CSCI 136
Data Structures & Advanced Programming

Lecture 27

Spring 2018

Profs Bill & Jon

Administrative Details

- Lab 9 Posted: Gardner's Hex-a-Pawn
  - Another partner lab!
  - Challenging to design & debug! A well-thought-out design document will be key
- `n x n` chessboard with `2n` pawns
- Task:
  - build a `GameTree` of all possible board states
  - Implement Player classes for human, random, comp.
    - Computer will "trim" its tree to eliminate losing moves!
  - Players can play against each other (like `CoinStrip`)

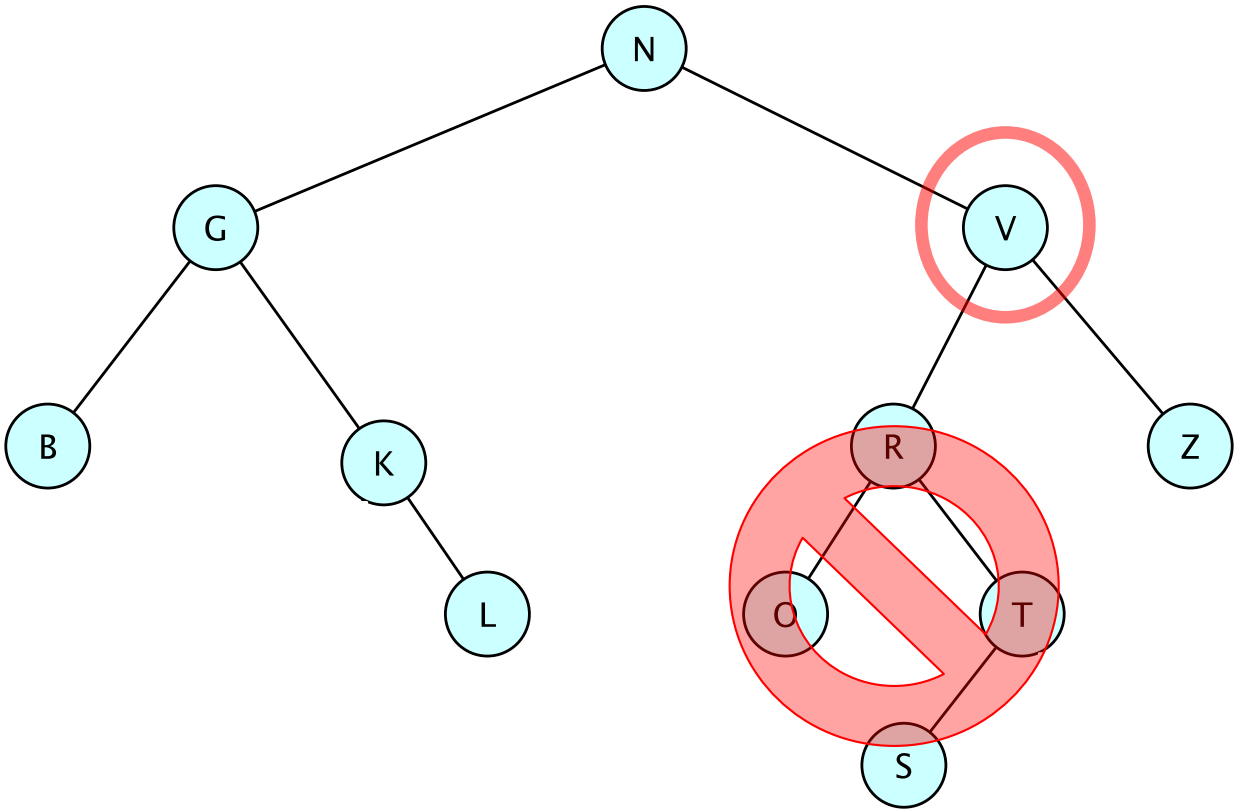
Last Time

- Search!
  - `OrderedVector`
  - `BinarySearchTree`
- BST Implementation details:
  - `locate`: return node containing value OR node where value should be added to the tree (as a child)
  - `predecessor`: return the node whose value precedes target value (i.e. immediately before it in the BST's ordering)
  - `add`

Binary Search Tree Add

- Remember!!! A binary tree is a binary *search* tree if it is:
  - Empty, or a binary tree where
  - (1) All nodes in the left subtree are less than or equal to the root (2) all nodes in the right subtree are greater than or equal to the root, and (3) the left and right subtrees are binary search trees.
- In our implementation, right subtrees only hold values that are *strictly greater than* the root

How to Add Duplicate Values



How to perform: bst.add(“v”) ???

locate(“v”).setLeft(new BinaryTree (“v”)); ???

