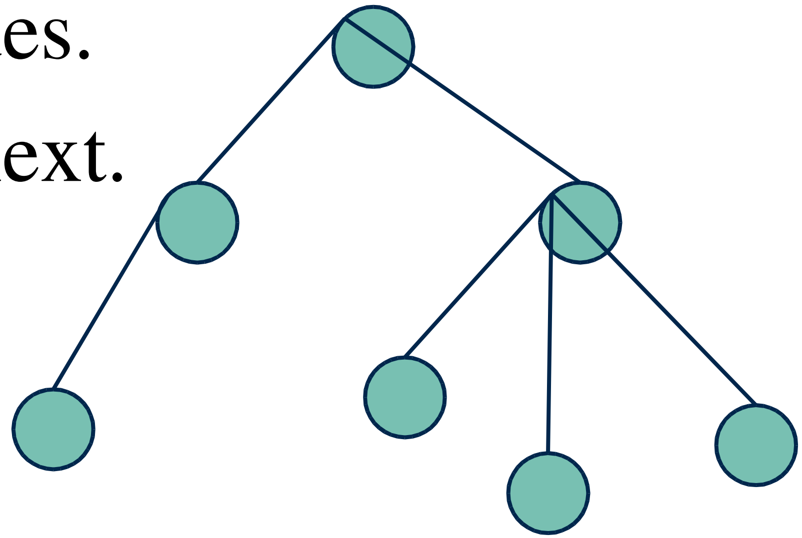
Why Trees?

- Alternatives
  - Ordered arrays
    - Fast searching (binary search)
    - Slow insertion (must shift)
  - Linked lists
    - Fast insertion
    - Slow searching (must start from head of list)
- Want:
  - A data structure that has quick insertion/deletion, as well as fast search

What is a Tree?

- A tree consists of nodes connected by edges.
- Tree nodes are similar to list nodes.
- Data is stored at the nodes.
- Edges tell where to go next.



Tree Terminology

- Root – The node at the top
- Parent – Any node having an edge running downward to another node.
- Child – Any node having an edge running upward to another node (parent).
- Sibling – A node having the same parent.
- Leaf – A node that has no children.
- Path – A list of successive nodes connected by edges.

