Problem 3481: Apply Substitutions

Problem Information

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

Problem Description

You are given a

replacements

mapping and a

text

string that may contain

placeholders

formatted as

%var%

, where each

var

corresponds to a key in the

replacements

mapping. Each replacement value may itself contain one or more

such

placeholders

. Each

placeholder

is replaced by the value associated with its corresponding replacement key.

Return the fully substituted

text

string which

does not

contain any

placeholders

.

Example 1:

Input:

replacements = [["A","abc"],["B","def"]], text = "%A%_%B%"

Output:

"abc_def"

Explanation:

The mapping associates

"A"

with

"abc"

and

"B"

with

"def"

.

Replace

%A%

with

"abc"

and

%B%

with

"def"

in the text.

The final text becomes

"abc_def"

.

Example 2:

Input:

replacements = [["A","bce"],["B","ace"],["C","abc%B%"]], text = "%A%_%B%_%C%"

Output:

"bce_ace_abcace"

Explanation:

The mapping associates

"A"

with

"bce"

,

"B"

with

"ace"

, and

"C"

with

"abc%B%"

.

Replace

%A%

with

"bce"

and

%B%

with

"ace"

in the text.

Then, for

%C%

, substitute

%B%

in

"abc%B%"

with

"ace"

to obtain

"abcace"

.

The final text becomes

"bce_ace_abcace"

.

Constraints:

1 <= replacements.length <= 10

Each element of

replacements

is a two-element list

[key, value]

, where:

key

is a single uppercase English letter.

value

is a non-empty string of at most 8 characters that may contain zero or more placeholders formatted as

%<key>%

.

All replacement keys are unique.

The

text

string is formed by concatenating all key placeholders (formatted as

%<key>%

) randomly from the replacements mapping, separated by underscores.

text.length == 4 * replacements.length - 1

Every placeholder in the

text

or in any replacement value corresponds to a key in the

replacements

mapping.

There are no cyclic dependencies between replacement keys.

Code Snippets

C++:

```
class Solution {
public:
string applySubstitutions(vector<vector<string>>& replacements, string text)
{

}
};
```

Java:

```
class Solution {
public String applySubstitutions(List<List<String>> replacements, String
text) {



}
}
```

Python3:

```
class Solution:
def applySubstitutions(self, replacements: List[List[str]], text: str) ->
str:
```

Python:

```
class Solution(object):
def applySubstitutions(self, replacements, text):
"""
:type replacements: List[List[str]]
:type text: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string[][]} replacements
* @param {string} text
* @return {string}
*/
var applySubstitutions = function(replacements, text) {

};
```

TypeScript:

```
function applySubstitutions(replacements: string[][], text: string): string {

};
```

C#:

```
public class Solution {
public string ApplySubstitutions(IList<IList<string>> replacements, string
text) {
}
}
```

C:

```
char* applySubstitutions(char*** replacements, int replacementsSize, int*
replacementsColSize, char* text) {

}
```

Go:

```
func applySubstitutions(replacements [][]string, text string) string {

}
```

Kotlin:

```
class Solution {
fun applySubstitutions(replacements: List<List<String>>, text: String):
String {

}
}
```

Swift:

```
class Solution {
func applySubstitutions(_ replacements: [[String]], _ text: String) -> String
{

}
}
```

Rust:

```
impl Solution {
pub fn apply_substitutions(replacements: Vec<Vec<String>>, text: String) ->
String {

}
}
```

Ruby:

```
# @param {String[][]} replacements
# @param {String} text
# @return {String}
def apply_substitutions(replacements, text)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $replacements
* @param String $text
* @return String
*/
function applySubstitutions($replacements, $text) {

}
}
```

Dart:

```
class Solution {
String applySubstitutions(List<List<String>> replacements, String text) {

}
}
```

Scala:

```
object Solution {
def applySubstitutions(replacements: List[List[String]], text: String):
String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec apply_substitutions(replacements :: [[String.t]], text :: String.t) ::
String.t
def apply_substitutions(replacements, text) do

end
end
```

