Problem 1048: Longest String Chain

Problem Information

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

Problem Description

You are given an array of

words

where each word consists of lowercase English letters.

word

A

is a

predecessor

of

word

B

if and only if we can insert

exactly one

letter anywhere in

word

A

without changing the order of the other characters

to make it equal to

word

B

.

For example,

"abc"

is a

predecessor

of

"ab

a

c"

, while

"cba"

is not a

predecessor

of

"bcad"

.

A

word chain

is a sequence of words

[word

1

, word

2

, ..., word

k

]

with

k >= 1

, where

word

1

is a

predecessor

of

word

2

,

word

2

is a

predecessor

of

word

3

, and so on. A single word is trivially a

word chain

with

k == 1

.

Return

the

length

of the

longest possible word chain

with words chosen from the given list of

words

.

Example 1:

Input:

words = ["a","b","ba","bca","bda","bdca"]

Output:

4

Explanation

: One of the longest word chains is ["a","

b

a","b

d

a","bd

c

a"].

Example 2:

Input:

words = ["xbc","pcxbcf","xb","cxbc","pcxbc"]

Output:

5

Explanation:

All the words can be put in a word chain ["xb", "xb

c

", "

c

xbc", "

p

cxbc", "pcxbc

f

"].

Example 3:

Input:

words = ["abcd","dbqca"]

Output:

1

Explanation:

The trivial word chain ["abcd"] is one of the longest word chains. ["abcd","dbqca"] is not a valid word chain because the ordering of the letters is changed.

Constraints:

1 <= words.length <= 1000

1 <= words[i].length <= 16

words[i]

only consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest String Chain
* Difficulty: Medium
* Tags: array, string, dp, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Longest String Chain
* Difficulty: Medium
* Tags: array, string, dp, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int longestStrChain(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest String Chain
Difficulty: Medium
Tags: array, string, dp, hash, sort
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest String Chain
* Difficulty: Medium
* Tags: array, string, dp, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Longest String Chain
* Difficulty: Medium
* Tags: array, string, dp, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function longestStrChain(words: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Longest String Chain
* Difficulty: Medium
* Tags: array, string, dp, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int LongestStrChain(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Longest String Chain
* Difficulty: Medium
* Tags: array, string, dp, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

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

}
```

Go Solution:

```
// Problem: Longest String Chain
// Difficulty: Medium
// Tags: array, string, dp, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func longestStrChain(words []string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest String Chain
// Difficulty: Medium
// Tags: array, string, dp, hash, sort
```

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

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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