Problem 833: Find And Replace in String

Problem Information

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

Problem Description

You are given a

0-indexed

string

s

that you must perform

k

replacement operations on. The replacement operations are given as three

0-indexed

parallel arrays,

indices

,

sources

, and

targets

, all of length

k

.

To complete the

i

th

replacement operation:

Check if the

substring

sources[i]

occurs at index

indices[i]

in the

original string

s

.

If it does not occur,

do nothing

.

Otherwise if it does occur,

replace

that substring with

targets[i]

.

For example, if

s = "

ab

cd"

,

indices[i] = 0

,

sources[i] = "ab"

, and

targets[i] = "eee"

, then the result of this replacement will be

"

eee

cd"

.

All replacement operations must occur

simultaneously

, meaning the replacement operations should not affect the indexing of each other. The testcases will be generated such that the replacements will

not overlap

.

For example, a testcase with

s = "abc"

,

indices = [0, 1]

, and

sources = ["ab","bc"]

will not be generated because the

"ab"

and

"bc"

replacements overlap.

Return

the

resulting string

after performing all replacement operations on

s

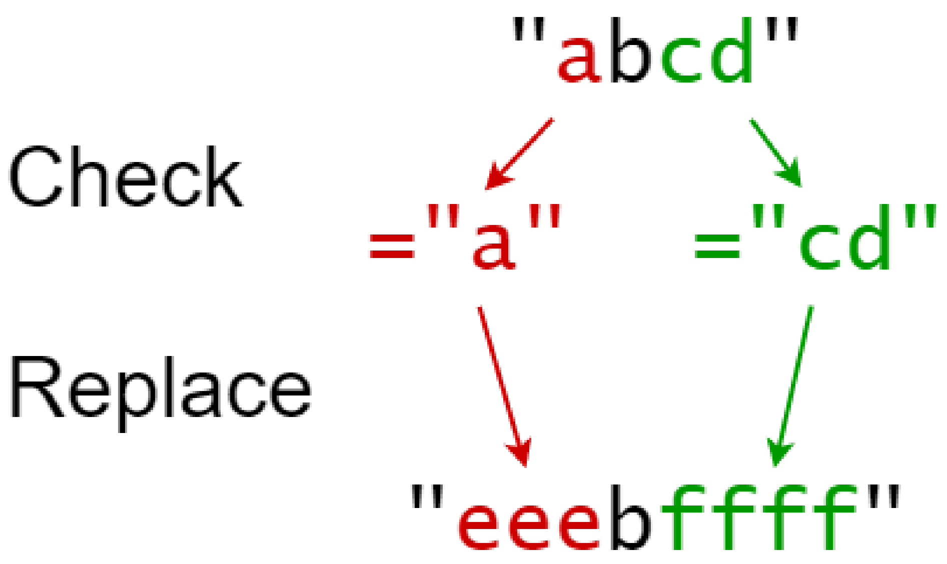
.

A

substring

is a contiguous sequence of characters in a string.

Example 1:



Input:

s = "abcd", indices = [0, 2], sources = ["a", "cd"], targets = ["eee", "ffff"]

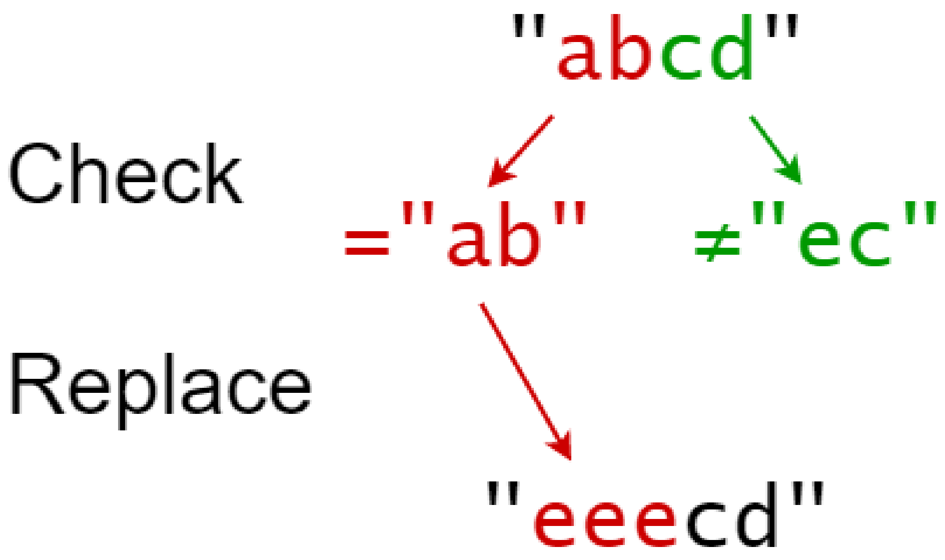
Output:

