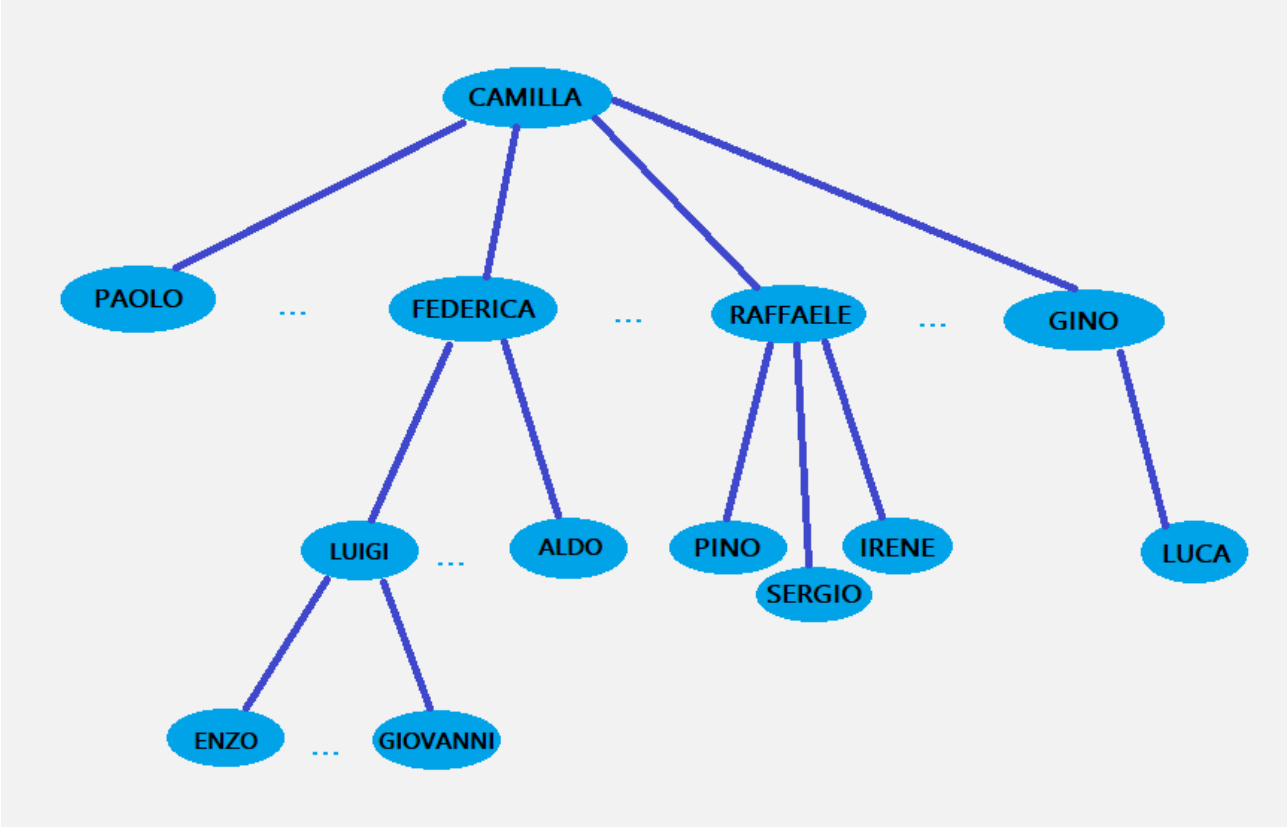
EXERCISE 4

In order to block the diffusion of misinformation on its social network, BaceFook has decided toinstall on the computers of some users a monitoringsoftware that is able to recognize and block the fake news. Specifically, BaceFook wants to install the software on the minimum number of users sothat for every pair of friends, at least one of them has the software.
Suppose that the BaceFooknetwork is a tree. Design and implement a Dynamic Programming algorithm that solve the problem in time O(n).

We thought about the BaceFook network as a n-ary tree, in which each person can have several friends. So, starting from a root, it has n children, so n friends, and each of these children has several other friends as children and so on. In other words, every node has as friends its root and its children. The root has as friends only its children.



Every tree has to be composed at least by 2 nodes, so the case in which there is only the root (it means that a user has no friends) it's not contemplated.

We also suppose that there are no repetitions in the tree. It means that people in different subtrees cannot have friends in common. This assumption is not very realistic, but considering repetition we wouldn't have a tree, but a graph.

We have to solve a minimum vertex cover problem: every node or its parent has to be marked (so must have the software) and in particular we want to find the minimum number of nodes to mark.

We implemented a simple TreeNode class in order to solve the problem, with methods *add_child()*, *height()*, *degree()* and *give_software()*. The method *give_software()* returns a tuple constituted by a counter and a dictionary. The dictionary contains the names of users (that are the values of the nodes) with a Boolean value that indicates if that person has the software or not, while the counter indicates the number of total software that have to be distributed.

We divide the problem in subproblems: we divide the tree in subtrees and for each vertex, we need to find the size of the smallest vertex cover in subtree rooted at that vertex.

We have to consider 2 cases: one in which we include the root and one in which we don't, and choose the solution with the minimum number of marked nodes. So, we call the function recursively on the children in the case in which we include the root and on the children's children in the case we exclude the root and then take in account the one with the minimum number of distributed software.

So, with dynamic programming we compute both cases simultaneously, avoiding the double call for each child.

We save the minimum number of software that have to be distributed in the variable count. If it's already calculated, we don't compute again the function, but return directly it and the dictionary containing the list of all the BaceFook's users(memoization phase).

We initially fill the dictionary with all the node's elements (name of users) as keys, and the boolean "False" as values. That value indicates if the node has the software or not. Other dictionaries used in the function have to collect all the nodes that have the software. *d_in* contains the minimum number of nodes with the software considering also the root, while *d_ex* doesn't contain the root. At the end, our first dictionary *d* is updated with the dict that contains the minimum number of nodes.

This dynamic programming solution actually solves the problem in time $\theta(n)$.