Trees, Binary Search Trees, Recursion, Project 2

Bryce Boe

2013/08/01

CS24, Summer 2013 C

Outline

- Stack/Queue Review
- Trees
- Recursion
- Binary Search Trees
- Project 2

Stack / Queue Review

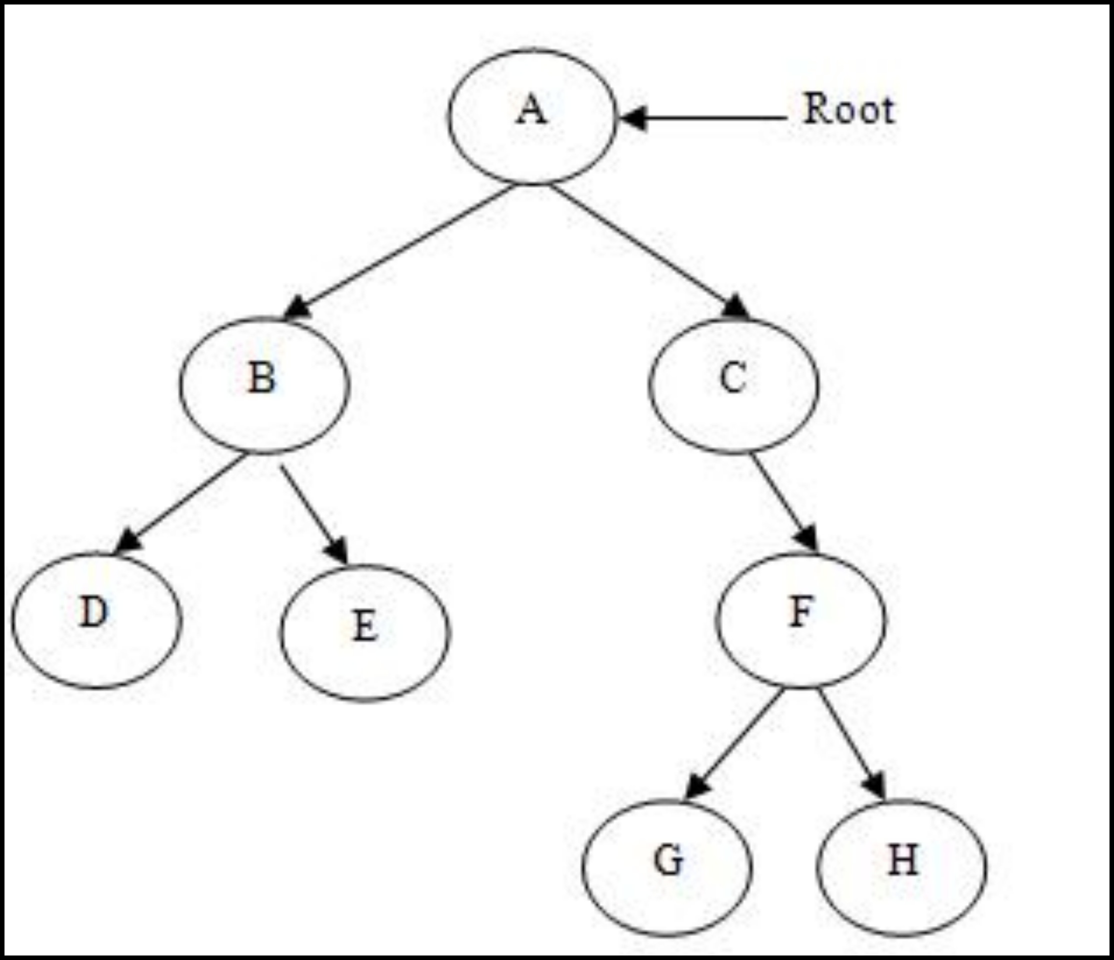
- Stack operations
  - push
  - pop
- Queue operations
  - enqueue
  - dequeue

TREES

Tree Explained

- Data structure composed of nodes (like a linked list)
- Each node in a tree can have one or more children (binary tree has at most two children)