"eeebffff"

Explanation:

"a" occurs at index 0 in s, so we replace it with "eee". "cd" occurs at index 2 in s, so we replace it with "ffff".

Example 2:



Input:

s = "abcd", indices = [0, 2], sources = ["ab","ec"], targets = ["eee","ffff"]

Output:

"eeecd"

Explanation:

"ab" occurs at index 0 in s, so we replace it with "eee". "ec" does not occur at index 2 in s, so we do nothing.

Constraints:

1 <= s.length <= 1000

k == indices.length == sources.length == targets.length

1 <= k <= 100

0 <= indexes[i] < s.length

1 <= sources[i].length, targets[i].length <= 50

s

consists of only lowercase English letters.

sources[i]

and

targets[i]

consist of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string findReplaceString(string s, vector<int>& indices, vector<string>&
sources, vector<string>& targets) {

}
};
```

Java:

```
class Solution {
public String findReplaceString(String s, int[] indices, String[] sources,
String[] targets) {

}
}
```

Python3:

```
class Solution:
def findReplaceString(self, s: str, indices: List[int], sources: List[str],
targets: List[str]) -> str:
```

Python:

```
class Solution(object):
def findReplaceString(self, s, indices, sources, targets):
"""
:type s: str
:type indices: List[int]
:type sources: List[str]
:type targets: List[str]
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number[]} indices
* @param {string[]} sources
* @param {string[]} targets
* @return {string}
*/
var findReplaceString = function(s, indices, sources, targets) {

};
```

TypeScript:

```
function findReplaceString(s: string, indices: number[], sources: string[],
targets: string[]): string {

};
```

C#:

```
public class Solution {
public string FindReplaceString(string s, int[] indices, string[] sources,
string[] targets) {

}
}
```

C:

```
char* findReplaceString(char* s, int* indices, int indicesSize, char**
sources, int sourcesSize, char** targets, int targetsSize) {

}
```

Go:

```
func findReplaceString(s string, indices []int, sources []string, targets
[]string) string {

}
```

Kotlin:

```
class Solution {
fun findReplaceString(s: String, indices: IntArray, sources: Array<String>,
targets: Array<String>): String {

}
}
```

Swift:

```
class Solution {
func findReplaceString(_ s: String, _ indices: [Int], _ sources: [String], _
targets: [String]) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn find_replace_string(s: String, indices: Vec<i32>, sources:
Vec<String>, targets: Vec<String>) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer[]} indices
# @param {String[]} sources
# @param {String[]} targets
# @return {String}
def find_replace_string(s, indices, sources, targets)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer[] $indices
* @param String[] $sources
* @param String[] $targets
* @return String
*/
function findReplaceString($s, $indices, $sources, $targets) {

}
}
```

Dart:

```
class Solution {
String findReplaceString(String s, List<int> indices, List<String> sources,
List<String> targets) {

}
}
```

Scala:

```
object Solution {
def findReplaceString(s: String, indices: Array[Int], sources: Array[String],
targets: Array[String]): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_replace_string(s :: String.t, indices :: [integer], sources ::
[String.t], targets :: [String.t]) :: String.t
def find_replace_string(s, indices, sources, targets) do

end
end
```

Erlang:

```
-spec find_replace_string(S :: unicode:unicode_binary(), Indices ::
[integer()], Sources :: [unicode:unicode_binary()], Targets ::
[unicode:unicode_binary()]) -> unicode:unicode_binary().
find_replace_string(S, Indices, Sources, Targets) ->
.
```

Racket:

