



Binary Trees

Wellesley College CS230
Lecture 17
Thursday, April 5
Handout #28

PS4 due 1:30pm Tuesday, April 10

17 - 1



Motivation: Inefficiency of Linear Structures

Up to this point our focus has been linear structures:
arrays, vectors, linked lists

Some operations on linear structures are cheap:

- **Constant time:**
 - Get/set elt at a given index in array/vector
 - Add elt to the end of a vector
 - Add elt to the front of a im/mutable list
 - Add elt to the back of a mutable list (if have pointer to last node)
- **Logarithmic time:**
 - Binary search on sorted array/vector

Other operations are expensive (linear in size):

Inserting/deleting elt at arbitrary position in an array/vector/list
Finding an element in a list (even a sorted one)

There is no linear structure for which insertion, deletion, and membership operations (e.g. in priority queues, sets, bags, and tables) are all cheaper than linear.

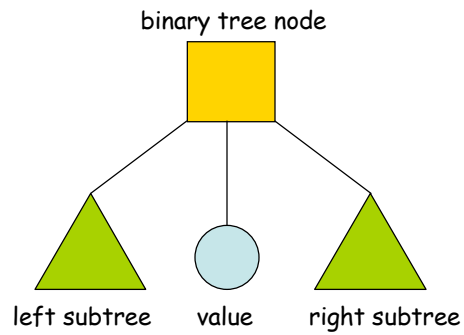
17 - 2



Idea: Branching of Binary Search + Linking of Linked Lists → Binary Trees

A binary tree is either

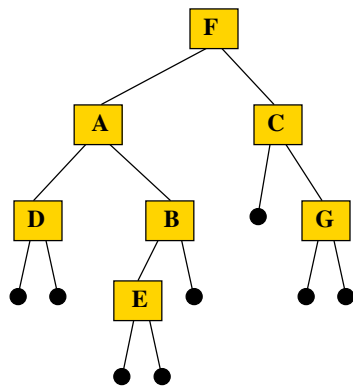
- a leaf: ●
- a node with (1) a left subtree (2) a value and (3) a right subtree:



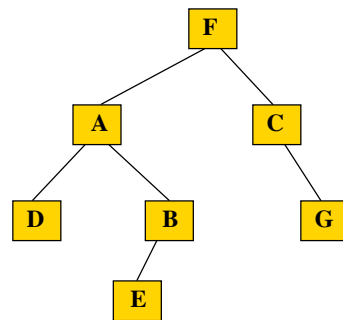
17 - 3



A Sample Binary Tree



Values are often shown in the nodes



Often, the leaves are not shown,
but they're still there!

17 - 4



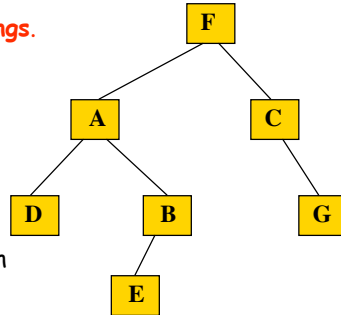
Tree Terminology

Nodes A and C are the **children** of node F; F is the **parent** of A and C; A and C are **siblings**. **Grandparents, grandchildren**, etc. defined similarly.

A node without a parent is a **root**. E.g. node F is the root of the tree, and node A is the root of F's left subtree (considered separately from the whole tree).

A **descendant** of a node n is a node on the path from n to a leaf. E.g. nodes D, B, and E are the descendants of A.

An **ancestor** of a node n is a node on the path from n to the root. E.g. Nodes B, A, and F are the ancestors of E.



17 - 5

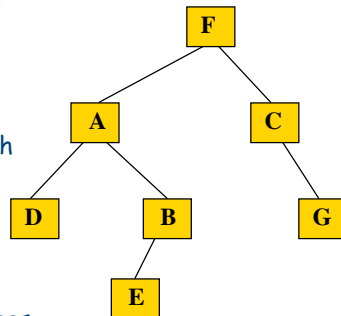


More Tree Terminology

The **height** of a node n is length of the longest path from n to a leaf below it. E.g., node G has height 1, node C has height 2, and node F has height 4.

The **depth** of a node n is the length of the path from n to the root. E.g., node F has depth 0, node C has depth 1, and node G has depth 2.

A binary tree is **height-balanced** iff at every node n , the heights of n 's left and right subtrees differ by no more than 1. The example tree is height-balanced, but would not be if G were removed. (There are other notions of tree balance in CS231.)



17 - 6



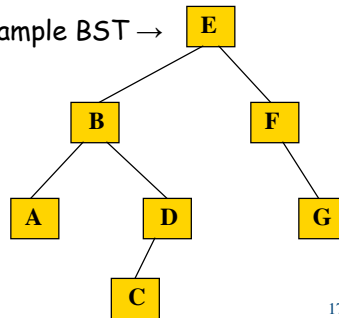
Binary Search Trees

To get full advantage of binary trees for data structures, need the values to be ordered.

A **binary search tree (BST)** is a binary tree in which the following properties hold at every node:

- (1) All elements in the left subtree are \leq the value;
- (2) All elements in the right subtree are \geq the value.

Here is a sample BST →



BSTs will be very important in the next lecture, when we use binary trees to implement data structures like sets, bags, and tables. For now, we'll stick with generic binary trees.

17 - 7

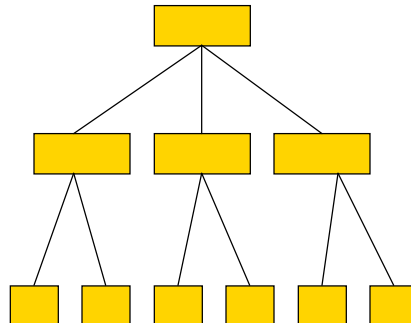


General Trees

In general, trees can have any number of nodes

Trees are used for many purposes:

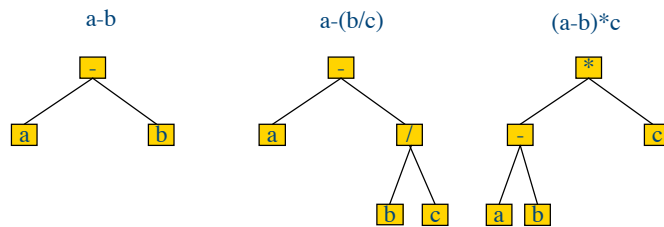
- Family trees
- Sport tournaments
- Animal kingdom
- Organizational chart
- A graph without cycles
- A home's plumbing and electrical systems
- File systems
- Document structure (e.g. HTML)
- Java class hierarchy
- Representing an expression/program
- ...



17 - 8



Arithmetic Expressions as Trees



Arithmetic expressions are often represented as binary trees.

Internal nodes are **operations** - Leaves are **numbers/variables**.

Operator **precedence** is enforced by the tree shape.

17 - 9



MBinTree Contract

```
public static <T> MBinTree<T> leaf();
public static <T> boolean isLeaf(MBinTree<T> tr);
public static <T> MBinTree<T> node(MBinTree<T> lt, T val, MBinTree<T> rt);
public static <T> T value (MBinTree<T> tr);
public static <T> MBinTree<T> left (MBinTree<T> tr);
public static <T> MBinTree<T> right (MBinTree<T> tr);
public static <T> void setValue (MBinTree<T> tr, T newValue);
public static <T> void setLeft (MBinTree<T> tr, MBinTree<T> newLeft);
public static <T> void setRight (MBinTree<T> tr, MBinTree<T> newRight);
```

17 - 10