CSCI 136:
Data Structures
and
Advanced Programming

Lecture 23

Trees, part 3

Instructor: Dan Barowy

Williams

Topics

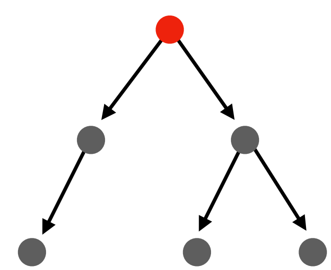
Tree terminology

Your to-dos

1. Read **before Fri**: review Bailey, Ch 14.
2. Lab 8 **(solo lab)**, **due Tuesday 11/15 by 10pm**. *Note: you will implement a tree data structure called a* ***trie*** *for lab 8; the structure is described in the lab handout. Please bring a short design document to your lab meeting.*

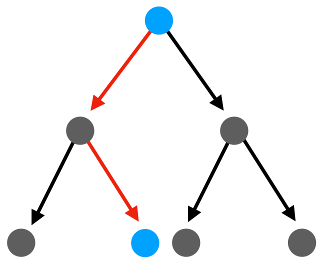
Properties of trees

Cycle-free: no path will ever revisit the same node.



Terminology

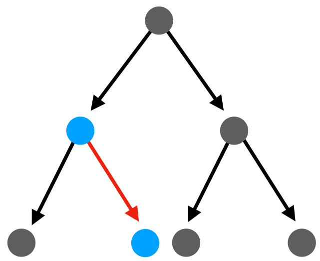
The **length** of a path is the **number of edges** in the path.



Length = **2**

Terminology

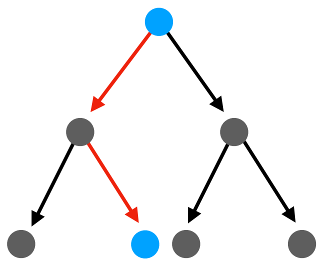
The **height** of **node n** is the length of the longest path between **n** and **any leaf**.



Height of **n** = **1**

Terminology

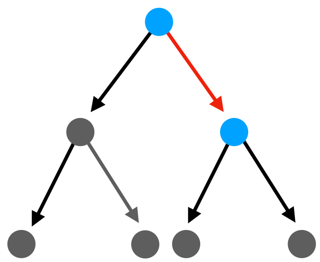
The **height** of **a tree** is the length of the longest path between **the root** and **any leaf**.



Height of **tree** = **2**

Terminology

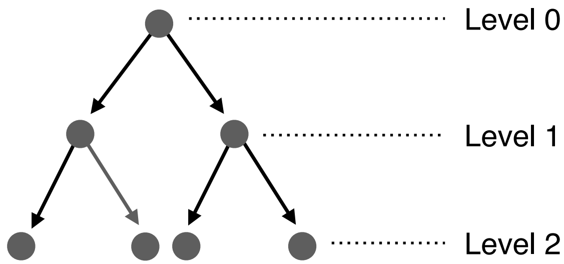
The **depth** of **node n** is the length of the path between **the root** and **n**.



Depth of **n** = **1**

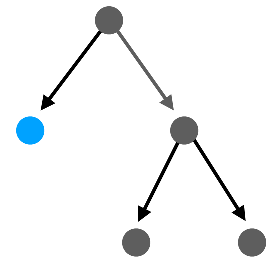
Terminology

The **level** of **any node** is its depth.



Terminology

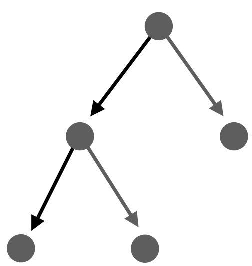
The depth of **n** + the height of **n** $\leq$ the height of the tree.



(depth of **n**: **1**) + (height of **n**: **0**) $\leq$ (height of **tree**: **2**)

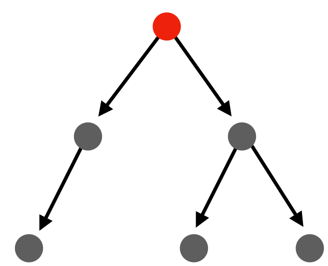
Terminology

A **complete** tree of **height h** is a full tree with zero or more rightmost leaves of **level h** removed.



