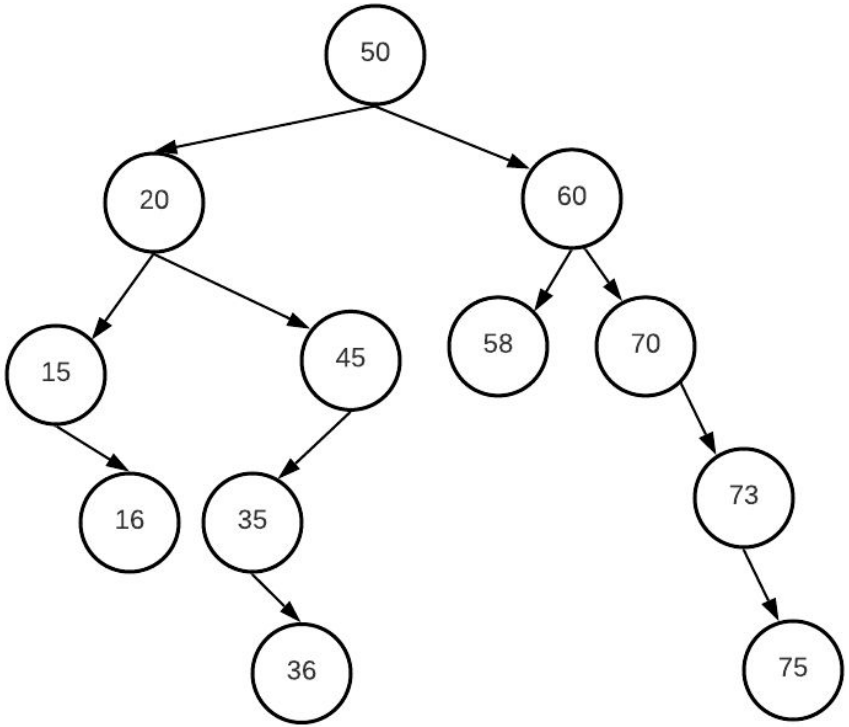
Data Structures

BST Deletion 1

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

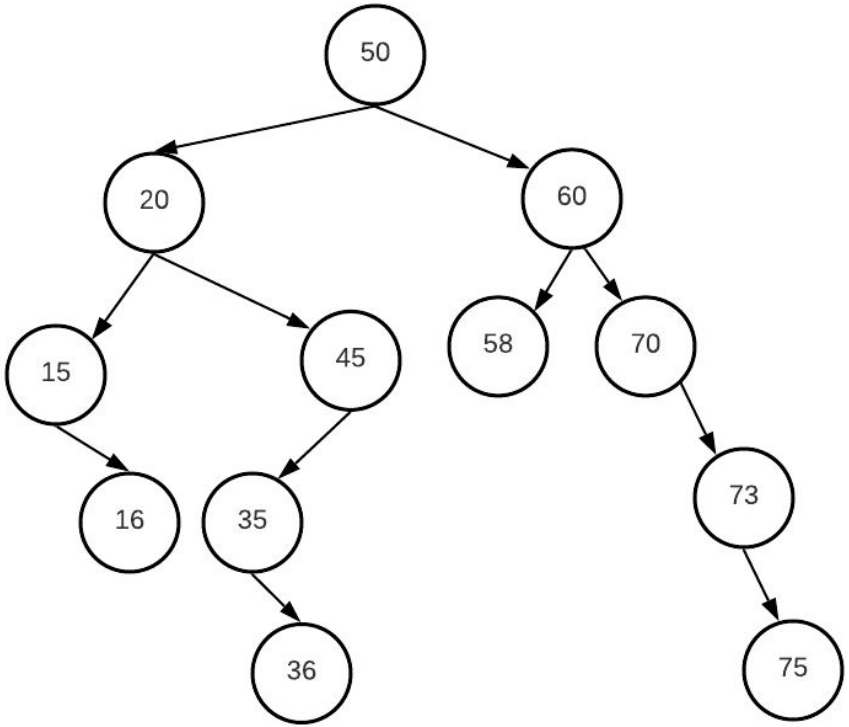
Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Node deletion



