Computer Science 102
Data Structures and Algorithms

Introduction to Trees

Professor: Evan Korth
New York University

Road Map

- Introduction to trees
  - Terminology
- Binary trees
- Tree traversal

Tree

- Tree defined recursively
- A tree is a collection of nodes. The collection can be empty; otherwise, a tree consists of a distinguished node r, called the root, and zero or more non-empty (sub) trees T_1, T_2, …, T_k each of whose roots are connected by a directed edge from r.
- A tree is a collection of N nodes, one of which is the root and N-1 edges.

Source:Mark Allen Weiss

Tree terminology

- The root of each subtree is said to be a **child** of r and r is said to be the **parent** of each subtree root.
- **Leaves**: nodes with no children (also known as external nodes)
- **Internal Nodes**: nodes with children
- **Siblings**: nodes with the same parent

Source:Mark Allen Weiss - edited by Evan Korth

Tree terminology (continued)

- A **path** from node n_1 to n_k is defined as a sequence of nodes n_1, n_2, …, n_k such that n_i is the parent of n_{i+1} for 1<= i <= k.
- The length of this path is the number of edges on the path namely k-1.
- The length of the path from a node to itself is 0.
- There is exactly one path from the root to each node in a tree.

Source:Mark Allen Weiss

Tree terminology (continued)

- Depth (of node): the length of the unique path from the root to a node.
- Depth (of tree): The depth of a tree is equal to the depth of its deepest leaf.
- Height (of node): the length of the longest path from a node to a leaf.
  - All leaves have a height of 0
  - The height of the root is equal to the depth of the tree

