TREES, PART 2

Lecture 12

CS2110 – Summer 2019

Announcements

- The regrading period has opened for Assignment 1.
- Deadline: Saturday, July 13th at 5PM

JavaHyperText topics

- Tree traversals (preorder, inorder, postorder)
  - http://www.cs.cornell.edu/courses/JavaAndDS/files/tree6BTreeTraversal.pdf
- Stack machines
  - http://www.cs.cornell.edu/courses/JavaAndDS/explainJava/03methodCalls.html

Trees, re-implemented

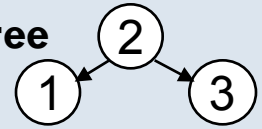
- Last time: lots of **`null`** comparisons to handle empty trees
- A more OO design:
  - Interface to represent operations on trees
  - Classes to represent behavior of empty vs. non-empty trees

Demo

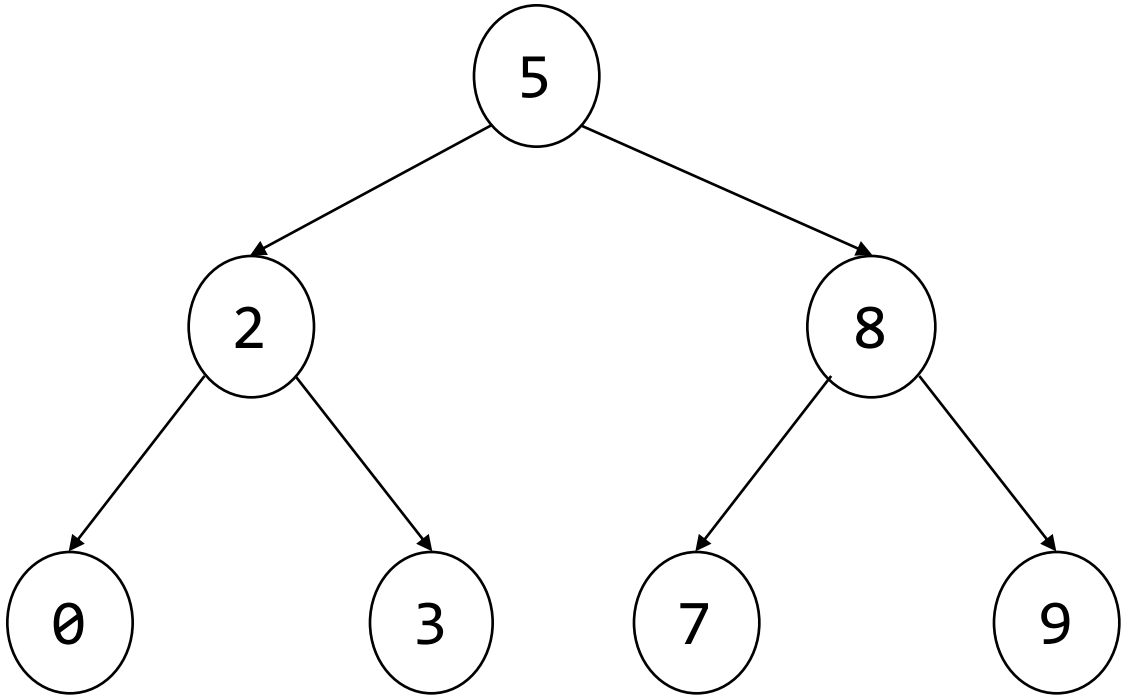
Iterate through data structure

Iterate: process elements of data structure

- Sum all elements
- Print each element
- …

Data Structure	Order to iterate
Array 	Forwards: 2, 1, 3, 0 Backwards: 0, 3, 1, 2
Linked List 	Forwards: 2, 1, 3, 0
Binary Tree 	**???**

Iterate through data structure



Discuss: What would a reasonable order be?

