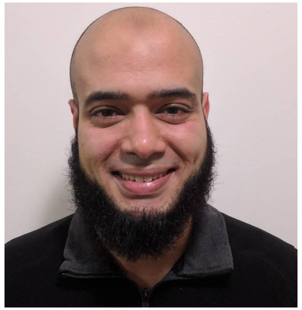
Data Structures
AVL Deletion

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

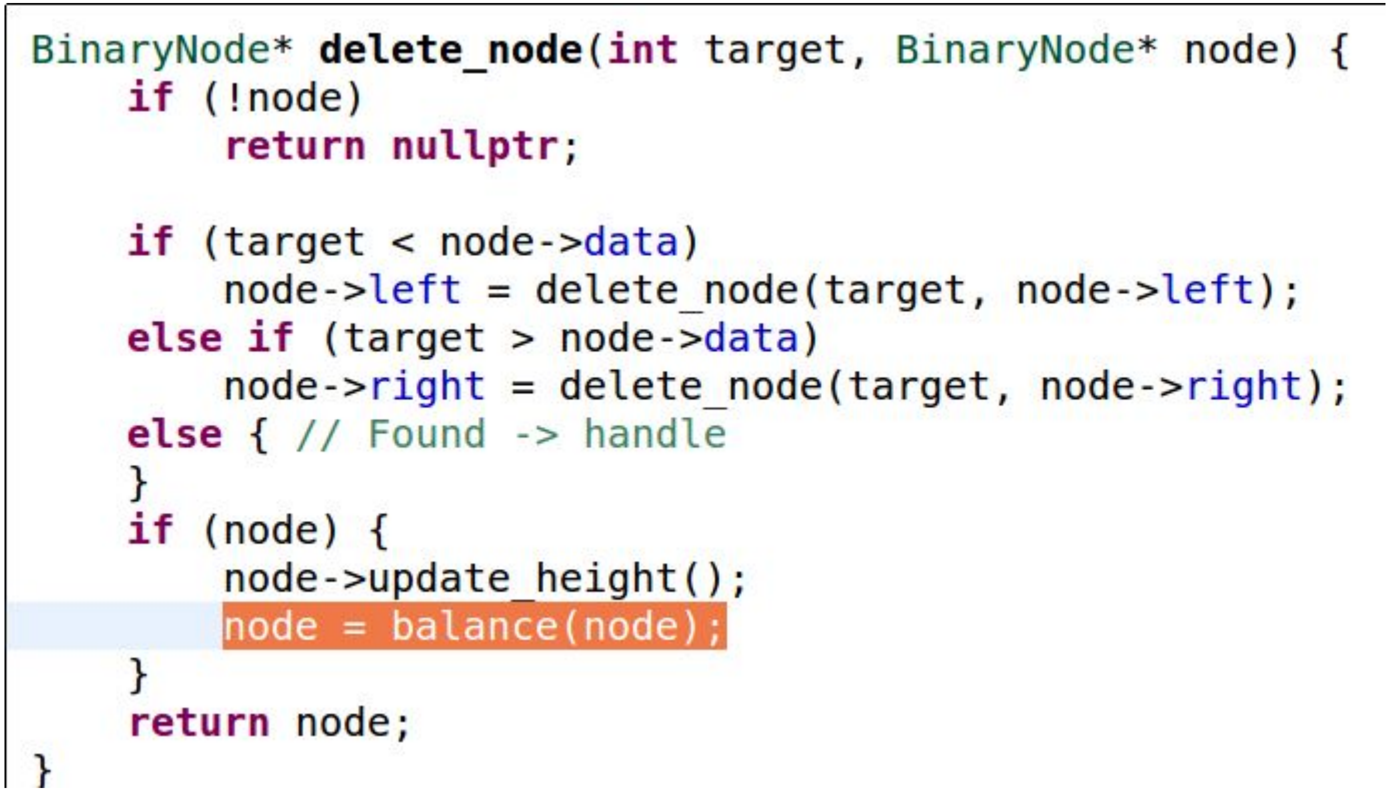
Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Node deletion

