VILLANOVA UNIVERSITY
IGNITE CHANGE. GO NOVA.

CSC 1052 – Algorithms & Data Structures II:

Binary Search Trees

Professor Henry Carter
Spring 2017

Housekeeping

- Remaining assignments:
  - 2 labs
  - 1 project (due in 1 week)
  - Final exam (May 6 @ 1:30 PM)
- Practice!
- Get assistance!
  - Office hours
  - Peer tutors

Recap

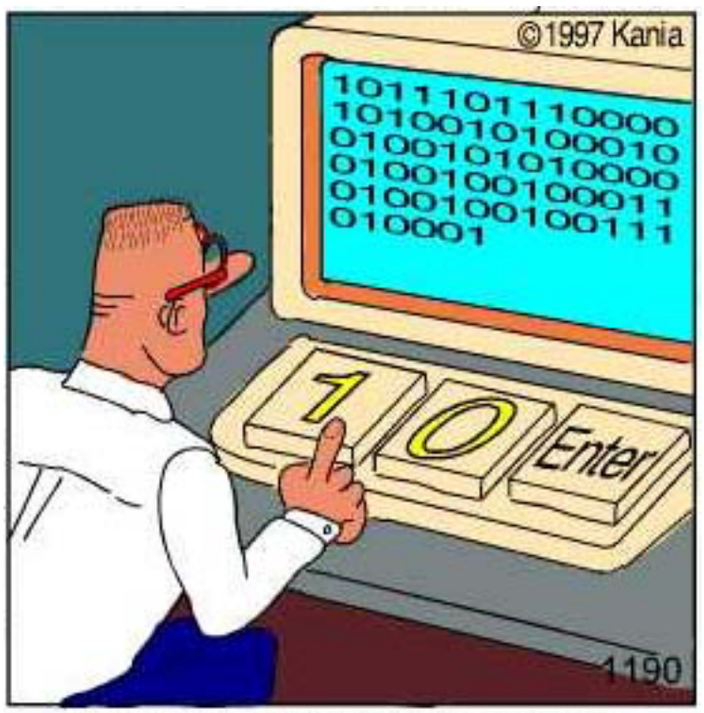
- Graphs represent structured data in a nonlinear fashion
  - Composed of nodes and links between the nodes
- Trees allow for storage of hierarchical data
  - Traversal in breadth-first or depth-first
- Binary Search Trees allow for a binary search to be embedded in the tree structure
  - Traversal in preorder, inorder, or postorder

Objective: Collection

- Original motivation to find a link-based collection that allows binary search
- Point of comparison: SortedArrayCollection
- What additional operations might be useful?

Binary Search Tree Interface

- Extends Collection and Iterable
  - What traversal order should iterator() use?
- Min()
- Max()
- getIterator()
  - In case you want a different traversal order



Real programmers code in binary.

Interface

```
import ch05.collections.CollectionInterface;
import java.util.Iterator;

public interface BSTInterface<T> extends CollectionInterface<T>, Iterable<T>
{
  // Used to specify traversal order.
  public enum Traversal {Inorder, Preorder, Postorder};

  T min();
  // If this BST is empty, returns null;
  // otherwise returns the smallest element of the tree.

  T max();
  // If this BST is empty, returns null;
  // otherwise returns the largest element of the tree.

  public Iterator<T> getIterator(Traversal orderType);
  // Creates and returns an Iterator providing a traversal of a "snapshot"
  // of the current tree in the order indicated by the argument.
}
```

Implementation Steps

- Collection methods
  - get(), add(), remove(), contains()…
- Iterator(s)
  - hasNext(), next(), ~~remove()~~
- New BST methods