Source:Mark Allen Weiss

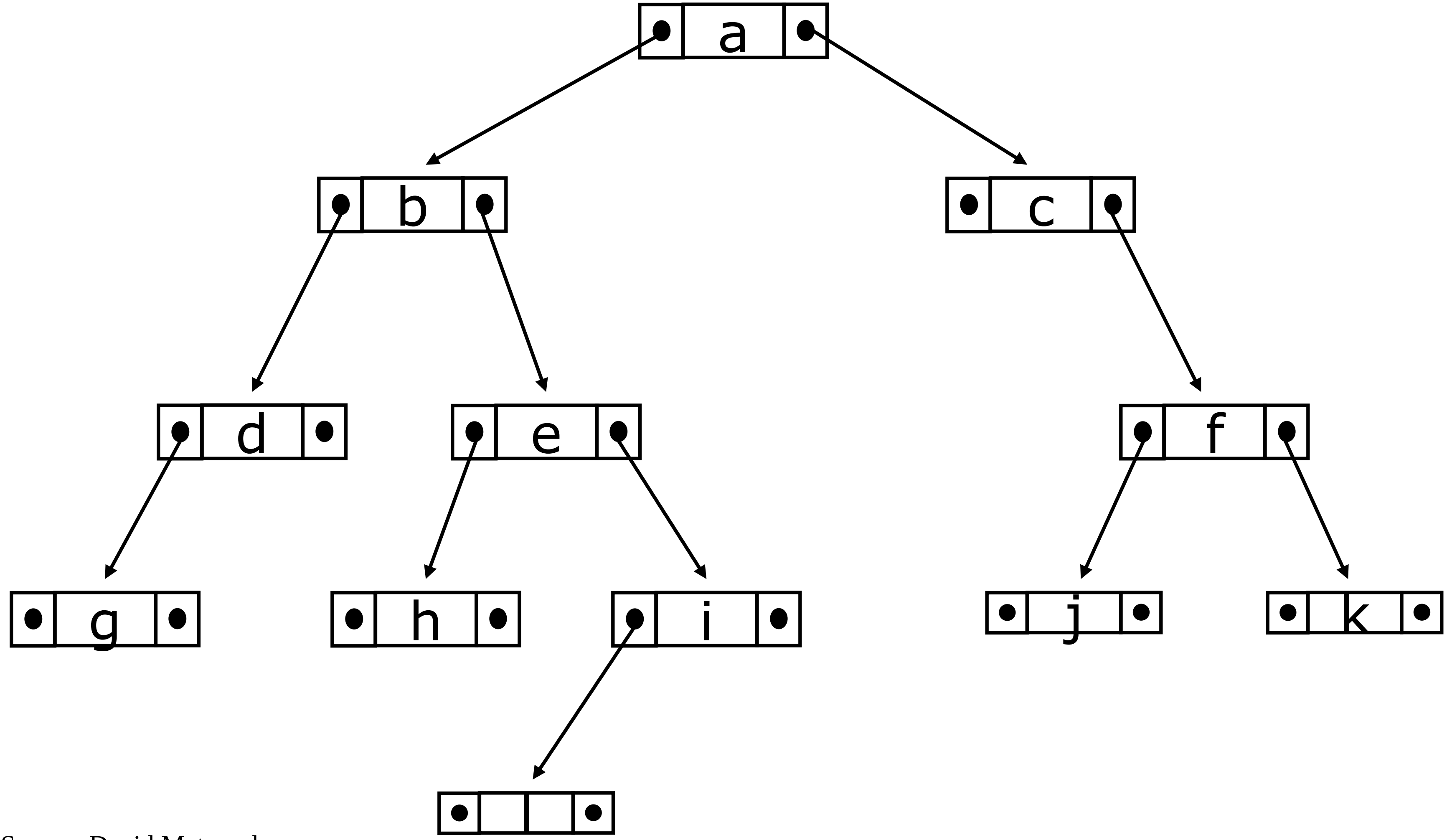
Tree Example

- On whiteboard

Binary trees

- A binary tree is a tree in which no node can have more than two children.
- Each node has an element, a reference to a left child and a reference to a right child.

Picture of a binary tree



Source: David Matuszek

Tree traversals

- A binary tree is defined recursively: it consists of a root, a left subtree, and a right subtree
- To traverse (or walk) the binary tree is to process each node in the binary tree exactly once
- Tree traversals are naturally recursive
- Since a binary tree has three “parts,” there are six possible ways to traverse the binary tree:
  - root, left, right
  - left, root, right
  - left, right, root
  - root, right, left
  - right, root, left
  - right, left, root

Source: David Matuszek

Preorder traversal

- In preorder, the root is visited *first*
- Here’s a preorder traversal to print out all the elements in the binary tree:

```
public void preorderPrint(BinaryTree bt)
{
     if (bt == null) return;
     System.out.println(bt.value);
     preorderPrint(bt.leftChild);
     preorderPrint(bt.rightChild);
}
```

Source: David Matuszek

Inorder traversal

- In inorder, the root is visited *in the middle*
- Here’s an inorder traversal to print out all the elements in the binary tree:

```
public void inorderPrint(BinaryTree bt)
{
    if (bt == null) return;
    inorderPrint(bt.leftChild);
    System.out.println(bt.value);
    inorderPrint(bt.rightChild);
}
```

Source: David Matuszek

Postorder traversal

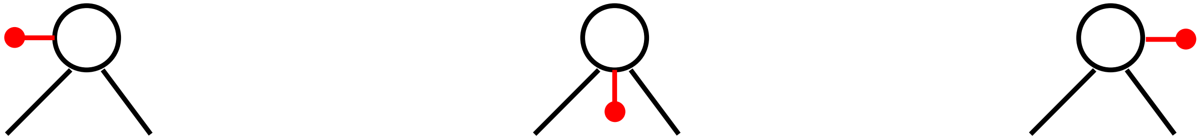
- In postorder, the root is visited *last*
- Here’s a postorder traversal to print out all the elements in the binary tree:

```
public void postorderPrint(BinaryTree
bt) {
    if (bt == null) return;
    postorderPrint(bt.leftChild);
    postorderPrint(bt.rightChild);
    System.out.println(bt.value);
}
```

Source: David Matuszek

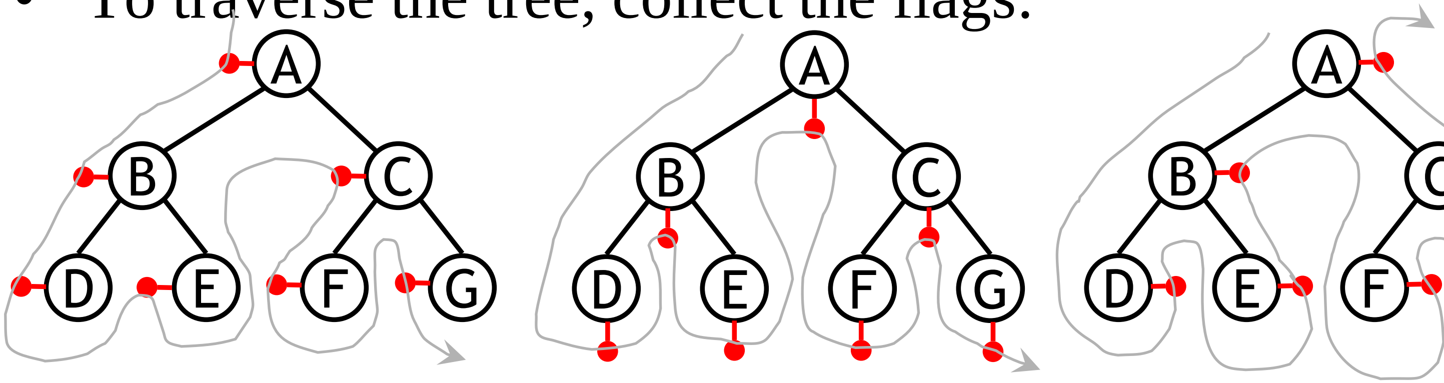
Tree traversals using “flags”

