Data Structures and Algorithms

CS245-2017S-07

Tree Operations

David Galles

Department of Computer Science
University of San Francisco

07-0: Binary Tree Definition

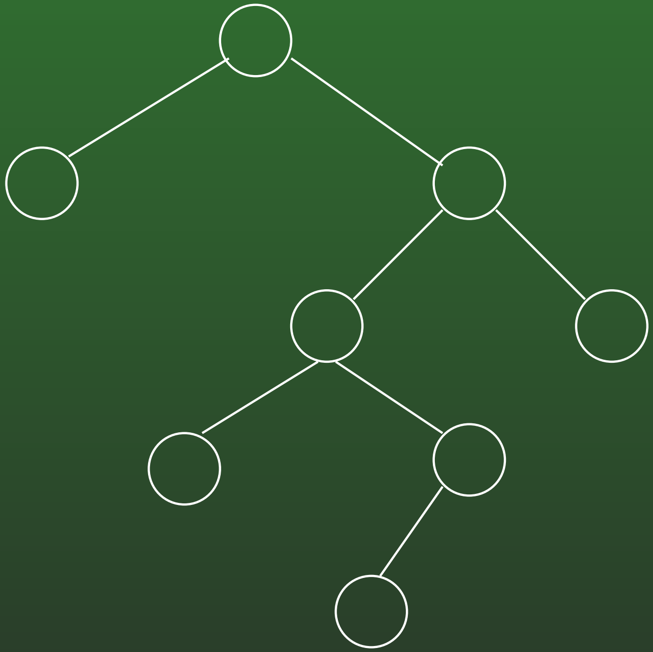
```
class Node {
  Node() { }
  Node(Comparable elem) {
    this.element = element;
  }
  Node(Object element, Node left, Node right) {
    this.element = element;
    this.left = left;
    this.right = right;
  }
   /* Access methods on next slide */
   private Node left;
   private Node right;
   private Comparable element;
}
```

07-1: Binary Tree Access Methods

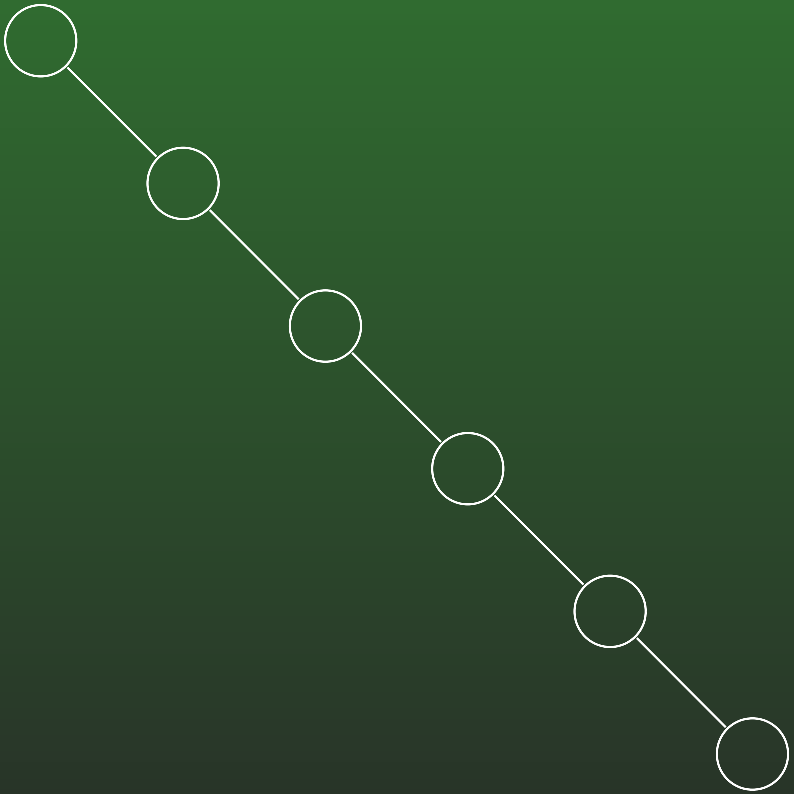
```
Node left() {       void setLeft(Node left) {
  return left;       this.left = left;
}                   }

Node right() {      void setRight(Node right) {
  return right;      this.right = right;
}                   }

Comparable element() {
  return element;
}

void setElement(Comparable element) {
  this.element = element;
}
```

07-2: Tree Operations – Height

- Returns the height of the tree
  - (Length of the path to the deepest leaf) + 1



