Operations on Binary Search Trees

Operations on Binary Search Trees

Following are the common operations that we can perform on the BSTs.

1. Insert a Node
2. Search a Value
3. Delete a Node
4. Traverse the Tree

Binary Search Trees

Let's make a class named BST.

```
struct TreeNode
{
    int val;
    TreeNode *left;
    TreeNode *right;
};
```

```
class binarySearchTree
{
    TreeNode *root;
public:
    binarySearchTree()
    {
        root = NULL;
    }
     TreeNode *createNode(int value)
    {
        TreeNode *record = new TreeNode();
        record->val = value;
        record->left = NULL;
        record->right = NULL;
        return record;
    }
```

Operations on Binary Search Trees

Following are the common operations that we can perform on the BSTs.

1. Insert a Node
2. Search a Value
3. Delete a Node
4. Traverse the Tree

Binary Search Trees

Let's insert a node in the BST.

```
void insert(TreeNode *node)
{
    TreeNode *prev = root;
    TreeNode *next = root;
    if (root == NULL)
    {
        root = node;
        return;
    }
    while (next != NULL)
    {
        prev = next;
        if (node->val < prev->val)
            next = prev->left;
        else
            next = prev->right;
    }
    if (node->val >= prev->val)
        prev->right = node;
    else
        prev->left = node;
}
```

Operations on Binary Search Trees

Following are the common operations that we can perform on the BSTs.

1. Insert a Node
2. Search a Value
3. Delete a Node
4. Traverse the Tree

Binary Search Trees

Let's search a value in the BST.

```
bool search(int value)
{
    TreeNode *temp = root;
    if (root == NULL)
    {
        return false;
    }
    while (temp != NULL)
    {
        if (temp->val == value)
            return true;
        if (value < temp->val)
            temp = temp->left;
        else
            temp = temp->right;
    }
    return false;
}
```

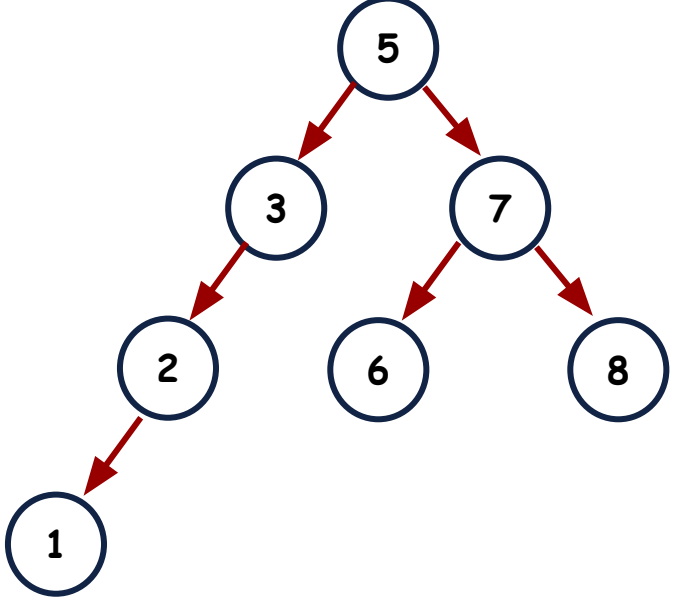
Operations on Binary Search Trees

Following are the common operations that we can perform on the BSTs.