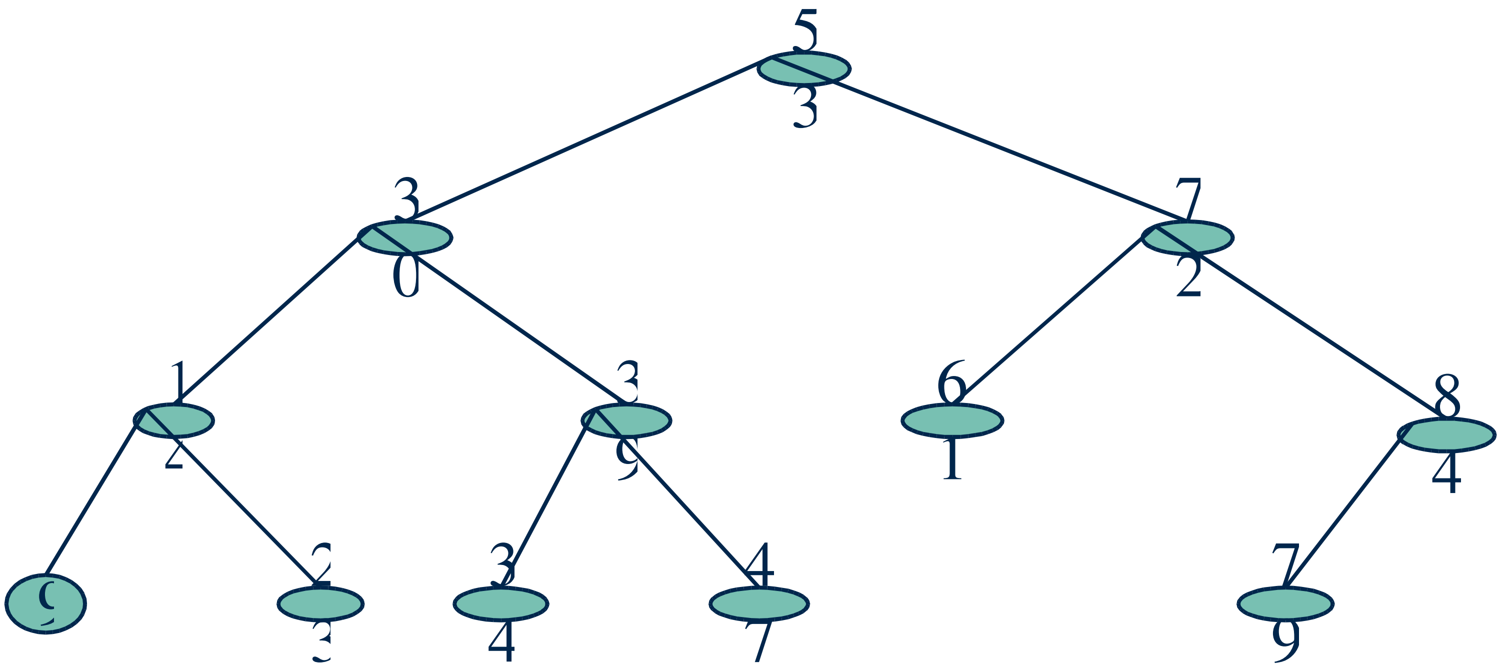
Tree Terminology

- Subtree – Any node may be considered the root of a subtree.
- Visit – A node is *visited* when program control arrives at the node.
- Traversal – To traverse a tree means to visit all nodes in some specific order.
- Levels – The level of a node refers to how far it is from the root (how many edges you need to traverse).
- Height/Depth – The height of a tree is the level of its youngest leaf.

Binary Trees

- A binary tree is a tree whose nodes may have at most two children.
- The two children are call the left and right child.
- Binary trees are data structures often with the binary search imbedded.
- Technically called the binary search tree.

