Lecture 16: Binary Trees

CS 62

Spring 2018

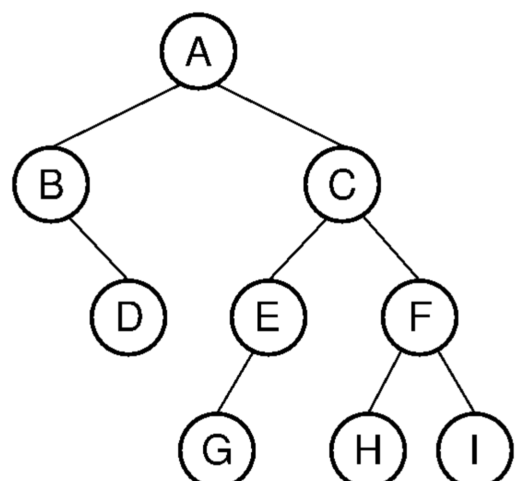
Alexandra Papoutsaki & William Devanny

Trees in CS

- Trees are abstract data types that store elements hierarchically
- Great when the linear, “before” and “after”, relationship is not enough
  - Certain operations are much faster too
- Hierarchical: Each element in a tree has a parent (an immediate ancestor) and zero or more children (immediate descendant)
- Trees in CS grow upside down!

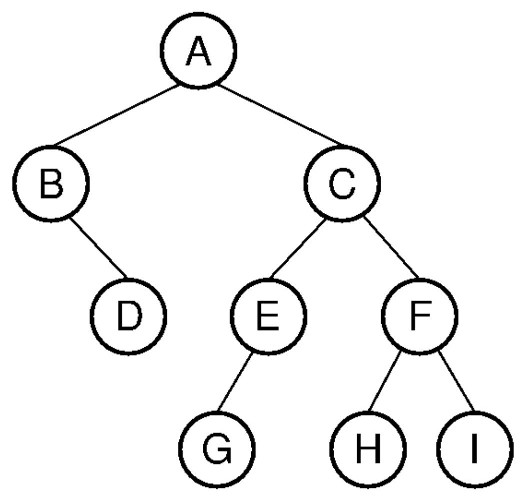
Definition of a tree

- A tree T is a set of nodes that store elements based on a *parent-child* relationship:
  - If T is non-empty, it has a node called the **root** of T, that has no parent
  - Each node v, other than the root, has a unique **parent** node u. Every node with parent u is a **child** of u.