First (Bad) Attempt: add(E value)

```
public void add(E value) {
    BinaryTree<E> newNode = new BinaryTree<E>(value,EMPTY,EMPTY);
    if (root.isEmpty()) {
        root = newNode;
    } else {
        BinaryTree<E> insertLocation = locate(root,value);
        E nodeValue = insertLocation.value();
        if (ordering.compare(value,nodeValue) > 0)
            insertLocation.setRight(newNode); // value > nodeValue
        else
            insertLocation.setLeft(newNode); // value <= nodeValue
    }
    count++;
}
```

Problem: If duplicate values are allowed in the BST, the left subtree might not be empty when setLeft is called

Strategy: Add Duplicates to Predecessor

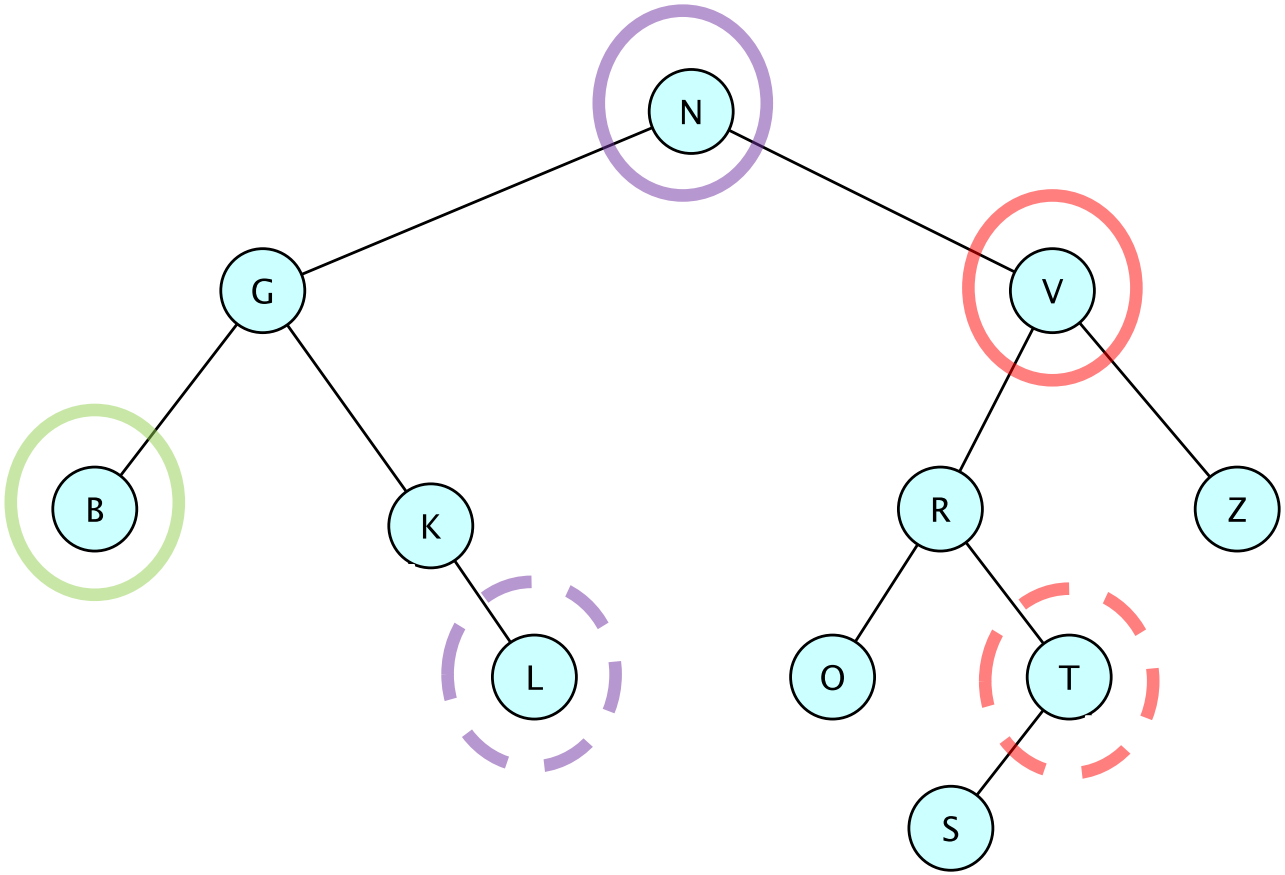
- If `insertLocation` has a left child:
  - Find `insertLocation`'s *predecessor*, then
  - Add duplicate node as *right child* of its predecessor
    - Why?
    - What are the relationships among `root`, `pred(root)`, and `node`?
  - Make duplicate values the successor of their predecessor!