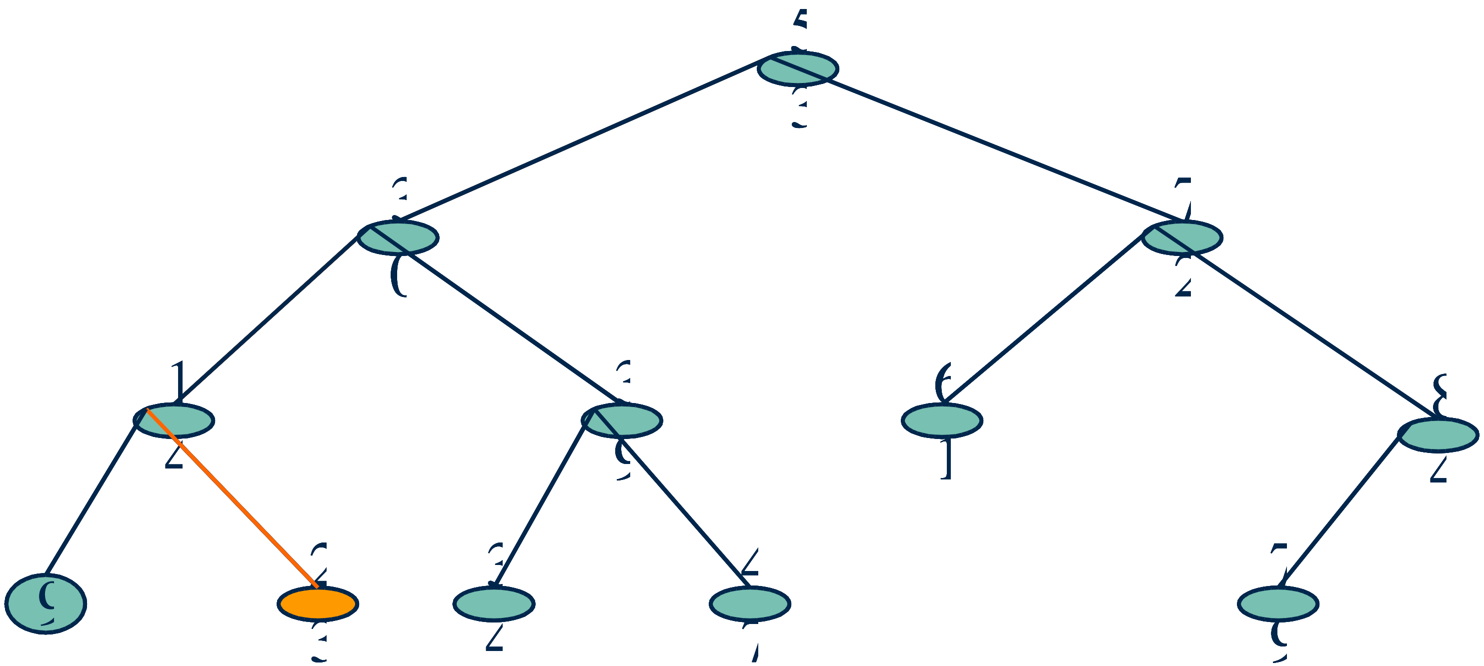
Binary Search Tree



Searching in a BST

- Start from the root
- Compare key with current node’s data
- If equal, report success.
- If less go to the left child.
- If more go to the right child.
- If no child present in the desired direction, report failure.

Looking for 23



Tree Node

```
class Node {
  private int idata;    // data
  Node left;    // reference
  Node right;      // reference
  Node (int key) {…}
  public void printNode() { … }
   ....
}
```

- Again, the references maintain the tree structure.
- If a child does not exist, the corresponding reference is set to `null`.

Data Embedding

```
class Node {
  Person p;     // reference to data
  Node left;    // reference
  Node right;     // reference
}
```

- It's not necessary to place data items directly in the node.
- A reference to an object is a common approach.

The Tree Class

```
class Tree {
  private Node root;

  public Tree() {root = null};

  public Node find(int key){…}
  public void insert(int key) {…}
  public void delete(int key) {…}
  //other methods …
}
```