Reorganized

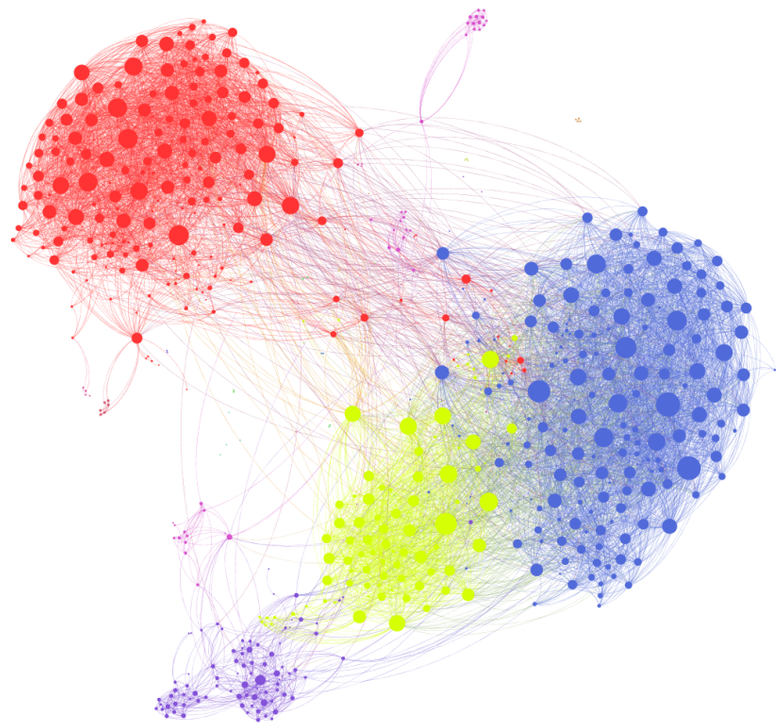
- Observers (today)
  - isEmpty, isFull, min, max, size, get, contains
- Iterator(s) (next week)
  - Preorder, inorder, postorder
- Transformers (next week)
  - Add, remove

First: Nodes

- Storing data and pointers

- Recall: Linked Node held data and next pointer

- What will change?



TreeNode Class

```
public class BSTNode<T>
{
  private T info;                    // The node info
  private BSTNode<T> left;           // A link to the left child node
  private BSTNode<T> right;          // A link to the right child node

  public BSTNode(T info)
  {
    this.info = info; left = null;  right = null;
  }

  public void setInfo(T info){this.info = info;}
  public T getInfo(){return info;}

  public void setLeft(BSTNode<T> link){left = link;}
  public void setRight(BSTNode<T> link){right = link;}

  public BSTNode<T> getLeft(){return left;}
  public BSTNode<T> getRight(){return right;}
}
```

BST Setup

- Class variables:
  - The root
  - A comparator
- Constructors
  - Allow the user to input a different comparator