Corrected: add(E value)

```
public void add(E value) {
    BinaryTree<E> newNode = new BinaryTree<E>(value,EMPTY,EMPTY);
    if (root.isEmpty()) {
        root = newNode;
    } else {
        BinaryTree<E> insertLocation = locate(root,value);
        E nodeValue = insertLocation.value();
        if (ordering.compare(value,nodeValue) > 0) {
            // value > nodeValue
            insertLocation.setRight(newNode);
        } else {
            // value <= nodeValue
            if (insertLocation.left().isEmpty())
               insertLocation.setLeft(newNode);
            else
               predecessor(insertLocation).setRight(newNode);
    }
    count++;
}
```

Recap: If value is in the tree, insert newNode immediately before it (successor of predecessor)

How to Find Predecessor?



predecessor(root) is the rightmost node in root's left subtree

Predecessor

```
// return node with largest value in root's left subtree
// pre: root is not empty, root's left child is not empty
protected BinaryTree<E> predecessor(BinaryTree<E> root) {
    BinaryTree<E> result = root.left();

    while (!result.right().isEmpty())
        result = result.right();

    return result;
}
```

"slide down" the left subtree

BST Operations

BST methods (`OrderedStructure` + friends):

- `locate(E item)`
- `contains(E item)`
- `get(E item)`
- `predecessor(E item)`
- `add(E item)`
- `remove(E item)`

Removal

- If we can remove the root, we can remove any element in a BST in the same way
  - Why?
- We need to implement:
  - `public E remove(E item)`
- We can benefit from a helper:
  - `protected BT removeTop(BT top)`
    - `removeTop(BT top)` removes `top`, and returns the root node of the resulting tree
- Assuming `removeTop` works, let's implement `remove`

BST remove()

```
public E remove(E value) {
    // base case 1: empty tree
    if (isEmpty()) return null;

    // base case 2: root contains value
    if (value.equals(root.value())) {
        E result = root.value();
        count--;
        root = removeTop(root);
        return result;
    }

    . . .
```

BST remove()

```
    // general case: find node that holds value, remove node,
    //      and re-attach resulting tree at node's old location
    BinaryTree<E> location = locate(root,value);
    if (value.equals(location.value())) {  // found node with value
        count--;                    // we are about to remove a node...
        BinaryTree<E> parent = location.parent();
        if (parent.right() == location) { // removing right child
            parent.setRight(removeTop(location));
        } else {                                // removing left child
            parent.setLeft(removeTop(location));
        }
        return location.value();
    }

    // if we got here, value not found in tree, nothing to do
    return null;
}
```

What about removeTop(BST top)

- Task:
  - Disconnect BST top from tree
    - This breaks the left and right subtrees
  - Reassemble left and right subtrees into a valid BST
  - Return root of newly assembled BST

- Let's look at some pseudo-code
  - (We will break it down into cases)

RemoveTop(topNode)

Detach left and right sub-trees from root (i.e. topNode)

