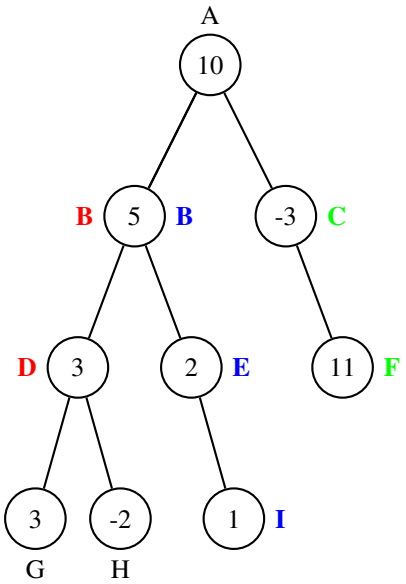

As highlighted in the picture, the solution is very general. It can

- begin at any node,
- a node may be part of multiple paths,
- the tree may contain negative nodes so you may never discard a node just because it's value is larger than targetSum.

Thus, it's very apparent the solution must traverse the full tree, and multiple times for all possible cases. Let's try to divide the solution. The solution

- can be sub-divided to solutions just in the left and right sub-trees,
- and solutions in left and right sub-trees including the current node.
- The total of these gives the final answer.

Let $\Pi(n, t)$ be short for pathSum(node, targetSum), and let $n.l$ and $n.r$ represent node.left and node.right, and let $n.v$ represent node.value. Let $\pi(n, t)$ is the number of paths starting from n with target sum t. So, a general strategy is that the solution $\Pi(n, t)$ is recursively

$$\Pi(n, t) = \Pi(n.l, t) + \Pi(n.r, t) + \pi(n.l, t - n.v) + \pi(n.r, t - n.v) \tag{1}$$

where, the first two terms count number of paths in either sub-trees that sum to t without including the current node, while the latter two terms include the current node in the target sum. This equation makes sense since each of the terms are mutually exclusive. So, they can be summed up.
Edge cases? Say in the figure below the **B-E-I** path, there was a long list of nodes with alternative +1, -1. How many nodes down a path must one go? Do they all count as their own unique paths? This algo will enumerate them all, but the worst case complexity would be poor. I don't see any opportunity to memoize or DP-ize the solution either. Maybe a bottom-up solution?