Documentation

Problem Overview

The problem is determining the **Minimum Height Trees (MHTs)** in a given tree structure. A tree is an undirected graph in which there are no cycles, and any two nodes are connected by exactly one path. The task is to identify the tree's root nodes such that the tree's height, when rooted at these nodes, is minimized. The height of a tree is defined as the length of the longest path from the root node to any leaf node. A minimum-height tree has the smallest possible height among all possible root nodes.

The problem provides a tree with n nodes and n-1 edges, and we are asked to return the nodes that can act as the roots of the minimum-height trees. This task can be solved efficiently by analyzing the properties of tree diameters and exploiting graph traversal techniques.

Key Insights

To find the Minimum Height Trees, it is crucial to understand the **diameter of the tree**. The diameter of a tree is the longest path between any two nodes in the tree. The center of this diameter is key to finding the MHT roots. The center of the tree lies on this longest path and may consist of one or two nodes. These nodes are the most balanced in terms of distance to the farthest leaf nodes, leading to the smallest height when used as the root.

By performing a series of graph traversals, we can identify these central nodes efficiently. The key idea is to first identify one endpoint of the tree's diameter using a breadth-first search (BFS). From this endpoint, we then perform another BFS to determine the other endpoint of the diameter. The path between these two endpoints forms the tree's diameter, and the nodes at or near the center of this path are the MHT roots.

Graph Representation

The input is a tree represented by an array of edges, where each edge connects two nodes. We can represent this tree as an undirected graph using an adjacency list, where each node is connected to a set of neighboring nodes. This adjacency list structure is particularly useful for performing BFS, as it allows for efficient lookups of each node's neighbors. Given the problem's constraints, using a dictionary of lists (or a defaultdict from Python's collections module) to represent the tree will allow for an efficient traversal.

Breadth-First Search (BFS) Approach

The solution relies heavily on BFS to determine the longest path in the tree (i.e., the tree's diameter). BFS is an ideal choice for this problem because it explores all nodes level by level, ensuring that the first time a node is encountered is along the shortest path from the starting node. By performing BFS from any node, we can determine the farthest node from it, which gives us one endpoint of the diameter. A second BFS from this farthest node will reveal the other endpoint of the diameter.

The BFS process also allows us to track the distance from the starting node to all other nodes, which is crucial for identifying the center of the tree. The distance information helps trace back the path between the two farthest nodes, and ultimately find the central nodes.

Tree Diameter and Center

The **center of the tree** is found along the path between the two endpoints of the tree's diameter. If the diameter has an odd length, the center consists of a single node, which will be the only root of the MHT. If the diameter has an even length, there will be two central nodes, both of which will be valid roots of the MHT. These central nodes are the most balanced in terms of their distances from the leaf nodes, ensuring that they minimize the height of the tree when used as the root.

This insight is crucial because it tells us that the problem reduces to finding the diameter of the tree and identifying the center(s) of this diameter, which is a significantly simpler task than trying all possible root nodes.

Time Complexity

The time complexity of the solution is linear, O(n)O(n), where n is the number of nodes in the tree. This is because each BFS operation runs in O(n)O(n) time, and we perform two BFS operations. The graph construction step is also O(n)O(n), as we only process n−1n-1 edges to build the adjacency list. Hence, the total time complexity is O(n)O(n), which is efficient for the problem's constraints.

This linear time complexity makes the solution scalable for the upper limits of the problem's input size, where nn can be as large as 20,000.

Conclusion

The problem of finding minimum-height trees can be efficiently solved using BFS and the concept of the tree's diameter. By performing two BFS operations, we can identify the endpoints of the tree's diameter and trace back to find the center(s) of this diameter. The nodes at the center of the tree's diameter are the roots of the MHTs, which minimizes the height of the tree. This approach ensures a time complexity of O(n)O(n), making it well-suited for large input sizes. Through this method, we not only solve the problem but also gain deeper insights into the structural properties of trees and their diameter.