ZAMAN UNIVERSITY

Data Structures and Algorithms

Chapter 5

Tree

Outline

- Binary Trees
- Traversing Binary Tree
- Red-Black Trees
- Red-Black Tree Insertions
- 2-3-4 Trees

Outline

- Binary Trees
- Traversing Binary Tree
- Red-Black Trees
- Red-Black Tree Insertions
- 2-3-4 Trees

Traversing the Tree

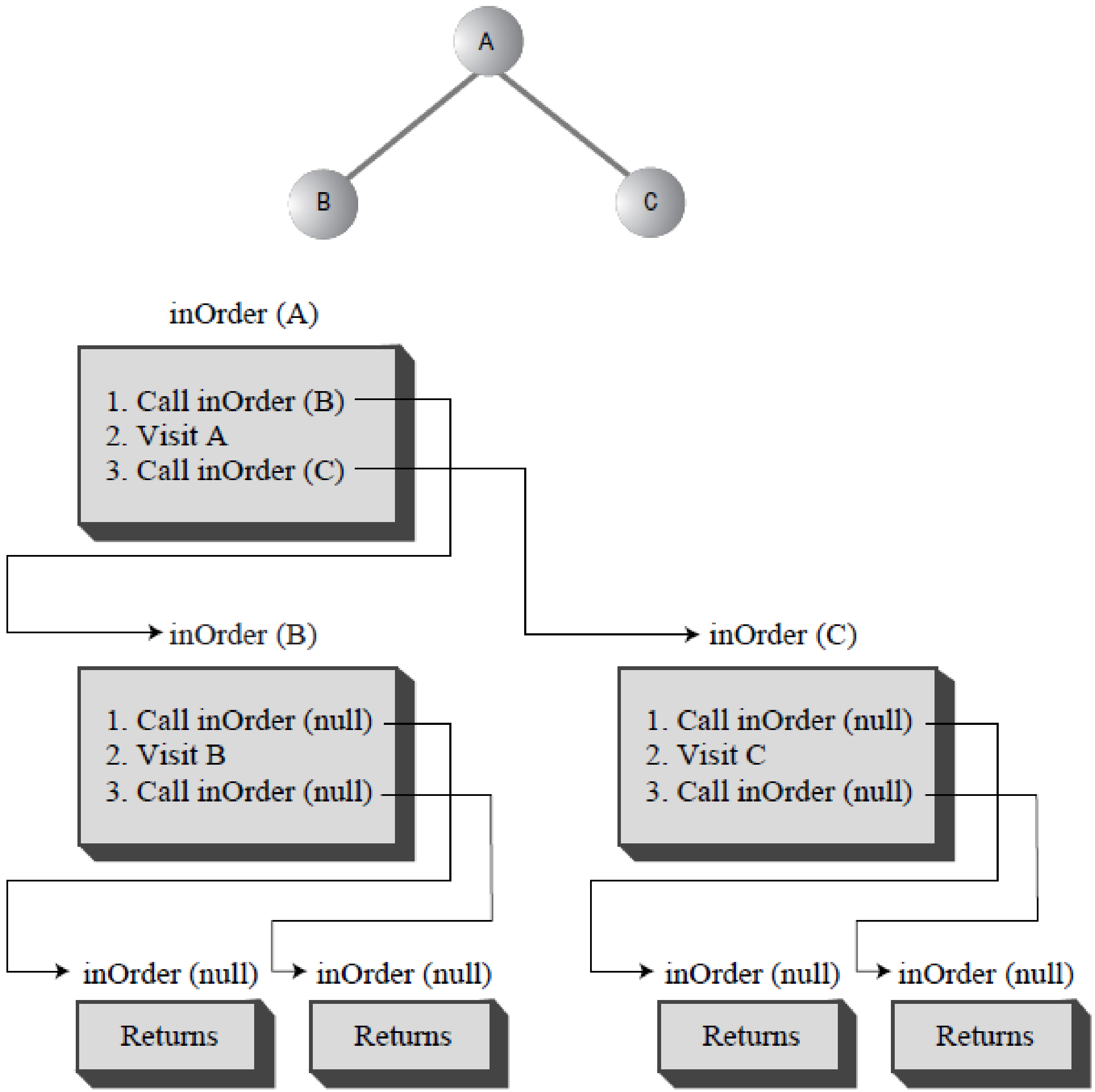
- Traversing a tree means visiting each node in a specified order
- There are three simple ways to traverse a tree: *preorder*, *inorder*, and *postorder*.
- The order most commonly used for binary search trees is inorder
- An *inorder traversal* of a binary search tree will cause all the nodes to be visited in ascending order, based on their key values.

