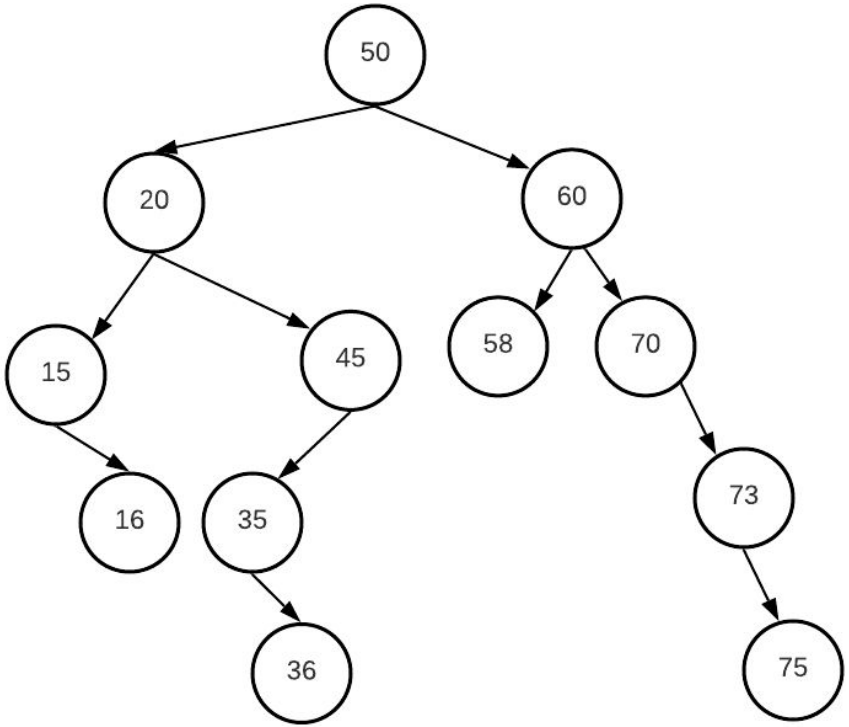
Data Structures

BST Deletion 1

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

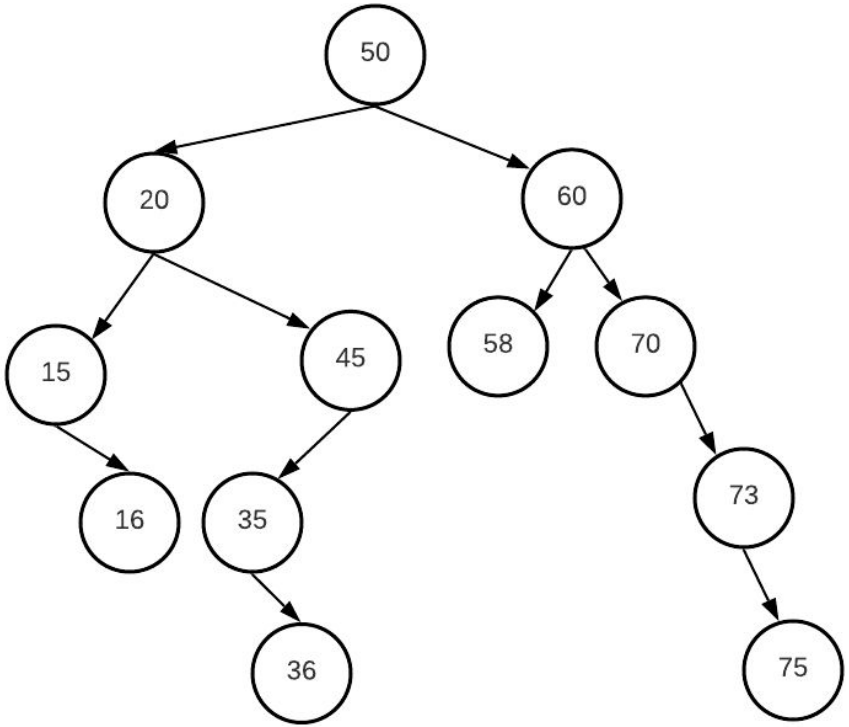
Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Node deletion