Setup Code

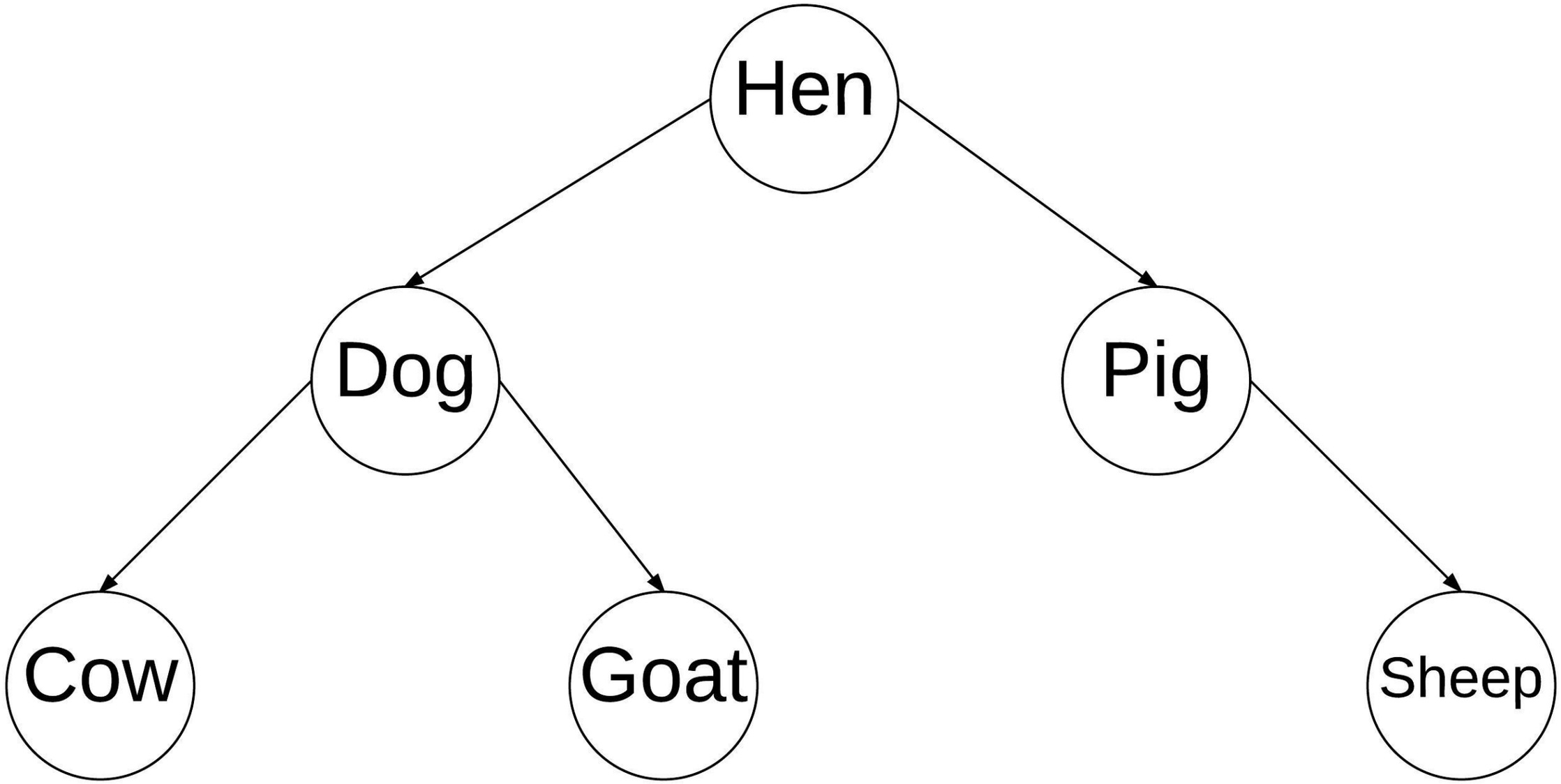
```
public class BinarySearchTree<T> implements BSTInterface<T>
{
  protected BSTNode<T> root;       // reference to the root of this BST
  protected Comparator<T> comp;    // used for all comparisons

  protected boolean found;    // used by remove

  public BinarySearchTree()
  // Precondition: T implements Comparable
  // Creates an empty BST object - uses the natural order of elements.
  {
    root = null;
    comp = new Comparator<T>()
    {
       public int compare(T element1, T element2)
       {
         return ((Comparable)element1).compareTo(element2);
       }
    };
  }

  public BinarySearchTree(Comparator<T> comp)
  // Creates an empty BST object - uses Comparator comp for order
  // of elements.
  {
    root = null;
    this.comp = comp;
  }
```

Easy Observers

- isEmpty
  - What do we check?
- isFull
  - What do we check?

- min and max
  - Where do we find these elements?

Example Tree



min (and max) code

```
public T min()
// If this BST is empty, returns null;
// otherwise returns the smallest element of the tree.
{
   if (isEmpty())
      return null;
   else
   {
      BSTNode<T> node = root;
      while (node.getLeft() != null)
        node = node.getLeft();
      return node.getInfo();
   }
}
```

Size, get, contains

- The remaining observers can be implemented in many ways
- Recall: the binary search tree has a recursive structure
  - Each node in the tree is the root of a smaller BST
- What programming technique should we try first?

Recursive Size

- Could maintain size as a parameter
- Consider: measuring the size of a subtree
- Instead, we implement size() recursively

