Deletion in Binary Tree

Deleting a node in BST

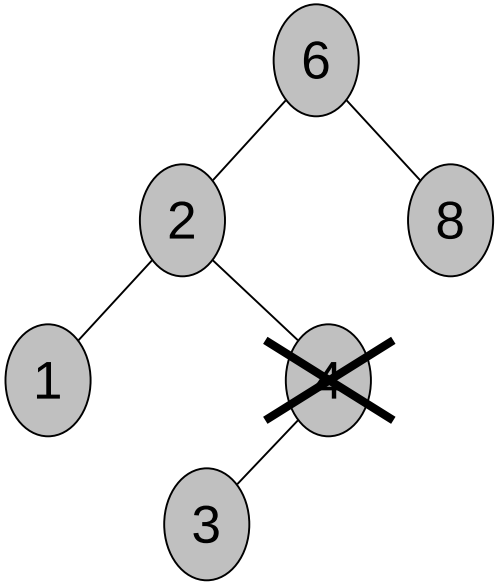
- As is common with many data structures, the hardest operation is deletion.
- Once we have found the node to be deleted, we need to consider several possibilities.
- Case 1:

If the node is a *leaf*, it can be deleted immediately.

Deleting a node in BST

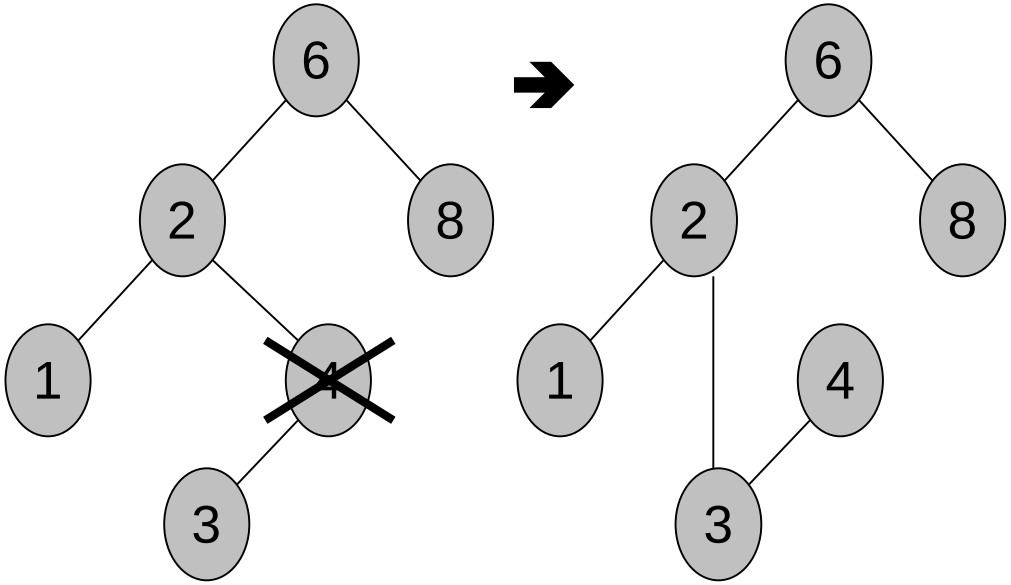
- Case 2:

If the node has one child, the node can be deleted after its parent adjusts a pointer to bypass the node and connect to inorder successor.