if either left or right is empty, **return** the other one

Cases 1 & 2

if left has no right child

Case 3

make right the right child of left then return left

Otherwise find largest node *C* in left

General case

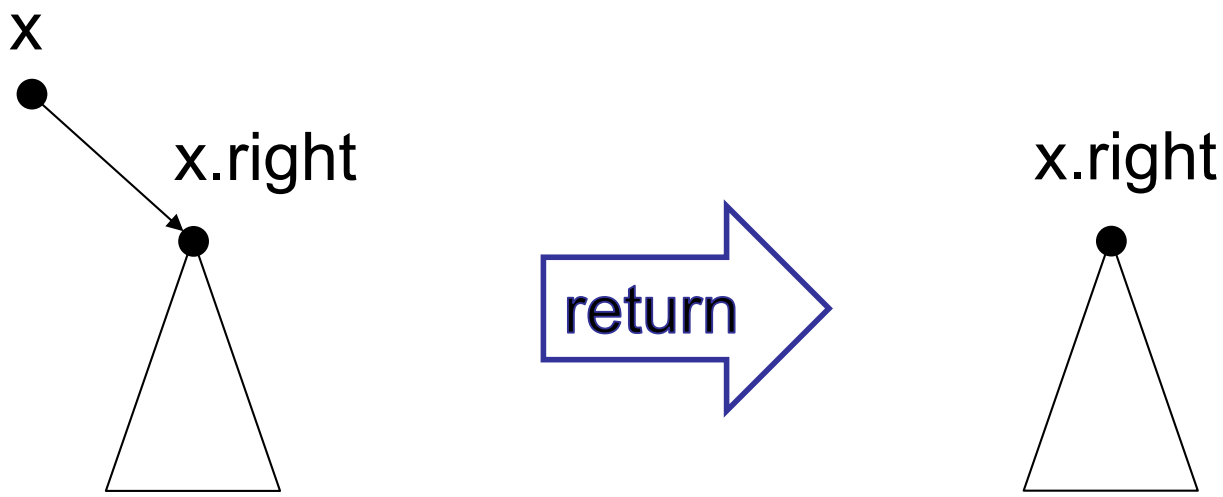
// C is the right child of its own parent P

// C is the predecessor of right (ignoring topNode)

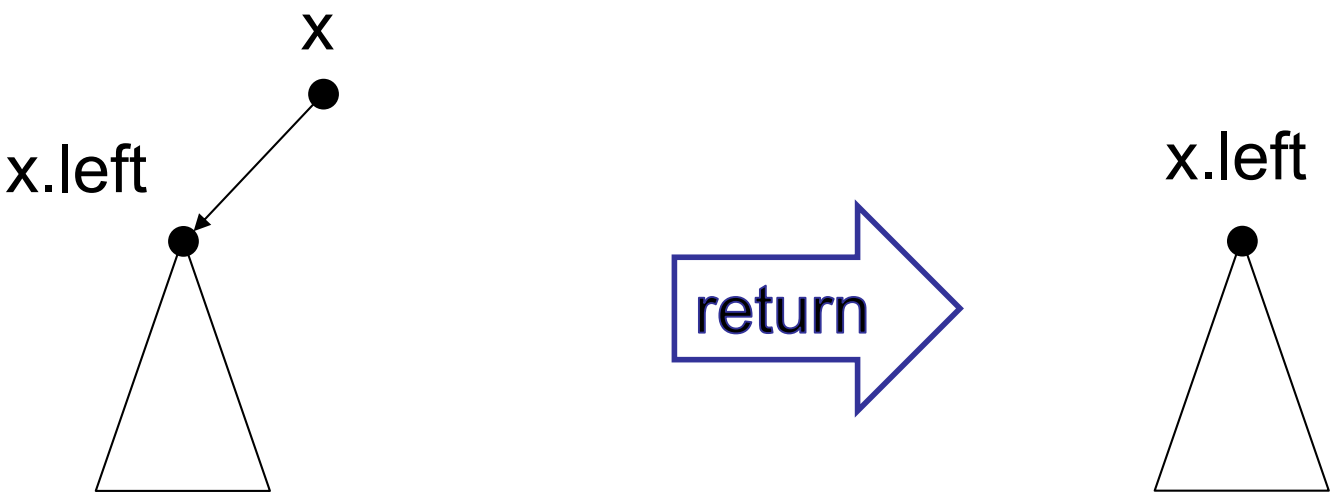
Detach C from P; make C's left child the right child of P

