Solution Review: Problem Challenge 2

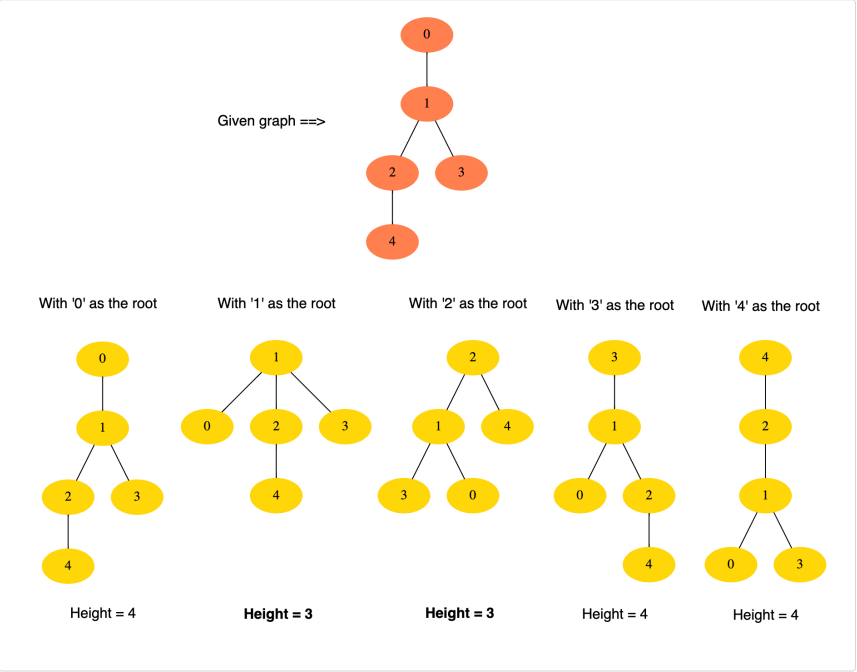
We'll cover the following

Minimum Height Trees (hard)

We are given an undirected graph that has characteristics of a k-ary tree. In such a graph, we can choose any node as the root to make a k-ary tree. The root (or the tree) with the minimum height will be called **Minimum Height Tree (MHT)**. There can be multiple MHTs for a graph. In this problem, we need to find all those roots which give us MHTs. Write a method to find all MHTs of the given graph and return a list of their roots.

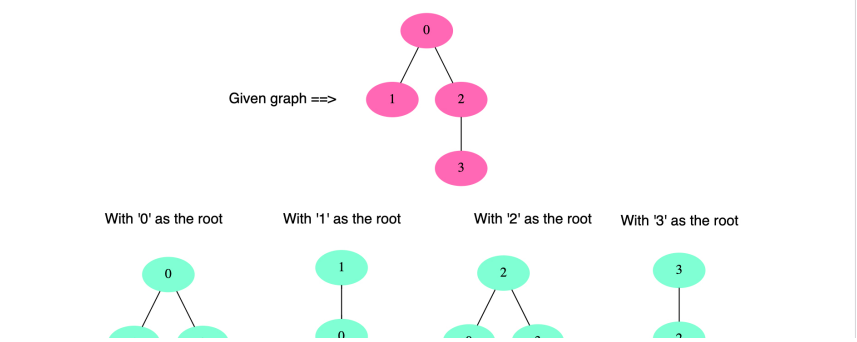
Example 1:

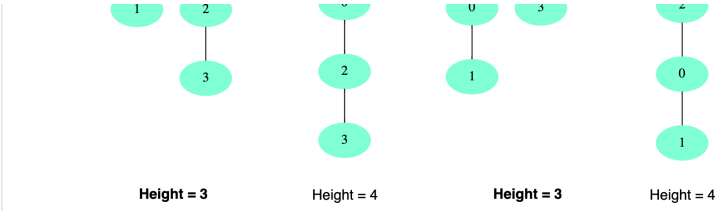
```
Input: vertices: 5, Edges: [[0, 1], [1, 2], [1, 3], [2, 4]]
Output:[1, 2]
Explanation: Choosing '1' or '2' as roots give us MHTs. In the below diagram, we can see that t
he
height of the trees with roots '1' or '2' is three which is minimum.
```



Example 2:

```
Input: vertices: 4, Edges: [[0, 1], [0, 2], [2, 3]]
Output:[0, 2]
Explanation: Choosing '0' or '2' as roots give us MHTs. In the below diagram, we can see that t
he
height of the trees with roots '0' or '2' is three which is minimum.
```





Example 3:

```
Input: vertices: 4, Edges: [[0, 1], [1, 2], [1, 3]]
Output:[1]
```

Solution

From the above-mentioned examples, we can clearly see that any leaf node (i.e., node with only one edge) can never give us an MHT because its adjacent non-leaf nodes will always give an MHT with a smaller height. All the adjacent non-leaf nodes will consider the leaf node as a subtree. Let's understand this with another example. Suppose we have a tree with root 'M' and height '5'. Now, if we take another node, say 'P', and make the 'M' tree as its subtree, then the height of the overall tree with root 'P' will be '6' (=5+1). Now, this whole tree can be considered a graph, where 'P' is a leaf as it has only one edge (connection with 'M'). This clearly shows that the leaf node ('P') gives us a tree of height '6' whereas its adjacent non-leaf node ('M') gives us a tree with smaller height '5' - since 'P' will be a child of 'M'.

This gives us a strategy to find MHTs. Since leaves can't give us MHT, we can remove them from the graph and remove their edges too. Once we remove the leaves, we will have new leaves. Since these new leaves can't give us MHT, we will repeat the process and remove them from the graph too. We will prune the leaves until we are left with one or two nodes which will be our answer and the roots for MHTs.

We can implement the above process using the topological sort. Any node with only one edge (i.e., a leaf) can be our source and, in a stepwise fashion, we can remove all sources from the graph to find new sources. We will repeat this process until we are left with one or two nodes in the graph, which will be our answer.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
const Deque = require('./collections/deque'); //http://www.collectionsjs.com


function find_trees(nodes, edges) {
  if (nodes <= 0) {
    return [];
  }

  // with only one node, since its in-degrees will be 0, therefore, we need to handle it separately
  if (nodes === 1) {
    return [0];
  }

  // a. Initialize the graph
  const inDegree = Array(nodes).fill(0); // count of incoming edges
  const graph = Array(nodes).fill(0).map(() => Array()); // adjacency list graph

  // b. Build the graph
  edges.forEach((edge) => {
    let n1 = edge[0],
      n2 = edge[1];
    // since this is an undirected graph, therefore, add a link for both the nodes
    graph[n1].push(n2);
    graph[n2].push(n1);
    // increment the in-degrees of both the nodes
    inDegree[n1] += 1;
    inDegree[n2] += 1;
  });
```

RUN SAVE RESET

Close

Output 3.859s

```
Roots of MHTs: 1,2
Roots of MHTs: 0,2
Roots of MHTs: 1
```

Time complexity

In step 'd', each node can become a source only once and each edge will be accessed and removed once. Therefore, the time complexity of the above algorithm will be $O(V + E)$, where 'V' is the total nodes and 'E' is the total number of the edges.

Space complexity

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