Height = 5



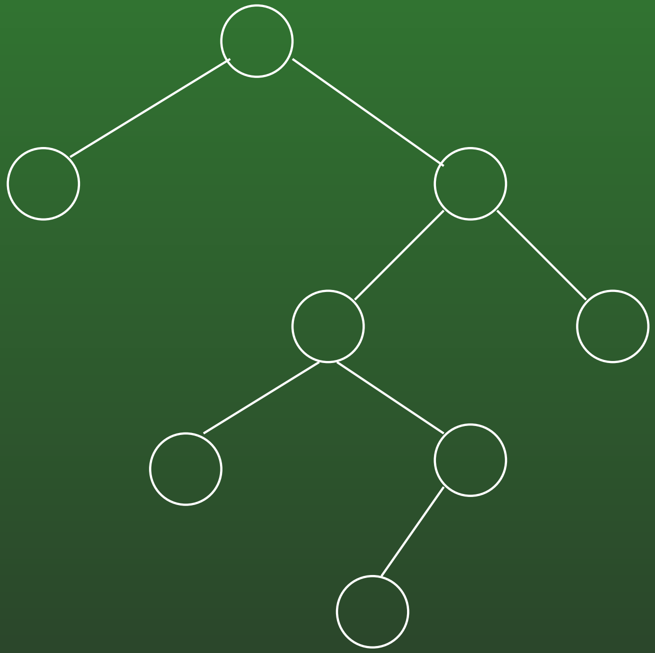
Height = 6

07-3: Tree Operations – Height

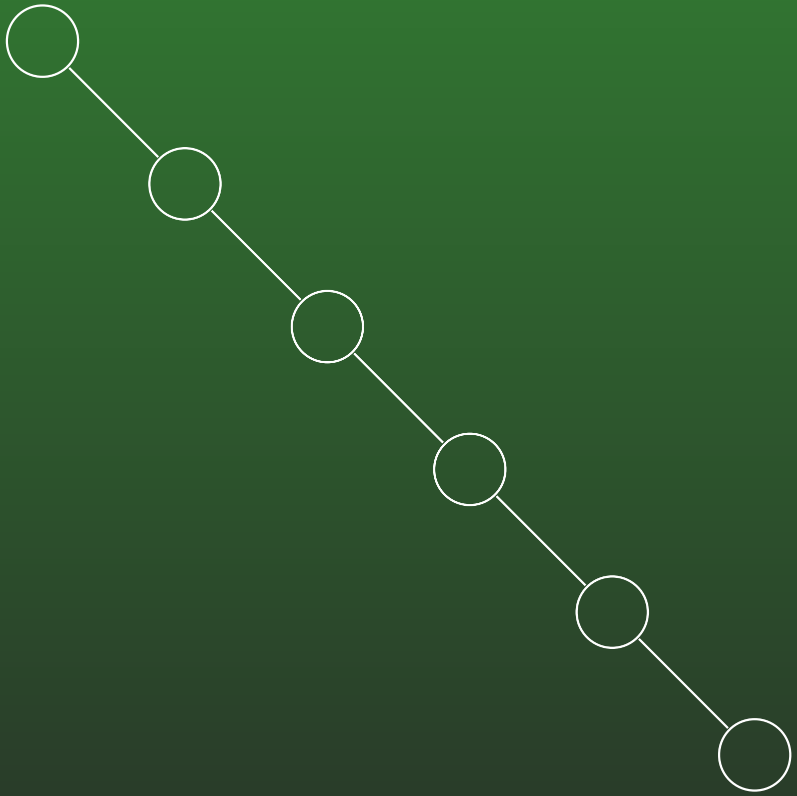
```
int height(Node tree) {
  if (tree == null)
    return 0;
  return 1 + MAX(height(tree.left()),
                 height(tree.right()));
}
```