Make C new root with left and right as its sub-trees

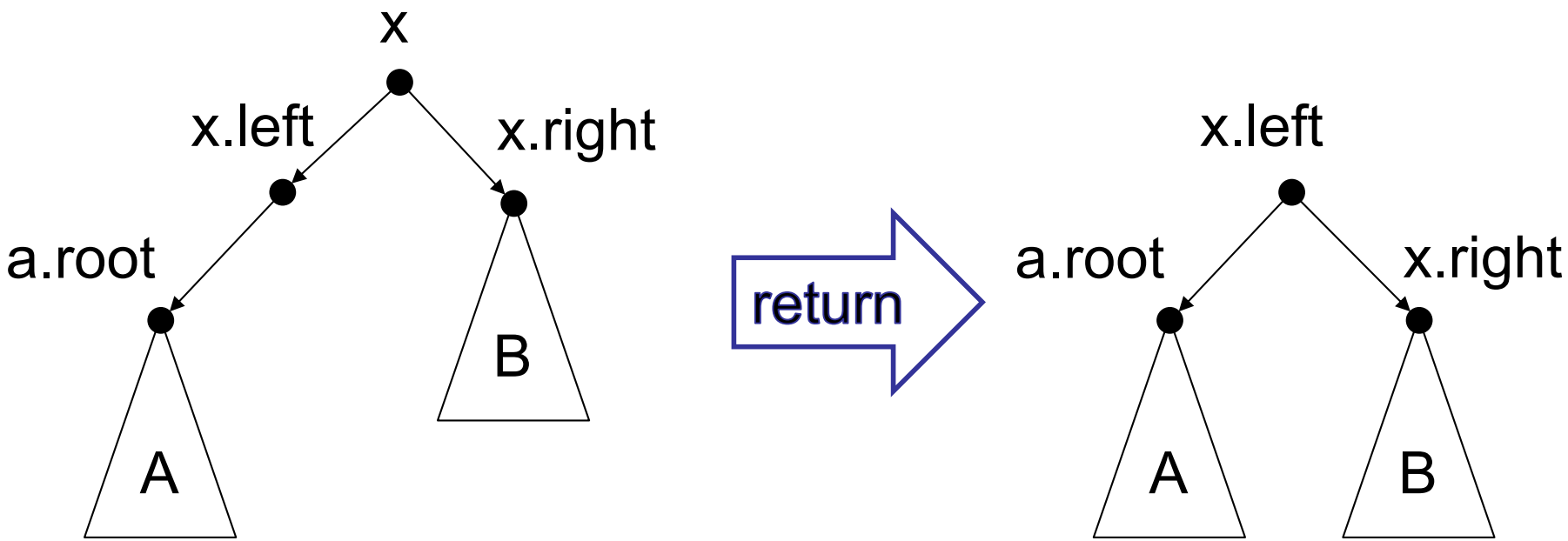
Case 1: No left binary tree



Case 2: No right binary tree



Case 3: Left has no right subtree



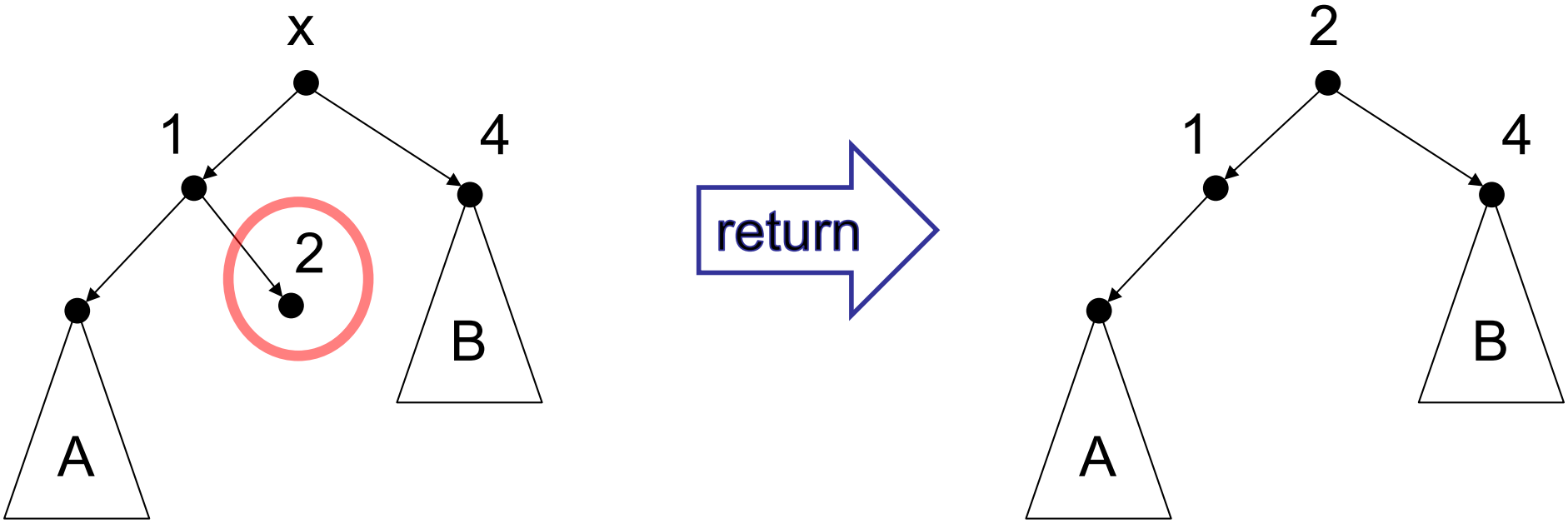
Remember, in our BST:

`x.left <= x < x.right`

Case 4: General Case (HARD!)