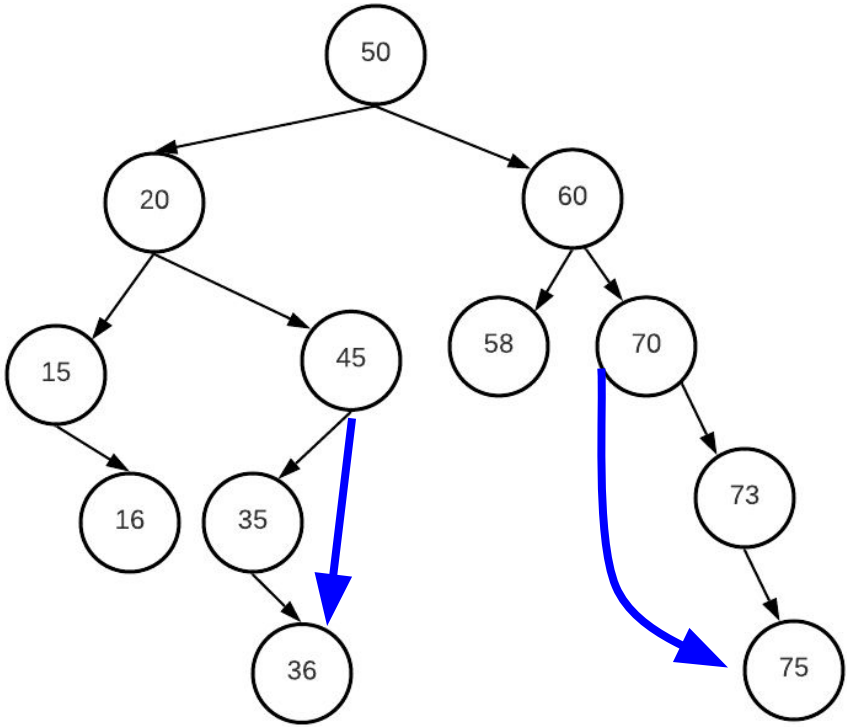
- Assume we have a tree of 2+ elements
- We want to delete a specific value
  - The remaining tree must be BST
- We have 3 cases:
  - 0 children (75) ⇒ Direct
  - 1 child (73) ⇒ Almost Direct
  - 2 children (20) ⇒ A bit tricky

Case 1: 0 Children node



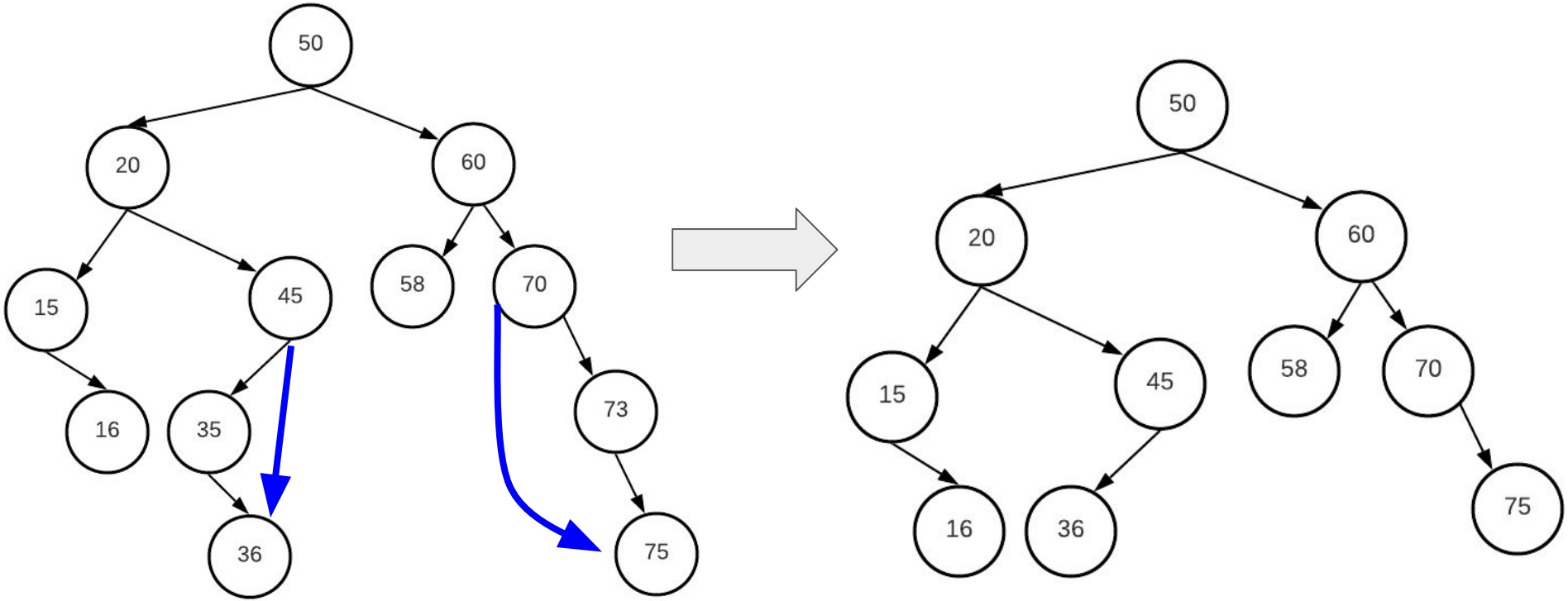
- If the node is leaf node, this is very straight forward. Just remove it
- Examples: 75, 58, 36, 16
  - Properly set parent child as null
    - was on parent left? set parent->left = null
    - Similarly, if was a right child
  - Then free the node!