Algorithm

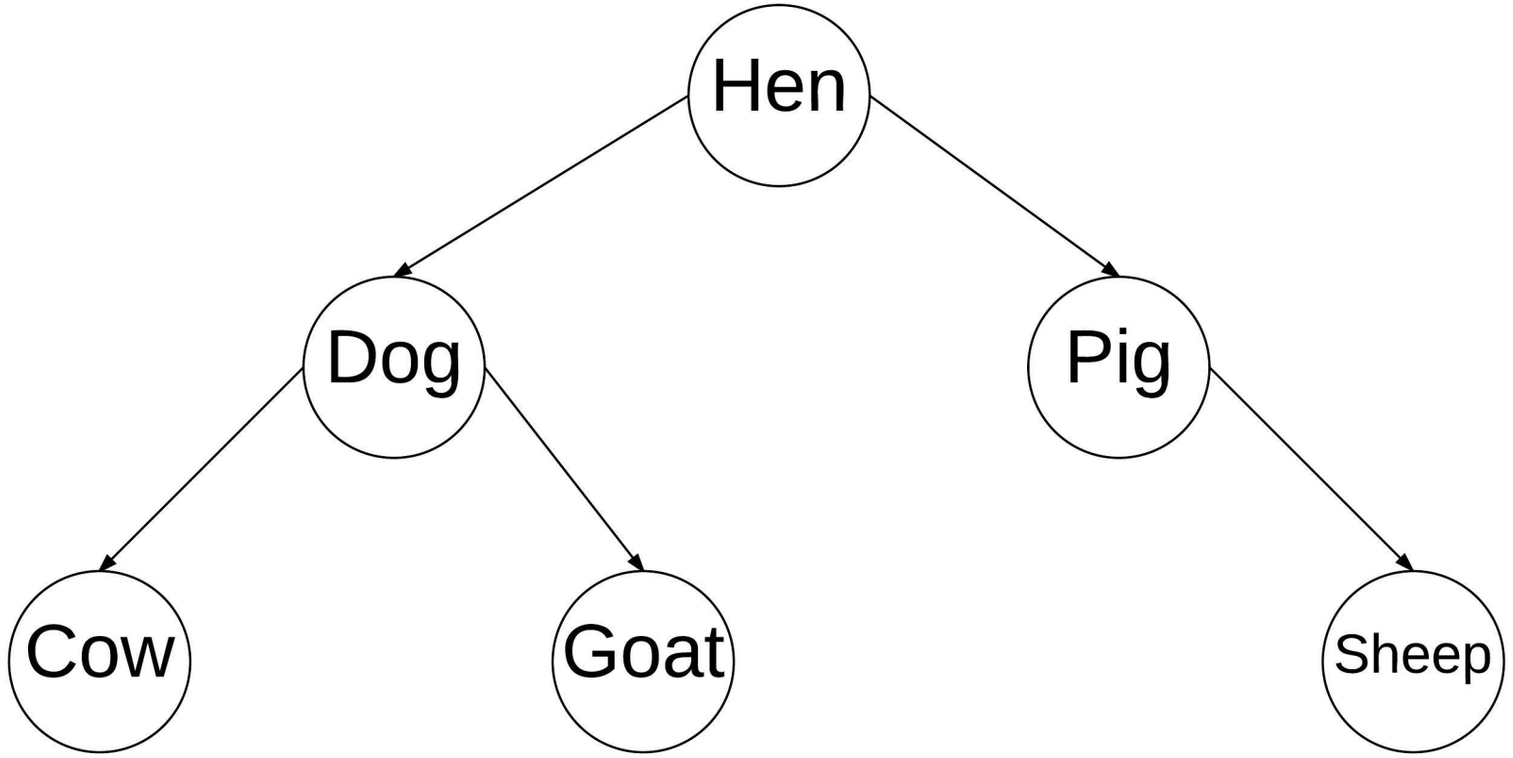
- Base case?
- Base case check?
- Recursive case?

recSize()

```
private int recSize(BSTNode<T> node)
// Returns the number of elements in subtree rooted at node.
{
  if (node == null)
    return 0;
  else
    return 1 + recSize(node.getLeft()) + recSize(node.getRight());
}
```

Practice

- Step through the size() code for the following tree. Determine the returned value at each recursive call and find a node that returns: 1, 2, 3, 4, 5, 6 (or determine if no such node exists)



Iterative Approach

- Bring your own stack
- Add nodes to the stack and count as they are popped off
- Implementation similar to our tree traversal algorithms

