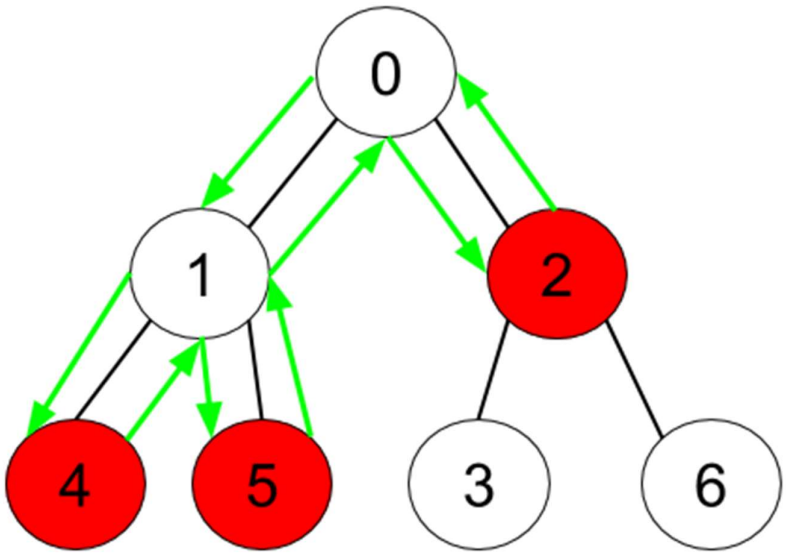
Approach For Solving a Medium Level LeetCode Problem

Question Link: https://leetcode.com/problems/minimum-time-to-collect-all-apples-in-a-tree/

Question in Simpler words:

In the question we are given an undirected tree consisting of n vertices numbered from 0 to n-1, which has some apples in their vertices. We spend 1 second to walk over one edge of the tree. We need to Return the minimum time in seconds that we must spend to collect all apples in the tree starting at **vertex 0** and coming back to this vertex.



Output: 8

Explanation: One optimal path to collect all apples is shown by the green arrows. The number of green arrows is 8 thus we require 8 seconds to collect all the apples

Solution:

Hint 1: Think about DFS (Depth First Search)

Hint 2: Think about what all paths we don't need to traverse in the tree.

Complete Solution:

This Problem can be solved using Depth First Search Algorithm.

Observation: We need to traverse an edge only if that edge is connected to a subtree in which is having apples.

How do we translate this Observation into code?

While doing DFS we can return whether the subtree of a node is having a node containing an apple is yes then we need to traverse that edge otherwise not. For doing this we increment the global answer (for traversing that edge) variable by one when we traverse the edge and when we return after traversing the subtree we if the DFS function returns true then that means that there exists a node in the subtree which is having an apple thus it is necessary to traverse this edge and we increment the answer variable again (for returning back from the subtree).

If the DFS function returns false then that means that there is no node in the subtree which is having an apple thus we don't need to traverse this edge and we decrement the answer variable by one (thereby eliminating the increment which we did earlier while traversing that edge)

Space Complexity: O(n)

The Function call stack is the only extra memory we take and there can be at max O(n) Functions in the call Stack at a time.

Time Complexity: O(n)

As we traverse the tree using DFS only once time complexity is O(n)

JAVA Code:

```
class Solution {
    boolean has[];
    int ans=0;
    boolean dfs(int start,int par){
```

```
    boolean flag=false;
    for(int u:graph[start]){
        if(u==par)
            continue;
        ans++;
        boolean b=dfs(u,start);
        flag=flag||b;
        if(b){
            ans++;
        }
        else{
            ans--;
        }
    }
     if(flag||has[start]){
        return true;
    }
    return false;
}
public int minTime(int n, int[][] edges, List<Boolean> hasApple) {
    Makegraph(n);
    for(int arr[]:edges){
        addEdge(arr[0],arr[1]);
        addEdge(arr[1],arr[0]);
    }
    has=new boolean[n];
    for(int i=0;i<n;i++){
```

```
            has[i]=hasApple.get(i);
        }
        ans=0;
        dfs(0,-1);
        return ans;
    }
    ArrayList<Integer> graph[];
    void Makegraph(int n){
        graph=new ArrayList[n];
        for(int i=0;i<n;i++)
            graph[i]=new ArrayList<>();
    }
    void addEdge(int a,int b){
        graph[a].add(b);
    }
}
```