1. Insert a Node
2. Search a Value
3. Delete a Node
4. Traverse the Tree

Binary Search Trees

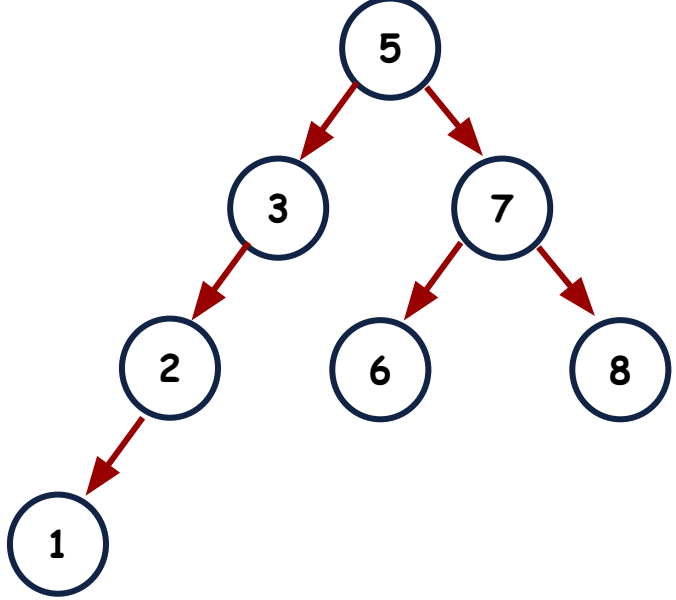
Let's delete a node in the BST.



Binary Search Trees

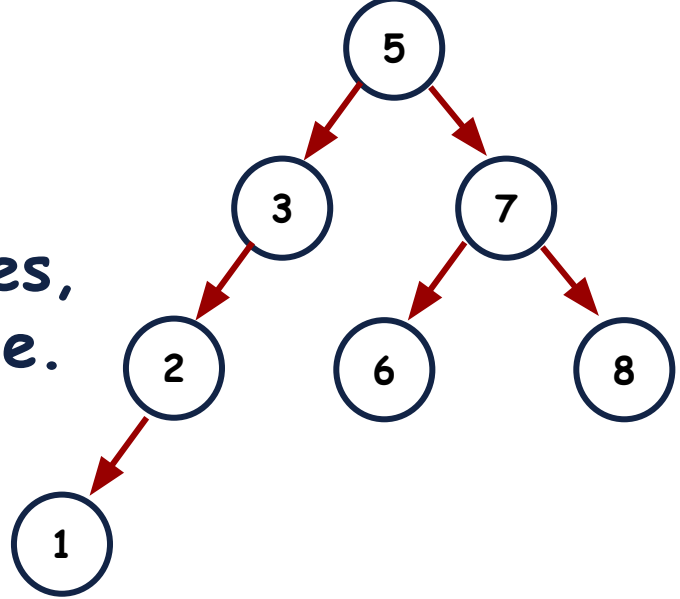
Let's delete a node in the BST.

Deleting a node is a little tricky. We should delete a node in such a way that the rest of the tree is again a BST.



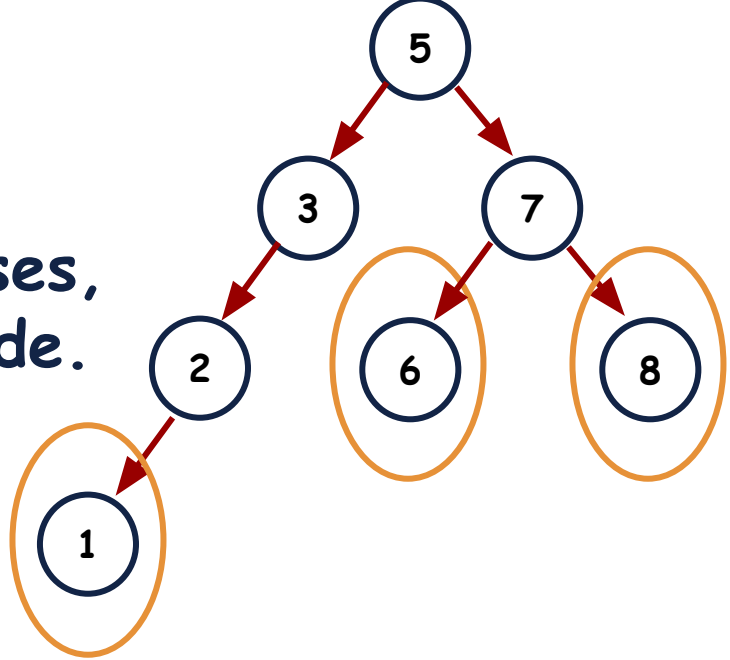
Binary Search Trees

There can be three different cases, according to the children of a node.