- Recall deletion has several cases (successor replacement in worst case)
- In fact, we follow same logic as insertion. If a tree node has changes
    - Update height. Do rebalance. That is it ⇒ Simple code changes

Balancing nodes

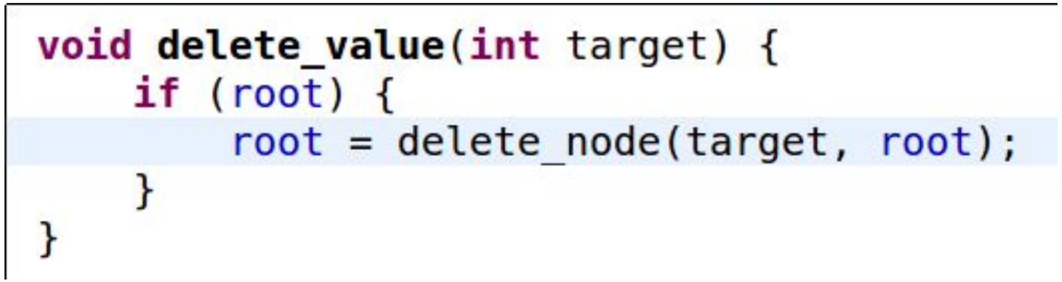


```
BinaryNode* delete_node(int target, BinaryNode* node) {
    if (!node)
        return nullptr;

    if (target < node->data)
        node->left = delete_node(target, node->left);
    else if (target > node->data)
        node->right = delete_node(target, node->right);
    else { // Found -> handle
    }
    if (node) {
        node->update_height();
        node = balance(node);
    }
    return node;
}
```

No changes to caller

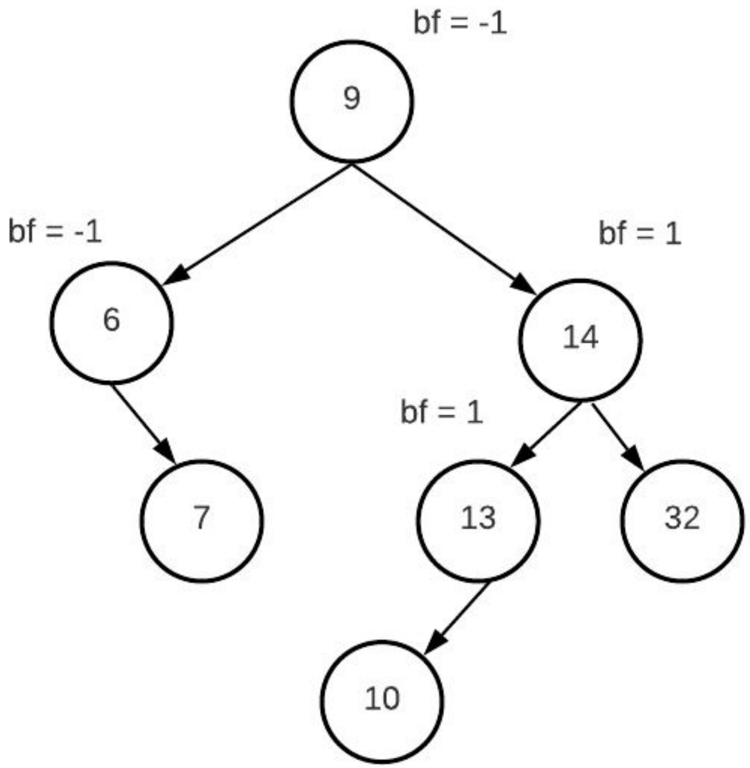
- Our old code already ready for changes



```
void delete_value(int target) {
    if (root) {
        root = delete_node(target, root);
    }
}
```