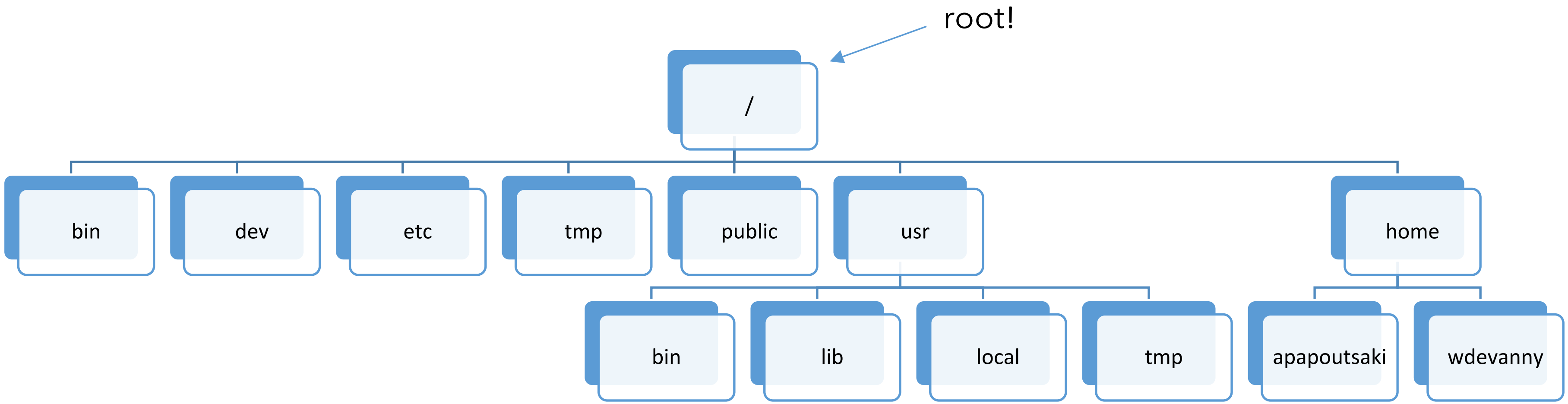


Recursive definition of a tree

- A tree T is either:
  - Empty or
  - Consists of a node r, called the root node of T, and a (possibly empty) disjoint set of trees, called its *subtrees*, whose roots are the children of r. These trees are disjoint from each other and the root.