07-4: Tree Operations – NumNodes

- Returns the number of nodes in a tree



Number of Nodes = 8

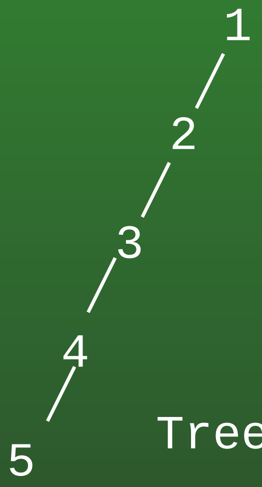


Number of Nodes = 6

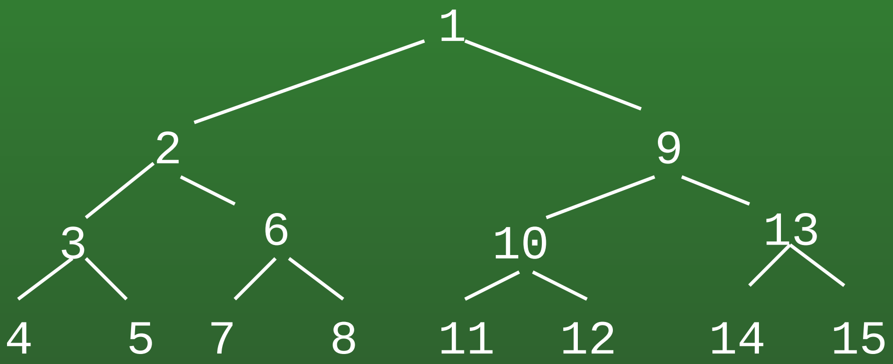
07-5: Tree Operations – NumNodes

```
int numNodes(Node tree) {
  if (tree == null)
    return 0;
  return 1 + numNodes(tree.left()) +
             numNodes(tree.right());
```

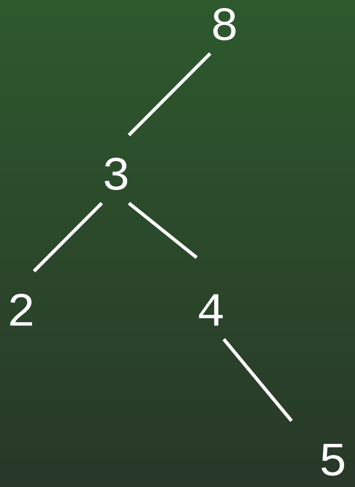
07-6: Writing Tree Functions



Tree1



Tree2

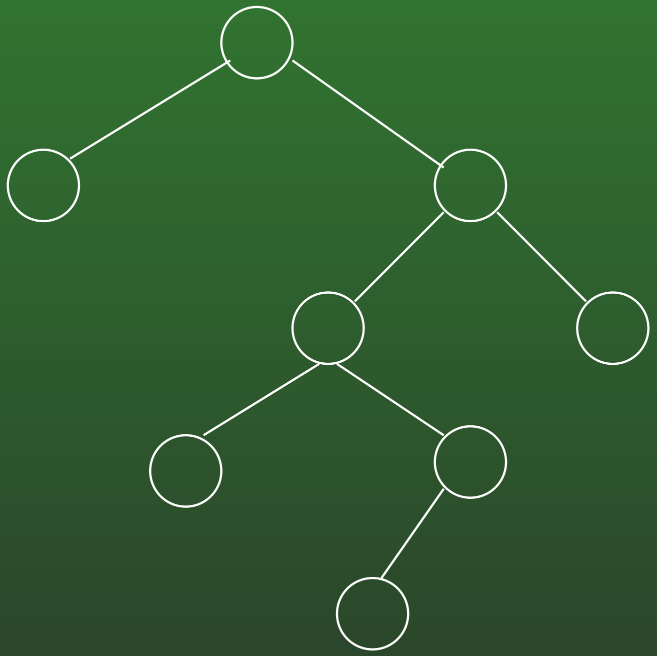


Tree3

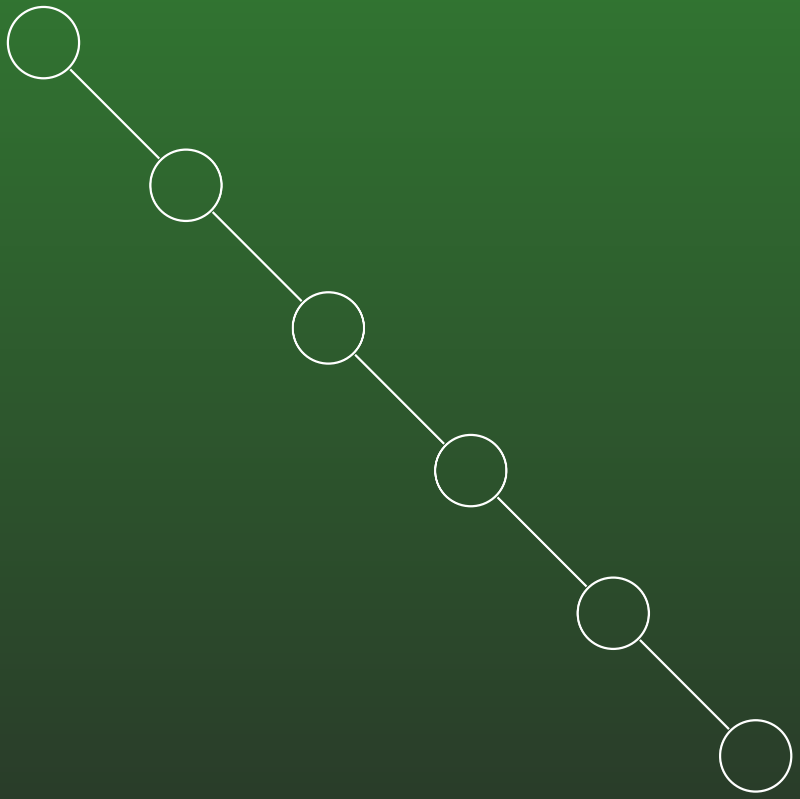
Write `find`, `numLeaves`, `shallowestleaf`