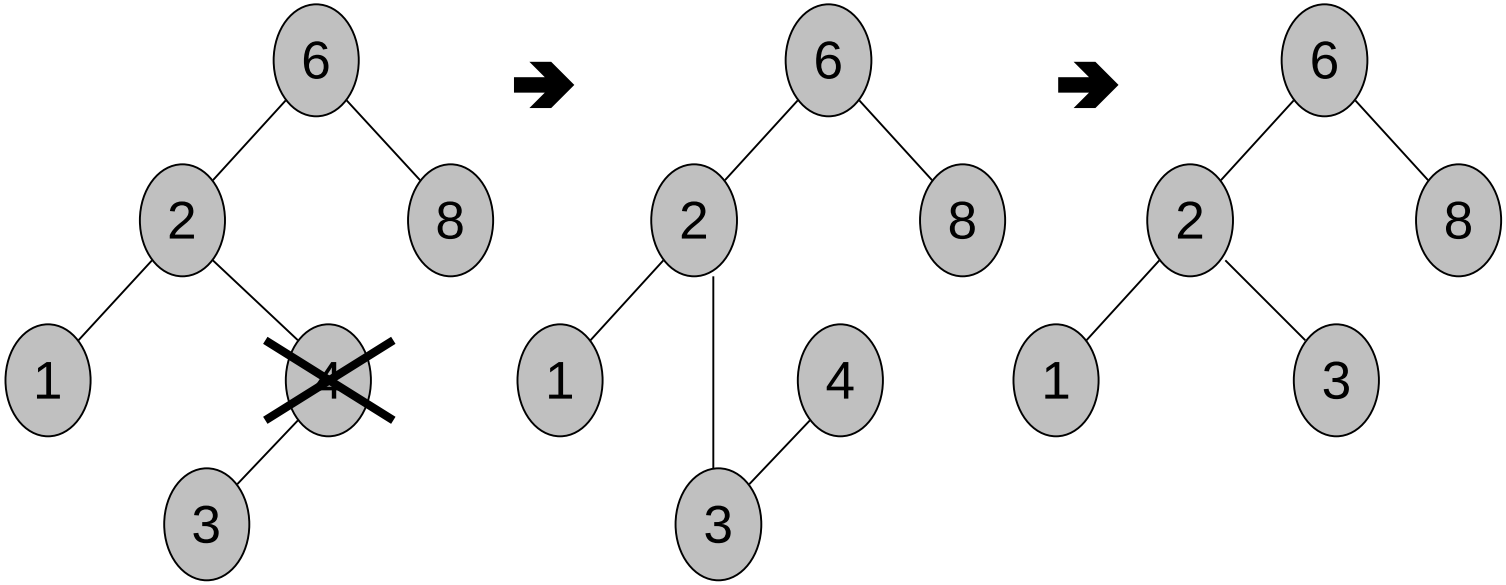
Deleting a node in BST

- The inorder traversal order has to be maintained after the delete.



Deleting a node in BST

- The inorder traversal order has to be maintained after the delete.

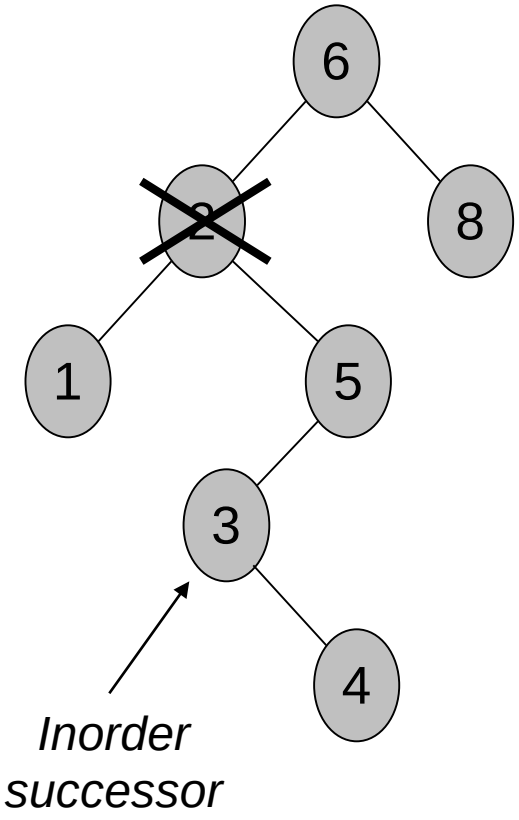


Deleting a node in BST

- Case 3:
- The complicated case is when the node to be deleted has both left and right subtrees.
- The strategy is to replace the data of this node with the smallest data of the right subtree and recursively delete that node.