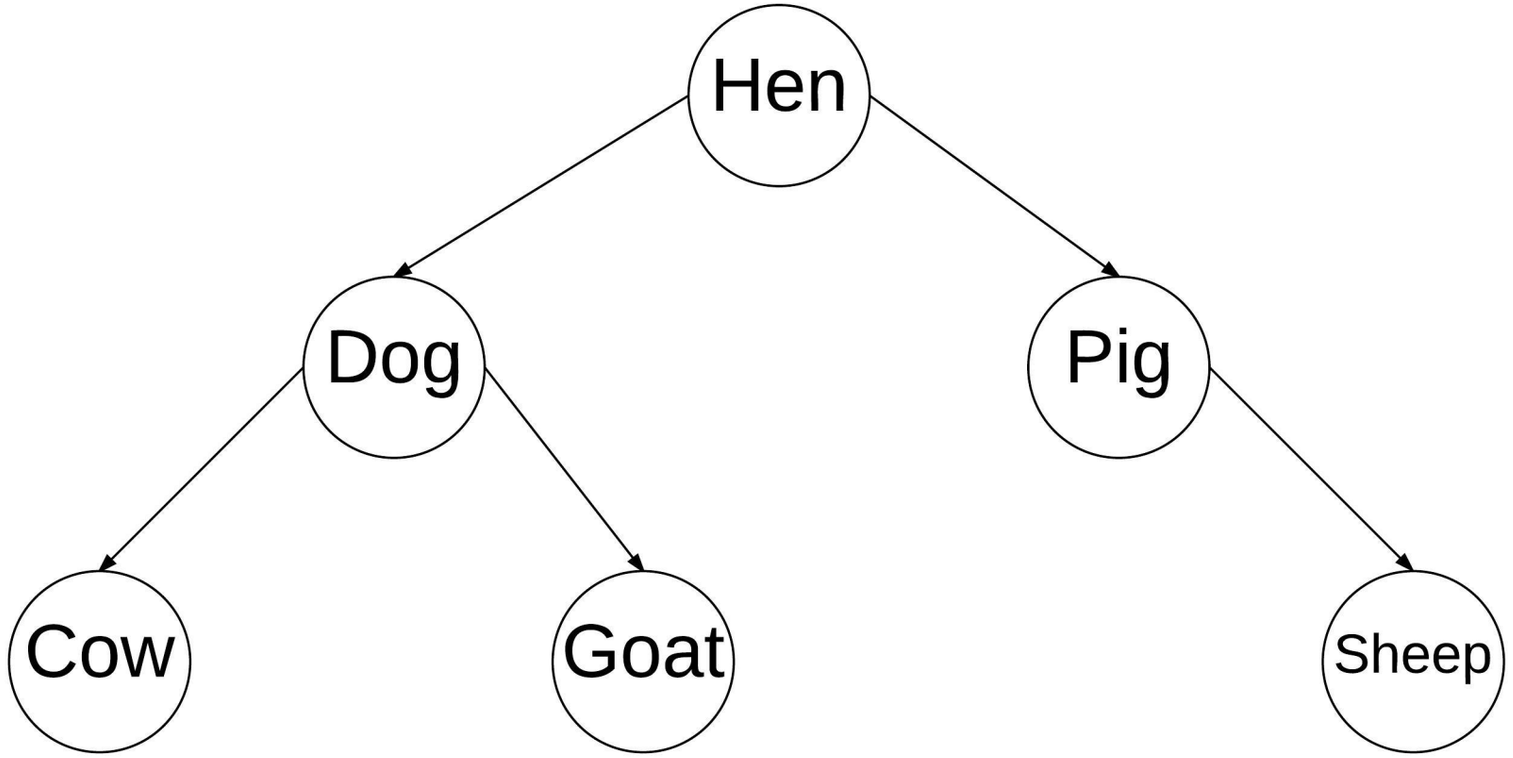
- Recall: depth-first search

Iterative Size

```
public int size2()
// Returns the number of elements in this BST.
{
  int count = 0;
  if (root != null)
  {
    LinkedStack<BSTNode<T>> nodeStack = new LinkedStack<BSTNode<T>>();
    BSTNode<T> currNode;
    nodeStack.push(root);
    while (!nodeStack.isEmpty())
    {
      currNode = nodeStack.top();
      nodeStack.pop();
      count++;
      if (currNode.getLeft() != null)
        nodeStack.push(currNode.getLeft());
      if (currNode.getRight() != null)
        nodeStack.push(currNode.getRight());
    }
  }
  return count;
}
```

Practice

- Step through the size2() code for the following tree. Show the stack contents after 2 iterations of the while() loop



Contains

- Recall: binary search implementation in SortedArrayCollection
  - Recursive solution using indices
- No need for index math in the BST

- How will we implement the recursive contains?