Binary Search Trees

There can be three different cases, according to the children of a node.

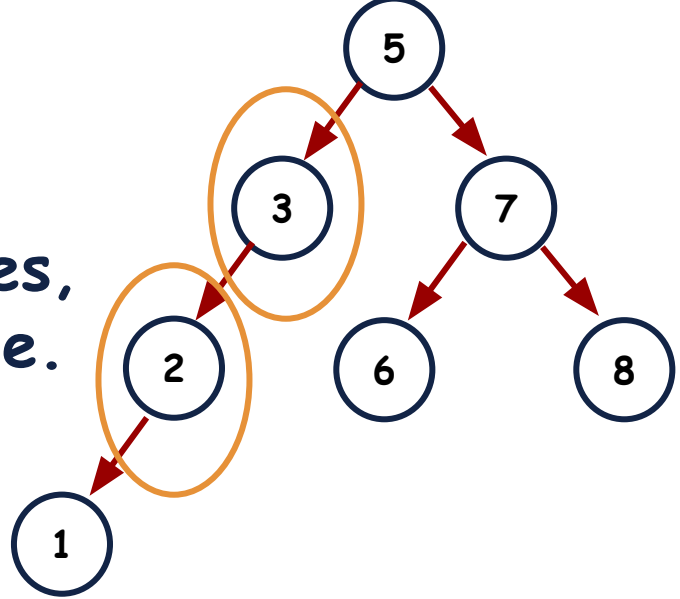


Case 1: (Easy One)

The node we want to delete has **no child**.

Binary Search Trees

There can be three different cases, according to the children of a node.

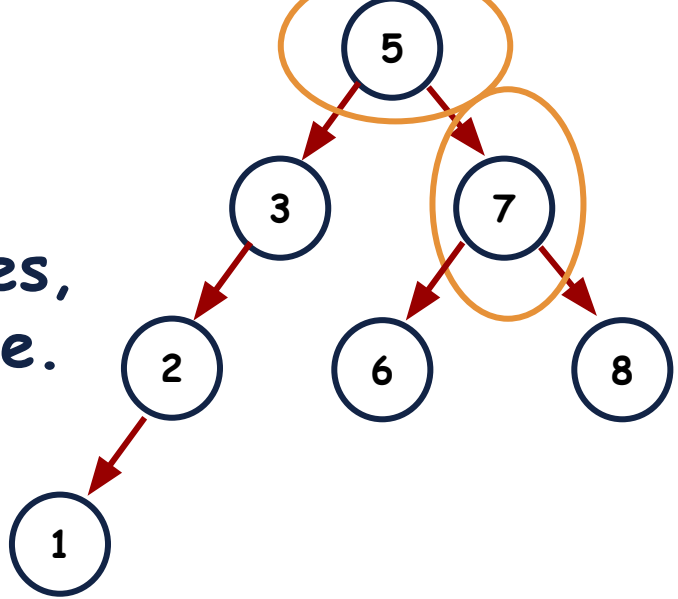


Case 2: (Moderate One)

The node we want to delete has **one child**.

Binary Search Trees

There can be three different cases, according to the children of a node.



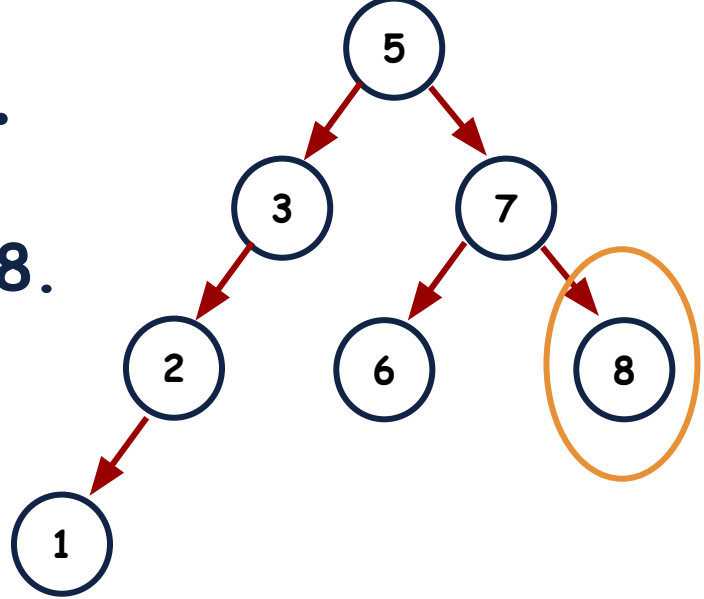
Case 3: (Difficult One)