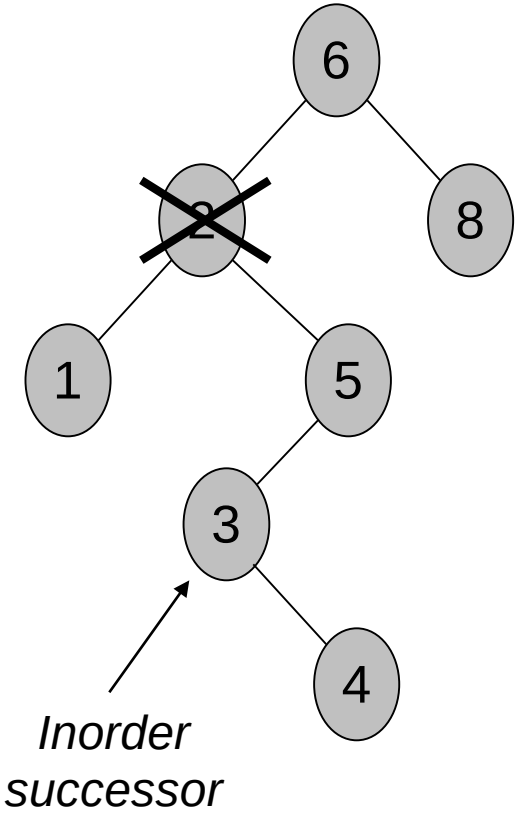
Deleting a node in BST

Delete(2): locate inorder successor



Deleting a node in BST

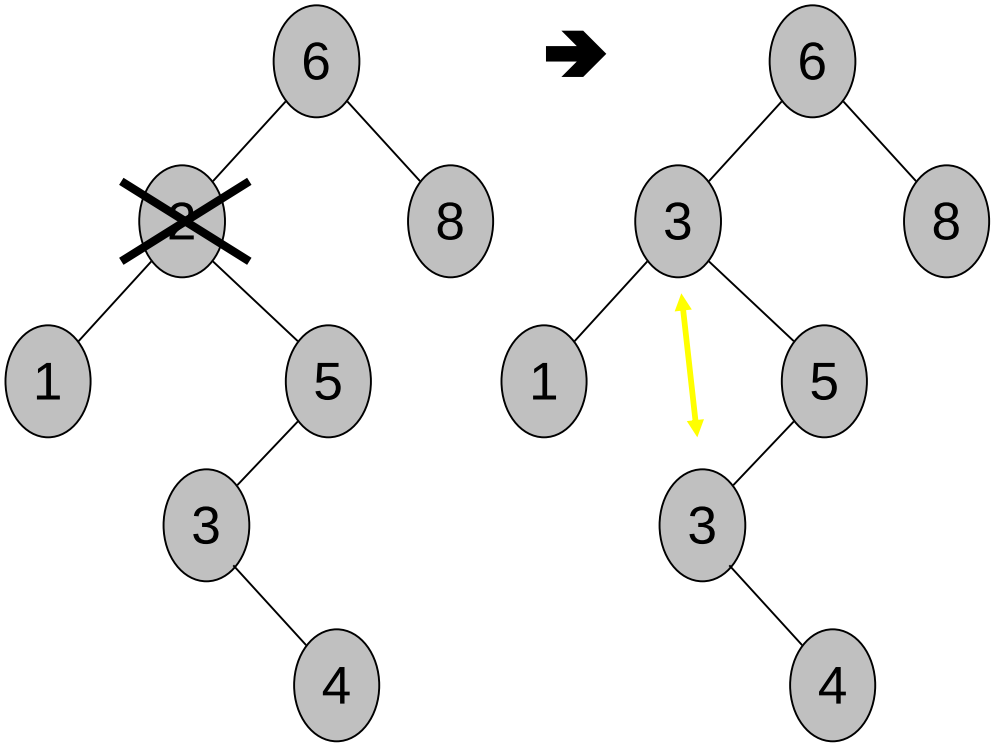
Delete(2): locate inorder successor



- Inorder successor will be the left-most node in the right subtree of 2.
- The inorder successor will not have a left child because if it did, that child would be the left-most node.