Erlang:

```
-spec apply_substitutions(Replacements :: [[unicode:unicode_binary()]], Text
:: unicode:unicode_binary()) -> unicode:unicode_binary().
apply_substitutions(Replacements, Text) ->
.
```

Racket:

```
(define/contract (apply-substitutions replacements text)
(-> (listof (listof string?)) string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Apply Substitutions
* Difficulty: Medium
* Tags: array, string, tree, graph, hash, sort, 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:
string applySubstitutions(vector<vector<string>>& replacements, string text)
{
}
};
```

Java Solution:

```
/**
* Problem: Apply Substitutions
* Difficulty: Medium
* Tags: array, string, tree, graph, hash, sort, 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 String applySubstitutions(List<List<String>> replacements, String
text) {

}
}
```

Python3 Solution:

```
"""
Problem: Apply Substitutions
Difficulty: Medium
Tags: array, string, tree, graph, hash, sort, 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 applySubstitutions(self, replacements: List[List[str]], text: str) ->
str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def applySubstitutions(self, replacements, text):
"""
:type replacements: List[List[str]]
:type text: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Apply Substitutions
* Difficulty: Medium
* Tags: array, string, tree, graph, hash, sort, 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[][]} replacements
* @param {string} text
* @return {string}
*/
var applySubstitutions = function(replacements, text) {

};
```

TypeScript Solution:

```
/**
* Problem: Apply Substitutions
* Difficulty: Medium
* Tags: array, string, tree, graph, hash, sort, 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 applySubstitutions(replacements: string[][], text: string): string {
};
```

C# Solution:

```
/*
* Problem: Apply Substitutions
* Difficulty: Medium
* Tags: array, string, tree, graph, hash, sort, 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 string ApplySubstitutions(IList<IList<string>> replacements, string
text) {

}
}
```

C Solution:

```
/*
* Problem: Apply Substitutions
* Difficulty: Medium
* Tags: array, string, tree, graph, hash, sort, 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
*/

char* applySubstitutions(char*** replacements, int replacementsSize, int*
replacementsColSize, char* text) {

}
```

Go Solution:

```
// Problem: Apply Substitutions
// Difficulty: Medium
// Tags: array, string, tree, graph, hash, sort, 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 applySubstitutions(replacements [][]string, text string) string {

}
```

Kotlin Solution:

```
class Solution {
fun applySubstitutions(replacements: List<List<String>>, text: String):
String {

}
}
```

Swift Solution:

```
class Solution {
func applySubstitutions(_ replacements: [[String]], _ text: String) -> String
{

}
}
```

Rust Solution:

```
// Problem: Apply Substitutions
// Difficulty: Medium
// Tags: array, string, tree, graph, hash, sort, 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 apply_substitutions(replacements: Vec<Vec<String>>, text: String) ->
String {

}
}
```

Ruby Solution:

```
# @param {String[][]} replacements
# @param {String} text
# @return {String}
def apply_substitutions(replacements, text)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $replacements
* @param String $text
* @return String
*/
function applySubstitutions($replacements, $text) {

}
}
```

Dart Solution:

```
class Solution {
String applySubstitutions(List<List<String>> replacements, String text) {

}
}
```

Scala Solution:

```
object Solution {
def applySubstitutions(replacements: List[List[String]], text: String):
String = {
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec apply_substitutions(replacements :: [[String.t]], text :: String.t) ::
String.t
def apply_substitutions(replacements, text) do

end
end
```

Erlang Solution:

```
-spec apply_substitutions(Replacements :: [[unicode:unicode_binary()]], Text
:: unicode:unicode_binary()) -> unicode:unicode_binary().
apply_substitutions(Replacements, Text) ->
.
```

Racket Solution:

```
(define/contract (apply-substitutions replacements text)
(-> (listof (listof string?)) string? string?)
)
```