The node we want to delete has **two children**.

Delete in BST: Case 1

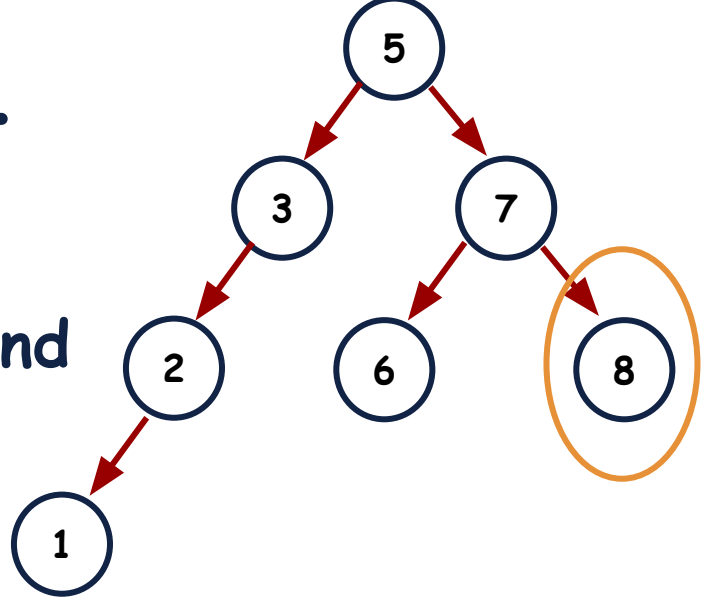
Let's say we want to delete node 8.
How can we do that?

OH, LET'S SEE...



Delete in BST: Case 1

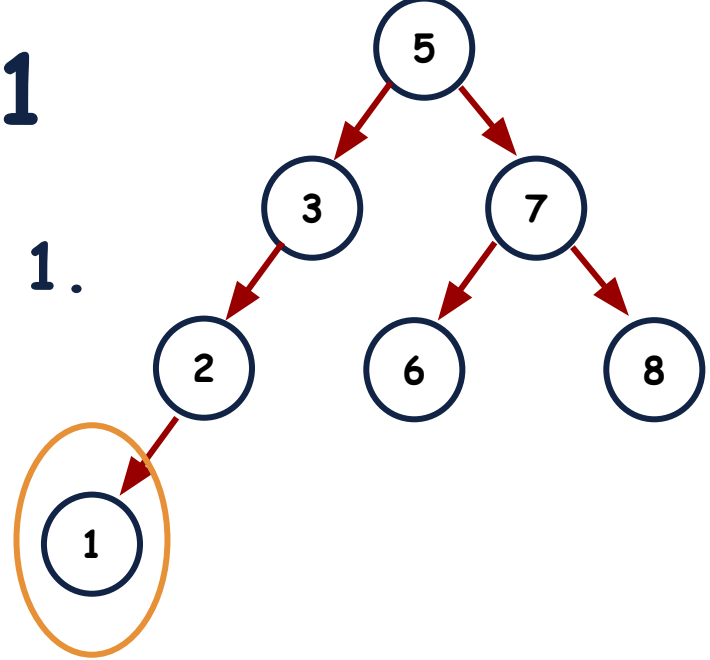
We just have to find the node containing the value 8, delete it and assign right pointer of the parent node to null.



