

Problem 444: Sequence Reconstruction

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an integer array

`nums`

of length

`n`

where

`nums`

is a permutation of the integers in the range

`[1, n]`

. You are also given a 2D integer array

`sequences`

where

`sequences[i]`

is a subsequence of

nums

.

Check if

nums

is the shortest possible and the only

supersequence

. The shortest

supersequence

is a sequence

with the shortest length

and has all

sequences[i]

as subsequences. There could be multiple valid

supersequences

for the given array

sequences

.

For example, for

sequences = [[1,2],[1,3]]

, there are two shortest

supersequences

,

[1,2,3]

and

[1,3,2]

.

While for

sequences = [[1,2],[1,3],[1,2,3]]

, the only shortest

supersequence

possible is

[1,2,3]

.

[1,2,3,4]

is a possible supersequence but not the shortest.

Return

true

if

nums

is the only shortest

supersequence

for

sequences

, or

false

otherwise

.

A

subsequence

is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

Example 1:

Input:

nums = [1,2,3], sequences = [[1,2],[1,3]]

Output:

false

Explanation:

There are two possible supersequences: [1,2,3] and [1,3,2]. The sequence [1,2] is a subsequence of both: [

,

2

,3] and [

1

,3,

2

]. The sequence [1,3] is a subsequence of both: [

1

,2,

3

] and [

1

,

3

,2]. Since nums is not the only shortest supersequence, we return false.

Example 2:

Input:

nums = [1,2,3], sequences = [[1,2]]

Output:

false

Explanation:

The shortest possible supersequence is [1,2]. The sequence [1,2] is a subsequence of it: [

1

,

2

]. Since nums is not the shortest supersequence, we return false.

Example 3:

Input:

nums = [1,2,3], sequences = [[1,2],[1,3],[2,3]]

Output:

true

Explanation:

The shortest possible supersequence is [1,2,3]. The sequence [1,2] is a subsequence of it: [

1

,

2

,3]. The sequence [1,3] is a subsequence of it: [

1

,2,

3

]. The sequence [2,3] is a subsequence of it: [1,

2

,

3

]. Since nums is the only shortest supersequence, we return true.

Constraints:

$n == \text{nums.length}$

$1 \leq n \leq 10$

4

nums

is a permutation of all the integers in the range

[1, n]

.

$1 \leq \text{sequences.length} \leq 10$

4

$1 \leq \text{sequences}[i].\text{length} \leq 10$

4

$1 \leq \sum(\text{sequences}[i].\text{length}) \leq 10$

5

$1 \leq \text{sequences}[i][j] \leq n$

All the arrays of

sequences

are

unique

.

sequences[i]

is a subsequence of

nums

.

Code Snippets

C++:

```
class Solution {
public:
    bool sequenceReconstruction(vector<int>& nums, vector<vector<int>>&
sequences) {

    }
};
```

Java:

```
class Solution {
    public boolean sequenceReconstruction(int[] nums, List<List<Integer>>
sequences) {
```

```
}  
}
```

Python3:

```
class Solution:  
    def sequenceReconstruction(self, nums: List[int], sequences: List[List[int]])  
    -> bool:
```

Python:

```
class Solution(object):  
    def sequenceReconstruction(self, nums, sequences):  
        """  
        :type nums: List[int]  
        :type sequences: List[List[int]]  
        :rtype: bool  
        """
```

JavaScript:

```
/**  
 * @param {number[]} nums  
 * @param {number[][]} sequences  
 * @return {boolean}  
 */  
var sequenceReconstruction = function(nums, sequences) {  
  
};
```

TypeScript:

```
function sequenceReconstruction(nums: number[], sequences: number[][]):  
    boolean {  
  
};
```

C#:

```
public class Solution {  
    public bool SequenceReconstruction(int[] nums, IList<IList<int>> sequences) {
```

```
}  
}
```

C:

```
bool sequenceReconstruction(int* nums, int numsSize, int** sequences, int  
sequencesSize, int* sequencesColSize) {  
  
}
```

Go:

```
func sequenceReconstruction(nums []int, sequences [][]int) bool {  
  
}
```

Kotlin:

```
class Solution {  
    fun sequenceReconstruction(nums: IntArray, sequences: List<List<Int>>):  
        Boolean {  
  
    }  
}
```

Swift:

```
class Solution {  
    func sequenceReconstruction(_ nums: [Int], _ sequences: [[Int]]) -> Bool {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn sequence_reconstruction(nums: Vec<i32>, sequences: Vec<Vec<i32>>) ->  
        bool {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[][]} sequences
# @return {Boolean}
def sequence_reconstruction(nums, sequences)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $nums
     * @param Integer[][] $sequences
     * @return Boolean
     */
    function sequenceReconstruction($nums, $sequences) {

    }

}
```

Dart:

```
class Solution {
  bool sequenceReconstruction(List<int> nums, List<List<int>> sequences) {

  }
}
```

Scala:

```
object Solution {
  def sequenceReconstruction(nums: Array[Int], sequences: List[List[Int]]):
    Boolean = {

  }
}
```

Elixir:

```

defmodule Solution do
  @spec sequence_reconstruction(nums :: [integer], sequences :: [[integer]]) ::
    boolean
  def sequence_reconstruction(nums, sequences) do

  end
end

```

Erlang:

```

-spec sequence_reconstruction(Nums :: [integer()], Sequences ::
[[integer()]]) -> boolean().
sequence_reconstruction(Nums, Sequences) ->
.