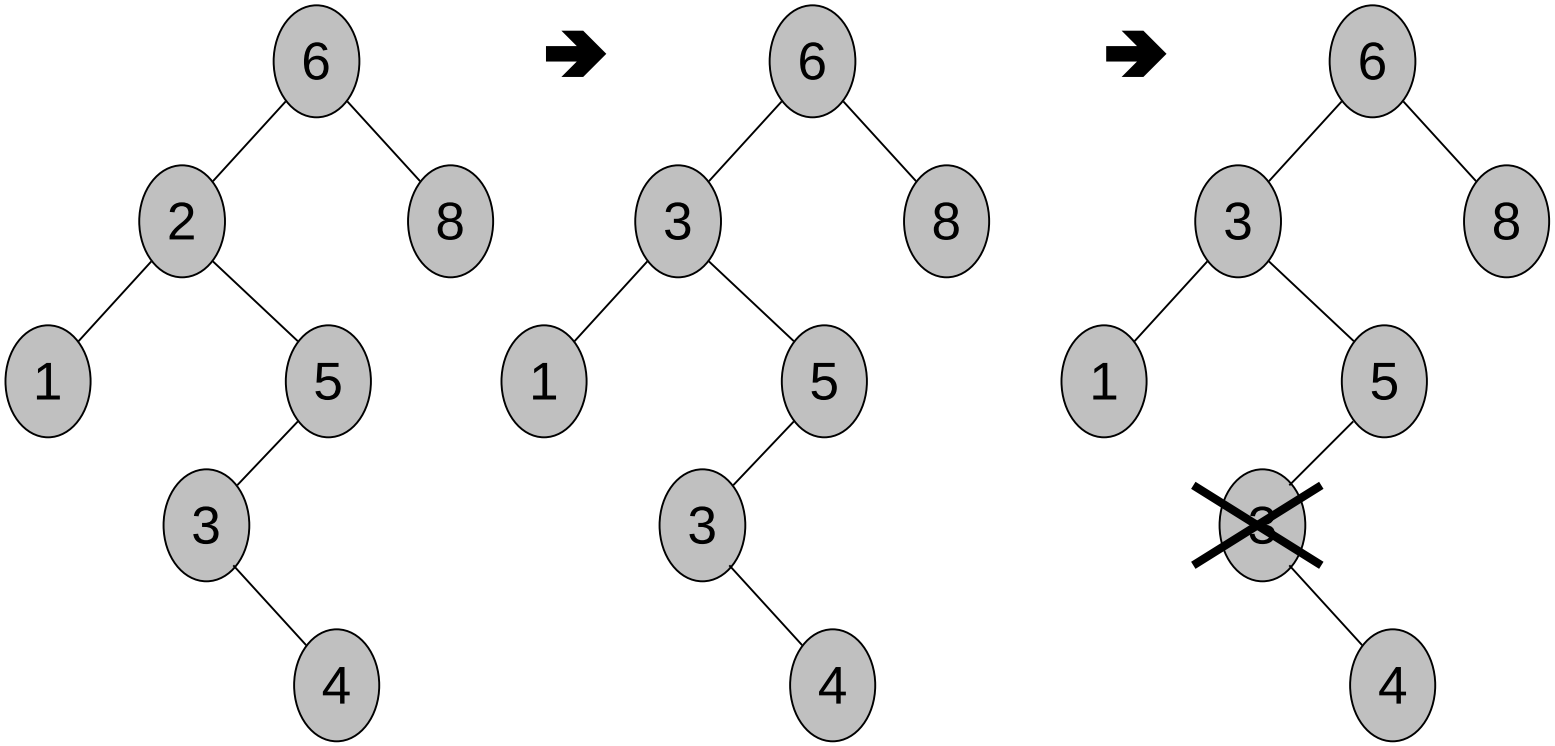
Deleting a node in BST

Delete(2): copy data from inorder successor



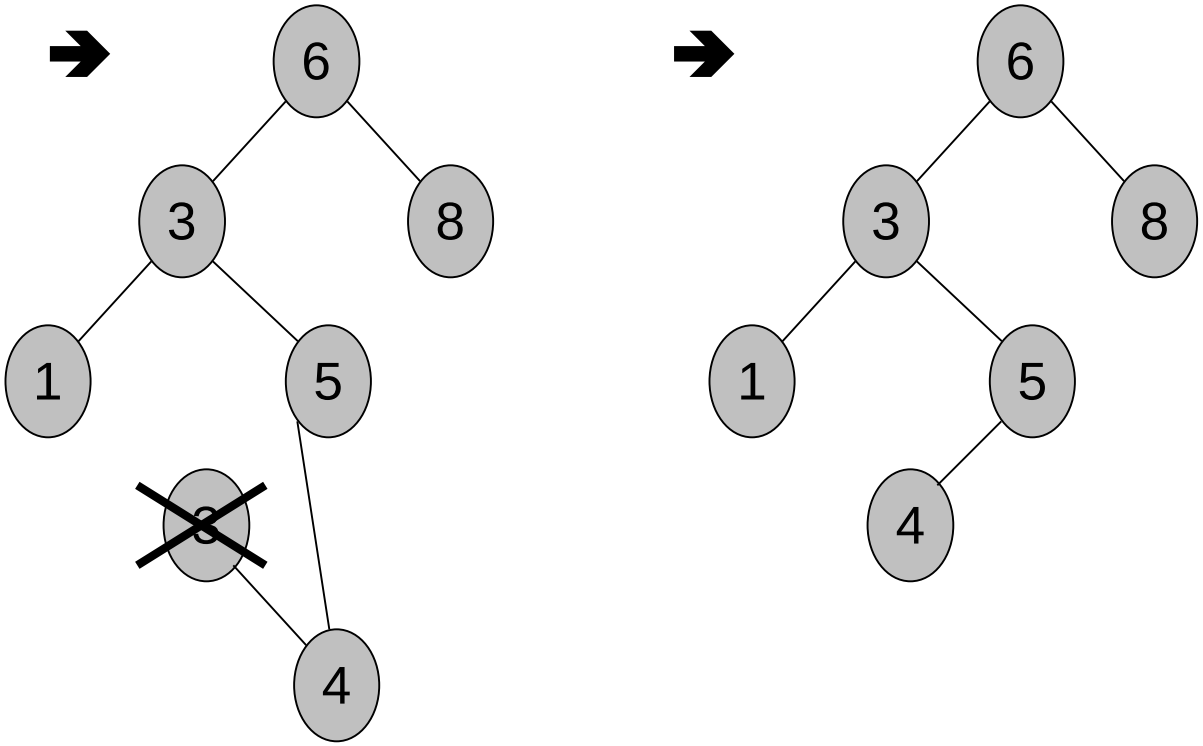
Deleting a node in BST

Delete(2): remove the inorder successor



Deleting a node in BST

Delete(2)



C++ code for delete

```
TreeNode* remove(TreeNode * tree, int info)
{
    TreeNode * t;
      if( info < tree->getinfo() ){
        t = remove(tree->getLeft(), info);
        tree->setLeft( t );
    }
    else if(info > tree->getinfo() ){
        t = remove(tree->getRight(), info);
        tree->setRight( t );
    }
```

C++ code for delete

```
TreeNode* remove(TreeNode * tree, int info)
{
    TreeNode * t;
      if( info < tree->getinfo() ){
        t = remove(tree->getLeft(), info);
        tree->setLeft( t );
    }
    else if(info > tree->getinfo() ){
        t = remove(tree->getRight(), info);
        tree->setRight( t );
    }
```

