Problem 2131: Longest Palindrome by Concatenating Two Letter Words

Problem Information

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

Problem Description

You are given an array of strings

words

. Each element of

words

consists of

two

lowercase English letters.

Create the

longest possible palindrome

by selecting some elements from

words

and concatenating them in

any order

. Each element can be selected

at most once

.

Return

the

length

of the longest palindrome that you can create

. If it is impossible to create any palindrome, return

0

.

A

palindrome

is a string that reads the same forward and backward.

Example 1:

Input:

words = ["lc","cl","gg"]

Output:

6

Explanation:

One longest palindrome is "lc" + "gg" + "cl" = "lcggcl", of length 6. Note that "clgglc" is another longest palindrome that can be created.

Example 2:

Input:

words = ["ab","ty","yt","lc","cl","ab"]

Output:

8

Explanation:

One longest palindrome is "ty" + "lc" + "cl" + "yt" = "tylcclyt", of length 8. Note that "lcyttycl" is another longest palindrome that can be created.

Example 3:

Input:

words = ["cc","ll","xx"]

Output:

2

Explanation:

One longest palindrome is "cc", of length 2. Note that "ll" is another longest palindrome that can be created, and so is "xx".

Constraints:

1 <= words.length <= 10

5

words[i].length == 2

words[i]

consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int longestPalindrome(String[] words) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int LongestPalindrome(string[] words) {

}
}
```

C:

```
int longestPalindrome(char** words, int wordsSize) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest Palindrome by Concatenating Two Letter Words
* Difficulty: Medium
* Tags: array, string, greedy, 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:
int longestPalindrome(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Palindrome by Concatenating Two Letter Words
* Difficulty: Medium
* Tags: array, string, greedy, 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 int longestPalindrome(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Palindrome by Concatenating Two Letter Words
Difficulty: Medium
Tags: array, string, greedy, 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 longestPalindrome(self, words: List[str]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Palindrome by Concatenating Two Letter Words
* Difficulty: Medium
* Tags: array, string, greedy, 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 longestPalindrome = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Palindrome by Concatenating Two Letter Words
* Difficulty: Medium
* Tags: array, string, greedy, 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 longestPalindrome(words: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Longest Palindrome by Concatenating Two Letter Words
* Difficulty: Medium
* Tags: array, string, greedy, 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 int LongestPalindrome(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Longest Palindrome by Concatenating Two Letter Words
* Difficulty: Medium
* Tags: array, string, greedy, 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
*/

int longestPalindrome(char** words, int wordsSize) {

}
```

Go Solution:

```
// Problem: Longest Palindrome by Concatenating Two Letter Words
// Difficulty: Medium
// Tags: array, string, greedy, 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 longestPalindrome(words []string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Palindrome by Concatenating Two Letter Words
// Difficulty: Medium
// Tags: array, string, greedy, 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 longest_palindrome(words: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```php
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```scala
object Solution {
def longestPalindrome(words: Array[String]): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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