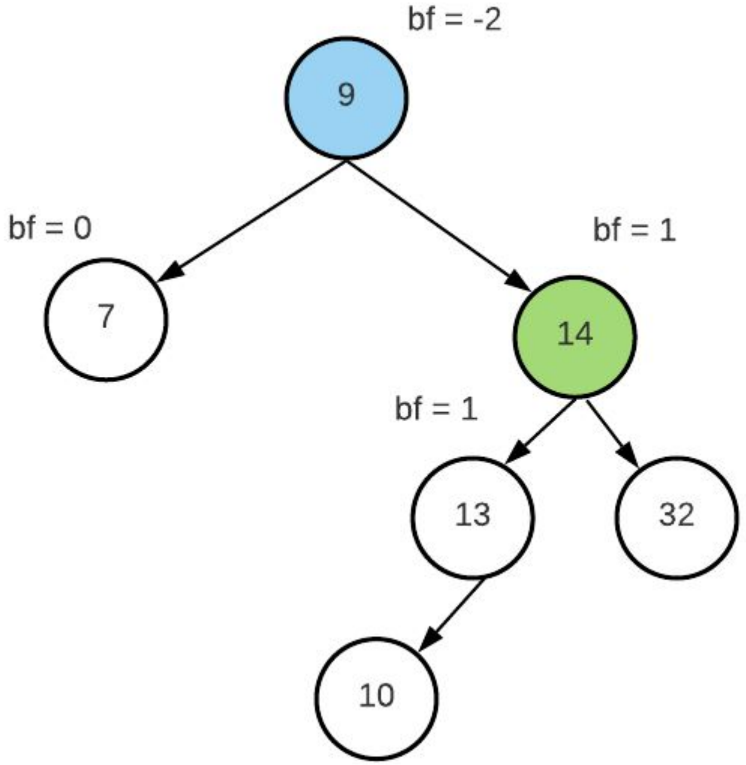
Let’s simulate

- Let’s create this tree first
  - Insert: 9 6 14 7 13 32 10
    - Tip: level-order traversal
- Next delete 6
  - 6 has 1 child only
  - Connect parent (9) to child (7)
  - Compute BF for 9



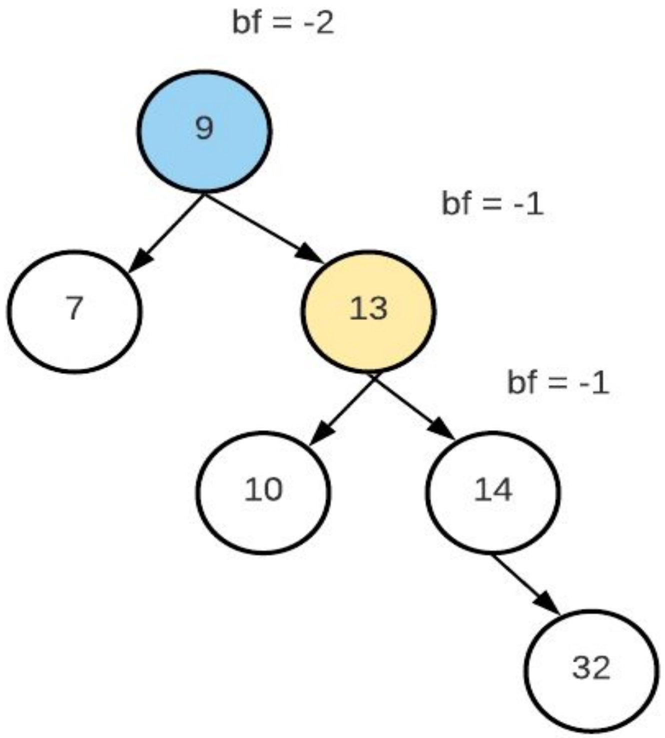
Delete 6

- BF [-2, 1] ⇒ Right-Left imbalanced tree
  - Right-Rotation(14)
    - Get’s 14 down and 13 up
  - Then Left-Rotation(9)



