Problem 1002: Find Common Characters

Problem Information

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

Problem Description

Given a string array

words

, return

an array of all characters that show up in all strings within the

words

(including duplicates)

. You may return the answer in

any order

.

Example 1:

Input:

words = ["bella","label","roller"]

Output:

["e","l","l"]

Example 2:

Input:

words = ["cool","lock","cook"]

Output:

["c","o"]

Constraints:

1 <= words.length <= 100

1 <= words[i].length <= 100

words[i]

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<string> commonChars(vector<string>& words) {

}
};
```

Java:

```
class Solution {
public List<String> commonChars(String[] words) {

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string[]} words
* @return {string[]}
*/
var commonChars = function(words) {

};
```

TypeScript:

```
function commonChars(words: string[]): string[] {

};
```

C#:

```
public class Solution {
public IList<string> CommonChars(string[] words) {

}
}
```

C:

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

}
```

Go:

```
func commonChars(words []string) []string {

}
```

Kotlin:

```
class Solution {
fun commonChars(words: Array<String>): List<String> {

}
}
```

Swift:

```
class Solution {
func commonChars(_ words: [String]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn common_chars(words: Vec<String>) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} words
# @return {String[]}
def common_chars(words)
```

```
end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @return String[]
*/
function commonChars($words) {

}
}
```

Dart:

```
class Solution {
List<String> commonChars(List<String> words) {

}
}
```

Scala:

```
object Solution {
def commonChars(words: Array[String]): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec common_chars(words :: [String.t]) :: [String.t]
def common_chars(words) do

end
end
```

Erlang:

```
-spec common_chars(Words :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
common_chars(Words) ->
.
```

Racket:

```
(define/contract (common-chars words)
(-> (listof string?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Common Characters
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<string> commonChars(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Find Common Characters
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) for hash map
*/

class Solution {
public List<String> commonChars(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Common Characters
Difficulty: Easy
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def commonChars(self, words: List[str]) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Common Characters
* Difficulty: Easy
* Tags: array, string, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[]} words
* @return {string[]}
*/
var commonChars = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Common Characters
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function commonChars(words: string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Find Common Characters
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
public class Solution {
public IList<string> CommonChars(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Find Common Characters
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
```

Go Solution:

```
// Problem: Find Common Characters
// Difficulty: Easy
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func commonChars(words []string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun commonChars(words: Array<String>): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func commonChars(_ words: [String]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Find Common Characters
// Difficulty: Easy
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn common_chars(words: Vec<String>) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @return {String[]}
def common_chars(words)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param String[] $words
* @return String[]
*/
function commonChars($words) {

}
}
```

Dart Solution:

```
class Solution {
List<String> commonChars(List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def commonChars(words: Array[String]): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec common_chars(words :: [String.t]) :: [String.t]
def common_chars(words) do

end
end
```

Erlang Solution:

```
-spec common_chars(Words :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
common_chars(Words) ->
.
```

Racket Solution:

```
(define/contract (common-chars words)
(-> (listof string?) (listof string?))
)
```