Problem 646: Maximum Length of Pair Chain

Problem Information

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

Problem Description

You are given an array of

n

pairs

pairs

where

pairs[i] = [left

i

, right

i

]

and

left

i

< right

i

.

A pair

p2 = [c, d]

follows

a pair

p1 = [a, b]

if

b < c

. A

chain

of pairs can be formed in this fashion.

Return

the length longest chain which can be formed

.

You do not need to use up all the given intervals. You can select pairs in any order.

Example 1:

Input:

pairs = [[1,2],[2,3],[3,4]]

Output:

2

Explanation:

The longest chain is [1,2] -> [3,4].

Example 2:

Input:

pairs = [[1,2],[7,8],[4,5]]

Output:

3

Explanation:

The longest chain is [1,2] -> [4,5] -> [7,8].

Constraints:

n == pairs.length

1 <= n <= 1000

-1000 <= left

i

< right

i

<= 1000

Code Snippets

C++:

```
class Solution {
public:
int findLongestChain(vector<vector<int>>& pairs) {

}
};
```

Java:

```
class Solution {
public int findLongestChain(int[][] pairs) {

}
}
```

Python3:

```
class Solution:
def findLongestChain(self, pairs: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def findLongestChain(self, pairs):
"""
:type pairs: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} pairs
* @return {number}
*/
var findLongestChain = function(pairs) {

};
```

TypeScript:

```
function findLongestChain(pairs: number[][]): number {

};
```

C#:

```
public class Solution {
public int FindLongestChain(int[][] pairs) {

}
}
```

C:

```
int findLongestChain(int** pairs, int pairsSize, int* pairsColSize) {

}
```

Go:

```
func findLongestChain(pairs [][]int) int {

}
```

Kotlin:

```
class Solution {
fun findLongestChain(pairs: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func findLongestChain(_ pairs: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_longest_chain(pairs: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} pairs
# @return {Integer}
def find_longest_chain(pairs)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $pairs
* @return Integer
*/
function findLongestChain($pairs) {

}
}
```

Dart:

```
class Solution {
int findLongestChain(List<List<int>> pairs) {

}
}
```

Scala:

```
object Solution {
def findLongestChain(pairs: Array[Array[Int]]): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec find_longest_chain(pairs :: [[integer]]) :: integer
def find_longest_chain(pairs) do

end
end
```

Erlang:

```
-spec find_longest_chain(Pairs :: [[integer()]]) -> integer().
find_longest_chain(Pairs) ->
.
```

Racket:

```
(define/contract (find-longest-chain pairs)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Length of Pair Chain
* Difficulty: Medium
* Tags: array, dp, greedy, 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 findLongestChain(vector<vector<int>>& pairs) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Maximum Length of Pair Chain
* Difficulty: Medium
* Tags: array, dp, greedy, 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 findLongestChain(int[][] pairs) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Length of Pair Chain
Difficulty: Medium
Tags: array, dp, greedy, 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 findLongestChain(self, pairs: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findLongestChain(self, pairs):
"""
:type pairs: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Length of Pair Chain
* Difficulty: Medium
* Tags: array, dp, greedy, 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 {number[][]} pairs
* @return {number}
*/
var findLongestChain = function(pairs) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Length of Pair Chain
* Difficulty: Medium
* Tags: array, dp, greedy, 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 findLongestChain(pairs: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Length of Pair Chain
* Difficulty: Medium
* Tags: array, dp, greedy, 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 FindLongestChain(int[][] pairs) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Length of Pair Chain
* Difficulty: Medium
* Tags: array, dp, greedy, 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 findLongestChain(int** pairs, int pairsSize, int* pairsColSize) {

}
```

Go Solution:

```
// Problem: Maximum Length of Pair Chain
// Difficulty: Medium
// Tags: array, dp, greedy, 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 findLongestChain(pairs [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findLongestChain(pairs: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func findLongestChain(_ pairs: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Length of Pair Chain
// Difficulty: Medium
// Tags: array, dp, greedy, 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 find_longest_chain(pairs: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} pairs
# @return {Integer}
def find_longest_chain(pairs)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $pairs
* @return Integer
*/
function findLongestChain($pairs) {

}
}
```

Dart Solution:

```
class Solution {
int findLongestChain(List<List<int>> pairs) {

}
}
```

Scala Solution:

```
object Solution {
def findLongestChain(pairs: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_longest_chain(pairs :: [[integer]]) :: integer
def find_longest_chain(pairs) do

end
```

```
end
```

Erlang Solution:

```
-spec find_longest_chain(Pairs :: [[integer()]]) -> integer().
find_longest_chain(Pairs) ->
.
```

Racket Solution:

```
(define/contract (find-longest-chain pairs)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```