Problem 1078: Occurrences After Bigram

Problem Information

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

Problem Description

Given two strings

first

and

second

, consider occurrences in some text of the form

"first second third"

, where

second

comes immediately after

first

, and

third

comes immediately after

second

.

Return

an array of all the words

third

for each occurrence of

"first second third"

.

Example 1:

Input:

text = "alice is a good girl she is a good student", first = "a", second = "good"

Output:

["girl","student"]

Example 2:

Input:

text = "we will we will rock you", first = "we", second = "will"

Output:

["we","rock"]

Constraints:

1 <= text.length <= 1000

text

consists of lowercase English letters and spaces.

All the words in

text

are separated by

a single space

.

1 <= first.length, second.length <= 10

first

and

second

consist of lowercase English letters.

text

will not have any leading or trailing spaces.

Code Snippets

C++:

```
class Solution {
public:
vector<string> findOcurrences(string text, string first, string second) {

}
};
```

Java:

```
class Solution {
public String[] findOcurrences(String text, String first, String second) {

}
}
```

Python3:

```
class Solution:
def findOcurrences(self, text: str, first: str, second: str) -> List[str]:
```

Python:

```
class Solution(object):
def findOcurrences(self, text, first, second):
"""
:type text: str
:type first: str
:type second: str
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string} text
* @param {string} first
* @param {string} second
* @return {string[]}
*/
var findOcurrences = function(text, first, second) {

};
```

TypeScript:

```
function findOcurrences(text: string, first: string, second: string):
string[] {

};
```

C#:

```
public class Solution {
public string[] FindOcurrences(string text, string first, string second) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** findOcurrences(char* text, char* first, char* second, int* returnSize)
{

}
```

Go:

```
func findOcurrences(text string, first string, second string) []string {

}
```

Kotlin:

```
class Solution {
fun findOcurrences(text: String, first: String, second: String):
Array<String> {

}
}
```

Swift:

```
class Solution {
func findOcurrences(_ text: String, _ first: String, _ second: String) ->
[String] {

}
}
```

Rust:

```
impl Solution {
pub fn find_ocurrences(text: String, first: String, second: String) ->
Vec<String> {

}
}
```

Ruby:

```
# @param {String} text
# @param {String} first
# @param {String} second
# @return {String[]}
def find_ocurrences(text, first, second)

end
```

PHP:

```
class Solution {

/**
* @param String $text
* @param String $first
* @param String $second
* @return String[]
*/
function findOcurrences($text, $first, $second) {

}
}
```

Dart:

```
class Solution {
List<String> findOcurrences(String text, String first, String second) {

}
}
```

Scala:

```
object Solution {
def findOcurrences(text: String, first: String, second: String):
Array[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_ocurrences(text :: String.t, first :: String.t, second ::
String.t) :: [String.t]
def find_ocurrences(text, first, second) do

end
end
```

Erlang:

```
-spec find_ocurrences(Text :: unicode:unicode_binary(), First ::
unicode:unicode_binary(), Second :: unicode:unicode_binary()) ->
[unicode:unicode_binary()].
find_ocurrences(Text, First, Second) ->
.
```

Racket:

```
(define/contract (find-ocurrences text first second)
(-> string? string? string? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Occurrences After Bigram
* Difficulty: Easy
* Tags: array, string
*
* 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<string> findOcurrences(string text, string first, string second) {

}
};
```

Java Solution:

```
/**
* Problem: Occurrences After Bigram
* Difficulty: Easy
* Tags: array, string
*
* 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 String[] findOcurrences(String text, String first, String second) {

}
}
```

Python3 Solution:

```
"""
Problem: Occurrences After Bigram
Difficulty: Easy
Tags: array, string

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 findOcurrences(self, text: str, first: str, second: str) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findOcurrences(self, text, first, second):
"""
:type text: str
:type first: str
:type second: str
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Occurrences After Bigram
* Difficulty: Easy
* Tags: array, string
*
* 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} text
* @param {string} first
* @param {string} second
* @return {string[]}
*/
var findOcurrences = function(text, first, second) {

};
```

TypeScript Solution:

```
/**
* Problem: Occurrences After Bigram
* Difficulty: Easy
* Tags: array, string
*
* 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 findOcurrences(text: string, first: string, second: string):
string[] {

};
```

C# Solution:

```
/*
* Problem: Occurrences After Bigram
* Difficulty: Easy
* Tags: array, string
*
* 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 string[] FindOcurrences(string text, string first, string second) {

}
}
```

C Solution:

```
/*
* Problem: Occurrences After Bigram
* Difficulty: Easy
* Tags: array, string
*
* 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().
*/
char** findOcurrences(char* text, char* first, char* second, int* returnSize)
{

}
```

Go Solution:

```
// Problem: Occurrences After Bigram
// Difficulty: Easy
// Tags: array, string
//
// 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 findOcurrences(text string, first string, second string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun findOcurrences(text: String, first: String, second: String):
Array<String> {

}
}
```

Swift Solution:

```
class Solution {
func findOcurrences(_ text: String, _ first: String, _ second: String) ->
[String] {

}
}
```

Rust Solution:

```
// Problem: Occurrences After Bigram
// Difficulty: Easy
// Tags: array, string
//
// 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 find_ocurrences(text: String, first: String, second: String) ->
Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String} text
# @param {String} first
# @param {String} second
# @return {String[]}
def find_ocurrences(text, first, second)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $text
* @param String $first
* @param String $second
* @return String[]
*/
function findOcurrences($text, $first, $second) {

}
}
```

Dart Solution:

```
class Solution {
List<String> findOcurrences(String text, String first, String second) {

}
}
```

Scala Solution:

```
object Solution {
def findOcurrences(text: String, first: String, second: String):
Array[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_ocurrences(text :: String.t, first :: String.t, second ::
String.t) :: [String.t]
def find_ocurrences(text, first, second) do

end
end
```

Erlang Solution:

```
-spec find_ocurrences(Text :: unicode:unicode_binary(), First ::
unicode:unicode_binary(), Second :: unicode:unicode_binary()) ->
[unicode:unicode_binary()].
find_ocurrences(Text, First, Second) ->
.
```

Racket Solution:

```
(define/contract (find-ocurrences text first second)
(-> string? string? string? (listof string?))
)
```