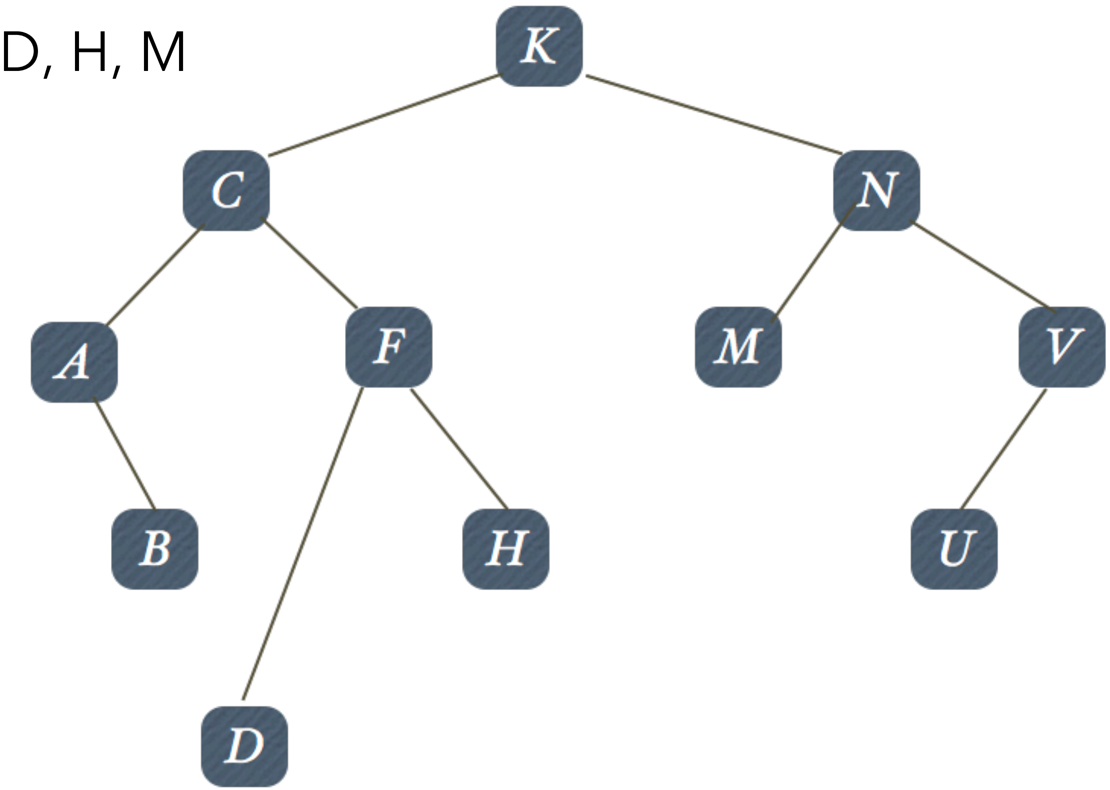


Example: Unix File System



Example: Binary Search Tree

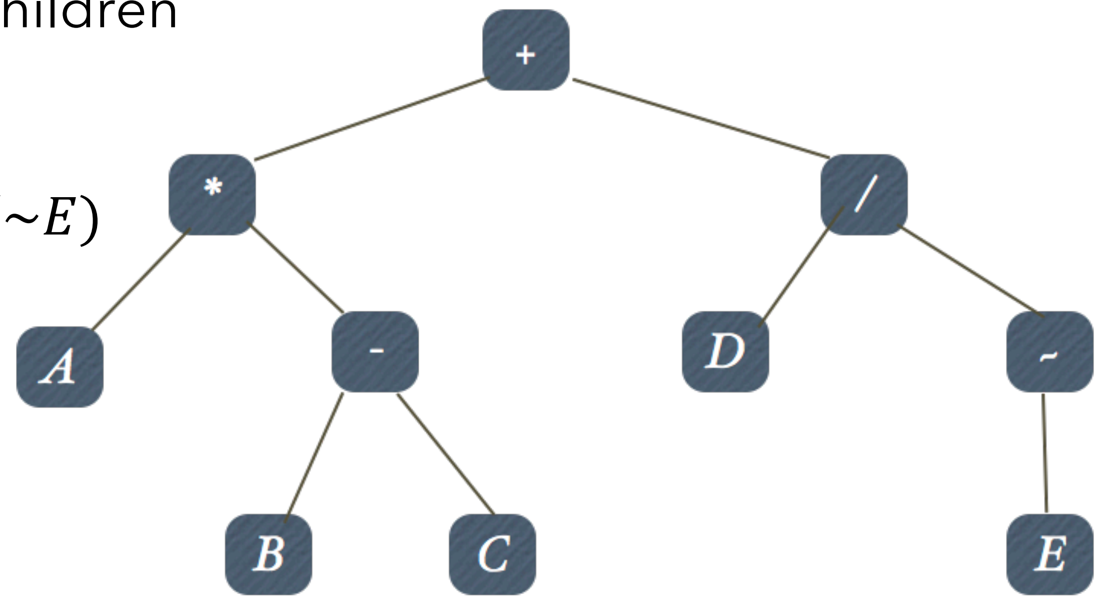
K, C, A, N, B, V, F, U, D, H, M



Example: Expression Tree

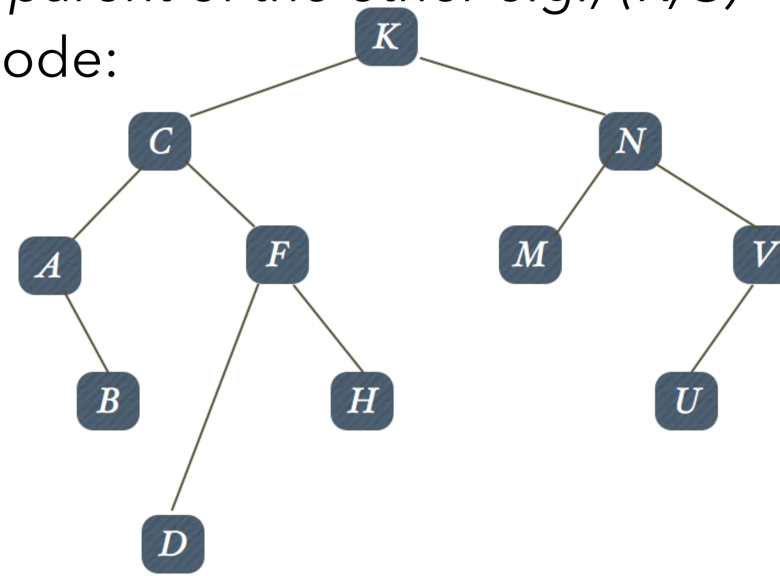
- If node is a leaf, then value is variable or constant
- If node is internal, then value calculated by applying operations on its children