07-7: Tree Operations – NumLeaves

- Returns the number of leaves in a tree



Number of Leaves = 4



Number of Leaves = 1

07-8: Tree Operations – NumLeaves

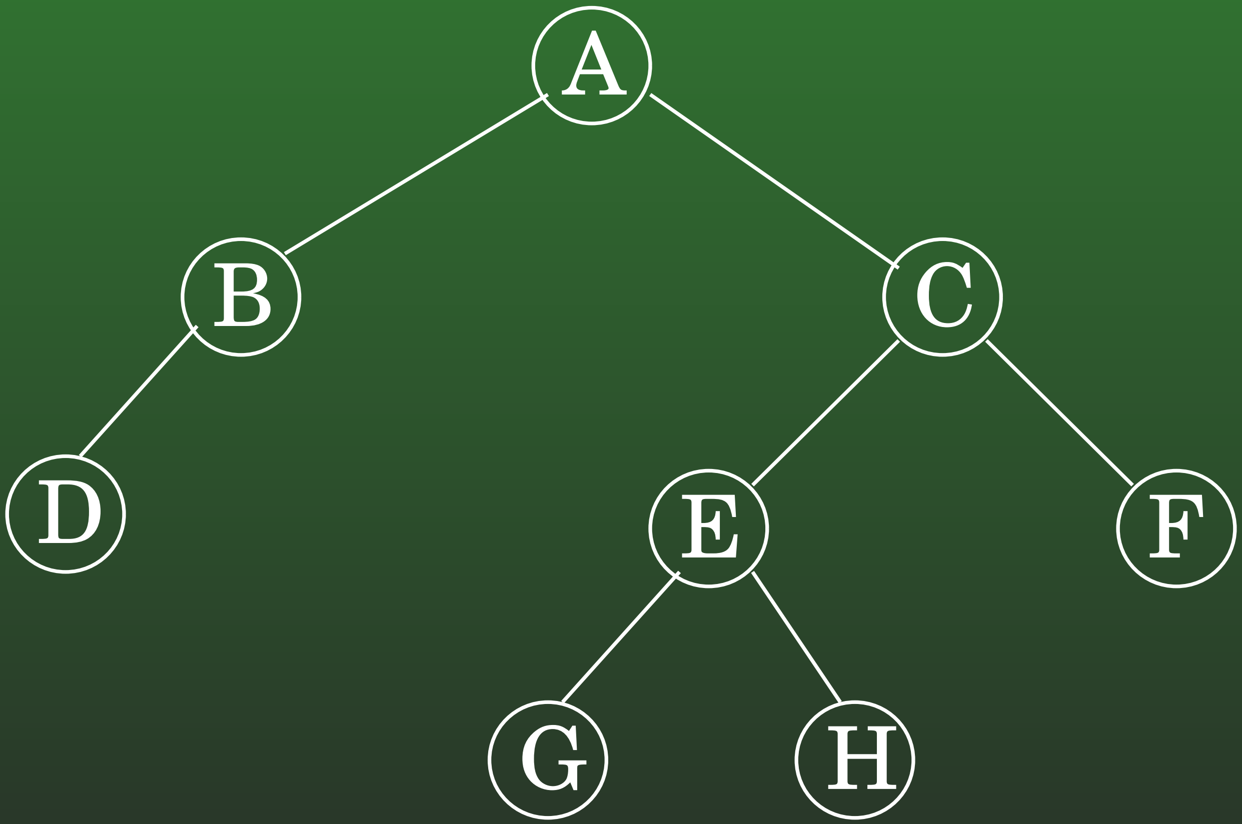
```
int numLeaves(Node tree) {
  if (tree == null)
    return 0;
  if ((tree.left() == null) &&
      (tree.right() == null))
    return 1;
  return numLeaves(tree.left()) +
         numLeaves(tree.right());
}
```