Binary Tree



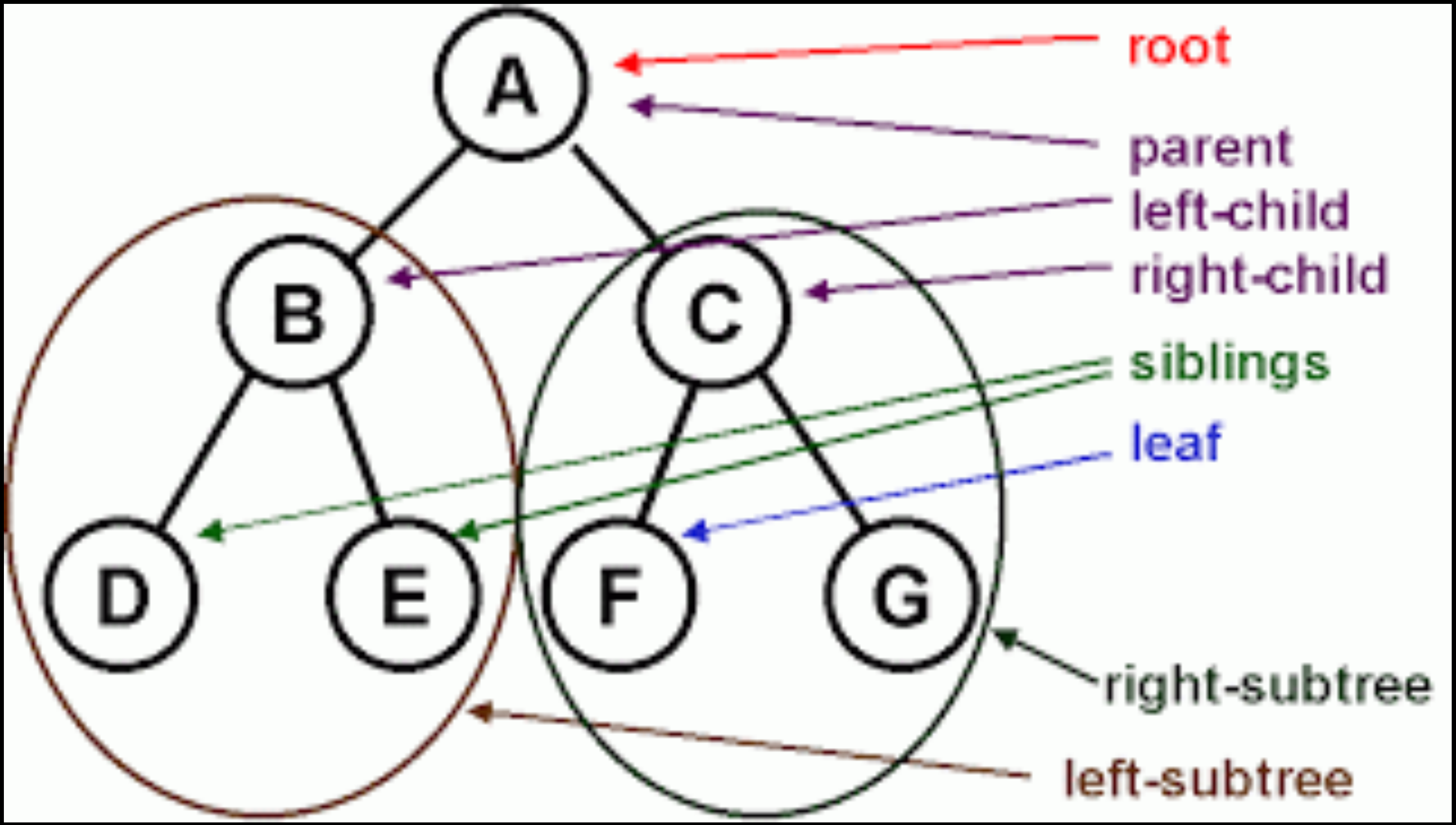
Tree Properties

- The **root** is the top-most node of the tree (has no parent)
- A node's **parent** is the node immediately preceding it (closer to the root)
- A node can have at most two **children** or **child** nodes
- A **leaf** is a node with no children

More Properties

- A node’s **ancestors** are all nodes preceding it
- A node’s **descendants** all all nodes succeeding it
- A **subtree** is the complete tree starting with a given node and including its descendants