- $[A * (B - C)] + (D/{\sim}E)$



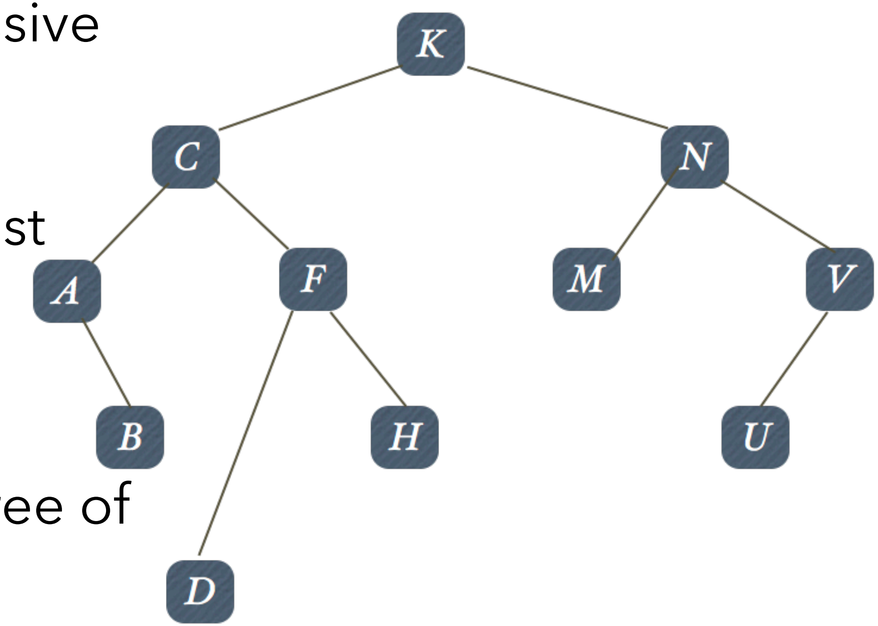
Family Tree Terminology

- ***Edge*** *is a pair of nodes s.t. one is the parent of the other e.g., (K,C)*
- ***Parent*** node is directly above ***child*** node:
  - K is parent to C, N.
- ***Sibling*** node has same parent:
  - A, F
- K is ***ancestor*** of B
- B is ***descendant*** of K
- Node plus all descendants gives ***subtree***
- Nodes without successors are called ***leaves*** *or* ***external***. The rest are called ***internal***
- A set of trees is called a forest



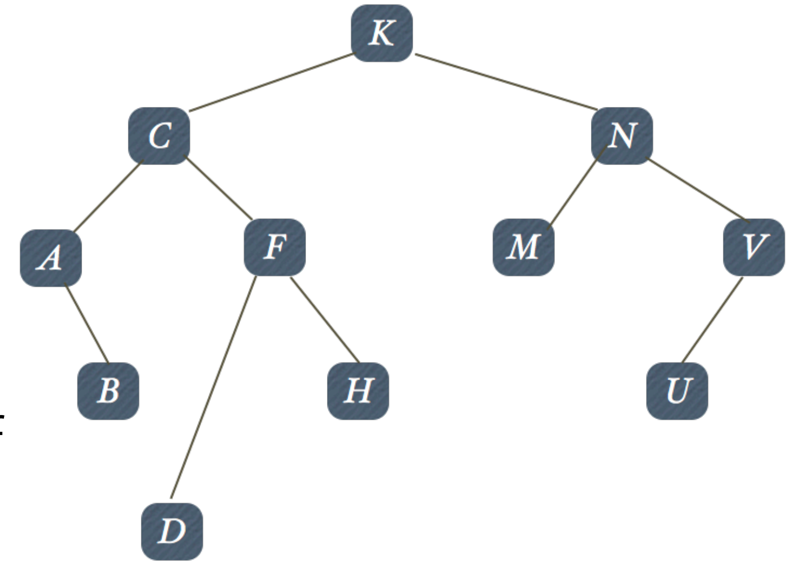
More Terminology

- **Simple path** is series of distinct nodes s.t. there is edge between successive nodes.
- ***Path length*** = # edges in path
- ***Height of node*** = length of longest path to a leaf
- ***Height of tree*** = height of root
- ***Degree of node*** is # of children
- ***Degree of tree (arity)*** = max degree of any its nodes
- Binary tree has arity ≤ 2.



