Outline

- Tree Terminologies
  - Relationships
  - Tree Node Classifications
  - Subtrees
  - Properties

- Binary Trees
  - Implementation
  - Traversal

- Binary Search Trees: Adding Nodes
  - Java's TreeSet<E> Class

- Conclusion
BST: add()

- Adding is like `contains()`, need to figure out *where to put* the new node
- Add:
  - "leg"
  - "cork"

(null reached - add here!!!)
BST: `add()`

- Adding is like `contains()`, need to figure out *where to put* the new node.
- Add:
  - "leg"
  - "cork"
BST: add()

- Adding is like `contains()`, need to figure out *where to put* the new node
- Add:
  - "leg"
  - "cork"
The Thing about `add()`...

- Does BST insertion order matter?
  - (I'm so glad you asked)

- Build a BST by adding the following items (in sequence):
  - 50, 75, 25, 0, 15, 80, 65

- Build a BST by adding the same items, just different ordering:
  - 0, 15, 25, 50, 65, 75, 80 or (80, 75, 65, 50, 25, 15, 0)
  - 65, 75, 80, 0, 15, 25, 50
Outline

- Tree Terminologies
  - Relationships
  - Tree Node Classifications
  - Subtrees
  - Properties

- Binary Trees
  - Implementation
  - Traversal

- Binary Search Trees: Removal
  - Java's TreeSet<E> Class

- Conclusion
BST: Node Removal

- Removing a node (victim) is a lot more complicated, depending on what node is removed,
  - Case 1: Victim is a leaf node
  - Case 2: Victim has one child
  - Case 3: Victim has both children
Case 1: Removing a leaf

- Easiest case; just remove the node with no further considerations

Example: remove "morn"
Case 1: Removing a leaf

- Easiest case; just remove the node with no further considerations

Example: remove "morn"
**Case 2: Removing a node with one child**

- Its child needs to take its place!

**Example: remove "tattered"**
Case 2: Removing a node with one child

- Its child needs to take its place!

Example: remove "tattered"
Case 3: Removing a node with two children

- For instance, "lay." What node takes its place?
  - *Hint: Resulting tree must still satisfy BST properties...*
Case 3: Removing a node with two children

- The replacement node must be:
  - Greater than all items in left subtree, and
  - Less than all items in right subtree

Two options: "lay's" in-order predecessor "lay's" in-order successor
Case 3: Removing a node with two children

- We'll replace using the in-order predecessor
BST: Node Removal (cont.)

- **Case 3**: Removing a node with two children
  - We'll replace using the in-order predecessor
**Case 3: Removing a node with two children**

- We'll replace using the in-order predecessor

But "killed" has a child node...
Case 3: Removing a node with two children

- We'll replace using the in-order predecessor
Check-in

- Using the following sequence of input, build a binary search tree:
  - 7, 5, 20, 4, 6, 19, 25, 30, 15, 3.

- For the binary search tree you just created, redraw the tree after deleting node 5.

- Redraw the tree one more time after deleting node 20.
Outline

▶ Tree Terminologies
  • Relationships
  • Tree Node Classifications
  • Subtrees
  • Properties

▶ Binary Trees
  • Implementation
  • Traversal

▶ Binary Search Trees
  • Performance

▶ Conclusion