Toy Train Tutorial

Pre-requisites:

This problem will require concepts of graph, graph traversal (dfs, bfs) and dynamic programming to solve. Make sure your basics of graph theory and dynamic programming are clear before attempting this problem. The following links can help you learn the prerequisites:

- http://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/
- http://www.geeksforgeeks.org/breadth-first-traversal-for-a-graph/
- https://www.topcoder.com/community/data-science/data-science-tutorials/introduction-to-graphs-and-their-data-structures-section-1/
- https://www.topcoder.com/community/data-science/data-science-tutorials/introduction-to-graphs-and-their-data-structures-section-3/
- https://www.topcoder.com/community/data-science/data-science-tutorials/introduction-to-graphs-and-their-data-structures-section-2/
- https://www.topcoder.com/community/data-science/data-science-tutorials/an-introduction-to-recursion-part-1/
- https://www.topcoder.com/community/data-science/data-science-tutorials/an-introduction-to-recursion-part-2/
- https://www.topcoder.com/community/data-science/data-science-tutorials/dynamic-programming-from-novice-to-advanced/

Problem Description:

There are n toy train stations, numbered 1 to n and there are a total of n-1 tracks. Some tracks are to be necessarily visited. The trains follow a specific path. If a train is placed on any station, it follows the path from that particular station to the 1st station. You have to decide the minimum number of trains (that you have to place wisely on stations) such that all necessary stations are visited.

Difficulty Level:

Medium-Hard

Editorial:

Consider all the tracks that we need to visit. Mark the ends of *u*, *v* white. After that, we will consider a simple dynamic programming *d*[*v*](*v* is the vertex) on the tree that for each vertex in the tree determines the number of white vertexes in the subtree. It is easy to calculate this by using a recursive function *calc*(*v*, *prev*) (*v* is the current node, and *prev* its immediate ancestor):

```
calc(v, prev)
{
        d[v] = 0;
        if (white[v])
                dv += 1;
        for all vertexes u such that there is the edge (u,v) or (v,u), u != prev:
                calc(u, v);
                d[v] += d[u];
}
```

After that we will add to answer all white vertexes *v* such that next condition is correct: *d*[*v*] = 1

Complexity of solution:

The algorithm we are implementing is a slight variation of dfs. So, the complexity is O(V+E). Where V is the number of vertices in the graph and E is the number of edges.