TREE TRAVERSALS

Tree Traversal

- Tree Traversal
- Preorder
- Inorder
- Postorder

Tree Traversal

- There are three commonly used patterns to visit all the nodes in a tree.
- The difference between these patterns is the order in which each node is visited (a “traversal”)
- The three traversals we will look at are called **preorder**, **inorder**, and **postorder**.

Preorder

- In a preorder traversal, we visit the root node first, then recursively do a preorder traversal of the left subtree, followed by a recursive preorder traversal of the right subtree.

Inorder

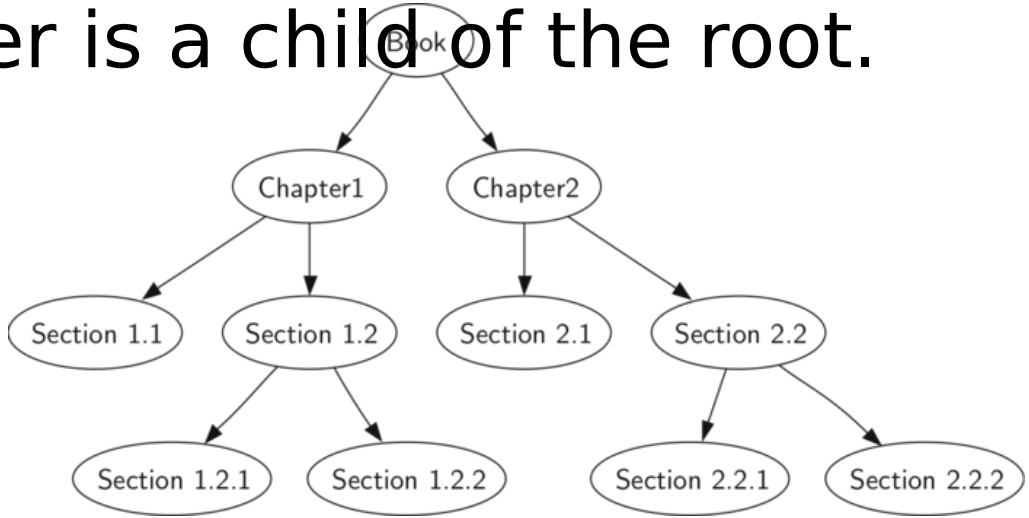
- In an inorder traversal, we recursively do an inorder traversal on the left subtree, visit the root node, and finally do a recursive inorder traversal of the right subtree.

Postorder

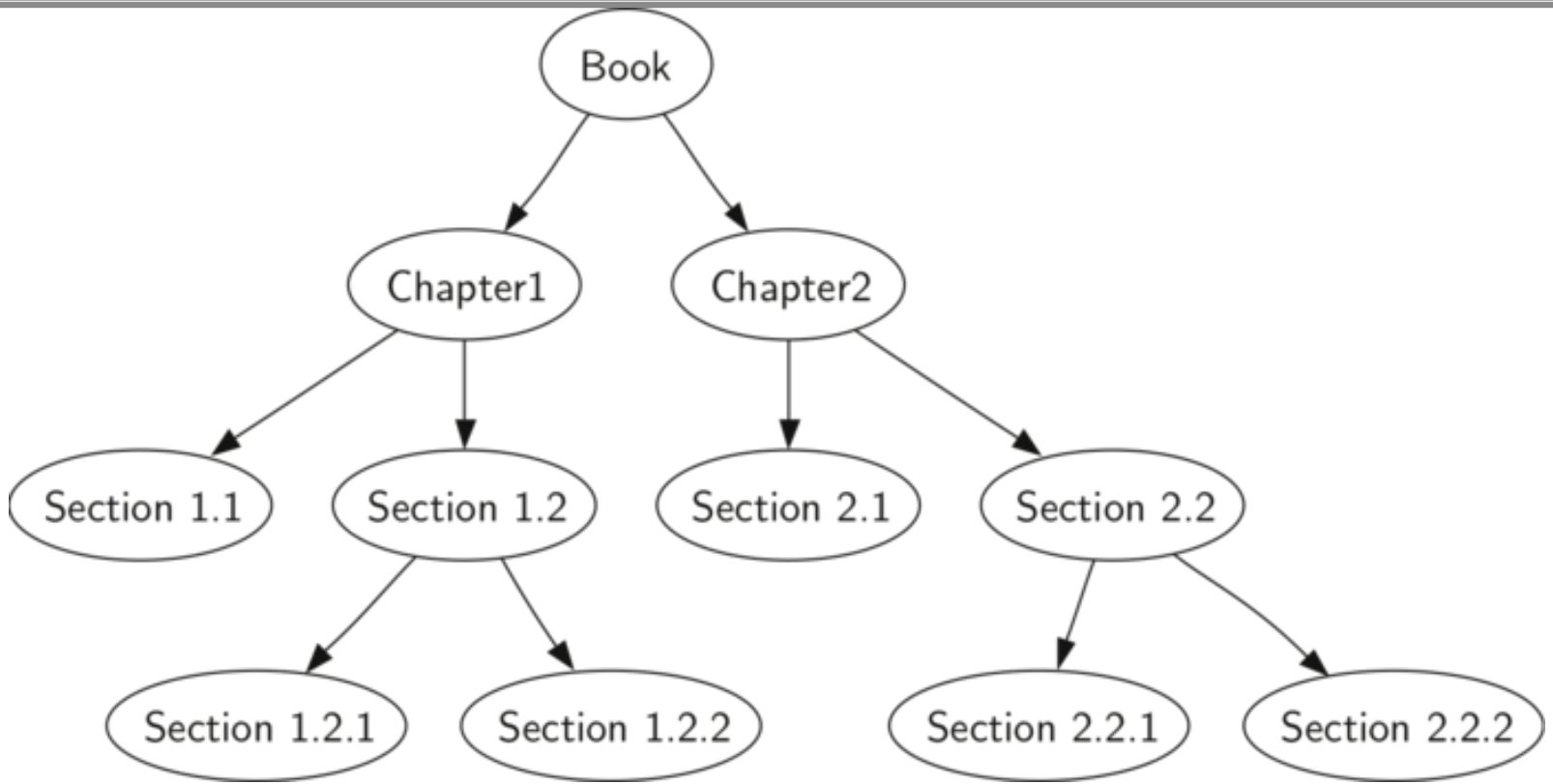
- In a postorder traversal, we recursively do a postorder traversal of the left subtree and the right subtree followed by a visit to the root node.

