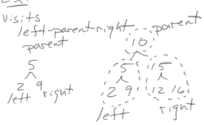
Tree traversals

Evaluate each node exactly once in the tree

BST is ordered. Traversal that produces ordered output is an inorder traversal.

Ex:

Visits left-parent-right, parent



Pattern is left-parent-right

```
InOrderPrint(node)
    if node->leftChild != nullptr
      InOrderPrint(node->leftChild)
    cout << node->value
    if node->rightChild != nullptr
      InOrderPrint(node->rightChild)
```

Pre-order and post-order

Pre-order evaluates parent before children

Post-order evaluates children before parent

Deleting nodes from a BST

Rule: Still needs to be a valid BST after delete

Ex:

Delete 1
- no children

Delete 6
- one child

Delete 15
- two children

No children case

Delete 1
- In this example, node is left child of its parent

```
node->parent->leftChild = nullptr
delete node
```

One child case

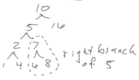
Delete 6

node is a right child

```
node->parent->rightChild = node->rightChild
delete node
```

Two child case

Ex:



Delete 5

Replace with minimum value in right branch of node to delete

6 is min in this example

Less common is to use max value in left branch

Two cases for min value

min is right child

Delete 5

min not right child

Delete 5

Find min by going left until you reach bottom of branch.

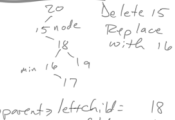
```
tmp = node->rightChild
while tmp->leftChild != nullptr
    tmp = tmp->leftChild
Min = tmp
```

Min is right child

```
if min == node->rightChild
  node->parent->leftChild = min
  Min->parent = node->parent
  Min->leftChild = node->left Child
  Min->leftChild->parent = Min
```

Min isn't right child

Ex:



Delete 15

Replace with 16

```
Min->parent->leftChild = Min->rightChild
Min->rightChild->parent = Min->parent
Min->parent = node->parent
node->parent->leftChild = Min
Min->leftChild = node->leftChild
Min->rightChild = node->rightChild
node->rightChild->parent = Min
node->leftChild->parent = Min
```

Final tree