

Problem 438: Find All Anagrams in a String

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given two strings

s

and

p

, return an array of all the start indices of

p

's

anagrams

in

s

. You may return the answer in

any order

Example 1:

Input:

s = "cbaebabacd", p = "abc"

Output:

[0,6]

Explanation:

The substring with start index = 0 is "cba", which is an anagram of "abc". The substring with start index = 6 is "bac", which is an anagram of "abc".

Example 2:

Input:

s = "abab", p = "ab"

Output:

[0,1,2]

Explanation:

The substring with start index = 0 is "ab", which is an anagram of "ab". The substring with start index = 1 is "ba", which is an anagram of "ab". The substring with start index = 2 is "ab", which is an anagram of "ab".

Constraints:

$1 \leq s.length, p.length \leq 3 * 10$

4

s

and

p

consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {  
public:  
vector<int> findAnagrams(string s, string p) {  
  
}  
};
```

Java:

```
class Solution {  
public List<Integer> findAnagrams(String s, String p) {  
  
}  
}
```

Python3:

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

Python:

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

JavaScript:

```
/**  
 * @param {string} s  
 * @param {string} p  
 * @return {number[]}   
 */  
var findAnagrams = function(s, p) {  
};
```

TypeScript:

```
function findAnagrams(s: string, p: string): number[] {  
};
```

C#:

```
public class Solution {  
    public IList<int> FindAnagrams(string s, string p) {  
        }  
    }
```

C:

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

Go:

```
func findAnagrams(s string, p string) []int {  
}
```

Kotlin:

```
class Solution {  
    fun findAnagrams(s: String, p: String): List<Int> {
```

}

Swift:

```
class Solution {
    func findAnagrams(_ s: String, _ p: String) -> [Int] {
        if s.count < p.count { return [] }
        var count = [Character: Int]()
        for c in p { count[c] = (count[c] ?? 0) + 1 }
        var start = 0, end = 0, matches = 0, result = []
        while end < s.count {
            let currentChar = s[end]
            if count[currentChar] != nil {
                if count[currentChar]! == 1 { matches += 1 }
                count[currentChar] = (count[currentChar] ?? 0) - 1
            }
            end += 1
            if end - start == p.count {
                if matches == p.count { result.append(start) }
                let startChar = s[start]
                if count[startChar] != nil {
                    if count[startChar]! == 0 { matches -= 1 }
                    count[startChar] = (count[startChar] ?? 0) + 1
                }
                start += 1
            }
        }
        return result
    }
}
```

Rust:

```
impl Solution {
    pub fn find_anagrams(s: String, p: String) -> Vec<i32> {
        if s.len() < p.len() {
            return vec![];
        }
        let mut count = [0; 26];
        for c in p.chars() {
            count[c as usize - 'a' as usize] += 1;
        }
        let mut start = 0;
        let mut end = 0;
        let mut matches = 0;
        let mut result = Vec::new();
        while end < s.len() {
            if count[s[end] as usize - 'a' as usize] == 0 {
                while start < end {
                    if count[s[start] as usize - 'a' as usize] > 0 {
                        break;
                    }
                    start += 1;
                }
            } else {
                if count[s[end] as usize - 'a' as usize] == 1 {
                    matches += 1;
                }
                count[s[end] as usize - 'a' as usize] -= 1;
            }
            if matches == p.len() {
                result.push(start);
            }
            end += 1;
        }
        result
    }
}
```

Ruby:

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

end
```

PHP:

```
class Solution {  
  
    /**  
     * @param String $s  
     * @param String $p  
     * @return Integer[]  
     */  
    function findAnagrams($s, $p) {  
  
        }  
    }  
}
```

Dart:

```
class Solution {  
    List<int> findAnagrams(String s, String p) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def findAnagrams(s: String, p: String): List[Int] = {  
  
    }  
}
```

Elixir:

```
defmodule Solution do  
    @spec find_anagrams(s :: String.t, p :: String.t) :: [integer]  
    def find_anagrams(s, p) do  
  
    end  
end
```

Erlang:

```
-spec find_anagrams(S :: unicode:unicode_binary(), P ::  
unicode:unicode_binary()) -> [integer()].  
find_anagrams(S, P) ->  
.
```

Racket:

```
(define/contract (find-anagrams s p)  
(-> string? string? (listof exact-integer?))  
)
```

Solutions

C++ Solution:

```

/*
 * Problem: Find All Anagrams in a String
 * Difficulty: Medium
 * 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> findAnagrams(string s, string p) {

}
};


```

Java Solution:

```

/**
 * Problem: Find All Anagrams in a String
 * Difficulty: Medium
 * 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> findAnagrams(String s, String p) {

}
};


```

Python3 Solution:

```

"""

Problem: Find All Anagrams in a String
Difficulty: Medium
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 findAnagrams(self, s: str, p: str) -> List[int]:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

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

```

JavaScript Solution:

```
/**
 * Problem: Find All Anagrams in a String
 * Difficulty: Medium
 * 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} p
 * @return {number[]}
 */
var findAnagrams = function(s, p) {

};
```

TypeScript Solution:

```
/**  
 * Problem: Find All Anagrams in a String  
 * Difficulty: Medium  
 * 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 findAnagrams(s: string, p: string): number[] {  
};
```

C# Solution:

```
/*  
 * Problem: Find All Anagrams in a String  
 * Difficulty: Medium  
 * 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> FindAnagrams(string s, string p) {  
        return null;  
    }  
}
```

C Solution:

```
/*  
 * Problem: Find All Anagrams in a String  
 * Difficulty: Medium  
 * 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* findAnagrams(char* s, char* p, int* returnSize) {
}

```

Go Solution:

```

// Problem: Find All Anagrams in a String
// Difficulty: Medium
// 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 findAnagrams(s string, p string) []int {
}

```

Kotlin Solution:

```

class Solution {
    fun findAnagrams(s: String, p: String): List<Int> {
        }
    }
}

```

Swift Solution:

```

class Solution {
    func findAnagrams(_ s: String, _ p: String) -> [Int] {
        }
    }
}

```

Rust Solution:

```

// Problem: Find All Anagrams in a String
// Difficulty: Medium
// 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_anagrams(s: String, p: String) -> Vec<i32> {
        }

    }
}

```

Ruby Solution:

```

# @param {String} s
# @param {String} p
# @return {Integer[]}
def find_anagrams(s, p)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param String $s
     * @param String $p
     * @return Integer[]
     */
    function findAnagrams($s, $p) {

    }
}

```

Dart Solution:

```

class Solution {
    List<int> findAnagrams(String s, String p) {

```

```
}
```

```
}
```

Scala Solution:

```
object Solution {  
    def findAnagrams(s: String, p: String): List[Int] = {  
  
    }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec find_anagrams(s :: String.t, p :: String.t) :: [integer]  
  def find_anagrams(s, p) do  
  
  end  
end
```

Erlang Solution:

```
-spec find_anagrams(S :: unicode:unicode_binary(), P ::  
  unicode:unicode_binary()) -> [integer()].  
find_anagrams(S, P) ->  
.
```

Racket Solution:

```
(define/contract (find-anagrams s p)  
  (-> string? string? (listof exact-integer?))  
)
```