Case 2: 1 Child node

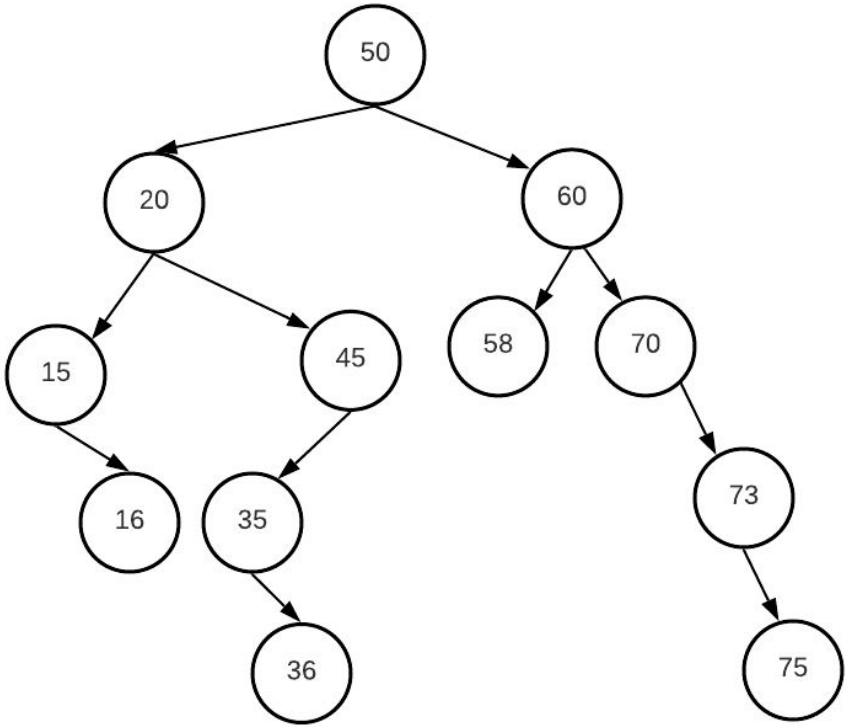


- If a node has only a single child, we can connect its parent with its 1 child
- Examples: 70, 73, 45, 35, 15
- Consider 73:
  - It is right of 70
  - After removal: 70's right = 75
- Consider 35:
  - It is left of 45
  - After removal: 45's left = 36
    - We don't care that it was 35's right

