CSCI 136:
Data Structures
and
Advanced Programming

Lecture 23

Trees, part 3

Instructor: Dan Barowy

Williams

Announcements

One-on-one: who's missing?

IntelliJ IDEA tutorial on website

Outline

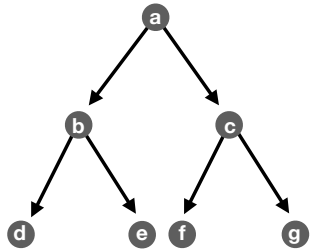
Traversals

Binary search tree

Binary tree traversals

Binary tree traversals

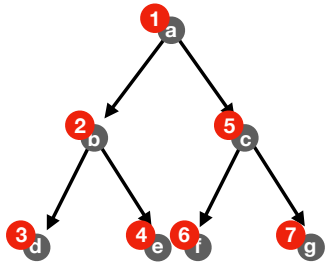
Suppose you are asked to write an `Iterator<T>` for a binary tree. What order do you choose?



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

Binary tree traversals

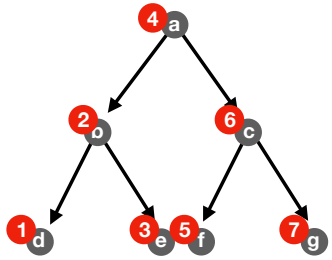
Pre-order traversal: Return data from each node **before its children**, and then return child data from **left to right**.



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

Binary tree traversals

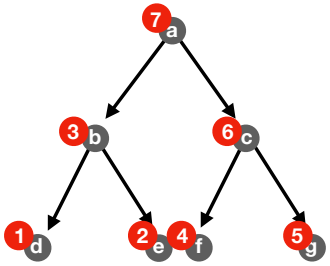
In-order traversal: Return data from each node **after its left child** and **before its right child**.



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

Binary tree traversals

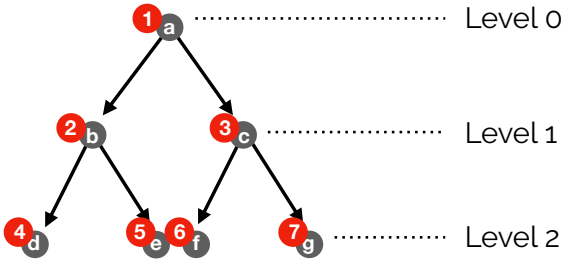
Post-order traversal: Return data from each node **after its children**; return child data from **left to right**.



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

Binary tree traversals

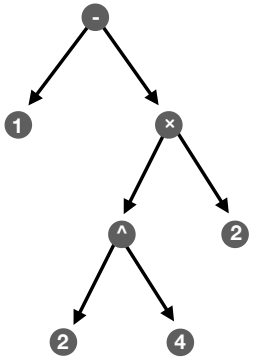
Level-order traversal (aka **breadth-first order**): Return data from each node in **level *i*** before data in **level *i+1***.



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.**

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

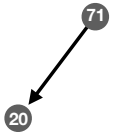
Insert the following elements: `71,20,27,17,91,14,87`

Assume K and V are the same.

Example

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

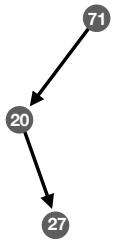
Assume K and V are the same.



Example

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

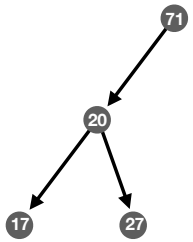
Assume K and V are the same.



Example

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

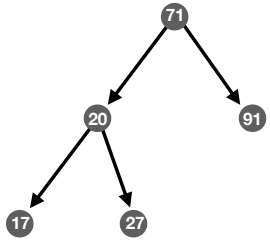
Assume K and V are the same.



Example

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

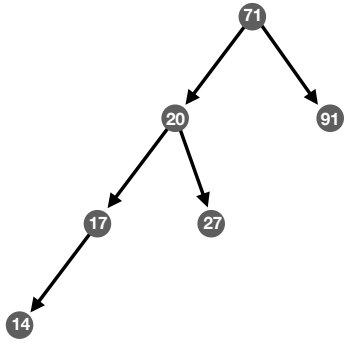
Assume K and V are the same.



Example

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

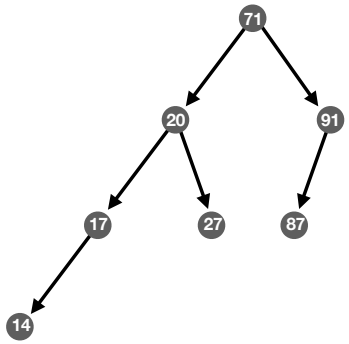
Assume K and V are the same.



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 implement `add`, and `toString` (printing in-order).

Binary Search Tree

How might we implement an in-order `iterator`?

In-order Iterator cases

Invariant: the current node is always the **leftmost unvisited node**.

1. If there is a right side, **go right**, then **go as far left** as possible.
2. Otherwise, find the **first parent of a left node**.
3. If there are no more parents, there are **no more elements**.

Recap & Next Class

Today we learned:

Binary tree traversals

Binary search trees

Next class:

Asymptotic performance for trees

Priority queues