Goals

Be able to remove a node from a BST on paper.

Be prepared to implement BST removal.
Warm-up

Write a method to find the smallest value in a BST:

```java
/** Returns min value in BST n.
 * pre: n is not null */
public int minimum(Node n) {
}
```
Write a method to find the smallest value in a BST:

```java
/** Returns min value in BST n. *
 * pre: n is not null */
public int minimum(Node n) {
    if (n.left == null) {
        return n.value;
    }
}
```
Warm-up

Write a method to find the smallest value in a BST:

```java
/** Returns min value in BST n.
 * pre: n is not null */
public int minimum(Node n) {
    if (n.left == null) {
        return n.value;
    }
}
```

3. Recursive definition:
Smallest(n) is:
- the smallest value in the left subtree, or
- n.value if no left subtree exists.
Warm-up

Write a method to find the smallest value in a BST:

```java
/**
 * Returns min value in BST n.
 * pre: n is not null */
public int minimum(Node n) {
    if (n.left == null)
        return n.value;
    return minimum(n.left);
}
```

3. Recursive definition:
Smallest(n) is:
- the smallest value in the left subtree, or
- n.value if no left subtree exists.
Deleting a node from a BST

Three possible cases:
1. \( n \) has no children (is a leaf)
2. \( n \) has one child
3. \( n \) has two children
Deleting a node from a BST:

Case 1

Three possible cases:

1. **n has no children (is a leaf)**
2. n has one child
3. n has two children

if (n is a leaf)
    replace parent’s child with null
Deleting a node from a BST:
Case 2

Three possible cases:
1. n has no children (is a leaf)
2. n has one child
3. n has two children

if (n has exactly one child)
   replace parent’s child with n’s child
   replace n’s child’s parent with n’s parent
Deleting a node from a BST: Case 2

Three possible cases:

1. n has no children (is a leaf)
2. **n has one child**
3. n has two children

if (n has exactly one child)
   replace parent’s child with n’s child
   replace n’s child’s parent with n’s parent
Deleting a node from a BST: 
Case 2

Three possible cases:
1. n has no children (is a leaf)
2. n has one child
3. n has two children

if (n has exactly one child)
   replace parent’s child with n’s child
   replace n’s child’s parent with n’s parent
Deleting a node from a BST:

Case 3

Three possible cases:

1. n has no children (is a leaf)
2. n has one child
3. n has two children

if (n has two children)
Deleting a node from a BST:

Case 3

Three possible cases:
1. n has no children (is a leaf)
2. n has one child
3. n has two children

if (n has two children)
    let k = min node in right subtree
Deleting a node from a BST:
Case 3

Three possible cases:
1. n has no children (is a leaf)
2. n has one child
3. n has two children

if (n has two children)
let k = min node in right subtree
replace n’s value with k’s value
Deleting a node from a BST: Case 3

Three possible cases:
1. n has no children (is a leaf)
2. n has one child
3. n has two children

if (n has two children)
    let k = min node in right subtree
    replace n’s value with k’s value

Can we do that?
• k is n’s successor (next in an in-order traversal)
• Everything else in n’s right subtree is bigger than it
• Everything in n’s left subtree is smaller than it
• k’s value can safely replace n’s…but now we have a duplicate.
Deleting a node from a BST: Case 3

Three possible cases:
1. n has no children (is a leaf)
2. n has one child
3. **n has two children**

if (n has two children)
   let k = min node in right subtree
   replace n’s value with k’s value
   remove k from n’s right subtree
Deleting a node from a BST:

Case 3

Three possible cases:
1. n has no children (is a leaf)
2. n has one child
3. n has two children

if (n has two children)
   let k = min node in right subtree
   replace n’s value with k’s value
   remove k from n’s right subtree
Deleting a node from a BST: Case 3

Three possible cases:

1. n has no children (is a leaf)
2. n has one child
3. n has two children

if (n has two children)
   let k = min node in right subtree
   replace n’s value with k’s value
   remove k from n’s right subtree
Deleting a node from a BST:

Case 3

Three possible cases:

1. n has no children (is a leaf)
2. n has one child
3. n has two children

if (n has two children)
   let k = min node in right subtree
   replace n’s value with k’s value
   remove k from n’s right subtree (recursively!)

Question: does this always make progress towards the base case?
Details

- Handle the root:
  - Update root pointer if root is removed.
  - Can’t assume n.parent is non-null

- To update parent’s child pointer, you need to know which (L or R) child pointer to update.

- The approach presented differs from that in CLRS and some other resources.