07-9: Tree Traversals

- PREORDER Traversal
  - Do operation on root of the tree
  - Traverse left subtree
  - Traverse right subtree
- INORDER Traversal
  - Traverse left subtree
  - Do operation on root of the tree
  - Traverse right subtree
- POSTORDER Traversal
  - Traverse left subtree
  - Traverse right subtree
  - Do operation on root of the tree

07-10: PREORDER Traversal

Printing out trees (Showing the shape of the tree in the printout)



```
A
  B
    D
  C
    E
      G
      H
    F
```

07-11: PREORDER Traversal

Printing out trees (Showing the shape of the tree in the printout)

- First print the root at current indent level
  - Print the left subtree with larger indentation
  - Print the right subtree with larger indentation

07-12: Printing Binary Trees

```
void print(Node tree, int indent) {
  if (tree != null) {
    for(int i=0; i<indent; i++) {
      System.out.print("\t");
    System.out.println(tree.element().toString())
  print(tree.left(), indent + 1);
  print(tree.right(), indent + 1);
}
```

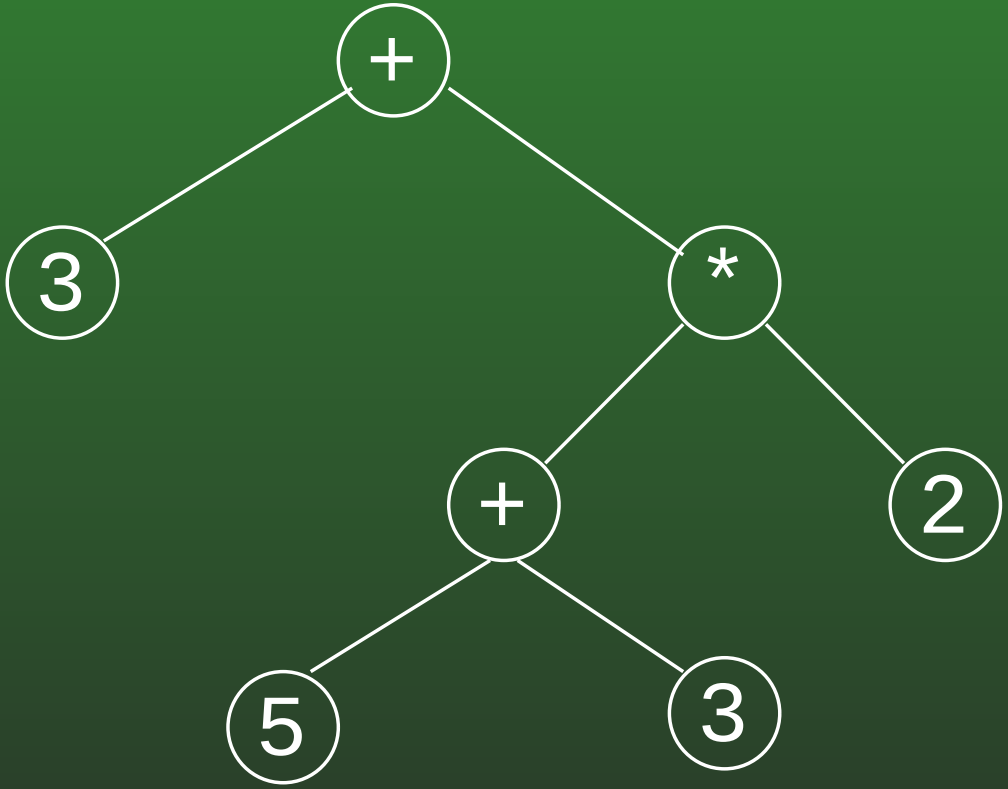
07-13: INORDER Traversal

Printing all elements in a Binary Search Tree in order

- (Already covered in previous slides)

07-14: POSTORDER Traversal

Calculating the Value of an expression tree



07-15: POSTORDER Traversal

Calculating the Value of an expression tree

- Base case:
  - Return value stored at leaf
- Recursive case:
  - Calculate value of left subtree
  - Calculate value of right subtree
  - Calculate expression value

07-16: Expression Tree Value

```
int value(Node tree) {
  if (tree.left() == null && tree.right() == null)
    return ((Integer) tree.element()).intValue();
  int left = value(tree.left());
  int right = value (tree.right());
  char op = ((Character) tree.element()).charValue();
  switch (op) {
    case ’+’:
      return left + right;
    case ’*’:
      return left * right;
      ...
  }
}
```