Tree Traversals

Tree traversals

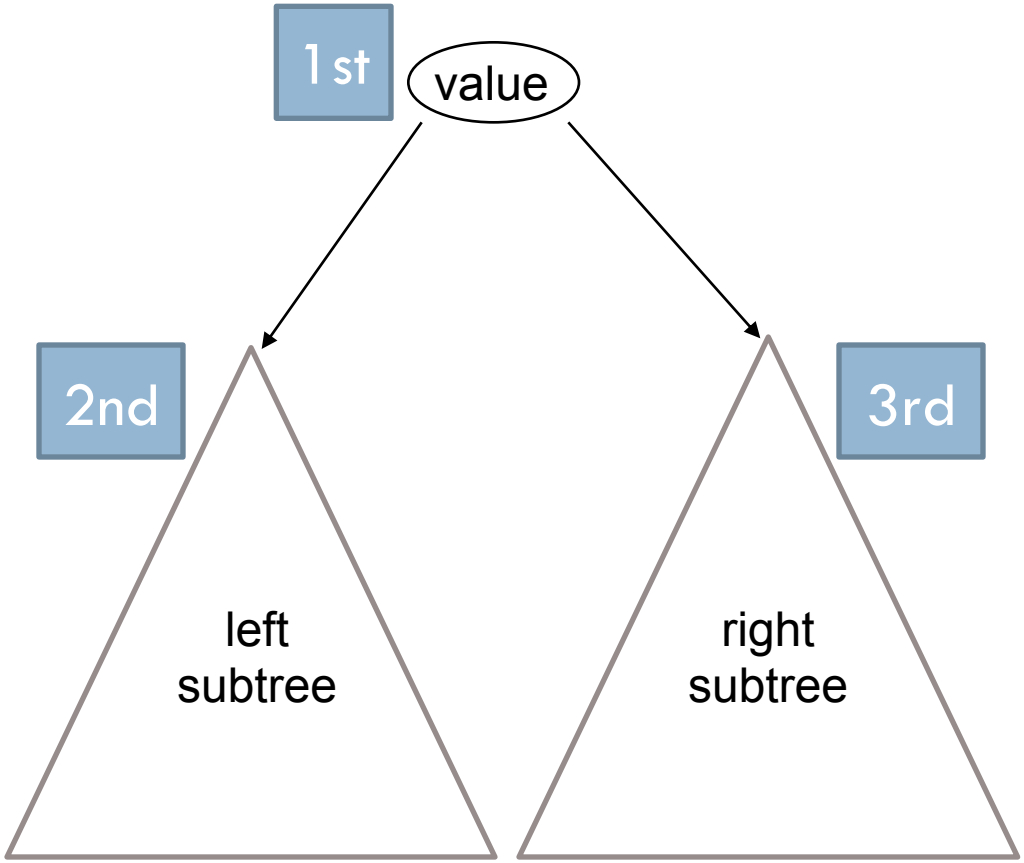
- Iterating through tree is aka tree traversal

- Well-known recursive tree traversal algorithms:
  - Preorder
  - Inorder
  - Postorder

- Another, non-recursive: level order (later in semester)