Inorder Traversal

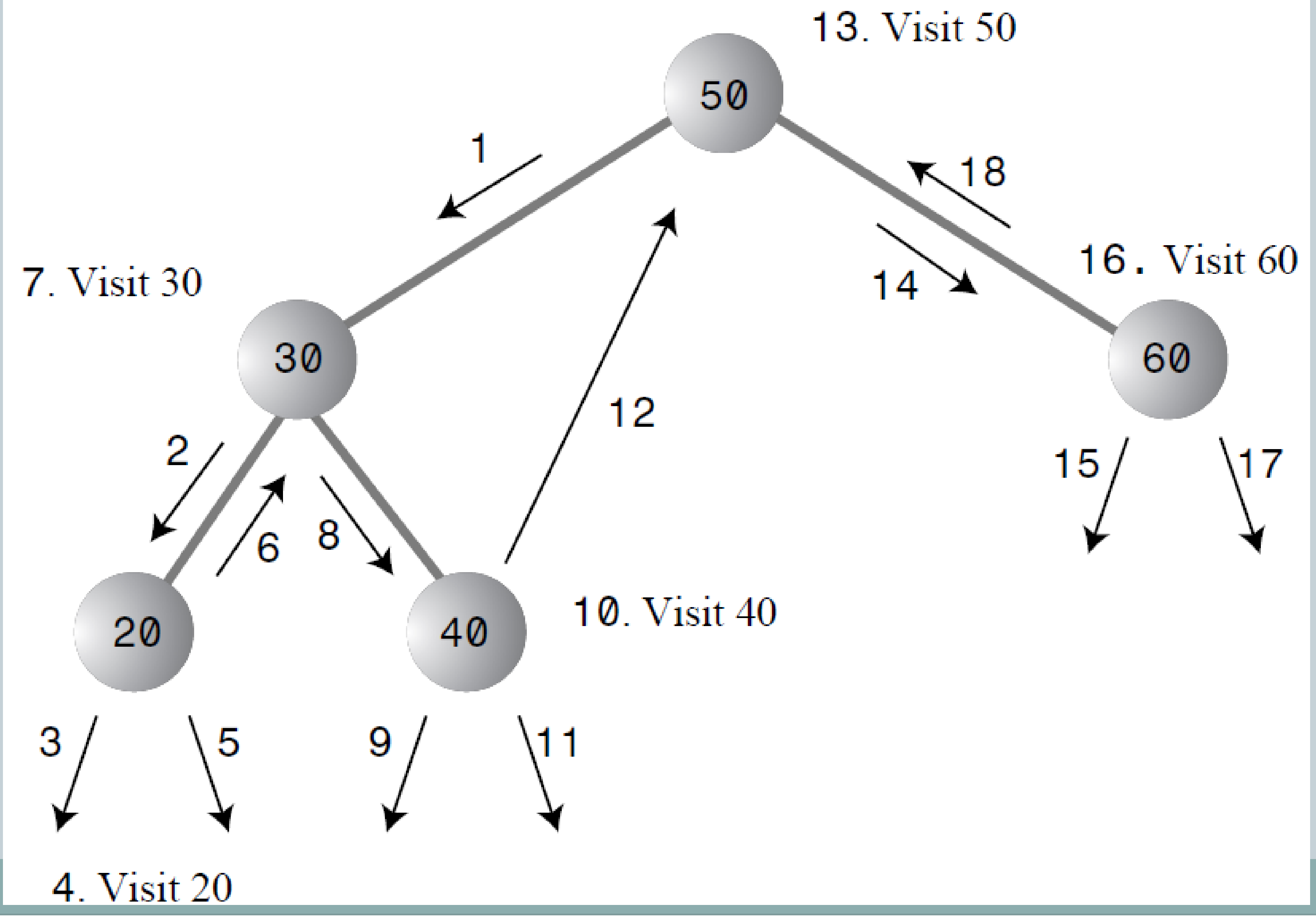
- The simplest way to carry out a traversal is the use of recursion
- A recursive function to traverse the tree is called with a node as an argument. Initially, this node is the root.
- The function must perform only three tasks:
    1. Call itself to traverse the node's left subtree.
    2. Visit the node.
    3. Call itself to traverse the node's right subtree.
- C++ Code for traversing:

```
void inOrder(Node* pLocalRoot) {
    if( pLocalRoot != NULL ){
        inOrder(pLocalRoot->pLeftChild); //left child
        cout << pLocalRoot->iData << " "; //display node
        inOrder(pLocalRoot->pRightChild); //right child
    }
}
```