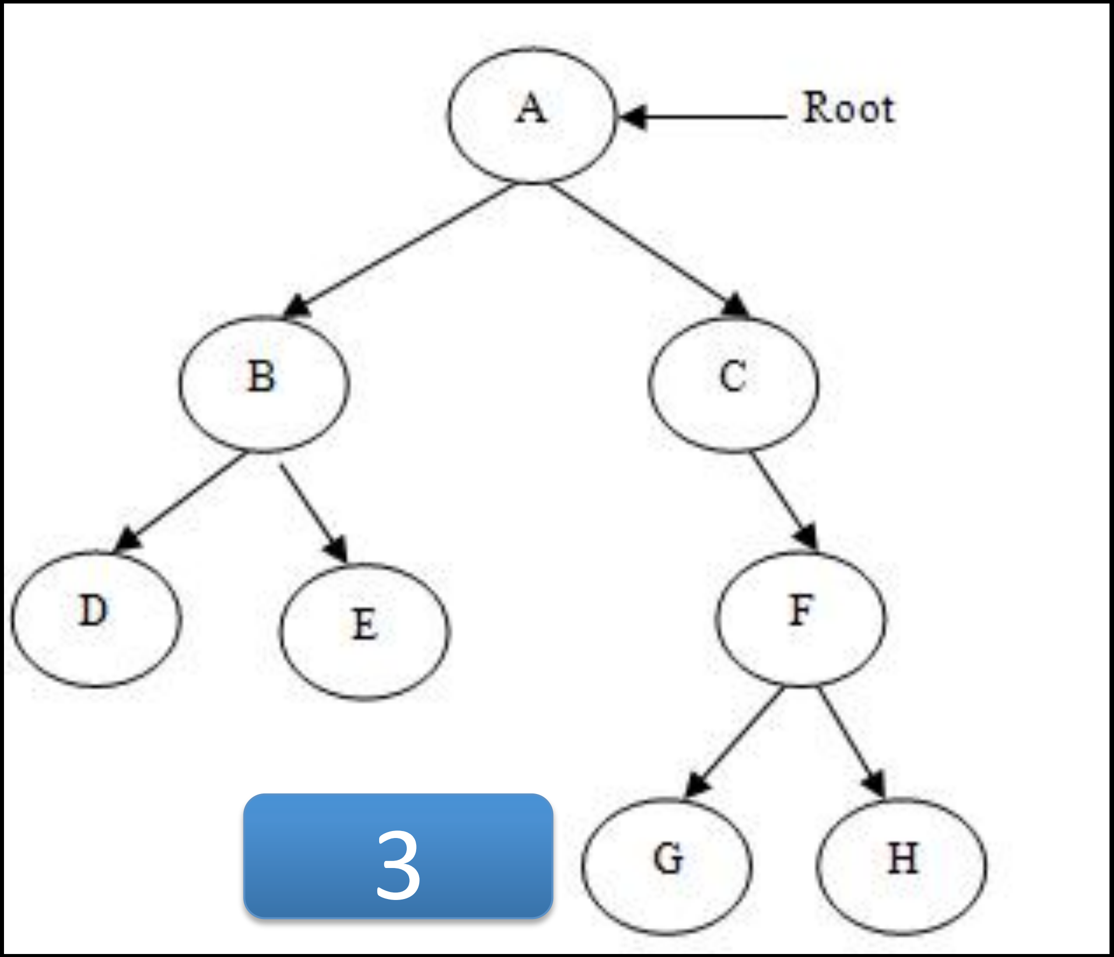
Tree properties



More Properties

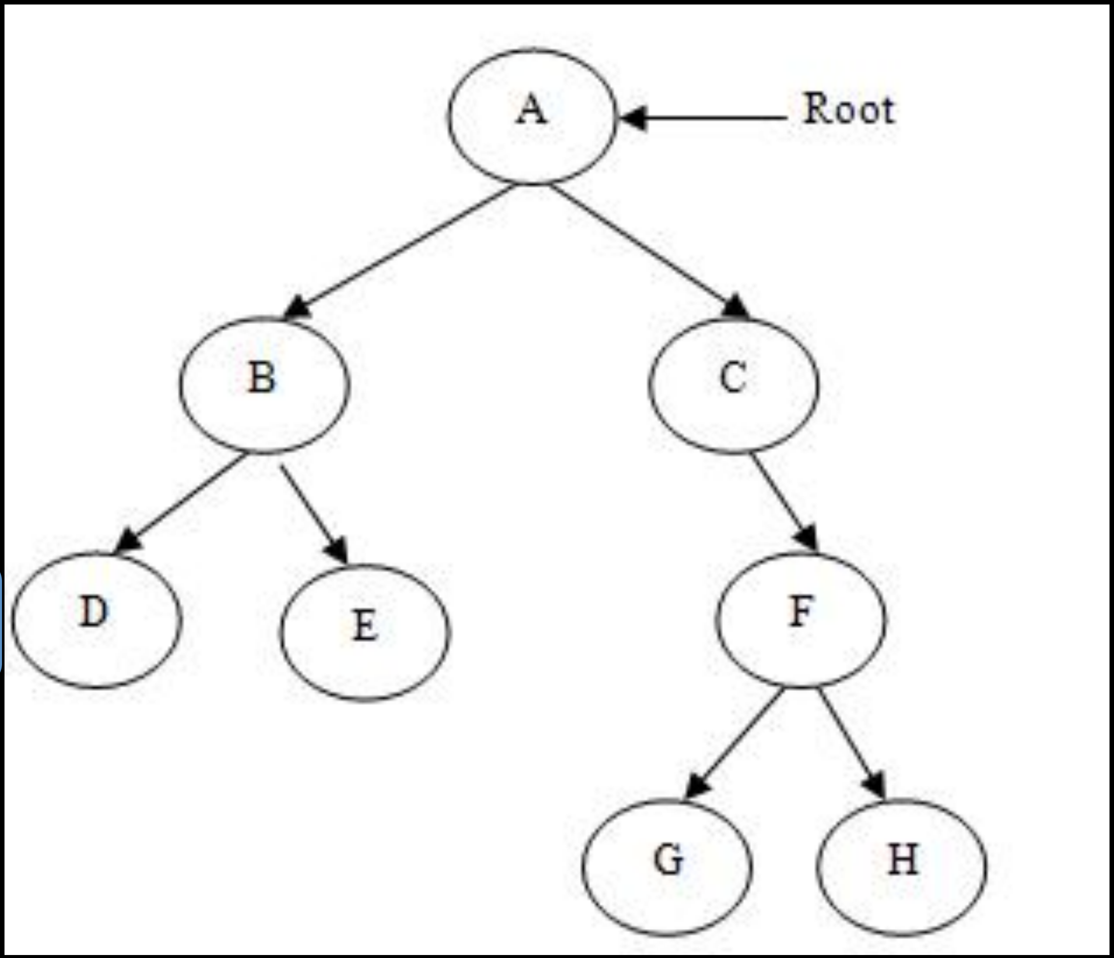
- The **depth** of a node is how far it is away from the root (the root is at depth 0)
- The **height** of a node is the maximum distance to one of its descendent leaf nodes (a leaf node is at height 0)
- The **height** of a tree is the height of the root node

What is the depth of G?



