BST Successor Search

Given a node `n` in a binary search tree, explain and code the most efficient way to find the successor of `n`.
Analyze the runtime complexity of your solution.

Hints & Tips

- A complete solution **must** consider the case when `n` has no successor in the tree. `null` or any other agreed value should be returned in this case. If your peer doesn't address this, ask about it, but consider is as a hint that would lower your problem solving feedback by one star.
- Some programming languages lack an implementation of a tree data structure. If this is the case with language used on the interview, simply use an object / associative array for each node, with a key, parent, left child and right child.
- Another approach to find a successor is to start searching from the root and traverse the tree in a binary search manner. However, the root is **not** given in this question, only the specific node `n`.
- If you or your peer have a hard time understanding the solution, or if you could use a reminder of how binary search trees work, take this interactive BST application for a spin.

Solution

An in order successor is the smallest key that is greater than `n`'s key.
Let's call `n`'s in-order successor in the tree `s`

We discern between 2 cases:

- **Node `n` has a right child:** in this case `s` is the node with the minimum key on `n`'s right sub-tree.
- **Node `n` doesn't have a right child:** in this case `s` is one of `n`'s ancestors. More specifically, within `n`'s ancestor chain (starting from `n` up to the root), `s` is the first parent that has a left child on that chain.

If `n` doesn't have a right child and all of its ancestors are right children to their parents, `n` doesn't have a successor (`s` is null).

Why is this always true?
If `n` was inserted to the tree before `s`, since `s` is greater than `n` but also smaller than all other keys greater than `n`, `s` has to be on `n`'s right sub-tree.
Otherwise, `n` was inserted to the tree after `s` was inserted. Since `n` is smaller than `s` but also larger than all other keys smaller than `s`, `n` has to be on `s`'s left sub-tree.

```
function findInOrderAncestor(n):
   if (n.right != null):
      return findMinKeyWithinTree(n.right)

   ancestor = n.parent
   child = n
   while (ancestor != null AND child == ancestor.right):
      child = ancestor
      ancestor = child.parent
   return ancestor


function findMinKeyWithinTree(root):
   while (root.left != null):
      root = root.left
   return root
```