Problem 2942: Find Words Containing Character

Problem Information

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

Problem Description

You are given a

0-indexed

array of strings

words

and a character

x

.

Return

an

array of indices

representing the words that contain the character

x

.

Note

that the returned array may be in

any

order.

Example 1:

Input:

words = ["leet","code"], x = "e"

Output:

[0,1]

Explanation:

"e" occurs in both words: "l

ee

t", and "cod

e

". Hence, we return indices 0 and 1.

Example 2:

Input:

words = ["abc","bcd","aaaa","cbc"], x = "a"

Output:

[0,2]

Explanation:

"a" occurs in "

a

bc", and "

aaaa

". Hence, we return indices 0 and 2.

Example 3:

Input:

words = ["abc","bcd","aaaa","cbc"], x = "z"

Output:

[]

Explanation:

"z" does not occur in any of the words. Hence, we return an empty array.

Constraints:

1 <= words.length <= 50

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

x

is a lowercase English letter.

words[i]

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<int> findWordsContaining(vector<string>& words, char x) {

}
};
```

Java:

```
class Solution {
public List<Integer> findWordsContaining(String[] words, char x) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string[]} words
* @param {character} x
* @return {number[]}
*/
var findWordsContaining = function(words, x) {

};
```

TypeScript:

```
function findWordsContaining(words: string[], x: string): number[] {

};
```

C#:

```
public class Solution {
public IList<int> FindWordsContaining(string[] words, char x) {

}
}
```

C:

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

}
```

Go:

```
func findWordsContaining(words []string, x byte) []int {

}
```

Kotlin:

```
class Solution {
fun findWordsContaining(words: Array<String>, x: Char): List<Int> {

}

}
```

Swift:

```
class Solution {
func findWordsContaining(_ words: [String], _ x: Character) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn find_words_containing(words: Vec<String>, x: char) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {String[]} words
# @param {Character} x
# @return {Integer[]}
def find_words_containing(words, x)

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<int> findWordsContaining(List<String> words, String x) {

}
}
```

Scala:

```
object Solution {
def findWordsContaining(words: Array[String], x: Char): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_words_containing(words :: [String.t], x :: char) :: [integer]
def find_words_containing(words, x) do

end
end
```

Erlang:

```
-spec find_words_containing(Words :: [unicode:unicode_binary()], X :: char())
-> [integer()].
find_words_containing(Words, X) ->
.
```

Racket:

```
(define/contract (find-words-containing words x)
(-> (listof string?) char? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Words Containing Character
* 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<int> findWordsContaining(vector<string>& words, char x) {

}
};
```

Java Solution:

```
/**
* Problem: Find Words Containing Character
* 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 List<Integer> findWordsContaining(String[] words, char x) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Words Containing Character
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 findWordsContaining(self, words: List[str], x: str) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Words Containing Character
* 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[]} words
* @param {character} x
* @return {number[]}
*/
var findWordsContaining = function(words, x) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Words Containing Character
* 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 findWordsContaining(words: string[], x: string): number[] {

};
```

C# Solution:

```
/*
* Problem: Find Words Containing Character
* 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 IList<int> FindWordsContaining(string[] words, char x) {

}
}
```

C Solution:

```
/*
* Problem: Find Words Containing Character
* 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().
*/
int* findWordsContaining(char** words, int wordsSize, char x, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Find Words Containing Character
// 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 findWordsContaining(words []string, x byte) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findWordsContaining(words: Array<String>, x: Char): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func findWordsContaining(_ words: [String], _ x: Character) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find Words Containing Character
// 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_words_containing(words: Vec<String>, x: char) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @param {Character} x
# @return {Integer[]}
def find_words_containing(words, x)

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<int> findWordsContaining(List<String> words, String x) {

}
}
```

Scala Solution:

```
object Solution {
def findWordsContaining(words: Array[String], x: Char): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_words_containing(words :: [String.t], x :: char) :: [integer]
def find_words_containing(words, x) do

end
end
```

Erlang Solution:

```
-spec find_words_containing(Words :: [unicode:unicode_binary()], X :: char())
-> [integer()].
find_words_containing(Words, X) ->
.
```

Racket Solution:

```
(define/contract (find-words-containing words x)
(-> (listof string?) char? (listof exact-integer?))
)
```