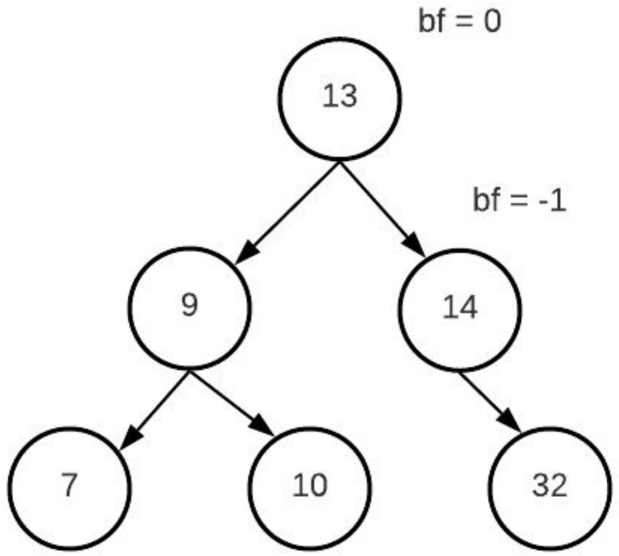
Delete 6

- Now it became BF [-2, -1]
  - Right-Right imbalance
- Left-Rotation(9)
  - Get 9 down and 13 up
  - B-subtree(10)
    - Moves from left of 13 to right of 9



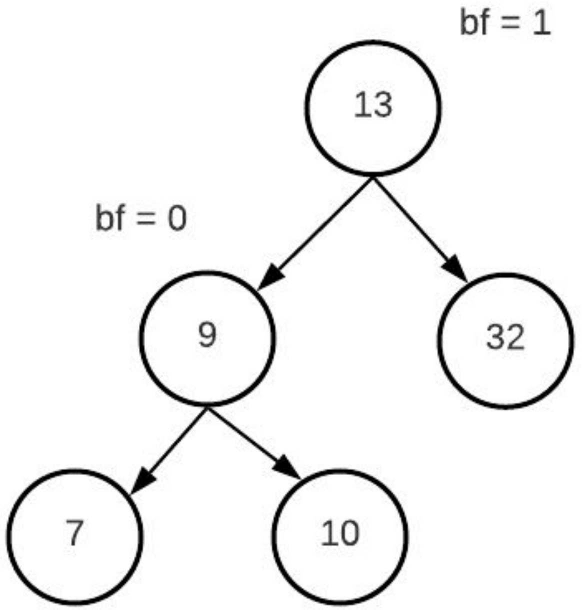
Delete 6

- Now fixed!
- Next delete 14
    - Again link 13 to 32
    - Update BF and check balance



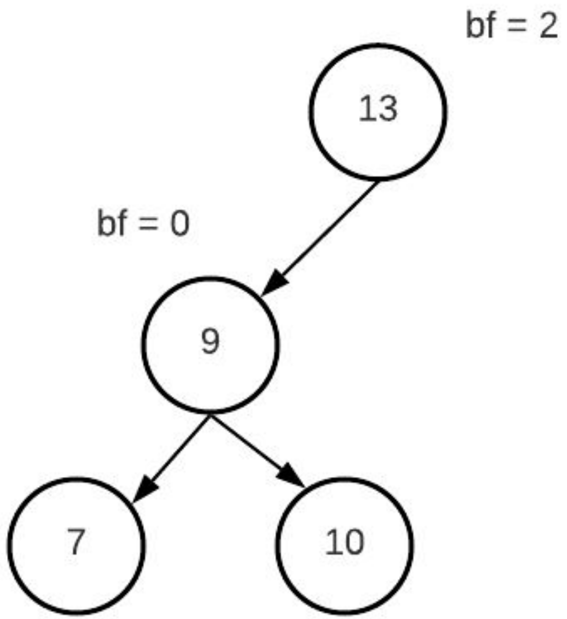
Delete 14

- No problems after deletion
- Next delete 32
  - Leaf node, just delete
  - Update height and check balance!