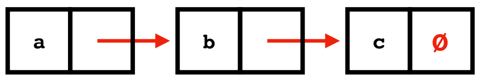
Properties of trees

Directed or **undirected**: trees can be either directed, meaning that traversals can only happen in one direction, or undirected, meaning that traversals can happen in any direction.



The tree shown here is directed.
We can represent an undirected tree using back edges.

Is a **list** a **tree**?

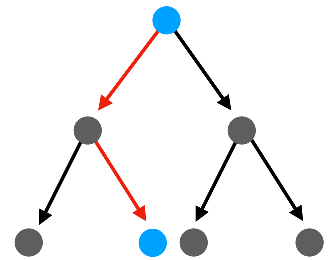


Yes, a list is a tree whose nodes have **degree** 1.

We call such trees **degenerate**.

Activity: Binary Tree Height

The **height** of **a tree** is the length of the longest path between **the root** and **any leaf**.



Height of **tree** = **2**

Binary Tree Height

Let's think about some corner cases.

What is the height of a tree with just one node?

The **height** of **a tree** is the length of the longest path between **the root** and **any leaf**.

Height of **tree** = **0**

Binary Tree Height

Let's think about some corner cases.

What about the empty tree?

The **height** of **a tree** is the length of the longest path between **the root** and **any leaf**.

Height of **tree** = **-1**

Binary Tree Height

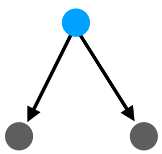
Here's a more formal definition.

The **height** of a tree is defined as:

- **-1** if the tree is empty, or
- **height(left)** or **height(right)**, whichever is bigger, **+ 1**

empty tree: -1

just a root: 0



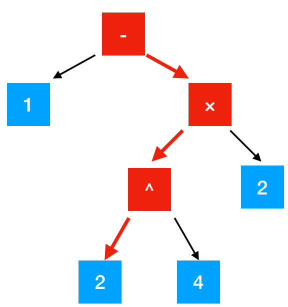
any other tree: longest path

Activity: Binary Tree Height

How might we implement `getHeight()`?

Height

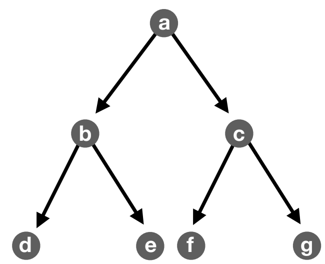
$$1 - 2^{4} \times 2$$



Binary tree traversals

Binary tree traversals

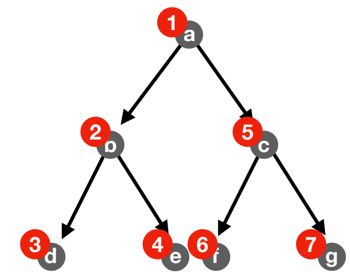
Suppose you are asked to write an `Iterator<T>` for a binary tree. What order do you choose?



Remember that tree nodes store data (`T`). A **traversal** corresponds with the order that data is returned.

Binary tree traversals

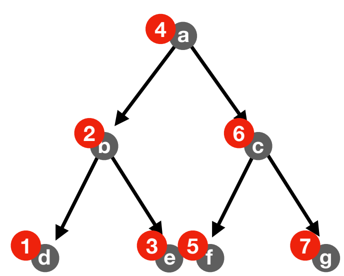
Pre-order traversal: Return data from each node **before its children**, and then return child data from **left to right**.



Returns the sequence: **a, b, d, e, c, f, g**

Binary tree traversals

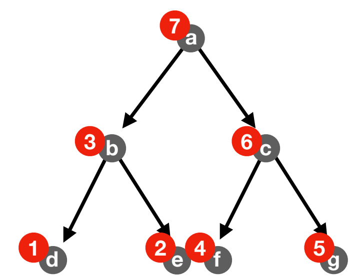
In-order traversal: Return data from each node **after its left child** and **before its right child**.



Returns the sequence: **d, b, e, a, f, c, g**

Binary tree traversals

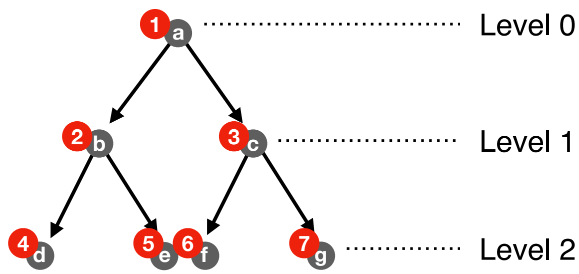
Post-order traversal: Return data from each node **after its children**; return child data from **left to right**.



