Week 6 - Friday

COMP 2100

Last time

- What did we talk about last time?
- Some binary tree facts
- Java implementation of binary search trees

Questions?

Assignment 3

Recursion

Project 2

Infix to Postfix Converter

Traversals

Basic BST class

```
public class Tree {
  private static class Node {
    public int key;
    public Object value;
    public Node left;
    public Node right;
  }

  private Node root = null;

  …
}
```

The book uses a generic approach, with keys of type **`Key`** and values of type **`Value`**. The algorithms we'll use are the same, but I use **`int`** keys to simplify comparison.

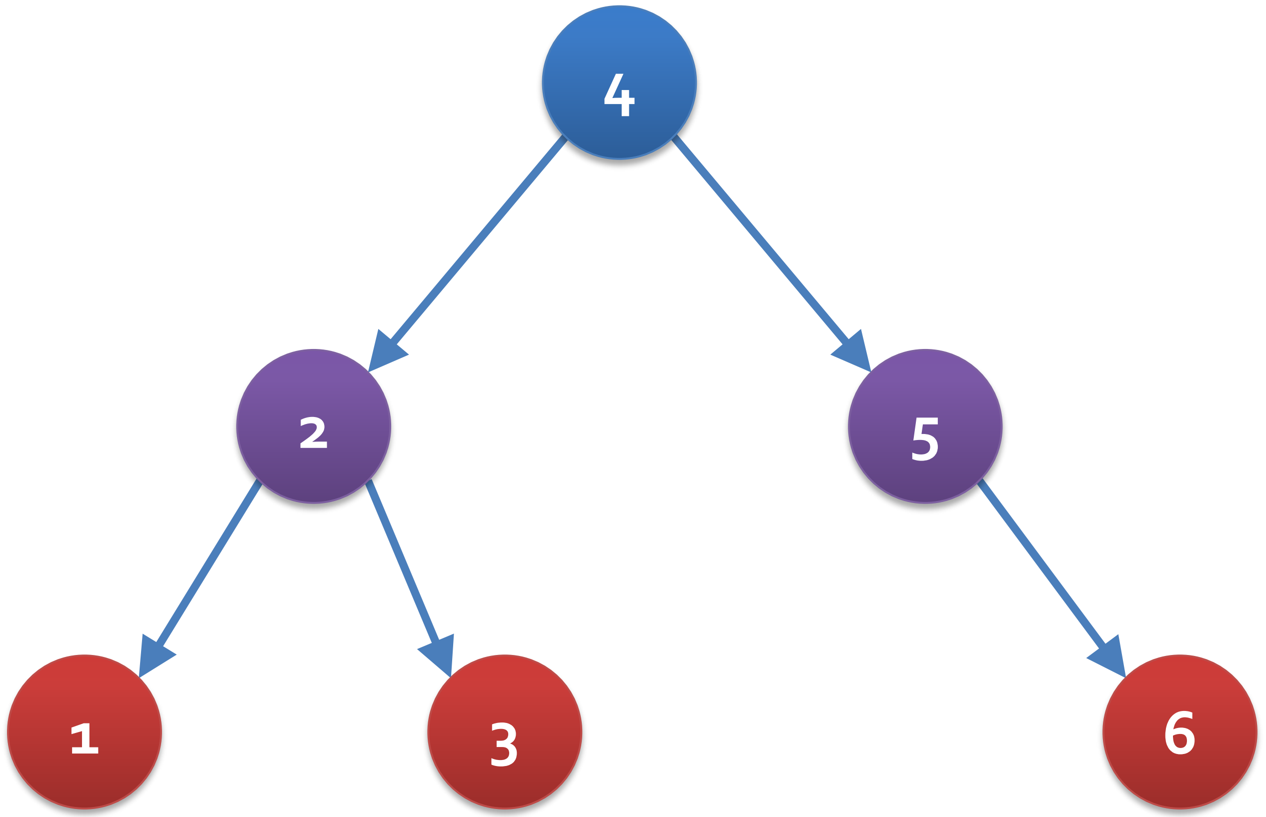
Traversals

- Visiting every node in a tree is called a **traversal**
- There are three traversals that we are interested in today:
    - Preorder
    - Postorder
    - Inorder
- We'll get to level order traversal in the future