C++ code for delete

```
TreeNode* remove(TreeNode * tree, int info)
{
    TreeNode * t;
      if( info < tree->getinfo() ){
        t = remove(tree->getLeft(), info);
        tree->setLeft( t );
    }
    else if(info > tree->getinfo() ){
        t = remove(tree->getRight(), info);
        tree->setRight( t );
    }
```

C++ code for delete

```
TreeNode* remove(TreeNode * tree, int info)
{
    TreeNode * t;
      if( info < tree->getinfo() ){
        t = remove(tree->getLeft(), info);
        tree->setLeft( t );
    }
    else if(info > tree->getinfo() ){
        t = remove(tree->getRight(), info);
        tree->setRight( t );
    }
```

C++ code for delete

```
//two children, replace with inorder successor
else if(tree->getLeft() != NULL
        && tree->getRight() != NULL ){
    TreeNode * minNode;
    minNode = findMin(tree->getRight());
    tree->setInfo( minNode->getInfo() );
    t = remove(tree->getRight(),(minNode->getInfo()));
    tree->setRight( t );
}
```

C++ code for delete

```
//two children, replace with inorder successor
   else if(tree->getLeft() != NULL
           && tree->getRight() != NULL ){
        TreeNode * minNode;
        minNode = findMin(tree->getRight());
        tree->setInfo( minNode->getInfo() );
        t = remove(tree->getRight(),(minNode->getInfo()));
        tree->setRight( t );
   }
```

C++ code for delete

```
TreeNode* findMin(TreeNode * tree)
{
    if( tree == NULL )
        return NULL;
    if( tree->getLeft() == NULL )
        return tree; // this is it.
    return findMin( tree->getLeft() );
}
```

C++ code for delete

```
TreeNode* findMin(TreeNode * tree)
{
    if( tree == NULL )
        return NULL;
    if( tree->getLeft() == NULL )
        return tree; // this is it.
    return findMin( tree->getLeft() );
}
```

C++ code for delete

```
//two children, replace with inorder successor
   else if(tree->getLeft() != NULL
           && tree->getRight() != NULL ){
       TreeNode * minNode;
       minNode = findMin(tree->getRight());
       tree->setInfo( minNode->getInfo() );
       t = remove(tree->getRight(),(minNode->getInfo()));
       tree->setRight( t );
   }
```

C++ code for delete

```
//two children, replace with inorder successor
   else if(tree->getLeft() != NULL
           && tree->getRight() != NULL ){
       TreeNode * minNode;
       minNode = findMin(tree->getRight());
       tree->setInfo( minNode->getInfo() );
       t = remove(tree->getRight(),(minNode->getInfo()));
       tree->setRight( t );
   }
```

C++ code for delete

```
//two children, replace with inorder successor
   else if(tree->getLeft() != NULL
           && tree->getRight() != NULL ){
       TreeNode * minNode;
       minNode = findMin(tree->getRight());
       tree->setInfo( minNode->getInfo() );
       t = remove(tree->getRight(),(minNode->getInfo()));
       tree->setRight( t );
   }
```

C++ code for delete

```
    else {  // case 1
        TreeNode* nodeToDelete = tree;
        if( tree->getLeft() == NULL ) //will handle 0 children
            tree = tree->getRight();
        else if( tree->getRight() == NULL )
            tree = tree->getLeft();
        else tree = NULL;

        delete nodeToDelete;
    }
    return tree;
}
```

C++ code for delete

```
    else {  // case 1
        TreeNode* nodeToDelete = tree;
        if( tree->getLeft() == NULL ) //will handle 0 children
            tree = tree->getRight();
        else if( tree->getRight() == NULL )
            tree = tree->getLeft();
        else tree = NULL;

        delete nodeToDelete;
    }
    return tree;
}
```

C++ code for delete

```
    else {  // case 1
        TreeNode<int>* nodeToDelete = tree;
        if( tree->getLeft() == NULL ) //will handle 0 children
            tree = tree->getRight();
        else if( tree->getRight() == NULL )
            tree = tree->getLeft();
        else tree = NULL;

        delete nodeToDelete;
    }
    return tree;
}
```