Traversing a 3-Node Tree

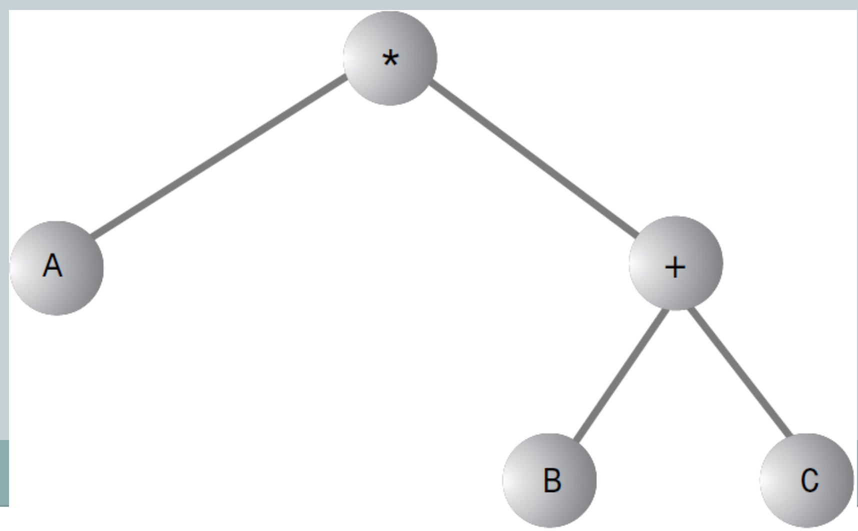


Example of Traversing a Tree Inorder



Preorder and Postorder Traversals

- A binary tree (not a binary search tree) can be used to represent an algebraic expression
- For example: algebraic expression A * (B + C) – this is called *infix* notation; it's the notation normally used in algebra
- Tree representing an algebraic expression:



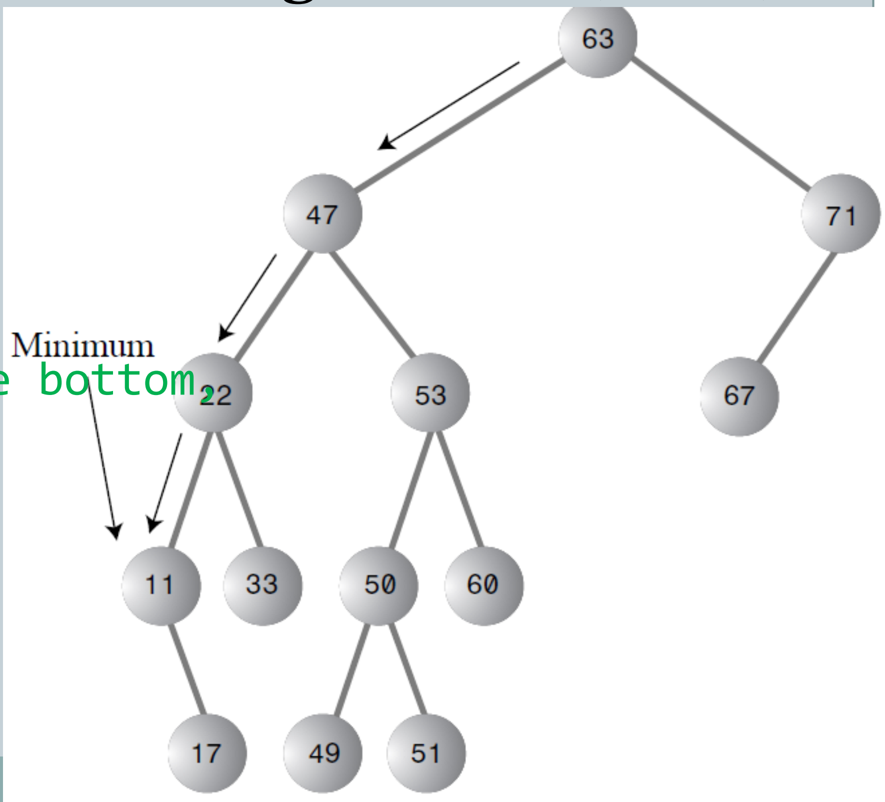
Preorder and Postorder Traversals (cont.)

