61) Minimum Time to Collect All Apples in a Tree
Given an undirected tree consisting of n vertices numbered from 0 to n-1, which has some apples in their vertices. You spend 1 second to walk over one edge of the tree. Return the minimum time in seconds you have to spend to collect all apples in the tree, starting at vertex 0 and coming back to this vertex.
The edges of the undirected tree are given in the array edges, where edges[i] = [ai, bi] means that exists an edge connecting the vertices ai and bi. Additionally, there is a boolean array hasApple, where hasApple[i] = true means that vertex i has an apple; otherwise, it does not have any apple.

Example 1:

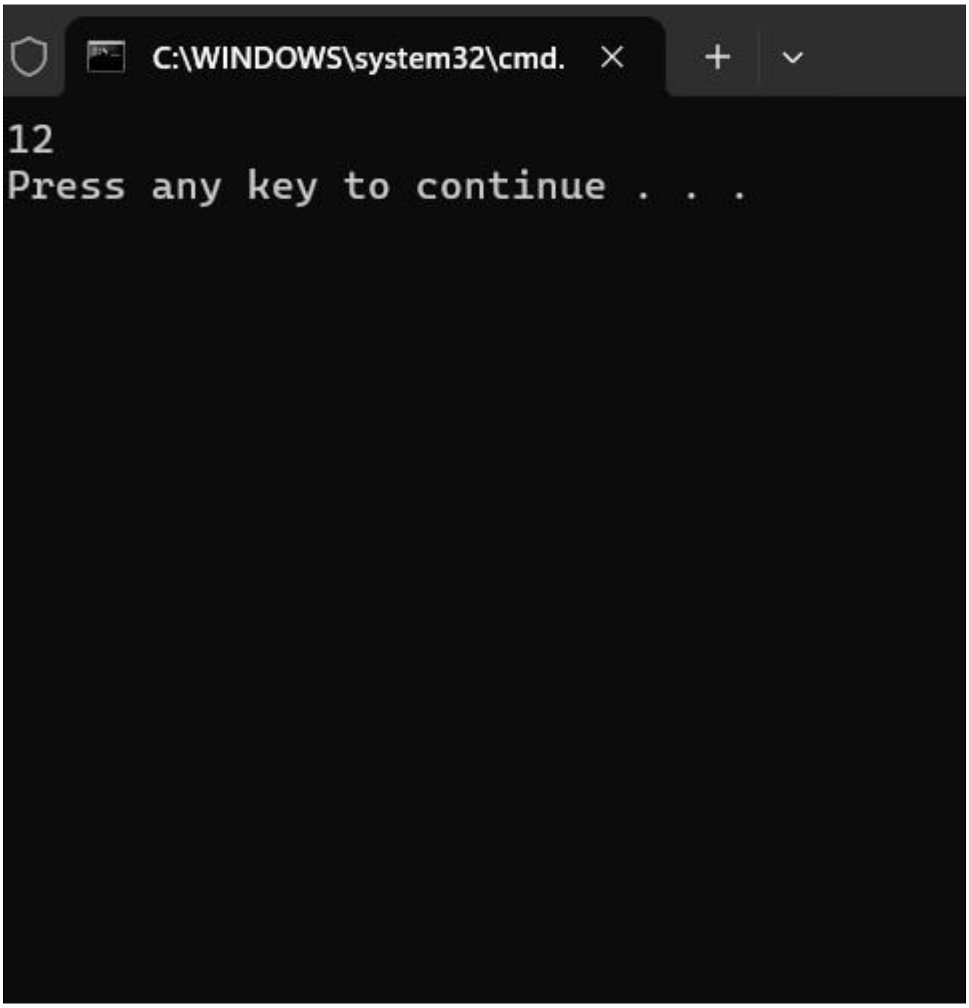
Input: n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], hasApple = [false,false,true,false,true,true,false]
Output: 8
Explanation: The figure above represents the given tree where red vertices have an apple. One optimal path to collect all apples is shown by the green arrows.

CODE:

```
def minTimeToCollectApples(n, edges,
hasApple):     graph = [[] for _ in range(n)]
for u, v in edges:         graph[u].append(v)
graph[v].append(u)
    def dfs(node, parent):         time = 0         for
neighbor in graph[node]:             if neighbor !=
parent:                 time += dfs(neighbor, node)         if
(time > 0 or hasApple[node]) and node != 0:
return time + 2         return time
```

```
    return max(0, 2 * (dfs(0, -1) - 2))

# Example
usage n = 7
edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]] hasApple
= [False,False,True,False,True,True,False]
print(minTimeToCollectApples(n, edges, hasApple))
```
OUTPUT:



TIME COMPLEXITY : O(n)