScript:

```
function movesToStamp(stamp: string, target: string): number[] {

};
```

C#:

```
public class Solution {
public int[] MovesToStamp(string stamp, string target) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* movesToStamp(char* stamp, char* target, int* returnSize) {

}
```

Go:

```
func movesToStamp(stamp string, target string) []int {

}
```

Kotlin:

```
class Solution {
fun movesToStamp(stamp: String, target: String): IntArray {

}
}
```

Swift:

```
class Solution {
func movesToStamp(_ stamp: String, _ target: String) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn moves_to_stamp(stamp: String, target: String) -> Vec<i32> {

}
```

```
}
```

Ruby:

```
# @param {String} stamp
# @param {String} target
# @return {Integer[]}
def moves_to_stamp(stamp, target)

end
```

PHP:

```
class Solution {

/**
* @param String $stamp
* @param String $target
* @return Integer[]
*/
function movesToStamp($stamp, $target) {

}
}
```

Dart:

```
class Solution {
List<int> movesToStamp(String stamp, String target) {

}
}
```

Scala:

```
object Solution {
def movesToStamp(stamp: String, target: String): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec moves_to_stamp(stamp :: String.t, target :: String.t) :: [integer]
def moves_to_stamp(stamp, target) do

end
end
```

Erlang:

```
-spec moves_to_stamp(Stamp :: unicode:unicode_binary(), Target ::
unicode:unicode_binary()) -> [integer()].
moves_to_stamp(Stamp, Target) ->
.
```

Racket:

```
(define/contract (moves-to-stamp stamp target)
(-> string? string? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Stamping The Sequence
* Difficulty: Hard
* Tags: array, string, greedy, stack, queue
*
* 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:
vector<int> movesToStamp(string stamp, string target) {

}
};
```

Java Solution:

```
/**
* Problem: Stamping The Sequence
* Difficulty: Hard
* Tags: array, string, greedy, stack, queue
*
* 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 int[] movesToStamp(String stamp, String target) {

}
}
```

Python3 Solution:

```
"""
Problem: Stamping The Sequence
Difficulty: Hard
Tags: array, string, greedy, stack, queue

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 movesToStamp(self, stamp: str, target: str) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def movesToStamp(self, stamp, target):
"""
:type stamp: str
:type target: str
:rtype: List[int]
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Stamping The Sequence
* Difficulty: Hard
* Tags: array, string, greedy, stack, queue
*
* 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} stamp
* @param {string} target
* @return {number[]}
*/
var movesToStamp = function(stamp, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Stamping The Sequence
* Difficulty: Hard
* Tags: array, string, greedy, stack, queue
*
* 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 movesToStamp(stamp: string, target: string): number[] {

};
```

C# Solution:

```
/*
* Problem: Stamping The Sequence
* Difficulty: Hard
* Tags: array, string, greedy, stack, queue
*
* 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 int[] MovesToStamp(string stamp, string target) {

}
}
```

C Solution:

```
/*
* Problem: Stamping The Sequence
* Difficulty: Hard
* Tags: array, string, greedy, stack, queue
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* movesToStamp(char* stamp, char* target, int* returnSize) {

}
```

Go Solution:

```
// Problem: Stamping The Sequence
// Difficulty: Hard
// Tags: array, string, greedy, stack, queue
//
// 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 movesToStamp(stamp string, target string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun movesToStamp(stamp: String, target: String): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func movesToStamp(_ stamp: String, _ target: String) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Stamping The Sequence
// Difficulty: Hard
// Tags: array, string, greedy, stack, queue
//
// 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 moves_to_stamp(stamp: String, target: String) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String} stamp
# @param {String} target
# @return {Integer[]}
def moves_to_stamp(stamp, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $stamp
* @param String $target
* @return Integer[]
*/
function movesToStamp($stamp, $target) {

}
}
```

Dart Solution:

```
class Solution {
List<int> movesToStamp(String stamp, String target) {

}
}
```

Scala Solution:

```
object Solution {
def movesToStamp(stamp: String, target: String): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec moves_to_stamp(stamp :: String.t, target :: String.t) :: [integer]
def moves_to_stamp(stamp, target) do
```

```
end
end
```

Erlang Solution:

```
-spec moves_to_stamp(Stamp :: unicode:unicode_binary(), Target ::
unicode:unicode_binary()) -> [integer()].
moves_to_stamp(Stamp, Target) ->
.
```

Racket Solution:

```
(define/contract (moves-to-stamp stamp target)
(-> string? string? (listof exact-integer?))
)
```