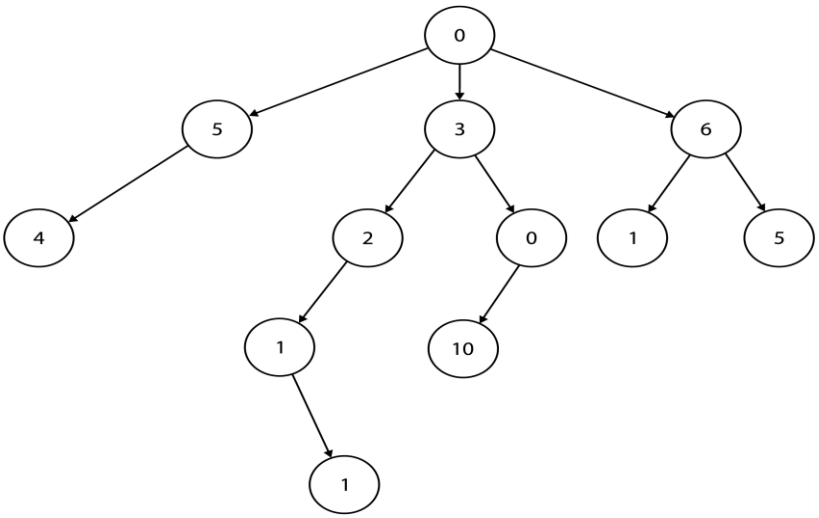
Sales Path

The car manufacturer Honda holds their distribution system in the form of a tree (not necessarily binary). The root is the company itself, and every node in the tree represents a car distributor that receives cars from the parent node and ships them to its children nodes. The leaf nodes are car dealerships that sell cars direct to consumers. In addition, every node holds an integer that is the cost of shipping a car to it.

Take for example the tree below:



A path from Honda's factory to a car dealership, which is a path from the root to a leaf in the tree, is called a Sales Path. The cost of a Sales Path is the sum of the costs for every node in the path. For example, in the tree above one Sales Path is 0→3→0→10, and its cost is `13` (0+3+0+10).

Honda wishes to find the minimal Sales Path cost in its distribution tree. Given a node `rootNode`, write an function `getCheapestCost` that calculates the minimal Sales Path cost in the tree.

Implement your function in the most efficient manner and analyze its time and space complexities.

For example:

Given the `rootNode` of the tree in diagram above

Your function would return:

`7` since it's the minimal Sales Path cost (there are actually two Sales Paths in the tree whose cost is `7`: 0→6→1 and 0→3→2→1→1)

Constraints:

- **[time limit] 5000ms**
- **[input] Node** `rootNode`
  - $0 \leq \text{rootNode.cost} \leq 100000$
- **[output] integer**

Hints

- Make sure your peer understands what is required from them in the question. You can check that by asking them to find the minimal Sales Path in the example.
- If your peer is stuck, ask them to write down all the possible paths in some example tree. After this is done, you may advise them to divide the paths per child of the root node.
- If your peer is still stuck, ask them to test on an example what happens if you input the function the children of the root node. After that, advise them to look for the connection between the function's value on the root, and on every child node.
- Your peer should get full score, only if they manage to find a solution that is $O(N)$ in both runtime and memory space, and only if they do so without any hints on your part. Make sure they know how to explain the correctness of the algorithm, along with the bounds on asymptotic complexity.
- While the tree in the question is abstract, if your peer needs an implementation you may state that function initially gets a tree node, and that every node has a field holding the cost, and and array holding its children nodes.
- A good followup question is how to alter the function in order to return all the Sales Paths with minimal cost in an array. Another good question, is how to use the function above to determine the longest or shortest Sales Path path.

Answer

Obviously iterating through all paths again and again is not a good solution, since its wasteful in terms of time and memory. But intuitively if we find a solution that uses previous calculations somehow. This hints that the solution should involve recursion in some manner.

First we notice that if the root is also a leaf, the best Sales Path, is simply the value in the node itself. This is the base case for the solution. If the root has children, then the minimal Sales Path is also a minimal path from the root's child. Thus, if we already know the minimal cost for the root's children, then the minimal cost for the root is simply the minimum of the values for its children plus the value stored in the root itself. A solution to this question, using these facts is given below:

```
function getCheapestCost(rootNode):
    n = rootNode.numberOfChildren()

    if (n == 0):
        return rootNode.cost
```

```
else:
    # initialize minCost to the largest integer in the system
    minCost = MAX_INT
    for i from 0 to n-1:
        tempCost = getCheapestCost(rootNode.child[i])
        if (tempCost < minCost):
            minCost = tempCost

return minCost + rootNode.cost
```

Time Complexity: let `N` be the number of nodes in the tree. Notice that `getCheapestCost` is applied to every node exactly once. Therefore, there are overall `O(N)` calls to `getCheapestCost`.

Space Complexity: every time the function recurses, it consumes only a constant amount of space. However, due to the nature of the recursion we used, the stack call holds `N` instances of `getCheapestCost` which makes the total space complexity to be `O(N)`.