```

Racket:

```

(define/contract (sequence-reconstruction nums sequences)
  (-> (listof exact-integer?) (listof (listof exact-integer?)) boolean?)
)

```

Solutions

C++ Solution:

```

/*
 * Problem: Sequence Reconstruction
 * Difficulty: Medium
 * Tags: array, graph, sort
 *
 * 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:
    bool sequenceReconstruction(vector<int>& nums, vector<vector<int>>&
sequences) {

    }
}

```

```
};
```

Java Solution:

```
/**
 * Problem: Sequence Reconstruction
 * Difficulty: Medium
 * Tags: array, graph, sort
 *
 * 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 boolean sequenceReconstruction(int[] nums, List<List<Integer>>
    sequences) {

    }

}
```

Python3 Solution:

```
"""
Problem: Sequence Reconstruction
Difficulty: Medium
Tags: array, graph, sort

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 sequenceReconstruction(self, nums: List[int], sequences: List[List[int]])
    -> bool:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

```

class Solution(object):
    def sequenceReconstruction(self, nums, sequences):
        """
        :type nums: List[int]
        :type sequences: List[List[int]]
        :rtype: bool
        """

```

JavaScript Solution:

```

/**
 * Problem: Sequence Reconstruction
 * Difficulty: Medium
 * Tags: array, graph, sort
 *
 * 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[]} nums
 * @param {number[][]} sequences
 * @return {boolean}
 */
var sequenceReconstruction = function(nums, sequences) {

};

```

TypeScript Solution:

```

/**
 * Problem: Sequence Reconstruction
 * Difficulty: Medium
 * Tags: array, graph, sort
 *
 * 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 sequenceReconstruction(nums: number[], sequences: number[][]):

```

```
boolean {  
  
};
```

C# Solution:

```
/*  
 * Problem: Sequence Reconstruction  
 * Difficulty: Medium  
 * Tags: array, graph, sort  
 *  
 * 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 bool SequenceReconstruction(int[] nums, IList<IList<int>> sequences) {  
  
    }  
}
```

C Solution:

```
/*  
 * Problem: Sequence Reconstruction  
 * Difficulty: Medium  
 * Tags: array, graph, sort  
 *  
 * 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  
 */  
  
bool sequenceReconstruction(int* nums, int numsSize, int** sequences, int  
sequencesSize, int* sequencesColSize) {  
  
}
```

Go Solution:

```

// Problem: Sequence Reconstruction
// Difficulty: Medium
// Tags: array, graph, sort
//
// 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 sequenceReconstruction(nums []int, sequences [][]int) bool {

}

```

Kotlin Solution:

```

class Solution {
    fun sequenceReconstruction(nums: IntArray, sequences: List<List<Int>>):
        Boolean {

    }
}

```

Swift Solution:

```

class Solution {
    func sequenceReconstruction(_ nums: [Int], _ sequences: [[Int]]) -> Bool {

    }
}

```

Rust Solution:

```

// Problem: Sequence Reconstruction
// Difficulty: Medium
// Tags: array, graph, sort
//
// 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 sequence_reconstruction(nums: Vec<i32>, sequences: Vec<Vec<i32>>) ->
        bool {

```

```
}  
}
```

Ruby Solution:

```
# @param {Integer[]} nums  
# @param {Integer[][]} sequences  
# @return {Boolean}  
def sequence_reconstruction(nums, sequences)  
  
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @param Integer[][] $sequences  
     * @return Boolean  
     */  
    function sequenceReconstruction($nums, $sequences) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
    bool sequenceReconstruction(List<int> nums, List<List<int>> sequences) {  
  
    }  
}
```

Scala Solution:

```
object Solution {  
    def sequenceReconstruction(nums: Array[Int], sequences: List[List[Int]]):  
    Boolean = {
```

```
}  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec sequence_reconstruction(nums :: [integer], sequences :: [[integer]]) ::  
    boolean  
  def sequence_reconstruction(nums, sequences) do  
  
  end  
end
```

Erlang Solution:

```
-spec sequence_reconstruction(Nums :: [integer()], Sequences ::  
  [[integer()]]) -> boolean().  
sequence_reconstruction(Nums, Sequences) ->  
  .
```

Racket Solution:

```
(define/contract (sequence-reconstruction nums sequences)  
  (-> (listof exact-integer?) (listof (listof exact-integer?)) boolean?)  
  )
```