Preorder	Postorder
Preorder() member function: 1. `Visit the node.` 2. `Call itself to traverse the node's left subtree.` 3. `Call itself to traverse the node's right subtree.`	*Postorder()* member function: 1. `Call itself to traverse the node's left subtree.` 2. `Call itself to traverse the node's right subtree.` 3. `Visit the node.`
Prefix: `* A + B C`	Postfix: A B C + *
`Prefix -` Starting on the left, each operator is applied to the next two things in the expression.	Postfix - Starting on the right, each operator is applied to the two things on its left.
Apply: * to A and +BC, in turn the expression +BC mean apply + to B and C give us (B+C). Inserting that into the original expression *A+BC (preorder) gives us A*(B+C) .	Apply: 1. + to B and C give us (B+C); 2. * to A and (B+C) give us infix A*(B+C)

Finding Maximum and Minimum Values

- Minimum value – go to the end of left child (node) of root;
- Maximum value – go to the end of right child (node) of root;

```
Node* minimum() {
  Node* pCurrent, pLast;
  pCurrent = pRoot; //start at root
  while(pCurrent != NULL){//until the bottom
   pLast = pCurrent; //remember node
   //go to left child
   pCurrent = pCurrent->pLeftChild;
  }
  return pLast;
}
```



Efficiency of Binary Trees

Number of Nodes	Number of Levels
1	1
3	2
7	3
15	4
31	5
…	…
1,023	10
…	…
32,767	15
…	…
1,048,575	20
…	…
33,554,432	25

L – Number of Levels;
N – Number of Nodes.

Thus,

$$N = 2^L - 1$$
$$L = \log_2(N+1)$$

Outline

- Binary Trees
- Traversing Binary Tree
- Red-Black Trees
- Red-Black Tree Insertions
- 2-3-4 Trees

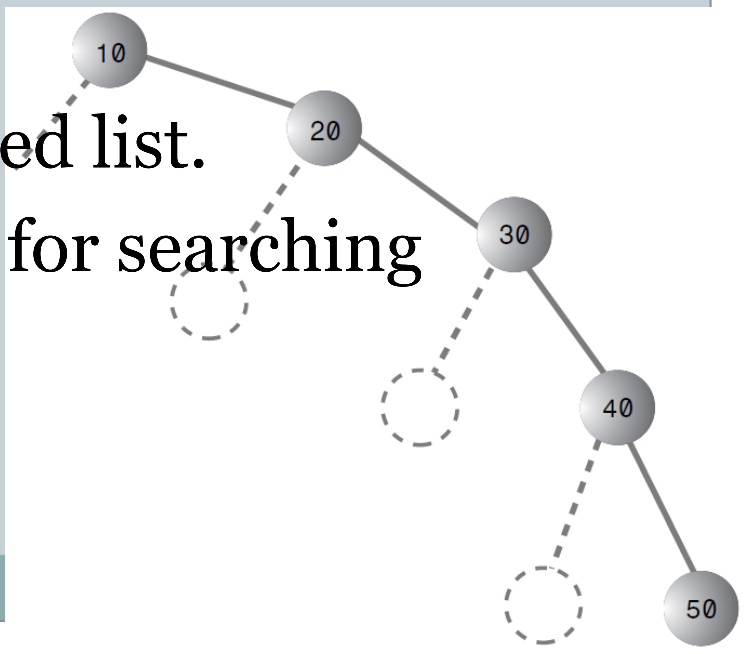
What is Red-Black Trees?

- Binary Search Trees, if data is inserted in a non-random sequence, the tree might become unbalanced, seriously degrading its performance.
- A Red-Black Tree can fix this by ensuring that the tree remains balanced at all times.

Balanced and Unbalanced Trees

- During insertion a series of nodes whose keys are in either ascending or descending order, thus binary tree becomes unbalanced
- With an unbalanced tree, the ability to quickly find (or insert or delete) a given element is lost
- When there are no branches,

  the tree becomes, in effect, a linked list.
- With linked list, we need O(n/2) for searching



Balanced Trees to the Rescue

- To guarantee the quick O(log N) search times a tree, we need to ensure that our tree is always balanced (or at least almost balanced)
- This means that each node in a tree must have roughly the same number of children on its left side as it has on its right

- In a red-black tree, balance is achieved during insertion and deletion.

Red-Black Tree Characteristics

- Red-Black Tree Characteristics:
  - The nodes are colored;
  - During insertion and deletion, rules are followed.

- Colored nodes, in Red-Black tree every node is either red or black.
- Red-Black rules:

1. Every node is either red or black.
2. The root is always black.
3. If a node is red, its children must be black.
4. Every path from the root to a leaf, or to a null child, must contain the same number of black nodes.

The Actions

- What actions can you take if one of the red-black rules is broken? There are two, and only two, possibilities:
  - You can change the colors of nodes.
  - You can perform rotations.

Try red-black tree on https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

To be continued…