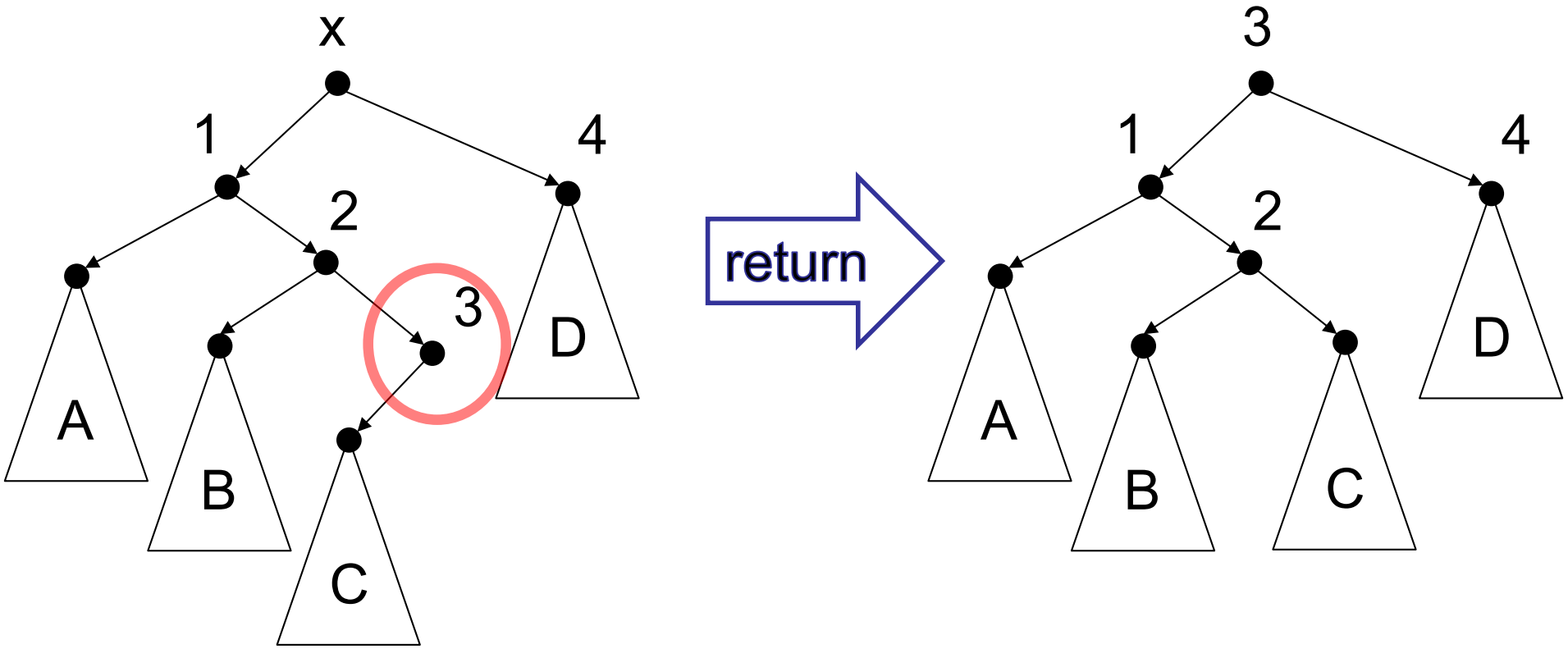
- Consider our BST requirements:
  - Left subtree must be <= root
  - Right subtree must be >= root
- Strategy: replace the root with the largest value that is less than or equal to it
  - `predecessor(root)` : rightmost left descendant
- This may require reattaching the predecessor's left subtree!

Case 4: General Case (HARD!)



Replace root with predecessor(root), then patch up the remaining tree

Case 4: General Case (HARD!)



Replace root with predecessor(root), then patch up the remaining tree

Let's Write Some Code

- removeTop.java