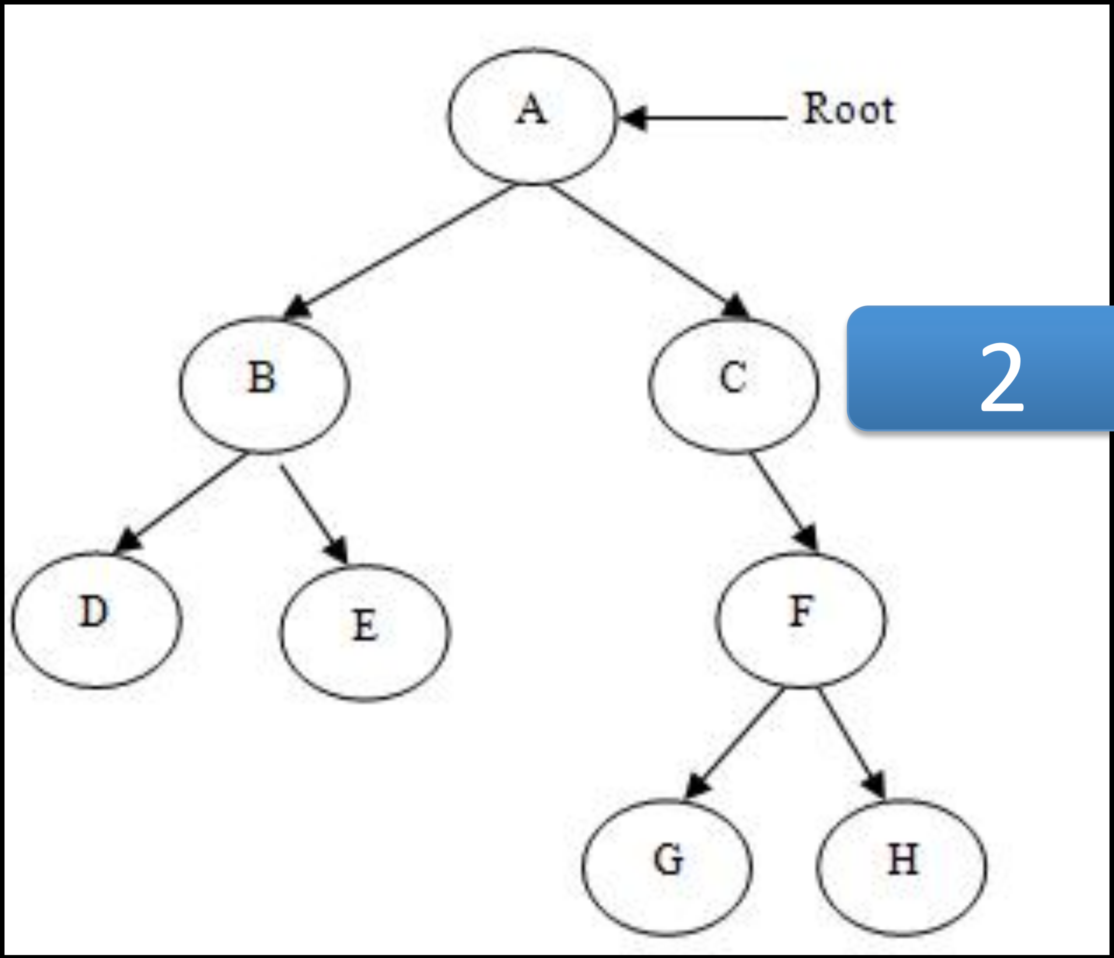
3

What is the depth of D?



