Data Structures

BST Deletion 2

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)

Deletion Implementation

- It is not easy to code a clean code for it
- Implementation depends: Does not has a parent link or not
- If no parent link (my style)
  - You either send the node* with reference to be able to affect the parent pointer
  - Another clever way: let the deletion node return the updated tree, and use it (My way)
- Due to my tree style, I assume there is 2+ nodes in the tree
  - If 1 node, we need to destroy this

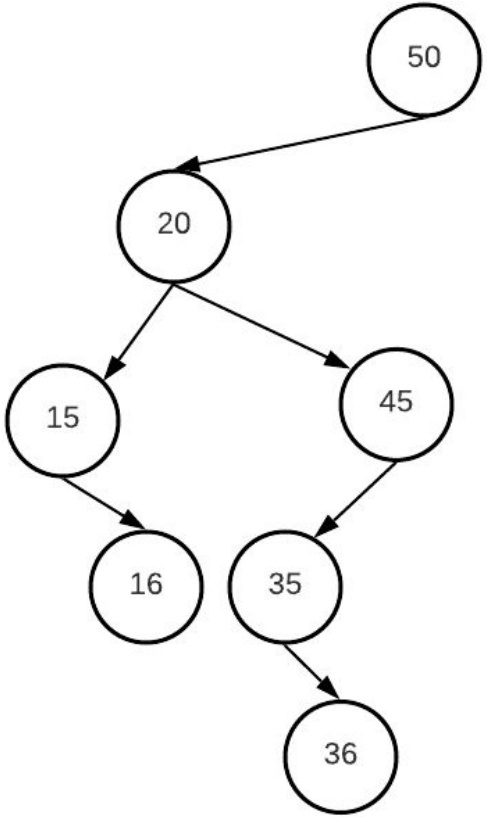
```
BinarySearchTree* min_node() {
    BinarySearchTree* cur = this;
    while (cur && cur->left)
        cur = cur->left;
    return cur;
}

// Delete the target node in the tree and return updated tree
// Caller use updated tree to re-link with children!
BinarySearchTree* delete_node(int target, BinarySearchTree* node) {
```

```
void delete_value(int target) {
    if (target == data && !left && !right)
        return; // can't remove root in this structure
    delete_node(target, this);
}
```

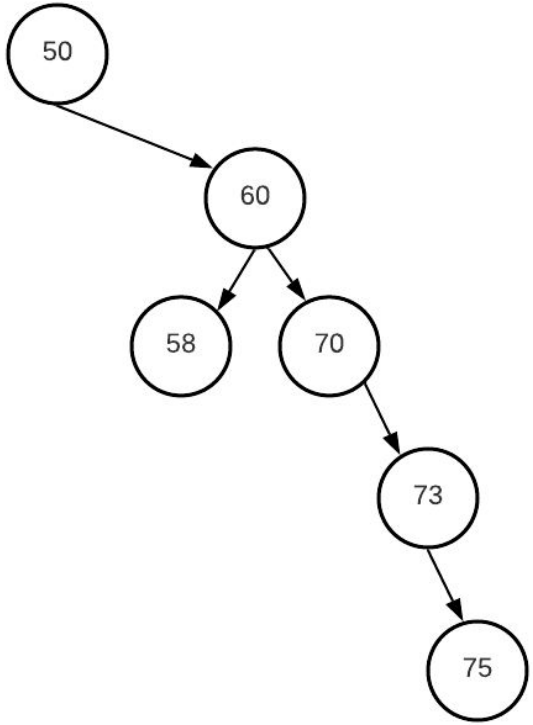
```
BinarySearchTree* delete_node(int target, BinarySearchTree* node) {
    if (!node)
        return nullptr;

    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 the node: Handle deletion
    }
    return node;
}
```

```
// Found the node: Handle deletion
BinarySearchTree* tmp = node;

if (!node->left && !node->right)    // case 1: no child
    node = nullptr;
else if (!node->right)  // case 2: has left only
    node = node->left;       // connect with child
else if (!node->left)   // case 2: has right only
    node = node->right;
else {  // 2 children: Use successor
    BinarySearchTree* mn = node->right->min_node();
    node->data = mn->data;  // copy & go delete
    node->right = delete_node(node->data, node->right);
    tmp = nullptr;
}
if (tmp)
    delete tmp;
```

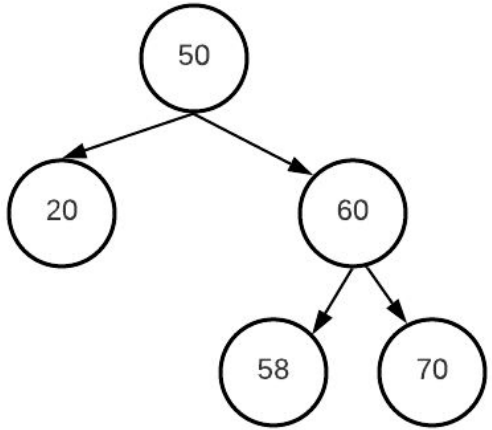
```
BinarySearchTree* delete_node(int target, BinarySearchTree* node) {
    if (!node)
        return nullptr;

    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 the node: Handle deletion
        BinarySearchTree* tmp = node;

        if (!node->left && !node->right)    // case 1: no child
            node = nullptr;
        else if (!node->right)  // case 2: has left only
            node = node->left;      // connect with child
        else if (!node->left)   // case 2: has right only
            node = node->right;
        else {  // 2 children: Use successor
            BinarySearchTree* mn = node->right->min_node();
            node->data = mn->data;  // copy & go delete
            node->right = delete_node(node->data, node->right);
            tmp = nullptr;
        }
        if (tmp)
            delete tmp;
    }
    return node;
}
```



```
BinarySearchTree* delete_node(int target, BinarySearchTree* 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 the node: Handle deletion
        BinarySearchTree* tmp = node;

        if (!node->left && !node->right)    // case 1: no child
            node = nullptr;
        else if (!node->right)  // case 2: has left only
            node = node->left;      // connect with child
        else if (!node->left)   // case 2: has right only
            node = node->right;
        else {  // 2 children: Use successor
            BinarySearchTree* mn = node->right->min_node();
            node->data = mn->data;  // copy & go delete
            node->right = delete_node(node->data, node->right);
            tmp = nullptr;
        }
        if (tmp)
            delete tmp;
    }
    return node;
}
```



```
BinarySearchTree* delete_node(int target, BinarySearchTree* 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 the node: Handle deletion
        BinarySearchTree* tmp = node;

        if (!node->left && !node->right)    // case 1: no child
            node = nullptr;
        else if (!node->right)  // case 2: has left only
            node = node->left;      // connect with child
        else if (!node->left)   // case 2: has right only
            node = node->right;
        else {  // 2 children: Use successor
            BinarySearchTree* mn = node->right->min_node();
            node->data = mn->data;  // copy & go delete
            node->right = delete_node(node->data, node->right);
            tmp = nullptr;
        }
        if (tmp)
            delete tmp;
    }
    return node;
}
```



"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."