Even More Terminology

- **Level/depth** of node defined recursively:
  - Root is at level 0
  - Level of any other node is one greater than level of parent
- Level of node is also length of path from root to the node or number of ancestors
- **Height** of node defined recursively:
  - If node is leaf then 0
  - Else height is max height of child + 1



But wait, there's more!

A tree is **ordered** if there is a meaningful linear order among the children of each node, e.g., when modeling books.

In contrast, when we're modeling an organization tree is unordered.

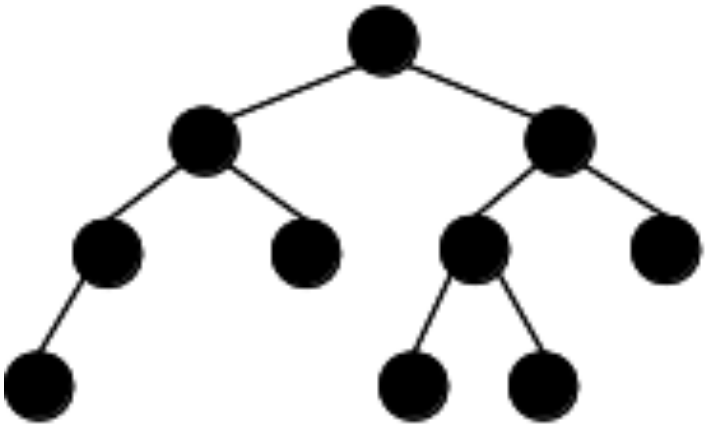
A binary tree is **full** (or proper) if every node has 0 or 2 children

A **complete** tree has minimal height and any holes in tree would appear in last level to right, i.e. all nodes are as far left as possible.

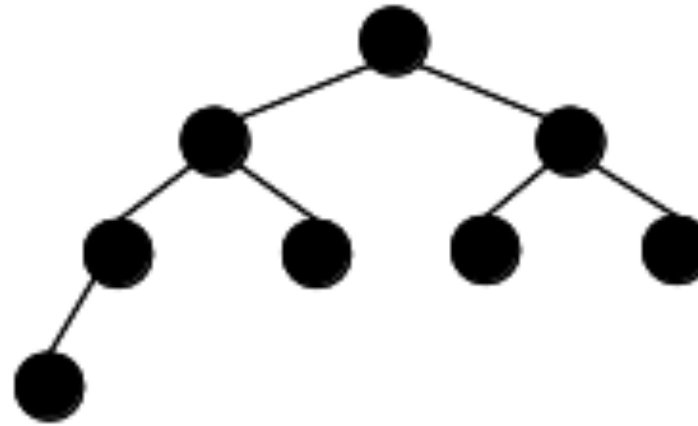
In a **perfect** binary tree all internal nodes have two children, ie. all leaves are at the same level.

A tree is height **balanced** iff at every node the difference in heights of subtrees is no greater than one and both left and right subtrees are balanced.