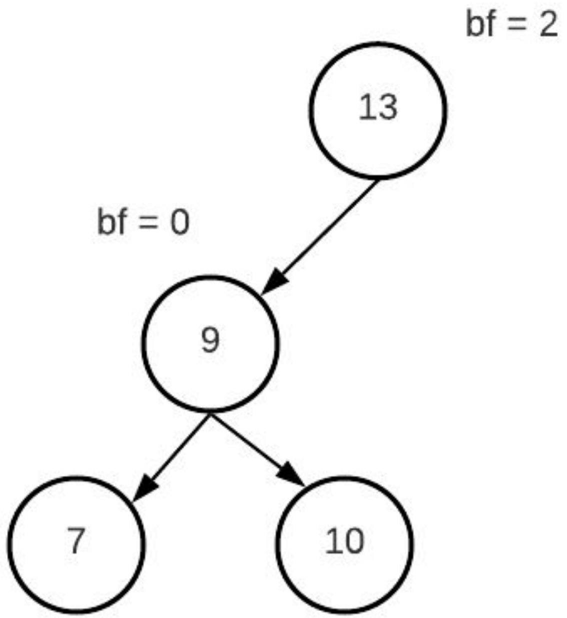
Delete 32

- Now we have imbalance
- Bf (2) so we need to check left
- In insertion left is either {1 or -1}
    - 1 ⇒ Left-Left rotation
    - -1 ⇒ Left-right rotation (perform double rotation)
- Deletion here created NEW scenario!
    - BF = 0
    - So now possible values: {-1, 0, 1}
    - It simply means **both** left-left or left-right are ok
    - Going left-left is more efficient
    - This means our balance code **don’t need** changes
        - It only check for left-right and right-left



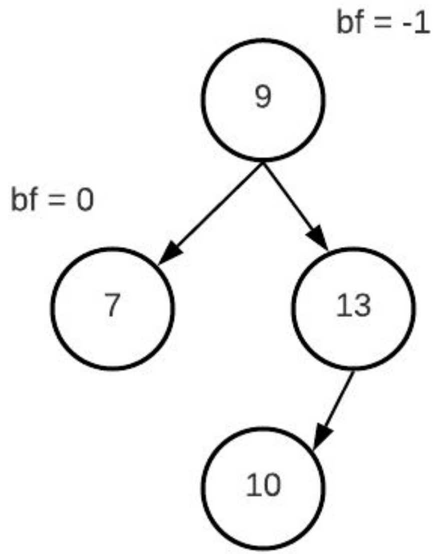
Delete 32

- BF [2, 0] ⇒ Left-left rotation
- Perform Right-rotation(13)
  - Get 13 down and 9 up
  - Subtree B(10) changes parent
-



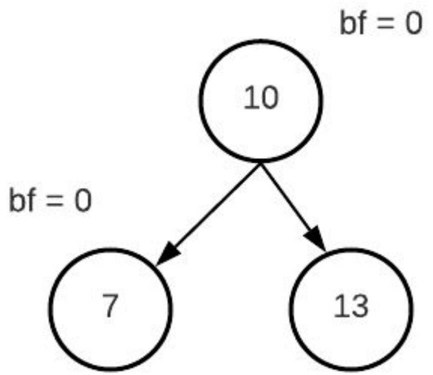
Delete 32

- Now fixed
- Next delete 9
    - 9 has 2 children
    - Find successor(9) = 10
    - Copy successor value
    - Remove node (10), which has no child
- So deletion in successor case ends up
  Just delete a node with 0-1 child as we learned



Done

- As we see deletion effect is direct
- Only it created the BF=0 case
    - BF {2, 0}
    - BF {-2, 0}
- However this case can be handled easily same as
    - BF {2, 1}
    - BF {-2, -1}
    - So balance code did not change
        - *Review code and verify*



“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”