Contains

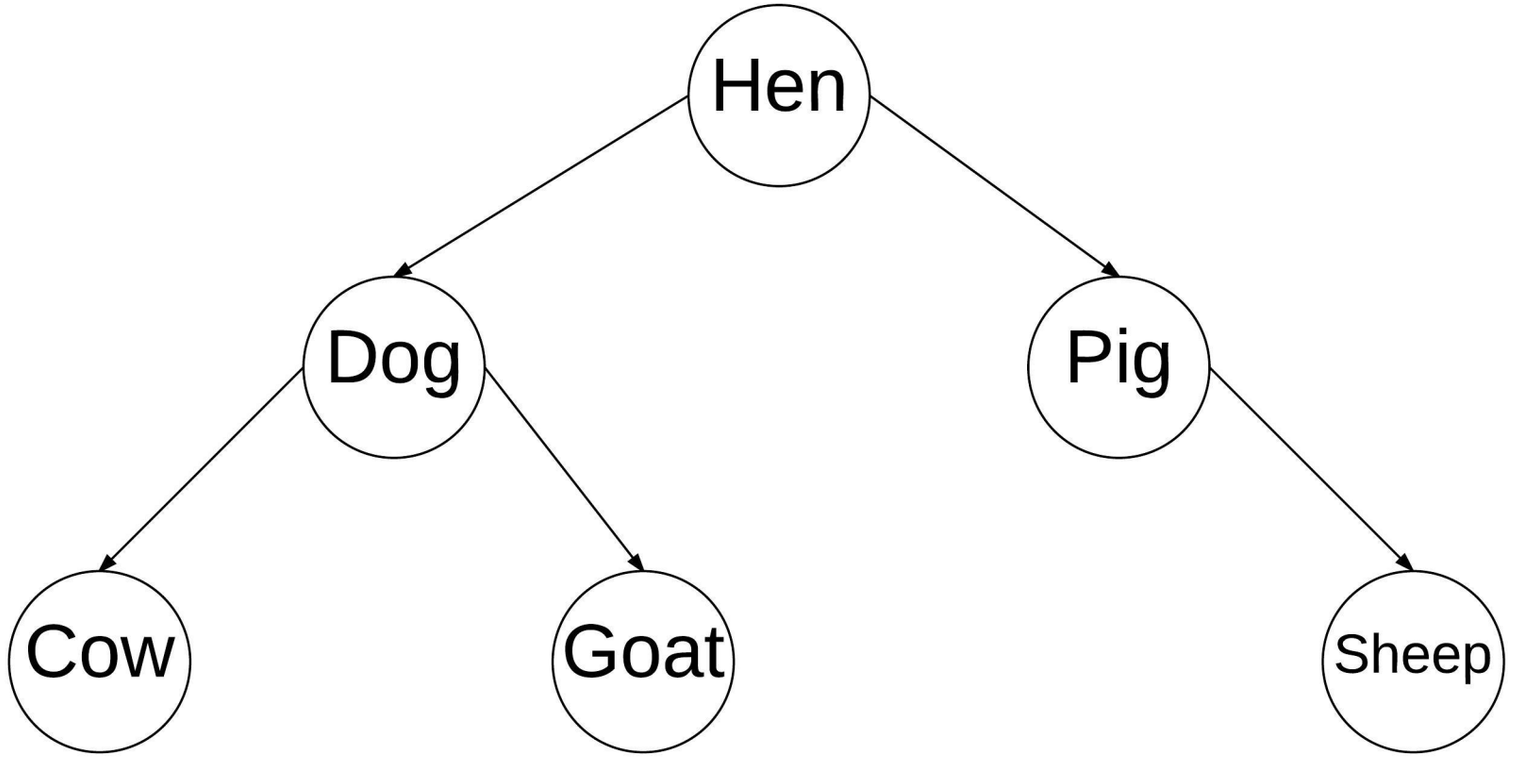
- Base case?
- Base case check?
- Recursive case(s)?

Code

```
 private boolean recContains(T target, BSTNode<T> node)
 // Returns true if the subtree rooted at node contains info i such that
 // comp.compare(target, i) == 0; otherwise, returns false.
{
   if (node == null)
     return false;          // target is not found
   else if (comp.compare(target, node.getInfo()) < 0)
     return recContains(target, node.getLeft());   // Search left subtree
   else if (comp.compare(target, node.getInfo()) > 0)
     return recContains(target, node.getRight());  // Search right subtree
   else
     return true;           // target is found
}
```

Practice Questions

- How many times is contains() called for the input "cow"?
- How many times for the input "Rooster"?