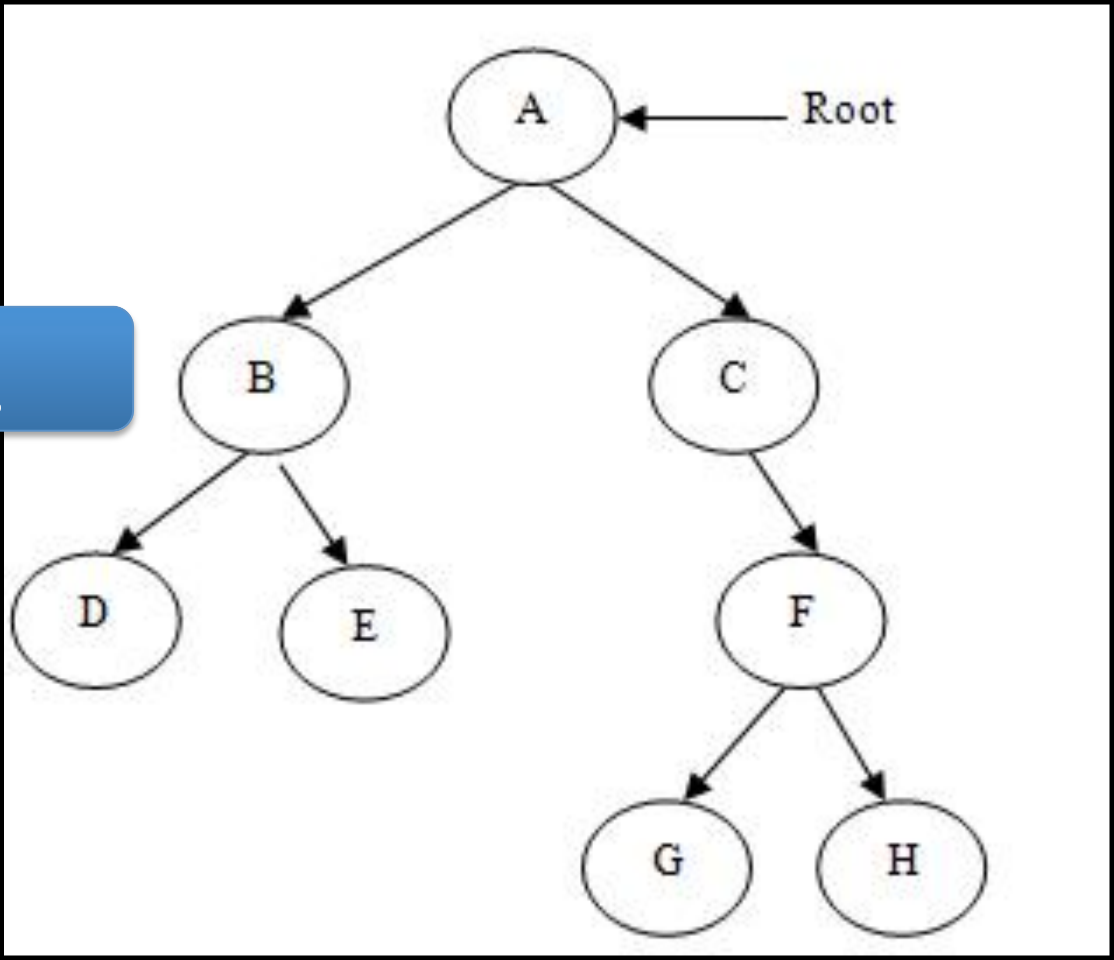
2

What is the height of C?



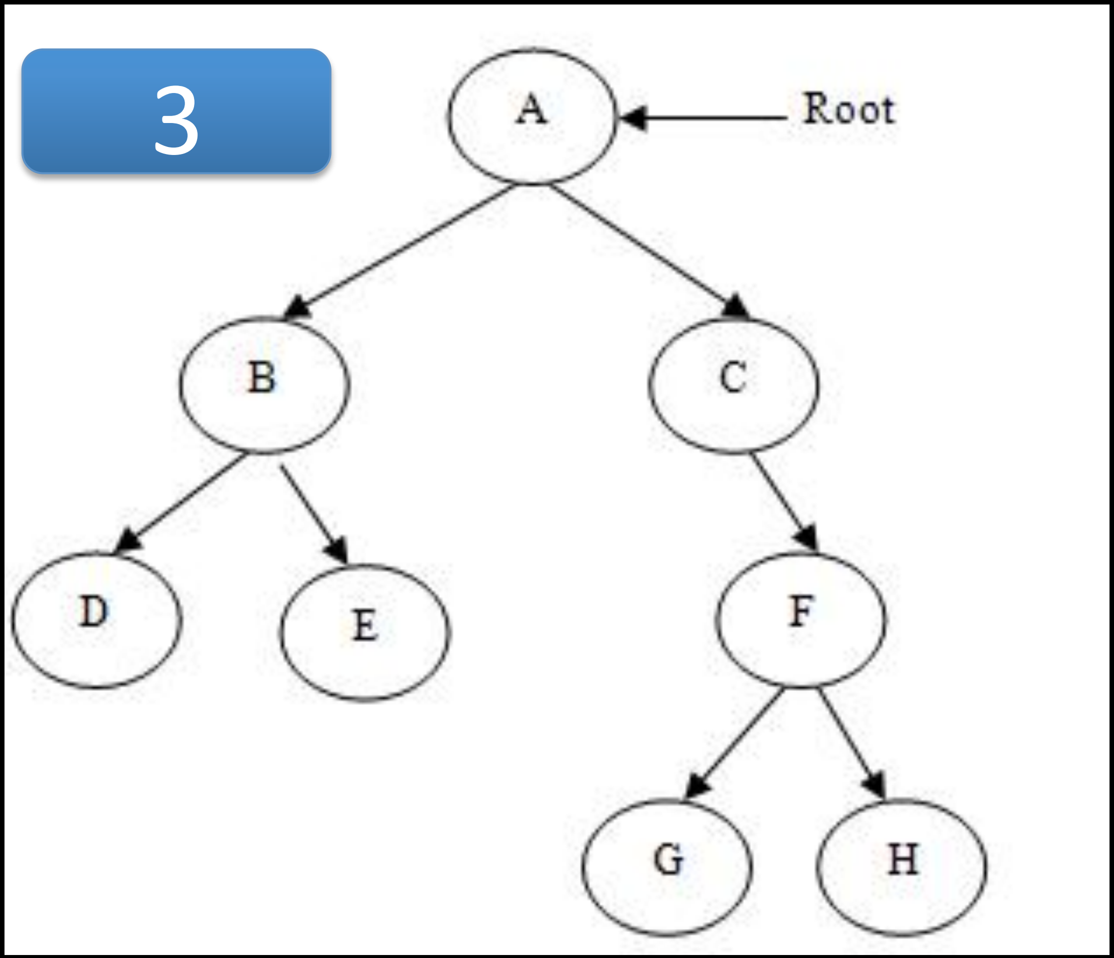
2

What is the height of B?