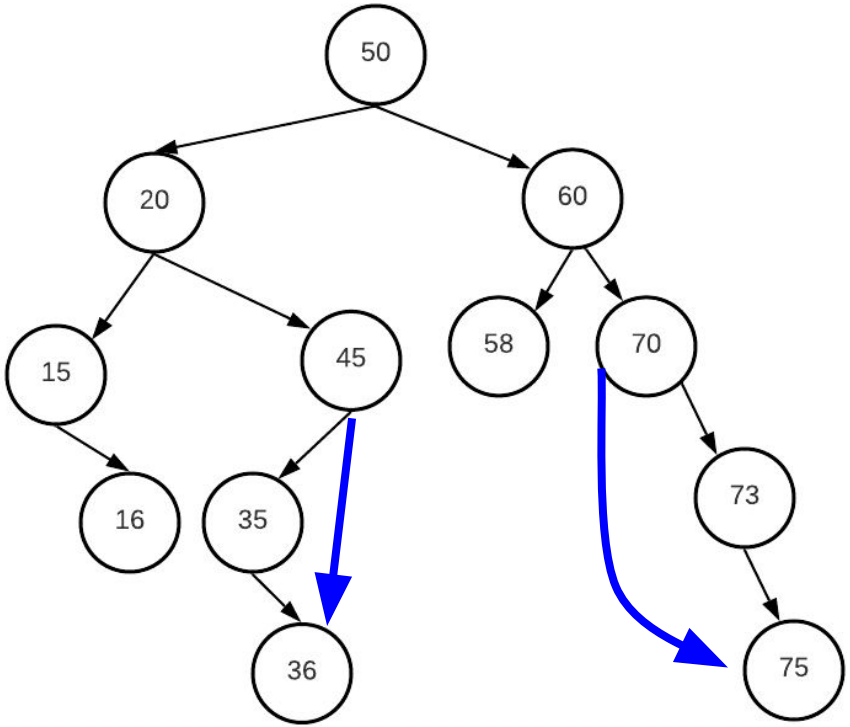
- Assume we have a tree of 2+ elements
- We want to delete a specific value
  - The tree must remain a BST after deletion
- We have 3 cases:
  - 0 children (75) ⇒ Direct
  - 1 child (73) ⇒ Almost Direct
  - 2 children (20) ⇒ A bit tricky

Case 1: 0 Children node



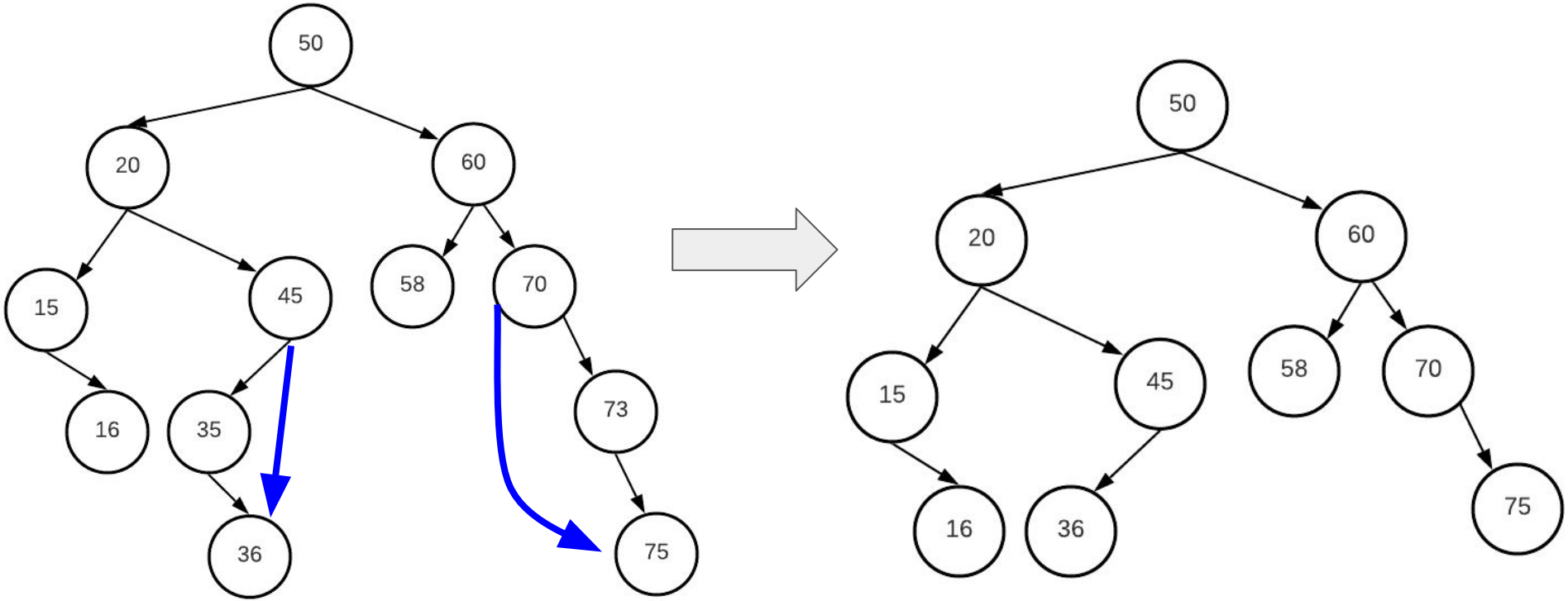
- If our node is a leaf node, it's straightforward. Just remove it
- Examples: 75, 58, 36, 16
  - Make sure you set the parent's child to None
    - Was it on the parent's left?
      Set parent.left = None
    - Apply the same logic if it was a right child!
  - Then free the node!

