perplexity

Visualizing Binary Search Tree Deletion

The code you provided implements deletion in a Binary Search Tree (BST), which the third image properly illustrates. This tree has the following structure:

```
      50
     /  \
   30    70
  / \   / \
20  40 60  80
```

Binary Search Tree Properties

In this BST:

- Every node in the left subtree of 50 (30, 20, 40) has a value less than 50
- Every node in the right subtree of 50 (70, 60, 80) has a value greater than 50
- Both left and right subtrees are also BSTs

Deletion Cases

The deletion algorithm handles three distinct cases, which we can visualize using the BST in the image:

Case 1: Deleting a Leaf Node (Node 20)

When deleting node 20:

- It has no children
- The function returns `None` to node 30's left pointer
- Node 20 is simply removed from the tree

After deletion:

```
    50
   /  \
 30    70
   \   / \
   40 60  80
```

Case 2: Deleting a Node with One Child (Node 30 after 20 is deleted)

If we delete node 30 after 20 is gone:

- It has only one child (40)
- The function returns the child node to replace 30
- Node 40 becomes the new left child of 50

After deletion:

```
    50
   /  \
  40   70
      / \
     60  80
```

Case 3: Deleting a Node with Two Children (Node 50)

When deleting the root node 50:

1. Find the inorder successor (60) - smallest node in right subtree
2. Replace node 50's value with 60
3. Delete the original 60 from its location

The `get_successor` function:

- Starts at the right child (70)
- Goes left as far as possible (to 60)
- Returns 60 as the successor to 50

After deletion:

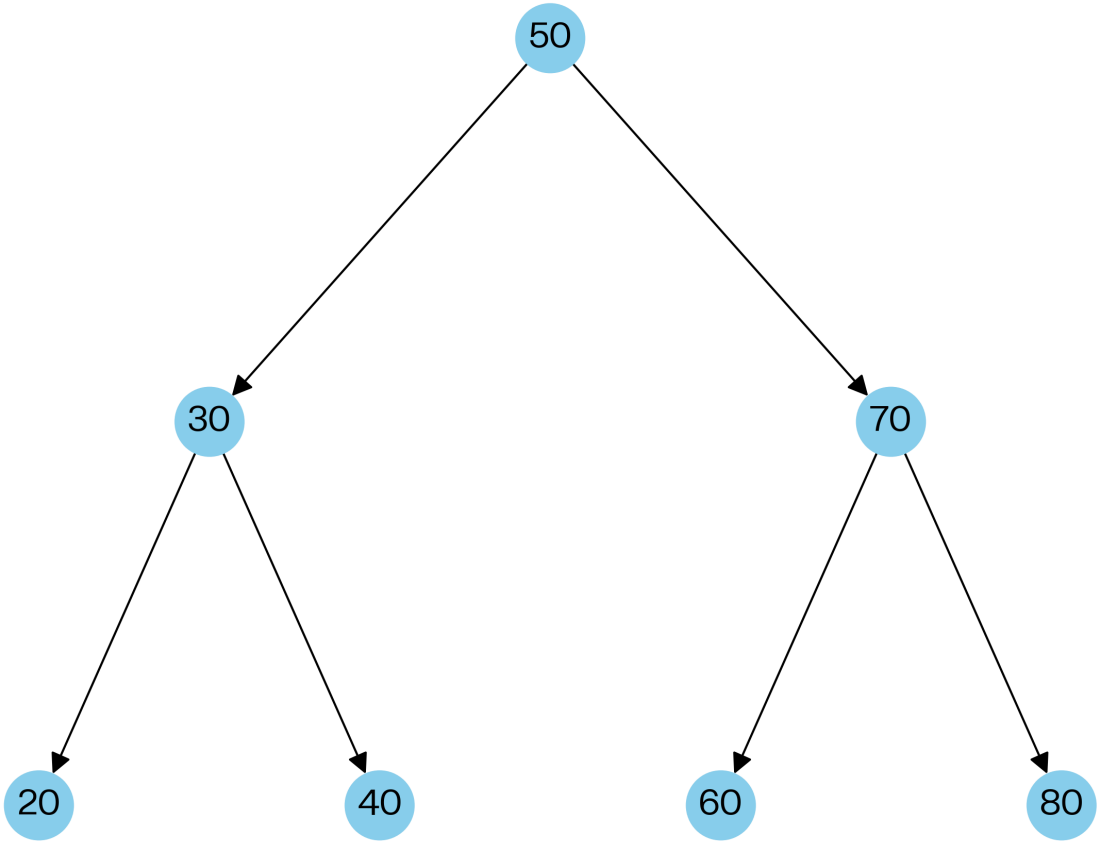
```
    60
   /  \
  40   70
         \
          80
```

Deletion Process Workflow

1. Search for the node to delete using BST properties
2. Once found, check which case applies
3. If leaf node: simply remove it
4. If one child: replace with its child
5. If two children: replace with inorder successor and recursively delete the successor

This deletion algorithm successfully maintains BST properties throughout the process, ensuring that for any node, all values in its left subtree are less than it, and all values in its right subtree are greater than it.

⁂



⁂