Traversals

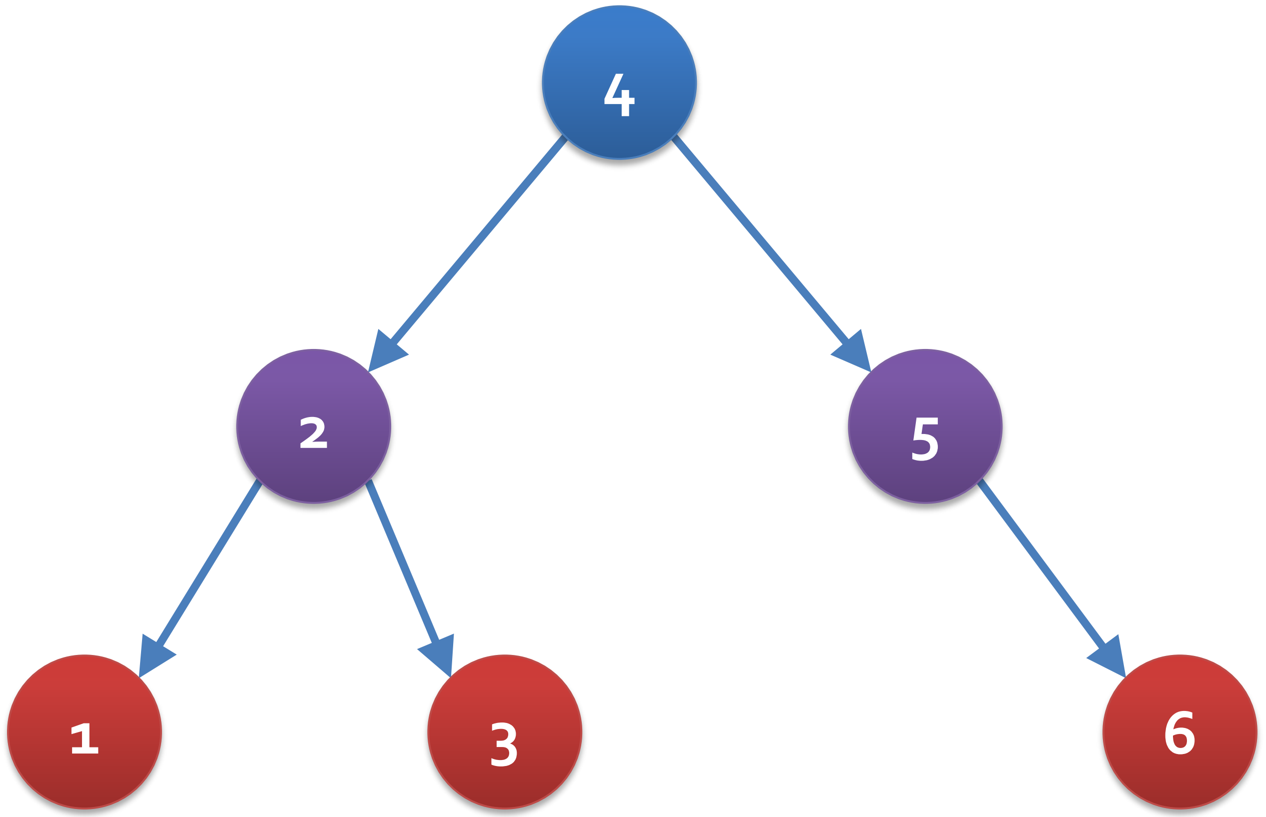
- Preorder:
  - Process the node, then recursively process its left subtree, finally recursively process its right subtree
  - **NLR**
- Postorder:
  - Recursively process the left subtree, recursively process the right subtree, and finally process the node
  - **LRN**
- Inorder:
  - Recursively process the left subtree, process the node, and finally recursively process the right subtree
  - **LNR**