Case 2: 1 Child node

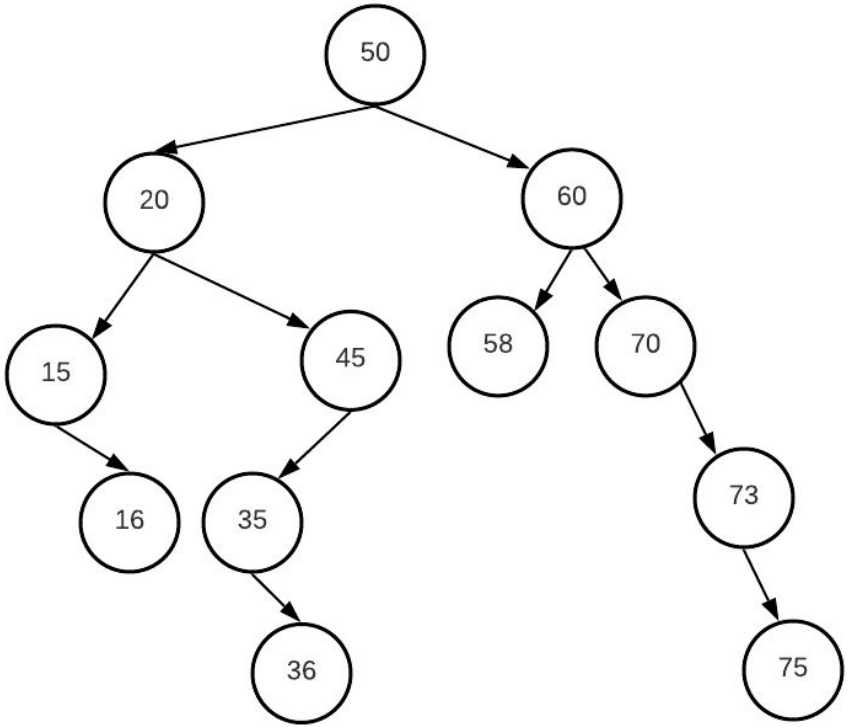


Case 3: 2 Children node



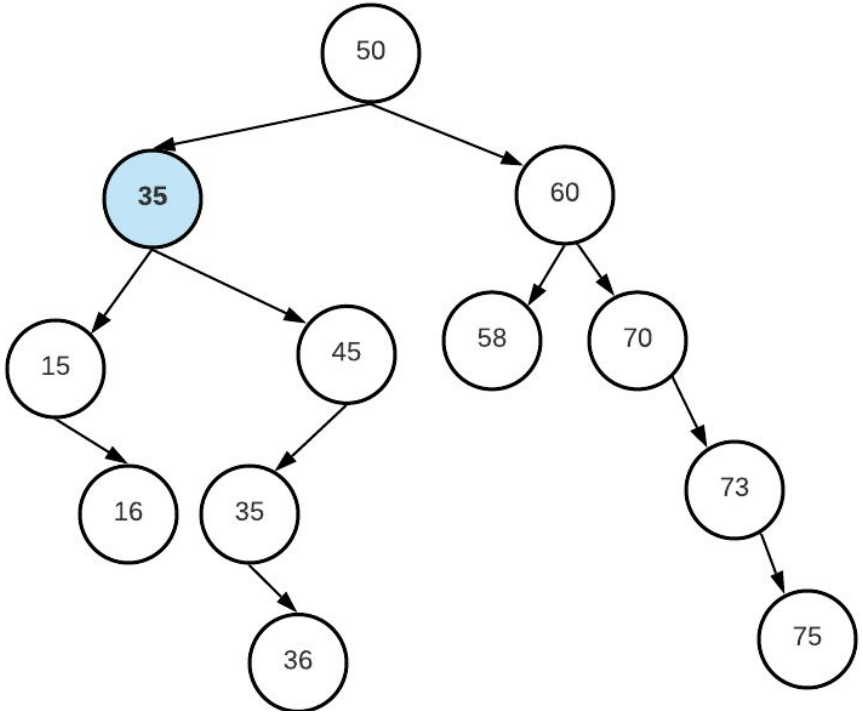
- Challenge: We need to make sure the tree remains BST
- Examples: 20, 50, 60
- What is this tree inorder traversal?
  - 15 16 **20** 35 36 45 80 58 60 70 73 75
- Consider node 20
  - After removal the inorder is
  - 15 **16 35** 36 45 80 58 60 70 73 75
- Observe: 35 is 20's **successor**
  - 16 is 20's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

Case 3: 2 Children node



- 3 critical observations about 20’s successor
  - As 20 has right, then it is = min(right subtree)
    - This is the easy case for a successor
  - By definition, this successor
    - Either has no children or a right child
    - It can’t have left child, otherwise it is not min
  - If we replaced a node with its successor, then BST is still valid
    - As it is > than all left subtree
    - And also <= than all right subtree

Case 3: 2 Children node



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

Lazy Processing Trick

- The lazy trick is very common in many data structures
- Instead of doing the operation now, we delay it later
- Here is it
- Add a bool flag in each node if it is deleted or not
- Whenever a node is deleted, just mark bool as true, with no remove
- After each N deletion, rebuild the whole tree
- This way:
  - We did not code a risky deletion
  - But, it might not be so efficient
- Observe: min/max/successor/search/Insert/delete are all O(h) time
  - If tree is balanced = very efficient. If tree is degenerate ⇒ O(n)

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."