Delete in BST: Case 1

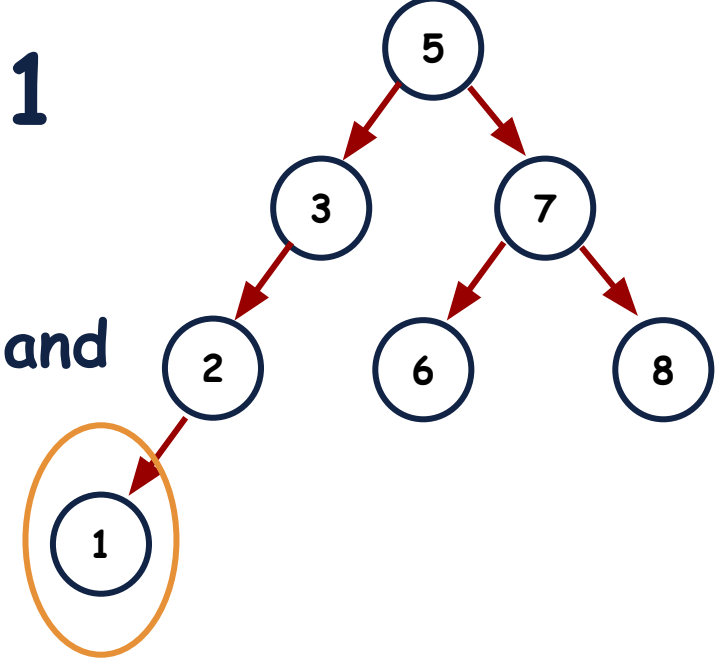
Let's say we want to delete node 1.
How can we do that?

OH, LET'S SEE...



Delete in BST: Case 1

We just have to find the node containing the value 1, delete it and assign left pointer of the parent node to null.



Delete in BST: Case 1

```
bool deleteValue(int value)
    {
        TreeNode *prev = root;
        TreeNode *next = root;
        while (next != NULL && next->val != value)
        {
            prev = next;
            if (value < prev->val)
                next = prev->left;
            else
                next = prev->right;
        }
        if (next == NULL)
        {
            cout << "Value not Found" << endl;
            return false;
        }
        else if (next->left == NULL && next->right == NULL)
        {
            if (next == prev->left)
                prev->left = NULL;
            else
                prev->right = NULL;
            delete next;
            return true;
        }
```

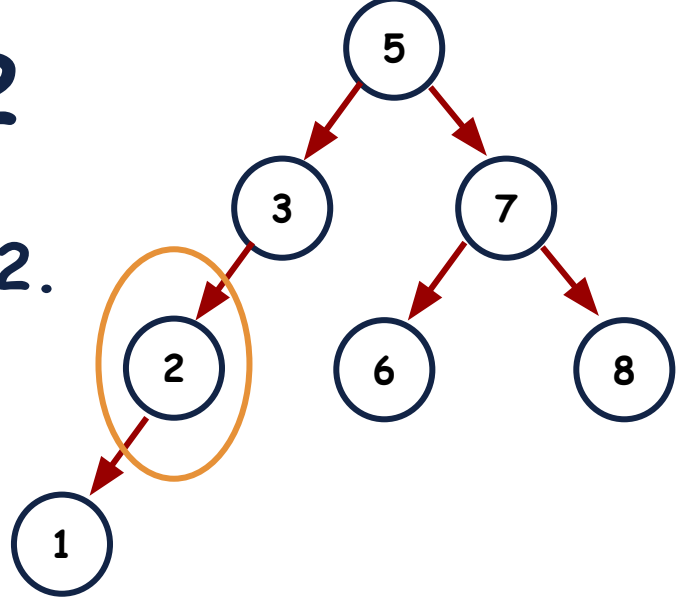
Delete in BST: Case 1

```
bool deleteValue(int value)
    {
        TreeNode *prev = root;
        TreeNode *next = root;
        while (next != NULL && next->val != value)
        {
            prev = next;
            if (value < prev->val)
                next = prev->left;
            else
                next = prev->right;
        }
        if (next == NULL)
        {
            cout << "Value not Found" << endl;
            return false;
        }
        else if (next->left == NULL && next->right == NULL)
        {
            if (next == prev->left)
                prev->left = NULL;
            else
                prev->right = NULL;
            delete next;
            return true;
        }
```