Learning through Example

- As an example of a tree to traverse, we will represent this book as a tree. The book is the root of the tree, and each chapter is a child of the root.



Learning through Example



Learning through Example

- Suppose that you wanted to read this book from front to back.
- The preorder traversal gives you exactly that ordering.

Learning through Example

- Starting at the root of the tree (the Book node) we will follow the preorder traversal instructions.
- We recursively call preorder on the left child, in this case Chapter1.
- We again recursively call **preorder** on the left child to get to Section 1.1.
- Since Section 1.1 has no children, we do not make any additional recursive calls.

Learning through Example

- When we are finished with Section 1.1, we move up the tree to Chapter 1.
- At this point we still need to visit the right subtree of Chapter 1, which is Section 1.2.
- As before we visit the left subtree, which brings us to Section 1.2.1, then we visit the node for Section 1.2.2.
- With Section 1.2 finished, we return to Chapter 1.
- Then we return to the Book node and follow the same procedure for Chapter 2.

Preorder – Recursive Implementation

- Base case is simply to check if the tree exists.
- If the tree parameter is None, then the fun
acti

```
def preorder(tree):
    if tree:
        print(tree.getRootVal())
        preorder(tree.getLeftChild())
        preorder(tree.getRightChild())
```

Preorder – Method Implementation

- We can also implement preorder as a method of the BinaryTree class.
- The internal method must check for the existence of the left and the right children *before* making the recursive call to preorder.

Preorder – Method Implementation

```
def preorder(self):
    print(self.key)
    if self.leftChild:
        self.leftChild.preorder()
    if self.rightChild:
        self.rightChild.preorder()
```

Preorder – Best Implementation

- Implementing preorder as an external function is probably better in this case.
- The reason is that you very rarely want to just traverse the tree.
- In most cases you are going to want to accomplish something else while using one of the basic traversal patterns.
- We will write the rest of the traversals as external functions.

Postorder

- The algorithm for the postorder traversal is nearly identical to preorder except that we move the call to print to the end of the function.

Postorder

```
def postorder(tree):
    if tree != None:
        postorder(tree.getLeftChild())
        postorder(tree.getRightChild())
        print(tree.getRootVal())
```

Inorder

- In the inorder traversal we visit the left subtree, followed by the root, and finally the right subtree.
- Notice that in all three of the traversal functions we are simply changing the position of the print statement with respect to the two recursive function calls.

Inorder

```
def inorder(tree):
  if tree != None:
      inorder(tree.getLeftChild())
      print(tree.getRootVal())
      inorder(tree.getRightChild())
```

Implementation

Your homework assignment is to implement each of these traversals as a function (not method)

Use the BinaryTree() class we made in the last lecture.

The code is in these slides for reference!