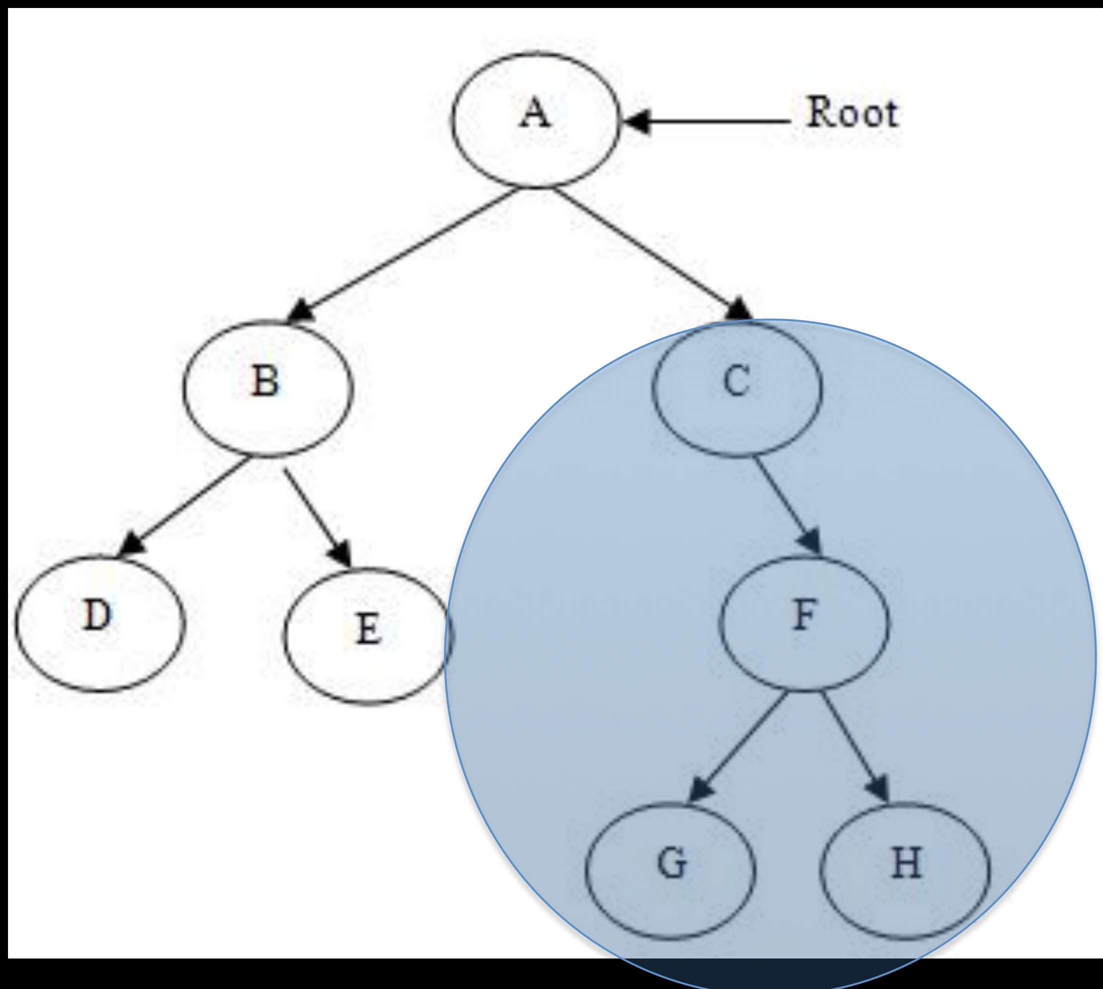


1

What is the height of the tree?



What nodes make up A's right subtree?



RECURSION

What is recursion?

- The process of solving a problem by dividing it into similar subproblems
- Examples
  - Factorial: 5! = 5 * 4! = 5 * 4 * 3!
  - Fibonacci Numbers: F(N) = F(n-1) + F(n-2)
  - Length of linked list: L(node) = 1 + L(node->next)

Factorial

- Base Case:
  - $F(1) = 1$
- General Case
  - $F(n) = n * F(n-1)$

Factorial

```
int factorial(n) {
        if (n < 1) throw 1;  // Error condition
        else if (n == 1)  // Base Case
                return 1;
        else  // General Case
                return n * factorial(n – 1);
}
```

Fibonacci Numbers

- Base Cases:
  - $F(0) = 0$
  - $F(1) = 1$
- General Case:
  - $F(n) = F(n-1) + F(n-2)$

Linked List Length

- Base Case:
  – Length(last node) = 1
- General Case:
  – Length(node) = 1 + Length(node->next);

Linked List Length (option 1)

