Problem 30: Substring with Concatenation of All Words

Problem Information

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

Problem Description

You are given a string

s

and an array of strings

words

. All the strings of

words

are of

the same length

.

A

concatenated string

is a string that exactly contains all the strings of any permutation of

words

concatenated.

For example, if

words = ["ab","cd","ef"]

, then

"abcdef"

,

"abefcd"

,

"cdabef"

,

"cdefab"

,

"efabcd"

, and

"efcdab"

are all concatenated strings.

"acdbef"

is not a concatenated string because it is not the concatenation of any permutation of

words

.

Return an array of

the starting indices

of all the concatenated substrings in

s

. You can return the answer in

any order

.

Example 1:

Input:

s = "barfoothefoobarman", words = ["foo","bar"]

Output:

[0,9]

Explanation:

The substring starting at 0 is

"barfoo"

. It is the concatenation of

["bar","foo"]

which is a permutation of

words

.

The substring starting at 9 is

"foobar"

. It is the concatenation of

["foo","bar"]

which is a permutation of

words

.

Example 2:

Input:

s = "wordgoodgoodgoodbestword", words = ["word","good","best","word"]

Output:

[]

Explanation:

There is no concatenated substring.

Example 3:

Input:

s = "barfoofoobarthefoobarman", words = ["bar","foo","the"]

Output:

[6,9,12]

Explanation:

The substring starting at 6 is

"foobarthe"

. It is the concatenation of

["foo","bar","the"]

.

The substring starting at 9 is

"barthefoo"

. It is the concatenation of

["bar","the","foo"]

.

The substring starting at 12 is

"thefoobar"

. It is the concatenation of

["the","foo","bar"]

.

Constraints:

1 <= s.length <= 10

4

1 <= words.length <= 5000

1 <= words[i].length <= 30

s

and

words[i]

consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<int> findSubstring(string s, vector<string>& words) {

}
};
```

Java:

```
class Solution {
public List<Integer> findSubstring(String s, String[] words) {

}
}
```

Python3:

```
class Solution:
def findSubstring(self, s: str, words: List[str]) -> List[int]:
```

Python:

```
class Solution(object):
def findSubstring(self, s, words):
```

```
"""
:type s: str
:type words: List[str]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {string[]} words
* @return {number[]}
*/
var findSubstring = function(s, words) {

};
```

TypeScript:

```
function findSubstring(s: string, words: string[]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> FindSubstring(string s, string[] words) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findSubstring(char* s, char** words, int wordsSize, int* returnSize) {

}
```

Go:

```
func findSubstring(s string, words []string) []int {

}
```

Kotlin:

```
class Solution {
fun findSubstring(s: String, words: Array<String>): List<Int> {

}
}
```

Swift:

```
class Solution {
func findSubstring(_ s: String, _ words: [String]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn find_substring(s: String, words: Vec<String>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {String} s
# @param {String[]} words
# @return {Integer[]}
def find_substring(s, words)

end
```

PHP:

```
class Solution {

/**
* @param String $s
```

```
* @param String[] $words
* @return Integer[]
*/
function findSubstring($s, $words) {

}
}
```

Dart:

```
class Solution {
List<int> findSubstring(String s, List<String> words) {

}
}
```

Scala:

```
object Solution {
def findSubstring(s: String, words: Array[String]): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_substring(s :: String.t, words :: [String.t]) :: [integer]
def find_substring(s, words) do

end
end
```

Erlang:

```
-spec find_substring(S :: unicode:unicode_binary(), Words ::
[unicode:unicode_binary()]) -> [integer()].
find_substring(S, Words) ->
.
```

Racket:

```
(define/contract (find-substring s words)
(-> string? (listof string?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Substring with Concatenation of All Words
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* 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:
vector<int> findSubstring(string s, vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Substring with Concatenation of All Words
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* 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 List<Integer> findSubstring(String s, String[] words) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Substring with Concatenation of All Words
Difficulty: Hard
Tags: array, string, tree, hash

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 findSubstring(self, s: str, words: List[str]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findSubstring(self, s, words):
"""
:type s: str
:type words: List[str]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Substring with Concatenation of All Words
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* 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[]} words
* @return {number[]}
*/
var findSubstring = function(s, words) {

};
```

TypeScript Solution:

```
/**
* Problem: Substring with Concatenation of All Words
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* 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 findSubstring(s: string, words: string[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Substring with Concatenation of All Words
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* 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 IList<int> FindSubstring(string s, string[] words) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Substring with Concatenation of All Words
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findSubstring(char* s, char** words, int wordsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Substring with Concatenation of All Words
// Difficulty: Hard
// Tags: array, string, tree, hash
//
// 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 findSubstring(s string, words []string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findSubstring(s: String, words: Array<String>): List<Int> {

}
```

```
}
```

Swift Solution:

```
class Solution {
func findSubstring(_ s: String, _ words: [String]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Substring with Concatenation of All Words
// Difficulty: Hard
// Tags: array, string, tree, hash
//
// 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_substring(s: String, words: Vec<String>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String[]} words
# @return {Integer[]}
def find_substring(s, words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
```

```
* @param String[] $words
* @return Integer[]
*/
function findSubstring($s, $words) {

}
}
```

Dart Solution:

```
class Solution {
List<int> findSubstring(String s, List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def findSubstring(s: String, words: Array[String]): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_substring(s :: String.t, words :: [String.t]) :: [integer]
def find_substring(s, words) do

end
end
```

Erlang Solution:

```
-spec find_substring(S :: unicode:unicode_binary(), Words ::
[unicode:unicode_binary()]) -> [integer()].
find_substring(S, Words) ->
.
```

Racket Solution:

```
(define/contract (find-substring s words)
(-> string? (listof string?) (listof exact-integer?))
)
```