Case 2: 1 Child node

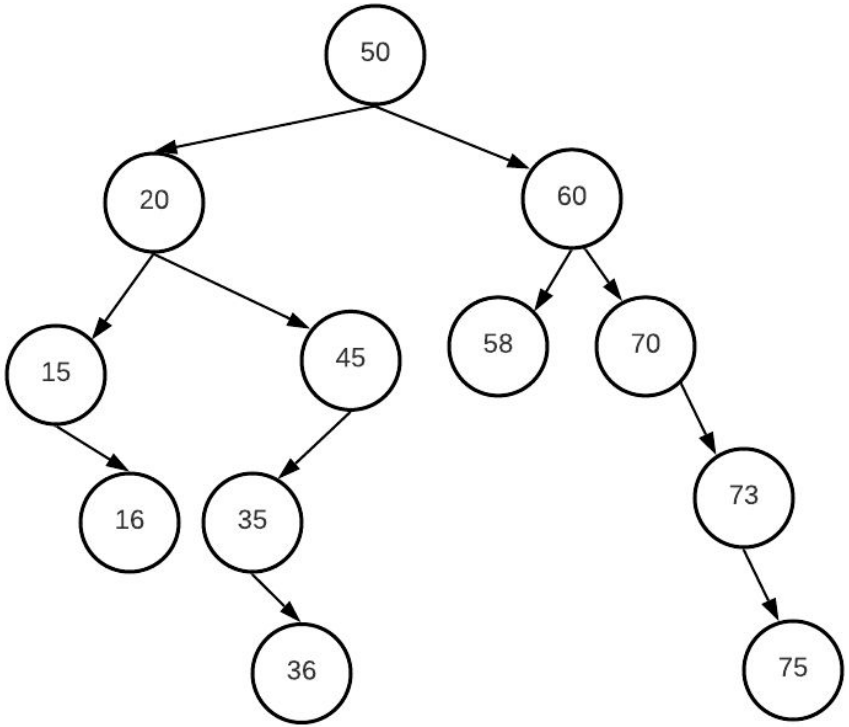


- If a node only has a single child, we can connect the parent to the lone child
- Examples: 70, 73, 45, 35, 15
- Consider 73:
  - It is right of 70
  - After removal: 70's right = 75
- Consider 35:
  - It is left of 45
  - After removal: 45's left = 36
    - We don't care that it was 35's right

