Problem 2097: Valid Arrangement of Pairs

Problem Information

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

Problem Description

You are given a

0-indexed

2D integer array

pairs

where

pairs[i] = [start

i

, end

i

]

. An arrangement of

pairs

is

valid

if for every index

i

where

1 <= i < pairs.length

, we have

end

i-1

== start

i

.

Return

any

valid arrangement of

pairs

.

Note:

The inputs will be generated such that there exists a valid arrangement of

pairs

.

Example 1:

Input:

pairs = [[5,1],[4,5],[11,9],[9,4]]

Output:

[[11,9],[9,4],[4,5],[5,1]]

Explanation:

This is a valid arrangement since end

i-1

always equals start

i

. end

0

= 9 == 9 = start

1

end

1

= 4 == 4 = start

2

end

2

= 5 == 5 = start

3

Example 2:

Input:

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

Output:

[[1,3],[3,2],[2,1]]

Explanation:

This is a valid arrangement since end

i-1

always equals start

i

. end

0

= 3 == 3 = start

1

end

1

= 2 == 2 = start

2

The arrangements [[2,1],[1,3],[3,2]] and [[3,2],[2,1],[1,3]] are also valid.

Example 3:

Input:

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

Output:

[[1,2],[2,1],[1,3]]

Explanation:

This is a valid arrangement since end

i-1

always equals start

i

. end

0

= 2 == 2 = start

1

end

1

= 1 == 1 = start

2

Constraints:

1 <= pairs.length <= 10

5

pairs[i].length == 2

0 <= start

i

, end

i

<= 10

9

start

i

!= end

i

No two pairs are exactly the same.

There

exists

a valid arrangement of

pairs

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[][]} pairs
* @return {number[][]}
*/
```

```
var validArrangement = function(pairs) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

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** validArrangement(int** pairs, int pairsSize, int* pairsColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

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

```
}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (valid-arrangement pairs)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Valid Arrangement of Pairs
```

```
* Difficulty: Hard
* Tags: array, graph, search
*
* 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:
vector<vector<int>> validArrangement(vector<vector<int>>& pairs) {

}
};
```

Java Solution:

```
/**
* Problem: Valid Arrangement of Pairs
* Difficulty: Hard
* Tags: array, graph, search
*
* 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[][] validArrangement(int[][] pairs) {

}
}
```

Python3 Solution:

```
"""
Problem: Valid Arrangement of Pairs
Difficulty: Hard
Tags: array, graph, search

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 validArrangement(self, pairs: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Valid Arrangement of Pairs
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 {number[][]} pairs
* @return {number[][]}
*/
var validArrangement = function(pairs) {

};
```

TypeScript Solution:

```
/**
* Problem: Valid Arrangement of Pairs
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 validArrangement(pairs: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Valid Arrangement of Pairs
* Difficulty: Hard
* Tags: array, graph, search
*
* 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[][] ValidArrangement(int[][] pairs) {

}
}
```

C Solution:

```
/*
* Problem: Valid Arrangement of Pairs
* Difficulty: Hard
* Tags: array, graph, search
*
* 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
```

```
*/

/**
* 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** validArrangement(int** pairs, int pairsSize, int* pairsColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Valid Arrangement of Pairs
// Difficulty: Hard
// Tags: array, graph, search
//
// 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 validArrangement(pairs [][]int) [][]int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
```

```
}
```

Rust Solution:

```
// Problem: Valid Arrangement of Pairs
// Difficulty: Hard
// Tags: array, graph, search
//
// 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 valid_arrangement(pairs: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (valid-arrangement pairs)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```