Solution Review: Problem Challenge 1

Reconstructing a Sequence (hard)

Given a sequence `originalSeq` and an array of sequences, write a method to find if `originalSeq` can be uniquely reconstructed from the array of sequences.

Unique reconstruction means that we need to find if `originalSeq` is the only sequence such that all sequences in the array are subsequences of it.

Example 1:

```
Input: originalSeq: [1, 2, 3, 4], seqs: [[1, 2], [2, 3], [3, 4]]
Output: true
Explanation: The sequences [1, 2], [2, 3], and [3, 4] can uniquely reconstruct
[1, 2, 3, 4], in other words, all the given sequences uniquely define the order of numbers
in the 'originalSeq'.
```

Example 2:

```
Input: originalSeq: [1, 2, 3, 4], seqs: [[1, 2], [2, 3], [2, 4]]
Output: false
Explanation: The sequences [1, 2], [2, 3], and [2, 4] cannot uniquely reconstruct
[1, 2, 3, 4]. There are two possible sequences we can construct from the given sequences:
1) [1, 2, 3, 4]
2) [1, 2, 4, 3]
```

Example 3:

```
Input: originalSeq: [3, 1, 4, 2, 5], seqs: [[3, 1, 5], [1, 4, 2, 5]]
Output: true
Explanation: The sequences [3, 1, 5] and [1, 4, 2, 5] can uniquely reconstruct
[3, 1, 4, 2, 5].
```

Solution

Since each sequence in the given array defines the ordering of some numbers, we need to combine all these ordering rules to find two things:

1. Is it possible to construct the `originalSeq` from all these rules?
2. Are these ordering rules not sufficient enough to define the unique ordering of all the numbers in the `originalSeq`? In other words, can these rules result in more than one sequence?

Take Example-1:

```
originalSeq: [1, 2, 3, 4], seqs:[[1, 2], [2, 3], [3, 4]]
```

The first sequence tells us that '1' comes before '2'; the second sequence tells us that '2' comes before '3'; the third sequence tells us that '3' comes before '4'. Combining all these sequences will result in a unique sequence: [1, 2, 3, 4].

The above explanation tells us that we are actually asked to find the topological ordering of all the numbers and also to verify that there is only one topological ordering of the numbers possible from the given array of the sequences.

This makes the current problem similar to Tasks Scheduling Order with two differences:

1. We need to build the graph of the numbers by comparing each pair of numbers in the given array of sequences.
2. We must perform the topological sort for the graph to determine two things:
   - Can the topological ordering construct the `originalSeq`?
   - That there is only one topological ordering of the numbers possible. This can be confirmed if we do not have more than one source at any time while finding the topological ordering of numbers.

Code

Here is what our algorithm will look like (only the highlighted lines have changed):

Java | Python3 | C++ | JS

```java
import java.util.*;

class SequenceReconstruction {
  public static boolean canConstruct(int[] originalSeq, int[][] sequences) {
    List<Integer> sortedOrder = new ArrayList<>();
    if (originalSeq.length <= 0)
      return false;

    // a. Initialize the graph
    HashMap<Integer, Integer> inDegree = new HashMap<>(); // count of incoming edges for every vertex
    HashMap<Integer, List<Integer>> graph = new HashMap<>(); // adjacency list graph
    for (int[] seq : sequences) {
      for (int i = 0; i < seq.length; i++) {
        inDegree.putIfAbsent(seq[i], 0);
        graph.putIfAbsent(seq[i], new ArrayList<Integer>());
      }
    }

    // b. Build the graph
    for (int[] seq : sequences) {
      for (int i = 1; i < seq.length; i++) {
        int parent = seq[i - 1], child = seq[i];
        graph.get(parent).add(child);
        inDegree.put(child, inDegree.get(child) + 1);
      }
    }

    // if we don't have ordering rules for all the numbers we'll not able to uniquely construct the seque
```

RUN SAVE RESET

Close

Output 1.736s

```
Can we uniquely construct the sequence: true
Can we uniquely construct the sequence: false
Can we uniquely construct the sequence: true
```

Time complexity

In step ‘d’, each number can become a source only once and each edge (a rule) will be accessed and removed once. Therefore, the time complexity of the above algorithm will be $O(V+E)$, where ‘V’ is the count of distinct numbers and ‘E’ is the total number of the rules. Since, at most, each pair of numbers can give us one rule, we can conclude that the upper bound for the rules is $O(N)$ where ‘N’ is the count of numbers in all sequences. So, we can say that the time complexity of our algorithm is $O(V+N)$.

Space complexity

The space complexity will be $O(V+N)$, since we are storing all of the rules for each number in an adjacency list.