Get

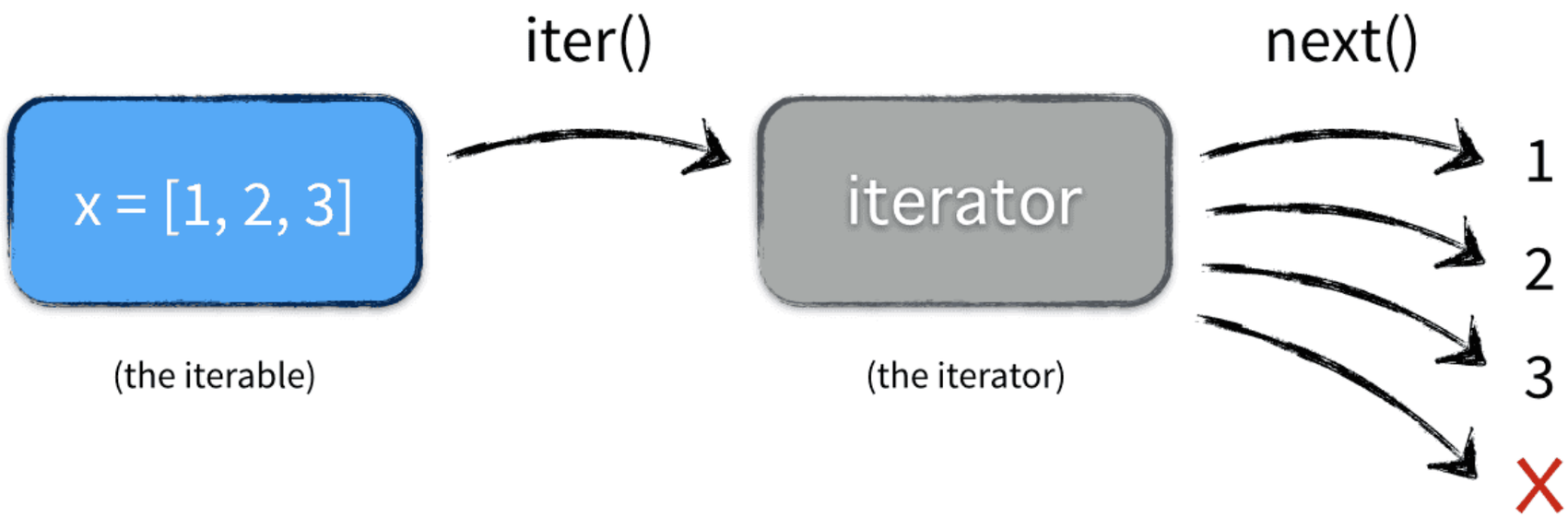
- Closely mirrors the contains implementation
- Rather than returning a boolean, we return the info stored at the node
- If we run off the end of the tree, return null

Practice

```
 private boolean recContains(T target, BSTNode<T> node)
 // Returns true if the subtree rooted at node contains info i such that
 // comp.compare(target, i) == 0; otherwise, returns false.
{
   if (node == null)
     return false;          // target is not found
   else if (comp.compare(target, node.getInfo()) < 0)
     return recContains(target, node.getLeft());   // Search left subtree
   else if (comp.compare(target, node.getInfo()) > 0)
     return recContains(target, node.getRight());  // Search right subtree
   else
     return true;           // target is found
}
```

What's the deal with iterators?

- Iterators allow observation of all data in the specified order
- The standard iterator() factory method and the for-each loop syntax will use inorder traversal
- Specify postorder or preorder with the getIterator() method



Implementation

- Create a ”snapshot” of the tree
  - Based on traversal order
- Store snapshot as a queue
- Perform iterator operations on the queue
  - next() -> ?
  - hasNext() -> ?
  - ~~remove()~~

