Problem 1065: Index Pairs of a String

Problem Information

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

Problem Description

Given a string

text

and an array of strings

words

, return

an array of all index pairs

[i, j]

so that the substring

text[i...j]

is in

words

.

Return the pairs

[i, j]

in sorted order (i.e., sort them by their first coordinate, and in case of ties sort them by their second coordinate).

Example 1:

Input:

text = "thestoryofleetcodeandme", words = ["story","fleet","leetcode"]

Output:

[[3,7],[9,13],[10,17]]

Example 2:

Input:

text = "ababa", words = ["aba","ab"]

Output:

[[0,1],[0,2],[2,3],[2,4]]

Explanation:

Notice that matches can overlap, see "aba" is found in [0,2] and [2,4].

Constraints:

1 <= text.length <= 100

1 <= words.length <= 20

1 <= words[i].length <= 50

text

and

words[i]

consist of lowercase English letters.

All the strings of

words

are

unique

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int[][] indexPairs(String text, String[] words) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function indexPairs(text: string, words: string[]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] IndexPairs(string text, string[] words) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
```

```
int** indexPairs(char* text, char** words, int wordsSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go:

```
func indexPairs(text string, words []string) [][]int {

}
```

Kotlin:

```
class Solution {
fun indexPairs(text: String, words: Array<String>): Array<IntArray> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```

end
```

PHP:

```
class Solution {

/**
* @param String $text
* @param String[] $words
* @return Integer[][]
*/
function indexPairs($text, $words) {

}
}
```

Dart:

```
class Solution {
List<List<int>> indexPairs(String text, List<String> words) {

}
}
```

Scala:

```
object Solution {
def indexPairs(text: String, words: Array[String]): Array[Array[Int]] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec index_pairs(Text :: unicode:unicode_binary(), Words ::
[unicode:unicode_binary()]) -> [[integer()]].
index_pairs(Text, Words) ->
.
```

Racket:

```
(define/contract (index-pairs text words)
(-> string? (listof string?) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Index Pairs of a String
* Difficulty: Easy
* Tags: array, string, tree, 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:
vector<vector<int>> indexPairs(string text, vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Index Pairs of a String
* Difficulty: Easy
* Tags: array, string, tree, 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 int[][] indexPairs(String text, String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Index Pairs of a String
Difficulty: Easy
Tags: array, string, tree, 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 indexPairs(self, text: str, words: List[str]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Index Pairs of a String
* Difficulty: Easy
* Tags: array, string, tree, 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} text
* @param {string[]} words
* @return {number[][]}
*/
var indexPairs = function(text, words) {

};
```

TypeScript Solution:

```
/**
* Problem: Index Pairs of a String
* Difficulty: Easy
* Tags: array, string, tree, 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 indexPairs(text: string, words: string[]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Index Pairs of a String
* Difficulty: Easy
* Tags: array, string, tree, 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 int[][] IndexPairs(string text, string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Index Pairs of a String
* Difficulty: Easy
* Tags: array, string, tree, 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** indexPairs(char* text, char** words, int wordsSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Index Pairs of a String
// Difficulty: Easy
// Tags: array, string, tree, 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 indexPairs(text string, words []string) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun indexPairs(text: String, words: Array<String>): Array<IntArray> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Index Pairs of a String
// Difficulty: Easy
// Tags: array, string, tree, 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 index_pairs(text: String, words: Vec<String>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param String $text
* @param String[] $words
* @return Integer[][]
*/
function indexPairs($text, $words) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> indexPairs(String text, List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def indexPairs(text: String, words: Array[String]): Array[Array[Int]] = {

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

```
-spec index_pairs(Text :: unicode:unicode_binary(), Words ::
[unicode:unicode_binary()]) -> [[integer()]].
index_pairs(Text, Words) ->
.
```

Racket Solution:

```
(define/contract (index-pairs text words)
(-> string? (listof string?) (listof (listof exact-integer?)))
)
```