Delete in BST: Case 2

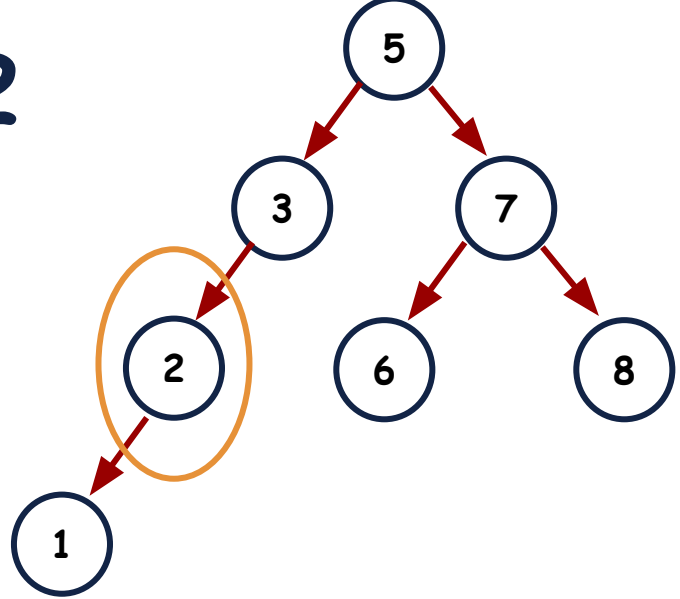
Let's say we want to delete node 2.
How can we do that?

OH, LET'S SEE...



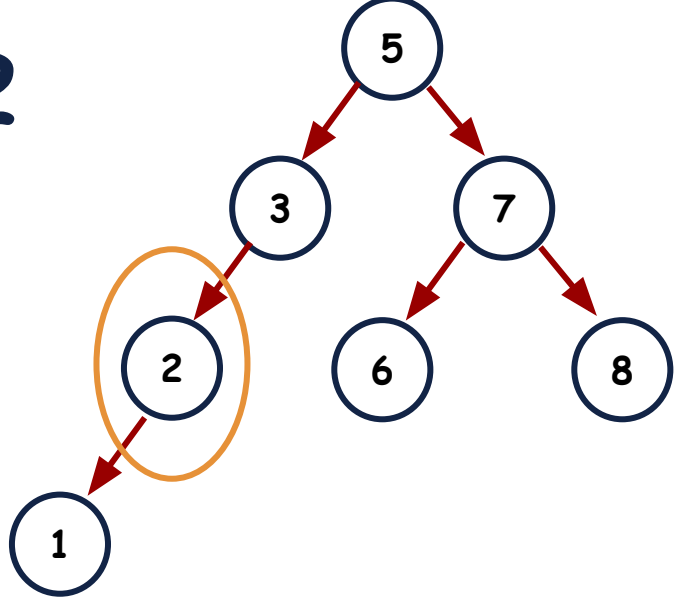
Delete in BST: Case 2

We just have to find the node containing the value 2 and then assign the parent of node 2 the child of node 2.