Snapshot

```
public Iterator<T> getIterator(BSTInterface.Traversal orderType)
// Creates and returns an Iterator providing a traversal of a "snapshot"
// of the current tree in the order indicated by the argument.
// Supports Preorder, Postorder, and Inorder traversal.
{
  final LinkedQueue<T> infoQueue = new LinkedQueue<T>();
  if (orderType == BSTInterface.Traversal.Preorder)
    preOrder(root, infoQueue);
else
if (orderType == BSTInterface.Traversal.Inorder)
    inOrder(root, infoQueue);
else
if (orderType == BSTInterface.Traversal.Postorder)
    postOrder(root, infoQueue);
```

Iterator() anonymous class

```
return new Iterator<T>()
{
  public boolean hasNext()
  // Returns true if the iteration has more elements; otherwise returns false.
  {
    return !infoQueue.isEmpty();
  }

  public T next()
  // Returns the next element in the iteration.
  // Throws NoSuchElementException - if the iteration has no more elements
  {
    if (!hasNext())
      throw new IndexOutOfBoundsException("illegal invocation of next " +
                                          " in BinarySearchTree iterator.\n");
    return infoQueue.dequeue();
  }

  public void remove()
  // Throws UnsupportedOperationException.
  // Not supported. Removal from snapshot iteration is meaningless.
  {
    throw new UnsupportedOperationException("Unsupported remove attempted on "
                                            + "BinarySearchTree iterator.\n");
  }
};
```

Snapshot

```
public Iterator<T> getIterator(BSTInterface.Traversal orderType)
// Creates and returns an Iterator providing a traversal of a "snapshot"
// of the current tree in the order indicated by the argument.
// Supports Preorder, Postorder, and Inorder traversal.
{
  final LinkedQueue<T> infoQueue = new LinkedQueue<T>();
  if (orderType == BSTInterface.Traversal.Preorder)
    preOrder(root, infoQueue);
else
if (orderType == BSTInterface.Traversal.Inorder)
    inOrder(root, infoQueue);
else
if (orderType == BSTInterface.Traversal.Postorder)
    postOrder(root, infoQueue);
```

Snapshot Recursion

- Base case?
- Base case check?
- Recursive case?

Preorder

```
private void preOrder(BSTNode<T> node, LinkedQueue<T> q)
// Enqueues the elements from the subtree rooted at node into q in preOrder.
{
  if (node != null)
  {
    q.enqueue(node.getInfo());
    preOrder(node.getLeft(), q);
    preOrder(node.getRight(), q);
  }
}
```

Inorder, Postorder

```
private void inOrder(BSTNode<T> node, LinkedQueue<T> q)
// Enqueues the elements from the subtree rooted at node into q in inOrder.
{
  if (node != null)
  {
    inOrder(node.getLeft(), q);
    q.enqueue(node.getInfo());
    inOrder(node.getRight(), q);
  }
}
```

```
private void postOrder(BSTNode<T> node, LinkedQueue<T> q)
// Enqueues the elements from the subtree rooted at node into q in postOrder.
{
  if (node != null)
  {
    postOrder(node.getLeft(), q);
    postOrder(node.getRight(), q);
    q.enqueue(node.getInfo());
  }
}
```

Recap

- The binary tree interface extends the collection with a few new methods
  - Min, max, getIterator
- The recursive structure of the tree lends itself to recursive implementations
  - Iterative versions will likely use a stack
- Careful definition of the base and recursive cases can lead to very simple code
  - E.g., considering the base case of an empty tree instead of a single node

Next Time...

- Dale, Joyce, Weems Chapter 7.6-8
  - Remember, you need to read it BEFORE you come to class!
- Check the course webpage for practice problems
- Peer Tutors
  - http://www.csc.villanova.edu/help/

Code Samples

```
BinarySearchTree<Character> example = new BinarySearchTree<Character>();
Iterator<Character> iter;

example.add('P'); example.add('F'); example.add('S'); example.add('B');
example.add('H'); example.add('R'); example.add('Y'); example.add('G');
example.add('T'); example.add('Z'); example.add('W');
```

Code Samples

```
// Preorder
System.out.print("\nPreorder:  ");
iter = example.getIterator(BSTInterface.Traversal.Preorder);
while (iter.hasNext())
  System.out.print(iter.next());


// Inorder
System.out.print("Inorder:   ");
iter = example.getIterator(BSTInterface.Traversal.Inorder);
while (iter.hasNext())
  System.out.print(iter.next());


// Inorder again
System.out.print("\nInorder:   ");
for (Character ch: example)
   System.out.print(ch);
```