>> 3. Explain the process of deleting a node from a binary search tree in Python. Discuss how you would handle different cases, such as deleting a node with one, or two children. Additionally, explain any potential challenges or edge cases that may arise during the deletion process and how you would address them.

ChatGPT link: https://chatgpt.com/share/68bf819d-34e4-8010-9134-1d3e35db8128

The deletion algorithm for a binary search tree has two phases. First, find the target key by traversing the tree and comparing keys to locate the target key. Tree traversal is done by comparing the key to the current node and then moving left if the key is smaller or right if the key is larger. Second, remove the target node while maintaining BEST structure by relinking subtrees back to the parent of the node being deleted.

When the target node is found, there are three scenarios:

1. No children. Delete a childless node by clearing the parent’s reference to the node. In a recursive Python implementation, return None so the caller assigns None to the left/right pointer.
2. One child. When the target node has one child, bypass the target node by linking the parent of the target node directly to the child node. To do so, return the child node from the recursive function and assign it to the left/right pointer of the parent.
3. Two children. When the target node has two children, replace the target node with a key in the left or right side of the subtree under the target node. If selecting from the left child, use the largest key value in the subtree. If selecting from the right child, use the smallest key value in the subtree. Replace the target node’s key with that of the chosen successor, then recursively delete the successor from the original location.

Edge cases may include:

1. Deleting the root can cause an issue because there isn’t a parent of the returned node. The caller must set root = delete(root, key) as the deletion may return a new root.
2. Deleting a non-existent node. If a key is not located, the code needs to handle the return. If a key is not found, the function should return the tree unchanged or raise a “not found” error.
3. Duplicates. The class videos talked about how to handle duplicates. For the binary search tree to function properly, the policy needs to be consistently maintained throughout the code. I would “pick a side” where deletion will remove the first found instance of the key and have duplicates stored as separate nodes on the left.
4. Empty Tree. If the root is None, then the deletion won’t work.
5. Complexity. BST time is O(h) where h is the tree height. You can do a tree-balancing algorithm to reduce the complexity and ensure O(log n) instead of O(n) – the worst case scenario where all nodes are one side of the tree.