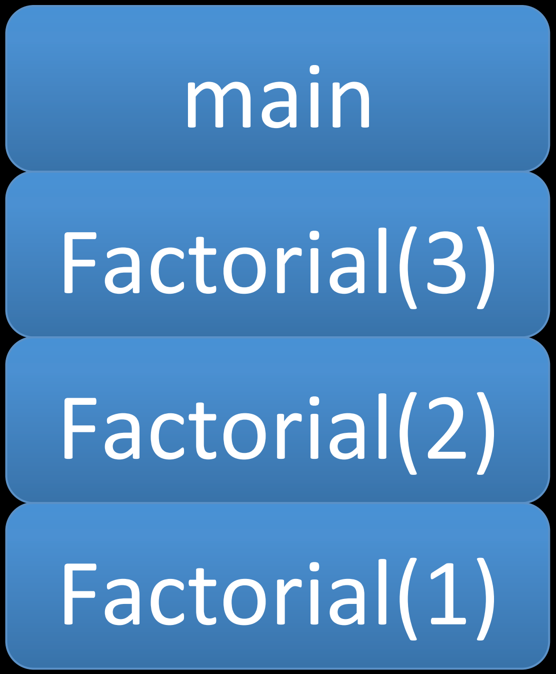
```
int length(Node *n) {
    if (n == NULL)  // Base Case
        return 0;
    else  // General Case
        return 1 + length(n->next);
}
```

Linked List Length (option 2)

```
int length(Node *n) {
        if (n == NULL) throw 1; // Error condition
        else if (n->next == NULL)  // Base Case
                return 1;
        else  // General Case
                return 1 + length(n->next);
}
```

Recursion and the Stack Segment

- main calls Factorial(3)



C++ Examples

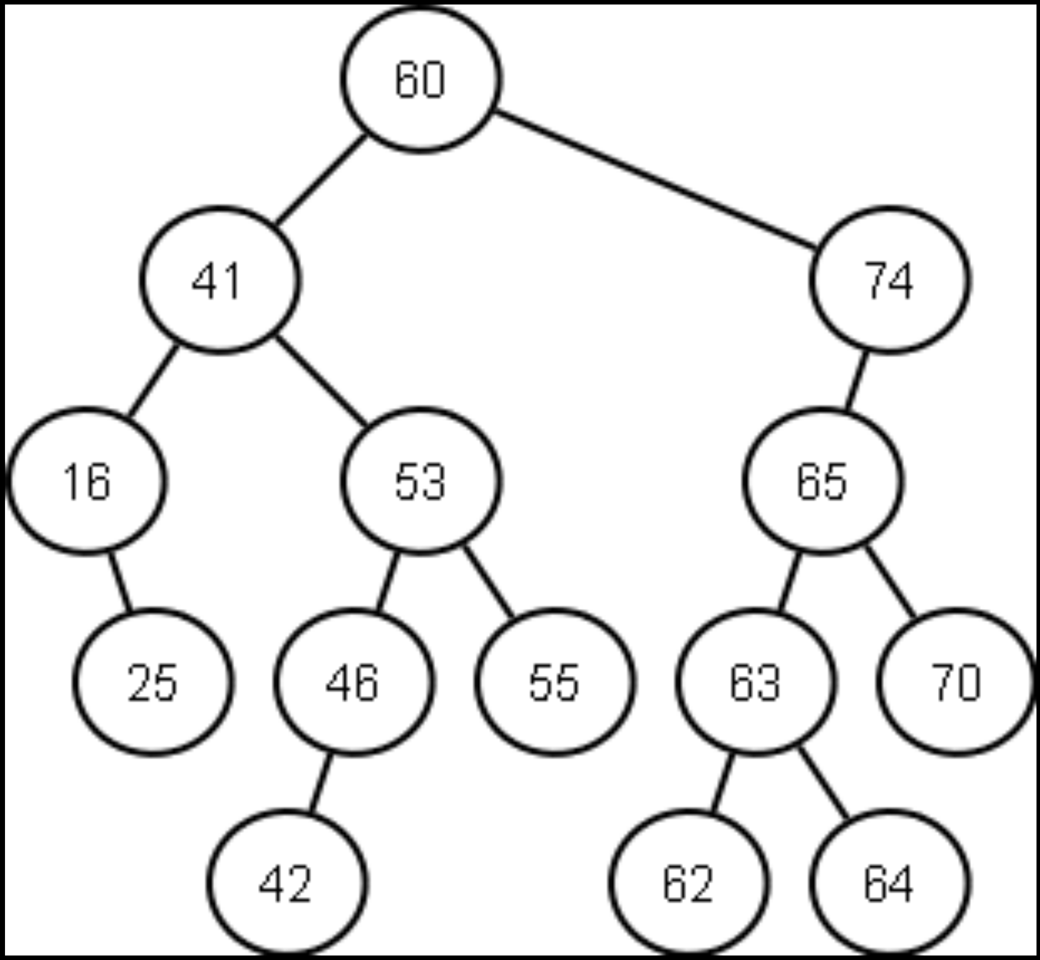
- See
  - week6/recursion.cpp
  - week6/trees.cpp

BINARY SEARCH TREES

Binary Search Trees

- A tree with the property that the value of all descendants of a node's left subtree are smaller, and the value of all descendants of a node's right subtree are larger

BST Example



BST Operations

- insert(item)
  - Add an item to the BST
- remove(item)
  - Remove an item from the BST
- contains(item)
  - Test whether or not the item is in the tree

BST Running Times

- All operations are O(n) in the worst case
  - Why?
- Assuming a balanced tree (CS132 material)
  - insert: O(log(n))
  - delete: O(log(n))
  - contains: O(log(n))