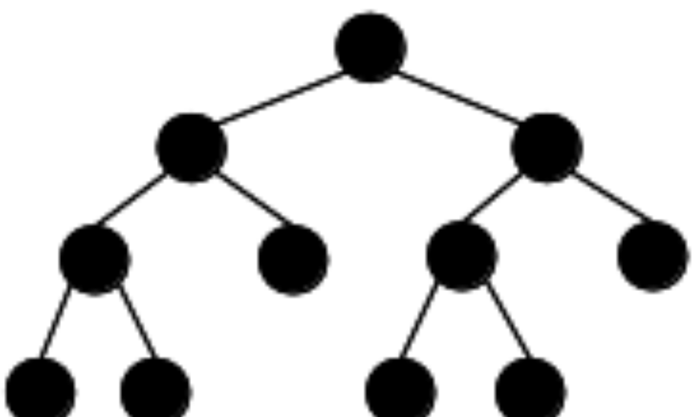
Neither complete nor full



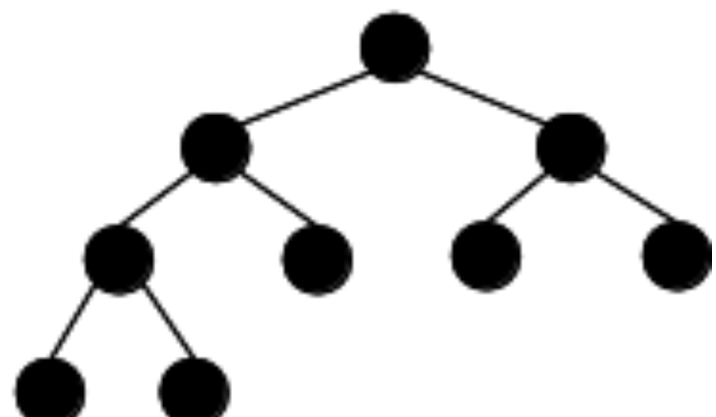
Complete but not full



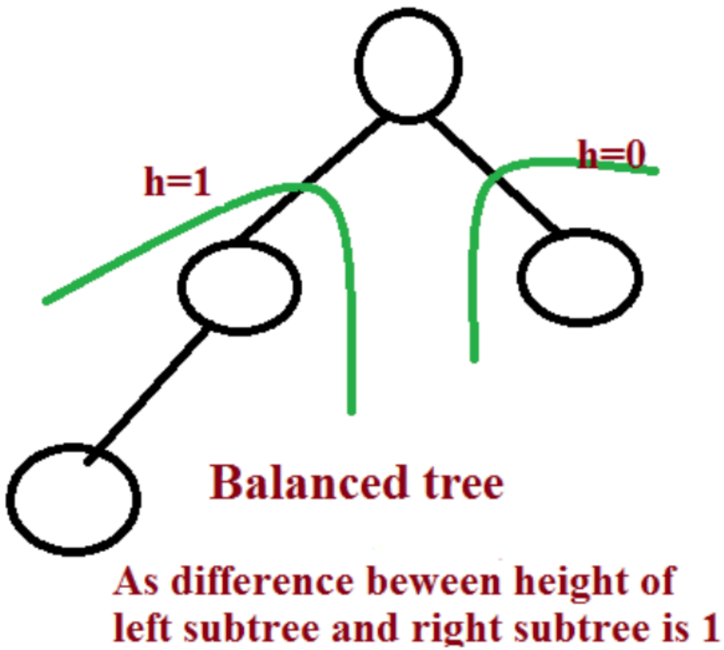
Full but not complete

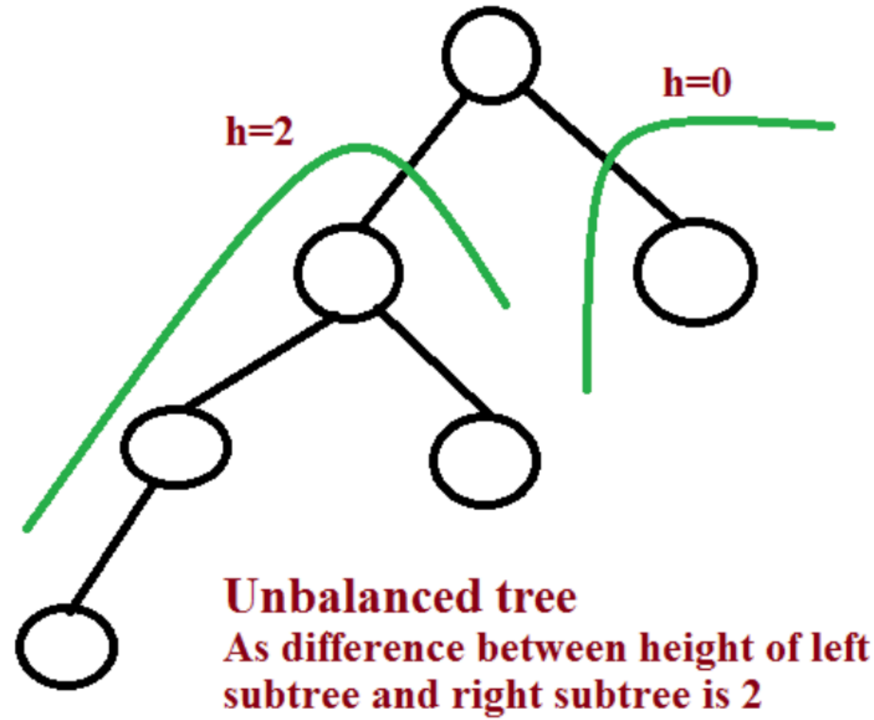


Complete and full



http://code.cloudkaksha.org/binary-tree/types-binary-tree

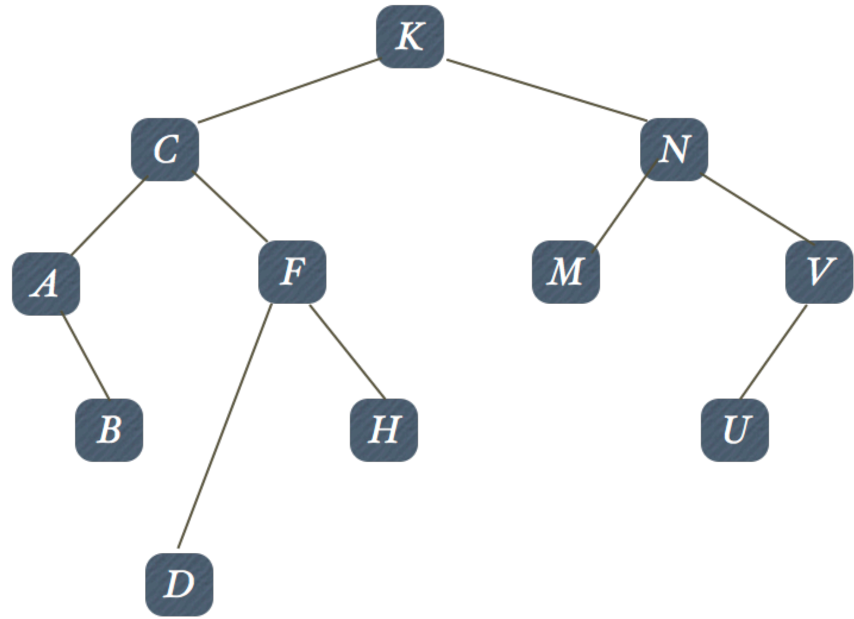




https://cs.stackexchange.com/questions/54171/is-a-balanced-binary-tree-a-complete-binary-tree

Counting

- Lemma: if T is a binary tree, then at level k, T has $\leq\ 2^k$ nodes.
- Theorem: If T has height h, then # of nodes n in T:
  $$h + 1 \leq n \leq\ 2^{h+1}\ - 1.$$
- Equivalently, if T has n nodes then
  $$\log(n+1) - 1 \leq h \leq n\ -\ 1$$



Binary Trees in Java

- No implementation in standard Java libraries
- `structure5` has `BinaryTree<E>` class, but no interface (*though we provide one!*).
- Like doubly-linked list:
  - value: E
  - parent, left, right: `BinaryTree<E>`