- The order in which the nodes are visited during a tree traversal can be easily determined by imagining there is a “flag” attached to each node, as follows:



preorder inorder postorder

- To traverse the tree, collect the flags:



A B D E C F G

D B E A F C G

D E B F G C A

Source: David Matuszek

Other traversals

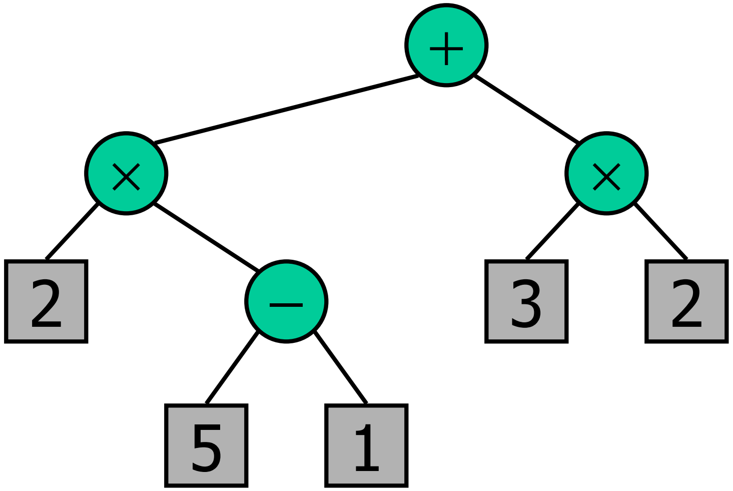
- The other traversals are the reverse of these three standard ones
  - That is, the right subtree is traversed before the left subtree is traversed
- Reverse preorder: root, right subtree, left subtree
- Reverse inorder: right subtree, root, left subtree
- Reverse postorder: right subtree, left subtree, root

Source: David Matuszek

Arithmetic Expression Tree

- Binary tree for an arithmetic expression
  - internal nodes: operators
  - leaves: operands
- Example: arithmetic expression tree for the expression

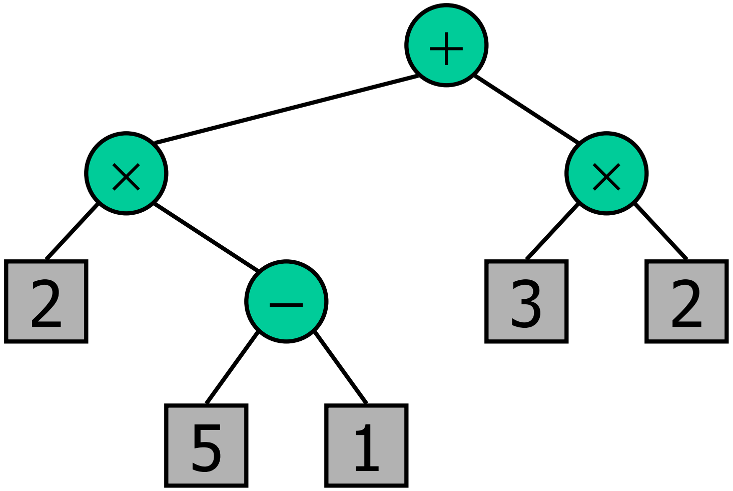
```
((2 × (5 − 1)) + (3 × 2))
```



Print Arithmetic Expressions

- inorder traversal:
  - print “(“ before traversing left subtree
  - print operand or operator when visiting node
  - print “)“ after traversing right subtree

```
void printTree(t)
//binary operands only
   if (t.left != null)
      print("(");
      printTree (t.left);
   print(t.element );
   if (t.right != null)
      printTree (t.right);
      print (")");
```



((2 × (5 – 1)) + (3 × 2))

Evaluate Arithmetic Expressions

- postorder traversal
  - Recursively evaluate subtrees
  - Apply the operator after subtrees are evaluated

```
int evaluate (t)
//binary operators only
    if (t.left == null)
      //external node
      return t.element;
    else  //internal node
      x = evaluate (t.left);
      y = evaluate (t.right);
      let o be the operator
        t.element
      z = apply o to x and y
      return z;
```