Preorder



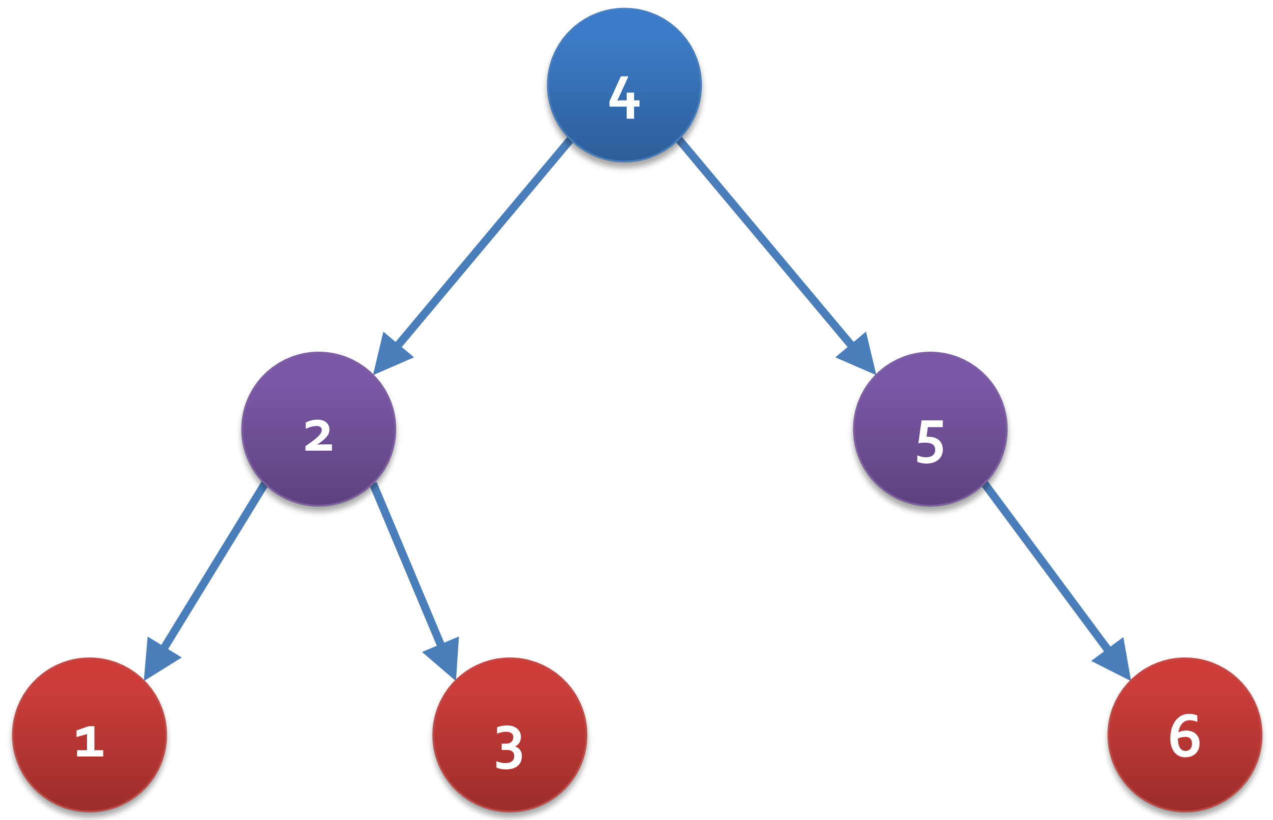
4 2 1 . . 3 . . 5 . 6 . .

Postorder



. . 1 . . 3 2 . . . 6 5 4

Inorder



. 1 . 2 . 3 . 4 . 5 . 6 .

Implement preorder

```
private static void preorder( Node node )
```

Proxy:

```
public void preorder() {
  preorder( root );
}
```

Just print out each node (or a dot). Real traversals will actually do something at each node.

Implement inorder

```
private static void inorder( Node node )
```

Proxy:

```
public void inorder() {
  inorder( root );
}
```

Just print out each node (or a dot). Real traversals will actually do something at each node.

Implement postorder

```
private static void postorder( Node node )
```

Proxy:

```
public void postorder() {
  postorder( root );
}
```

Just print out each node (or a dot). Real traversals will actually do something at each node.

Get range

- We can take the idea of an inorder traversal and use it to store a range of values into a queue
- We want to store all values greater than or equal to the min and less than the max

```
private static void getRange( Node node,
Queue<Object> queue, int min, int max )
```

Proxy:

```
public Queue<Object> getRange(int min, int max){
  Queue<Object> queue = new
        ArrayDeque<Object>();
  getRange( root, queue, min, max );
  return queue;
}
```

Delete

Delete

```
private static Node delete(Node node, int key)
```

Proxy:

```
public void delete(int key) {
  root = delete( root, key );
}
```

1. Find the node
2. Find its replacement (smallest right child)
3. Swap out the replacement

It's probably wise to find the replacement with iteration, though we could do it with recursion.

Note: This delete can cause an unbalanced tree.

Upcoming

Next time…

- Breadth-first traversal
- 2-3 trees
- Red-black trees
- Balancing trees by construction

Reminders

- Keep working on Project 2
- Finish Assignment 3
  - **Due tonight by midnight!**
- Read Section 3.3