Returns the sequence: **d, e, b, f, g, c, a**

Binary tree traversals

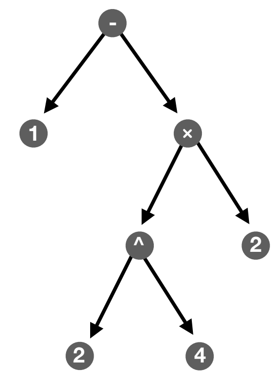
Level-order traversal (aka **breadth-first order**): Return data from each node in **level *i*** before data in **level *i+1***.



Returns the sequence: **a, b, c, d, e, f, g**

Activity: What traversal should I use?

Suppose I encode the arithmetic expression $1 - 2^4 \times 2$ using the following tree.



Ordered Trees

Binary search tree

A **binary search tree** is a binary tree that maintains the **binary search property** as elements are added or removed. In other words, the **key** in each node:

- must be > any **key** stored in the left subtree, and
- must be ≤ any **key** stored in the right subtree.

As with other ordered structures, order is maintained **on insertion.**

Binary search tree (alternative)

A **binary search tree** is a binary tree that maintains the **binary search property** as elements are added or removed. In other words, the **key** in each node:

- must be ≥ any **key** stored in the left subtree, and
- must be < any **key** stored in the right subtree.

As with other ordered structures, order is maintained **on insertion.**

Key, Value nodes

Note that I said **key** instead of **element**.

Storing a **key** and a **value** in each node allows the greatest flexibility when arranging a tree. I.e., the key type K need not be the value type V.

Restriction: keys must be **comparable** in some way (e.g., `Comparable<K>` or `Comparator<K>`).

Example

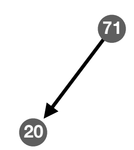
Insert the following elements: `71,20,27,17,91,14,87`

Assume K and V are the same.

Example

Insert the following elements: `71,20,27,17,91,14,87`

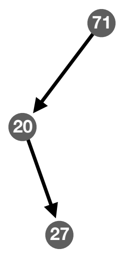
Assume K and V are the same.



Example

Insert the following elements: `71,20,27,17,91,14,87`

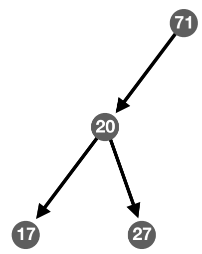
Assume K and V are the same.



Example

Insert the following elements: `71,20,27,17,91,14,87`

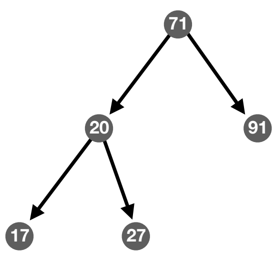
Assume K and V are the same.



Example

Insert the following elements: `71,20,27,17,91,14,87`

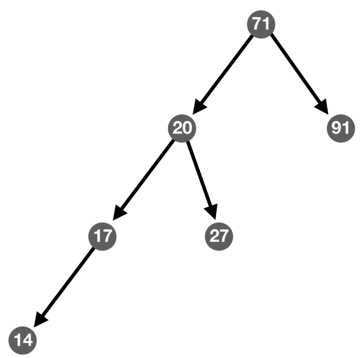
Assume K and V are the same.



Example

Insert the following elements: `71,20,27,17,91,14,87`

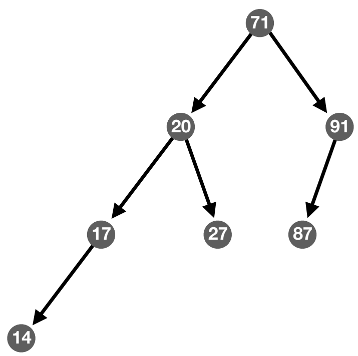
Assume K and V are the same.



Example

Insert the following elements: 71,20,27,17,91,14,87

Assume K and V are the same.



Activity

Insert the following elements:

Assume K and V are the same.

Binary Search Tree

Let's start implementing this together.

Recap & Next Class

Today:

Tree terminology

Tree traversals

Next class:

Binary search trees