Problem 318: Maximum Product of Word Lengths

Problem Information

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

Problem Description

Given a string array

words

, return

the maximum value of

length(word[i]) * length(word[j])

where the two words do not share common letters

. If no such two words exist, return

0

.

Example 1:

Input:

words = ["abcw","baz","foo","bar","xtfn","abcdef"]

Output:

16

Explanation:

The two words can be "abcw", "xtfn".

Example 2:

Input:

words = ["a","ab","abc","d","cd","bcd","abcd"]

Output:

4

Explanation:

The two words can be "ab", "cd".

Example 3:

Input:

words = ["a","aa","aaa","aaaa"]

Output:

0

Explanation:

No such pair of words.

Constraints:

2 <= words.length <= 1000

1 <= words[i].length <= 1000

words[i]

consists only of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-product words)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Product of Word Lengths
* Difficulty: Medium
* 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:
int maxProduct(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Product of Word Lengths
* Difficulty: Medium
* 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 int maxProduct(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Product of Word Lengths
Difficulty: Medium
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 maxProduct(self, words: List[str]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Product of Word Lengths
* Difficulty: Medium
* 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
* @return {number}
*/
var maxProduct = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Product of Word Lengths
* Difficulty: Medium
* 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 maxProduct(words: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Product of Word Lengths
* Difficulty: Medium
* 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 int MaxProduct(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Product of Word Lengths
* Difficulty: Medium
* 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
*/

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

}
```

Go Solution:

```
// Problem: Maximum Product of Word Lengths
// Difficulty: Medium
// 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 maxProduct(words []string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Product of Word Lengths
// Difficulty: Medium
// 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 max_product(words: Vec<String>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-product words)
(-> (listof string?) exact-integer?)
)
```