BST Insert

- If empty insert at the root
- If smaller than the current node
  - If no node on left: insert on the left
  - Otherwise: set the current node to the lhs (repeat)
- If larger than the current node
  - If no node on the right: insert on the right
  - Otherwise: set the current node to the rhs (repeat)

BST Contains

- Check the current node for a match
- If the value is smaller, check the left subtree
- If the value is larger, check the right subtree
- If the node is a leaf and the value does not match, return False

BST iterative traversal

```
ADT items;
items.add(root);  // Seed the ADT with the root
while(items.has_stuff() {
        Node *cur = items.random_remove();
        do_something(cur);
        items.add(cur.get_lhs()); // might fail
        items.add(cur.get_rhs()); // might fail
}
```

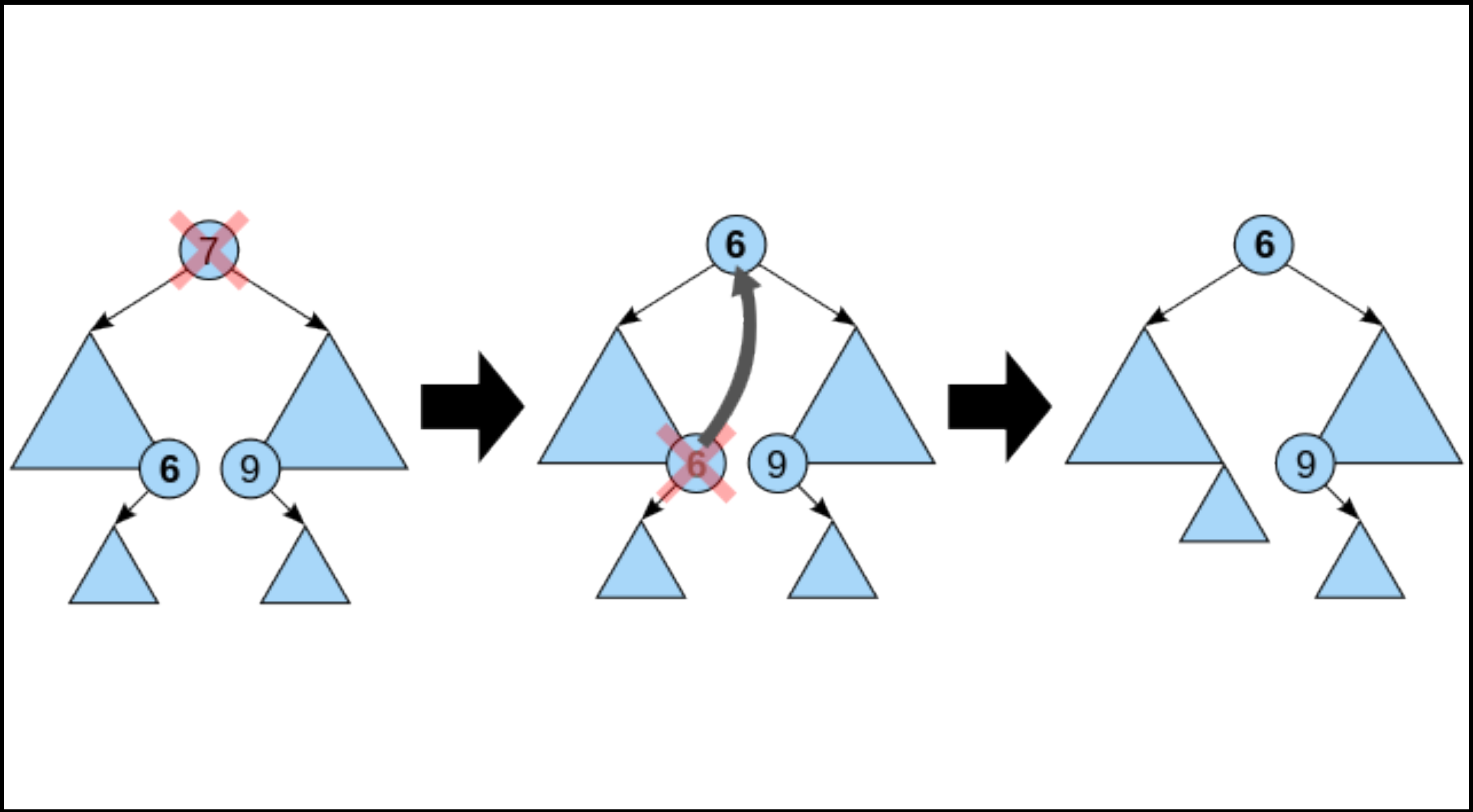
BST Remove

- If the node has no children simply remove it
- If the node has a single child, update its parent pointer to point to its child and remove the node

Removing a node with two children

- Replace the value of the node with the largest value in its left-subtree (right-most descendant on the left hand side)
- Then repeat the remove procedure to remove the node whose value was used in the replacement

Removing a node with two children



Project 2

- Add more functionality to the binary search tree
  - Implement ~Tree()
  - Implement remove(item)
  - Implement sorted_output() // Requires recursion
  - Implement distance(item_a, item_b);
  - Possibly implement one or two other functions (will be added later)