Case 2: 1 Child node

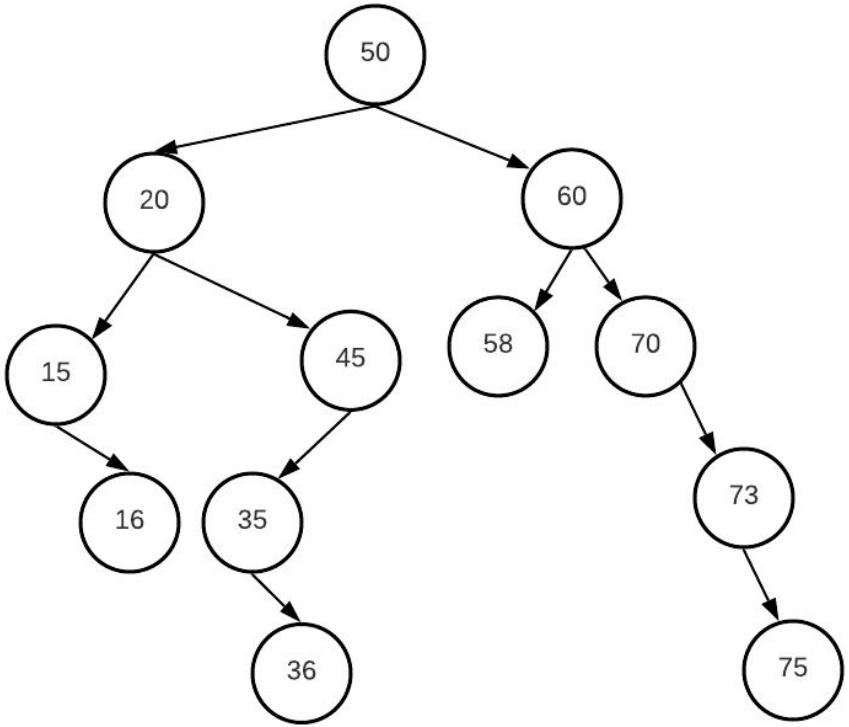


Case 3: 2 Child nodes



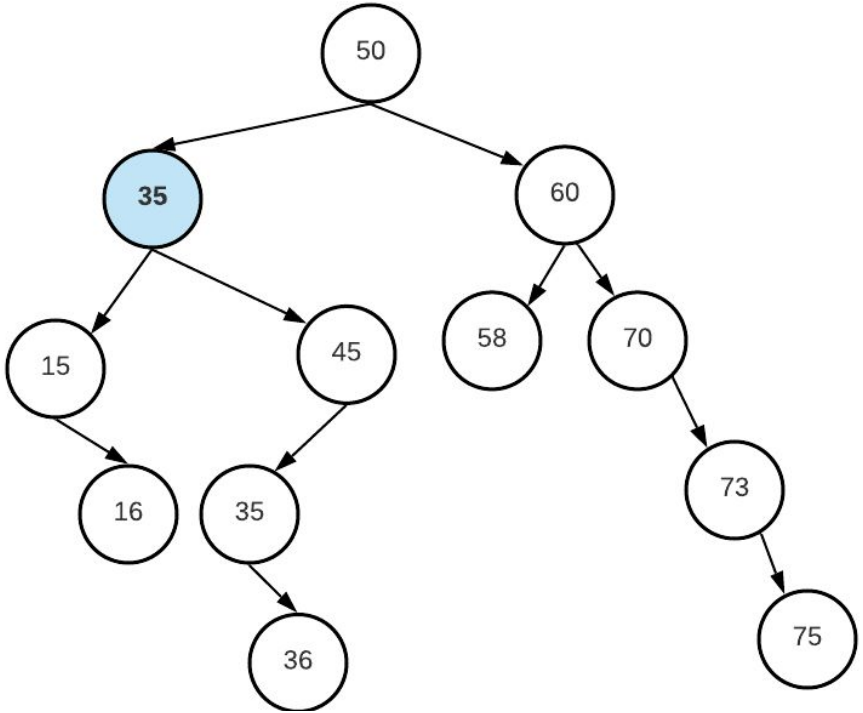
- Challenge: We need to make sure the tree remains a BST
- Examples: 20, 50, 60
- What is this tree inorder traversal?
  - 15 16 **20** 35 36 45 80 58 60 70 73 75
- Consider node 20
  - After removal, the inorder is:
  - 15 **16 35** 36 45 80 58 60 70 73 75
- Observe: 35 is 20's **successor**
  - 16 is 20's predecessor

Case 3: 2 Child nodes



- 3 critical observations about 20’s successor
  - As 20 has a right subtree, then the successor is the min node of the right subtree
    - This is the easy case for a successor
  - By definition, this successor:
    - Has either no children, or a right child
    - It is a min node, so it CAN'T have a left child
  - If we replace a node with its successor node, then the BST will remain valid
    - As it is > than the entire left subtree content
    - And also <= than all of the right subtree

Case 3: 2 Child nodes



- So successor(20) = min(right) = 35
- Let’s replace 20 with 35
- All that remains is to remove the successor node (35)
  - Either 0-children node or 1 RIGHT child node
- Overall, in the 2 child node case:
  - **Find** the successor in the right subtree
  - **Replace** the node value with that successor
  - **Delete** the actual successor node
    - Which has only 0-1 children
- Tip: we can also use predecessor instead of successor

Lazy Processing Trick

- The lazy trick is very common in many data structures
- Instead of doing the operation now, we delay it until later
- Simply add a bool flag in each node to indicate whether it is deleted or not
- Whenever a node is deleted, just mark bool as true, without completely removing it
- After each N deletion, rebuild the whole tree
- This way:
  - We did not code a risky deletion
  - However, this approach can be less efficient sometimes
- Observe: min/max/successor/search/Insert/delete are all O(h) time
  - If the tree is balanced: it's very efficient! If the tree is degenerate ⇒ O(n)

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."