```
(define/contract (find-replace-string s indices sources targets)
(-> string? (listof exact-integer?) (listof string?) (listof string?)
string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find And Replace in String
* Difficulty: Medium
* Tags: array, string, tree, hash, sort
*
* 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:
string findReplaceString(string s, vector<int>& indices, vector<string>&
sources, vector<string>& targets) {

}
};
```

Java Solution:

```
/**
* Problem: Find And Replace in String
* Difficulty: Medium
* Tags: array, string, tree, hash, sort
*
* 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 String findReplaceString(String s, int[] indices, String[] sources,
String[] targets) {

}
}
```

Python3 Solution:

```
"""
Problem: Find And Replace in String
Difficulty: Medium
Tags: array, string, tree, hash, sort

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 findReplaceString(self, s: str, indices: List[int], sources: List[str],
targets: List[str]) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findReplaceString(self, s, indices, sources, targets):
"""
:type s: str
:type indices: List[int]
:type sources: List[str]
:type targets: List[str]
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Find And Replace in String
* Difficulty: Medium
* Tags: array, string, tree, hash, sort
*
* 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 {number[]} indices
* @param {string[]} sources
* @param {string[]} targets
* @return {string}
*/
var findReplaceString = function(s, indices, sources, targets) {

};
```

TypeScript Solution:

```
/**
* Problem: Find And Replace in String
* Difficulty: Medium
* Tags: array, string, tree, hash, sort
*
* 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 findReplaceString(s: string, indices: number[], sources: string[],
targets: string[]): string {

};
```

C# Solution:

```
/*
* Problem: Find And Replace in String
* Difficulty: Medium
* Tags: array, string, tree, hash, sort
*
* 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 string FindReplaceString(string s, int[] indices, string[] sources,
string[] targets) {

}
}
```

C Solution:

```
/*
* Problem: Find And Replace in String
* Difficulty: Medium
* Tags: array, string, tree, hash, sort
*
* 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
*/

char* findReplaceString(char* s, int* indices, int indicesSize, char**
sources, int sourcesSize, char** targets, int targetsSize) {

}
```

Go Solution:

```
// Problem: Find And Replace in String
// Difficulty: Medium
// Tags: array, string, tree, hash, sort
//
// 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 findReplaceString(s string, indices []int, sources []string, targets
[]string) string {

}
```

Kotlin Solution:

```
class Solution {
fun findReplaceString(s: String, indices: IntArray, sources: Array<String>,
targets: Array<String>): String {

}
}
```

Swift Solution:

```
class Solution {
func findReplaceString(_ s: String, _ indices: [Int], _ sources: [String], _
targets: [String]) -> String {

}
}
```

Rust Solution:

```
// Problem: Find And Replace in String
// Difficulty: Medium
// Tags: array, string, tree, hash, sort
//
// 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 find_replace_string(s: String, indices: Vec<i32>, sources:
Vec<String>, targets: Vec<String>) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer[]} indices
# @param {String[]} sources
# @param {String[]} targets
# @return {String}
def find_replace_string(s, indices, sources, targets)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer[] $indices
* @param String[] $sources
* @param String[] $targets
* @return String
*/
function findReplaceString($s, $indices, $sources, $targets) {

}
}
```

Dart Solution:

```
class Solution {
String findReplaceString(String s, List<int> indices, List<String> sources,
List<String> targets) {

}
}
```

Scala Solution:

```
object Solution {
def findReplaceString(s: String, indices: Array[Int], sources: Array[String],
targets: Array[String]): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_replace_string(s :: String.t, indices :: [integer], sources ::
[String.t], targets :: [String.t]) :: String.t
def find_replace_string(s, indices, sources, targets) do

end
end
```

Erlang Solution:

```
-spec find_replace_string(S :: unicode:unicode_binary(), Indices ::
[integer()], Sources :: [unicode:unicode_binary()], Targets ::
[unicode:unicode_binary()]) -> unicode:unicode_binary().
find_replace_string(S, Indices, Sources, Targets) ->
.
```

Racket Solution:

```
(define/contract (find-replace-string s indices sources targets)
(-> string? (listof exact-integer?) (listof string?) (listof string?)
string?)
)
```