Delete in BST: Case 2

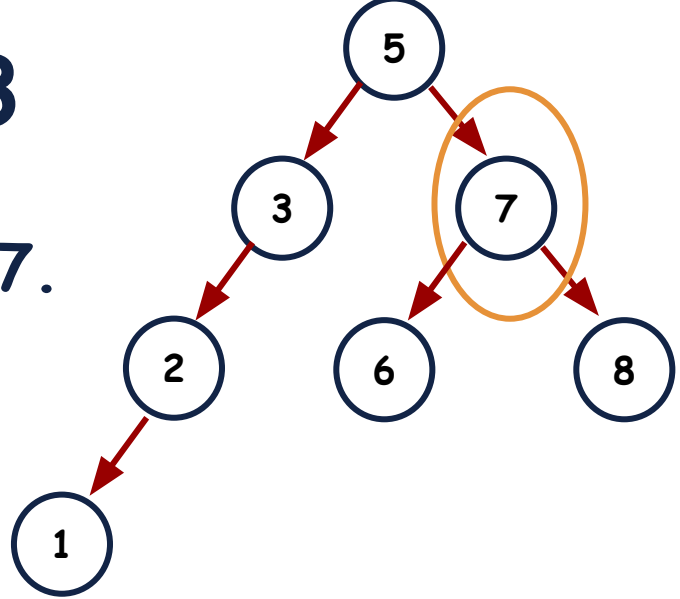
```
else if (next->left == NULL || next->right == NULL)
        {
            TreeNode *newCurr;
            if (next->left == NULL)
                newCurr = next->right;
            else
                newCurr = next->left;

            if (prev == root)
            {
                delete next;
                root = newCurr;
                return true;
            }

            if (next == prev->left)
                prev->left = newCurr;
            else
                prev->right = newCurr;

            delete next;
        }
```



Delete in BST: Case 3

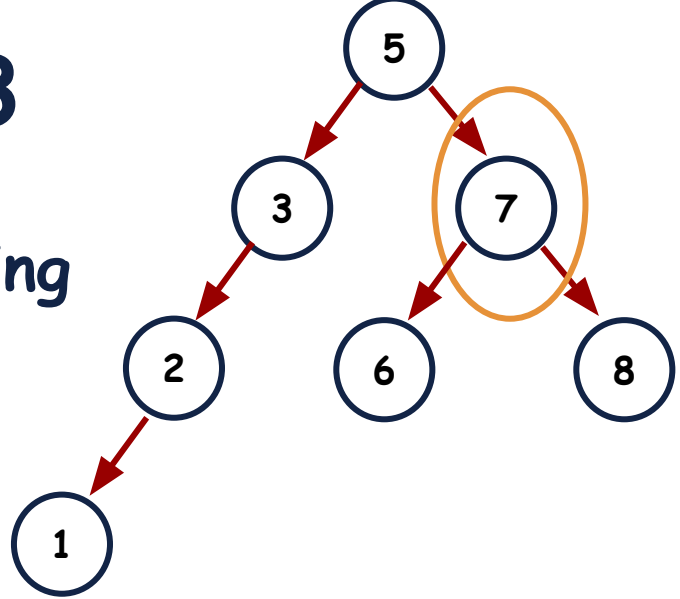
Let's say we want to delete node 7.
How can we do that?

OH, LET'S SEE...



Delete in BST: Case 3

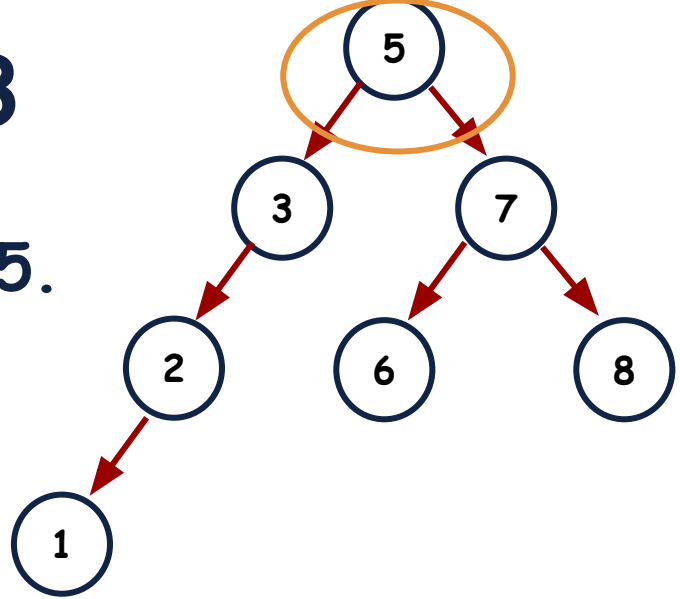
We have to find the node containing the value 7, replace it with its Inorder successor and delete the Successor node.