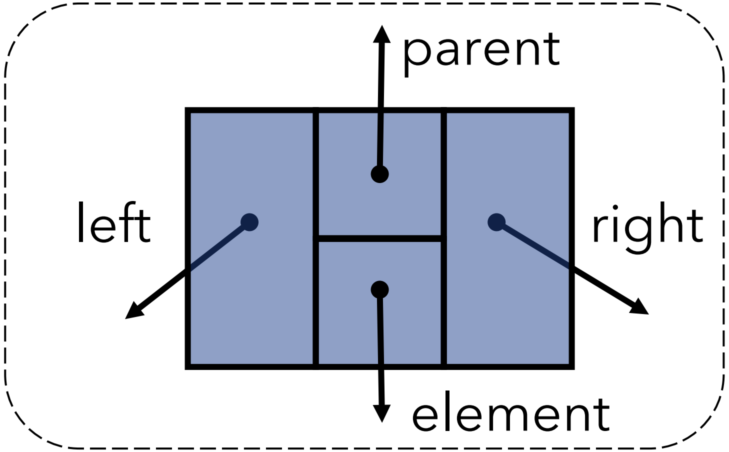
Binary Tree ADT

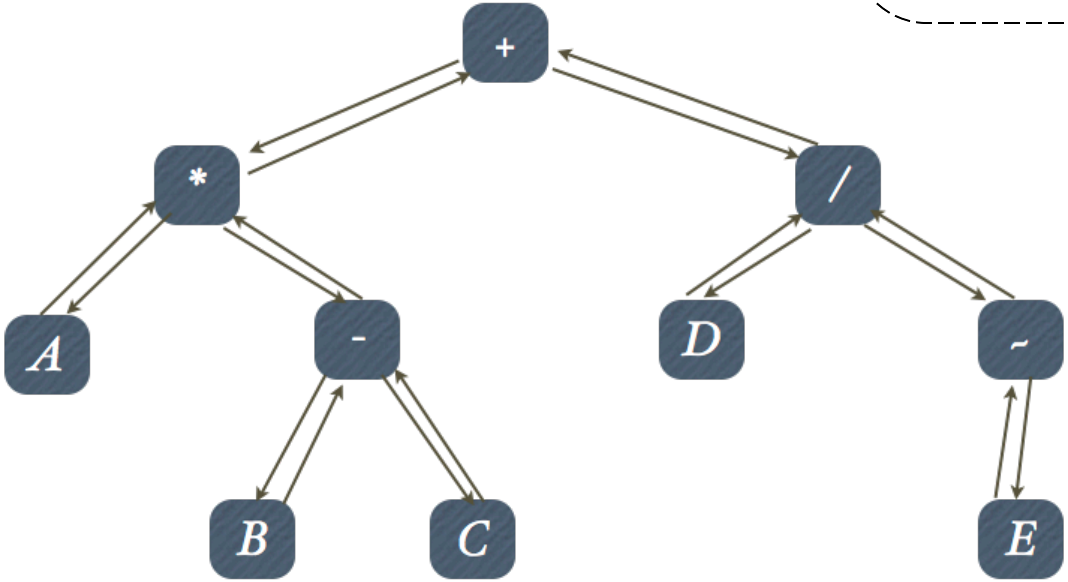
```
public interface BinaryTreeInterface<E> {
      public BinaryTreeInterface<E> left
      public BinaryTreeInterface<E> right
      public BinaryTreeInterface<E> parent();
      public E value;

      //getters, setters, iterators and other helper methods
}
```

This is just an example interface, `structure5.BinaryTree` *doesn't implement it!*

Linked Representation





Tree Traversals

- Traversals:
  - Pre-Order: root, left subtree, right subtree
  - In-Order: left subtree, root, right subtree
  - Post-Order: left subtree, right subtree, root
- Most algorithms have two parts:
  - Build tree
  - Traverse tree, performing operations on nodes

Tree traversals

- Pre-order: K C A B F D H N M V U
- In-order: A B C D F H K M N U V
- Post-order: B A D H F C M U V N K