Problem 336: Palindrome Pairs

Problem Information

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

Problem Description

You are given a

0-indexed

array of

unique

strings

words

.

A

palindrome pair

is a pair of integers

(i, j)

such that:

0 <= i, j < words.length

,

i != j

, and

words[i] + words[j]

(the concatenation of the two strings) is a

palindrome

.

Return

an array of all the

palindrome pairs

of

words

.

You must write an algorithm with

O(sum of words[i].length)

runtime complexity.

Example 1:

Input:

words = ["abcd","dcba","lls","s","sssll"]

Output:

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

Explanation:

The palindromes are ["abcddcba","dcbaabcd","slls","llssssll"]

Example 2:

Input:

words = ["bat","tab","cat"]

Output:

[[0,1],[1,0]]

Explanation:

The palindromes are ["battab","tabbat"]

Example 3:

Input:

words = ["a",""]

Output:

[[0,1],[1,0]]

Explanation:

The palindromes are ["a","a"]

Constraints:

1 <= words.length <= 5000

0 <= words[i].length <= 300

words[i]

consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function palindromePairs(words: string[]): number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> PalindromePairs(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** palindromePairs(char** words, int wordsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

```
func palindromePairs(words []string) [][]int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<List<int>> palindromePairs(List<String> words) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Palindrome Pairs
* Difficulty: Hard
* 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<vector<int>> palindromePairs(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Palindrome Pairs
* Difficulty: Hard
* 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<List<Integer>> palindromePairs(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Palindrome Pairs
Difficulty: Hard
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 palindromePairs(self, words: List[str]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Palindrome Pairs
* Difficulty: Hard
* 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 {number[][]}
*/
var palindromePairs = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Palindrome Pairs
* Difficulty: Hard
* 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 palindromePairs(words: string[]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Palindrome Pairs
* Difficulty: Hard
* 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<IList<int>> PalindromePairs(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Palindrome Pairs
* Difficulty: Hard
* 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
```

```
*/

/**
* 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** palindromePairs(char** words, int wordsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Palindrome Pairs
// Difficulty: Hard
// 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 palindromePairs(words []string) [][]int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
```

```
}
```

Rust Solution:

```
// Problem: Palindrome Pairs
// Difficulty: Hard
// 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 palindrome_pairs(words: Vec<String>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> palindromePairs(List<String> words) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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