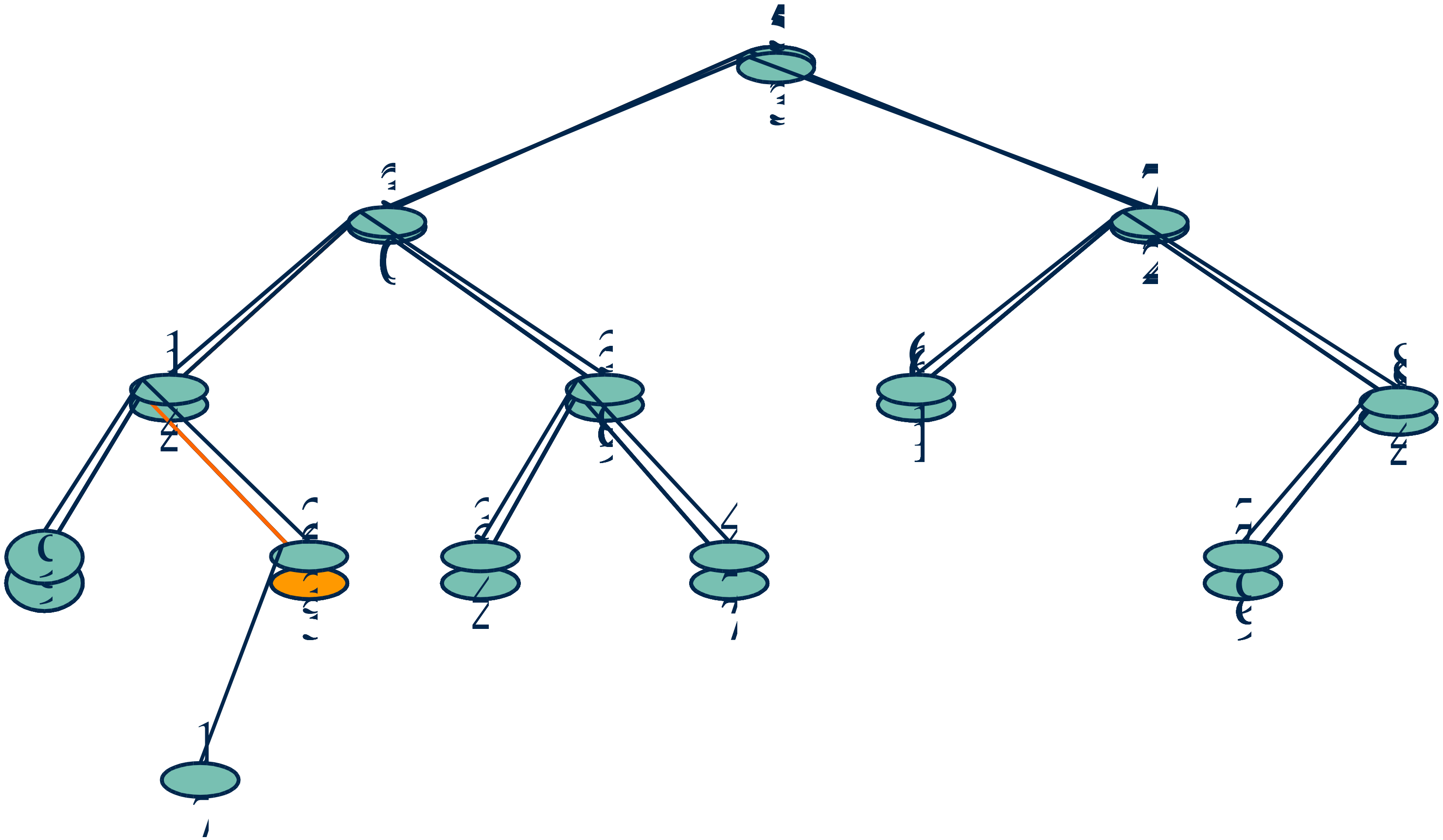
find

```
public Node find(int key) {
  Node current = root;
  while (current != null) {
    if (key == current.getData())
      return current;
    else if (key < current.getData())
       current = current.getLeft();
     else
      current = current.getRight();
  }
  return null;
}
```

Insertion

- Very similar to find, as we must first find the appropriate place to insert it.
- The difference is where `find` simply returns `null` when a node doesn't exist, `insert` creates the node.
- A newly inserted node is always a leaf.

Inserting 17



insert

```
public void insert(int key) {
  Node current = root;
  Node parent; // reference

  while (current != null) {
    parent = current;
    if (key < current.getData())
       current = current.getLeft();

     else
      current = current.getRight();
  }
```

insert

```
  Node newNode = new Node(key);
  if (key < parent.getData())
    parent.setLeft(newNode);
  else
    parent.setRight(newNode);
  return;
}
```

main

```
class TreeApp {
  public static void main(String[] args) {
    Tree theTree = new Tree();

    theTree.insert(50);
    theTree.insert(25);
    theTree.insert(75);
    Node result = theTree.find(25);
    if (result != null)
      result.printNode();
  }
}
```

Search Efficiency in a BST

- In the worst case, we must go to a leaf.
- Before we can get to the leaf, we must visit/compare with all its ancestors.
- The number of steps is equal to the leaf's level.
- A full BST has half of its nodes at the bottom level.
- A full BST with n nodes has $log(n+1)$ levels.

The Problem of an Unbalanced Tree

- Consider inserting this series of numbers into a BST: 1 2 3 4 5 6 7 8 9 10 …
- We end up with a one-sided tree!
- Unbalanced trees have terrible search efficiency.
- Unbalanced trees are unlikely if incoming data is random and many.
- Unbalanced trees can be balanced.