Delete in BST: Case 3

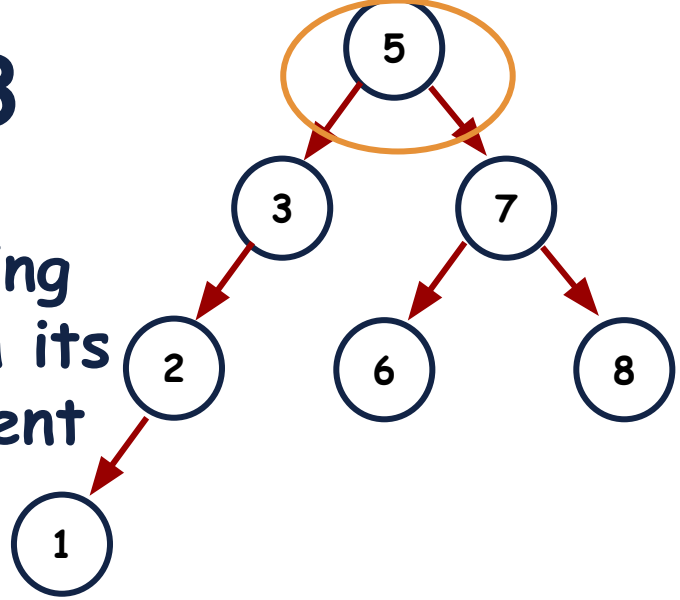
Let's say we want to delete node 5.
How can we do that?

OH, LET'S SEE...



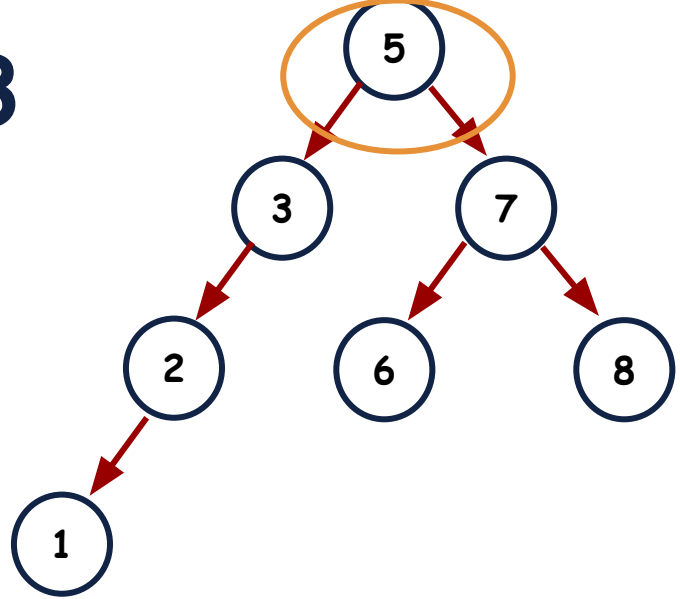
Delete in BST: Case 3

We have to find the node containing the value 5, replace its value with its Inorder successor, assign the parent of inorder successor to the right child of inorder successor and delete the inorder successor.



Delete in BST: Case 3

```
else
{
     TreeNode *p = NULL;
     TreeNode *temp;
     temp = next->right;
     while (temp->left != NULL)
     {
           p = temp;
           temp = temp->left;
     }

     if (p != NULL)
           p->left = temp->right;
     else
           next->right = temp->right;

     next->val = temp->val;
     delete temp;
}
return true;
```



Operations on Binary Search Trees

Following are the common operations that we can perform on the BSTs.

1. Insert a Node
2. Search a Value
3. Delete a Node
4. Traverse the Tree 

Operations on Binary Search Trees

We have already covered the 4 types of traversing (Level Order, In Order, Pre Order and Post Order).

Learning Objective

Students should be able to apply different operations on **Binary Search Trees** in order to solve the problems **efficiently**.

Online Links

btv.melezinek.cz/binary-search-tree.html
https://visualgo.net/en/bst