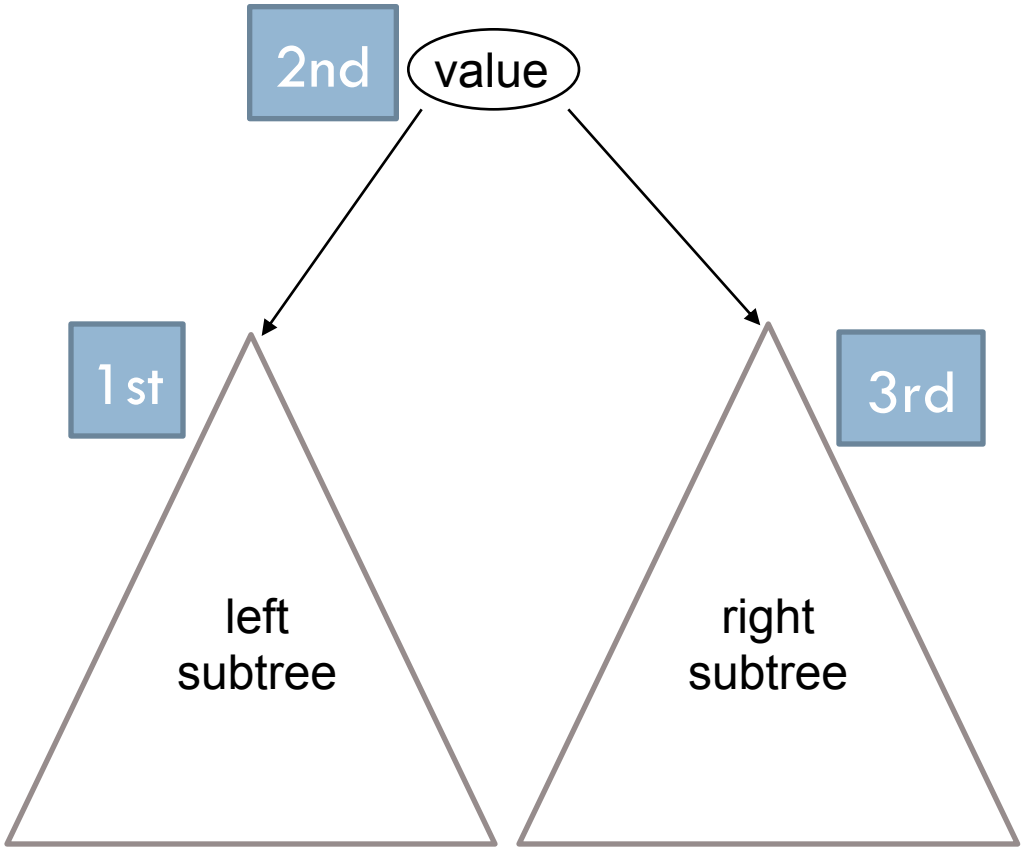
Preorder

"Pre:" process root before subtrees



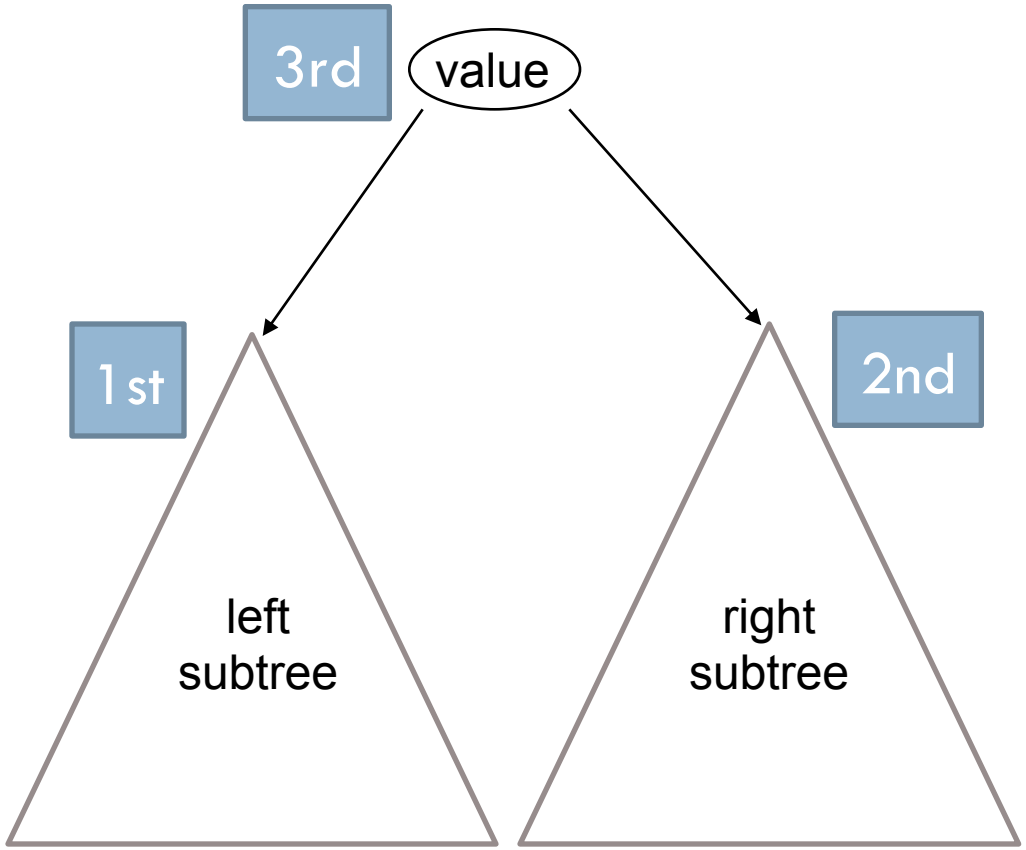
Inorder

"In:" process root in-between subtrees



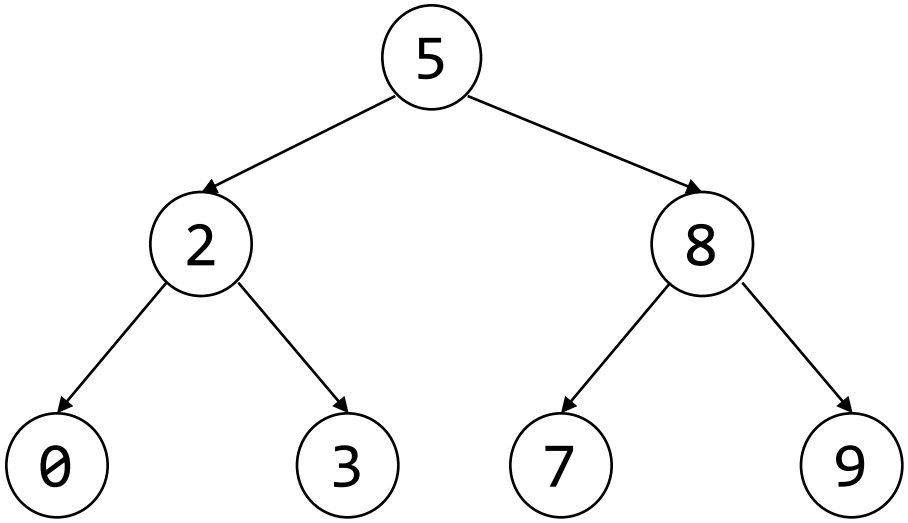
Postorder

"Post:" process root after subtrees



Poll

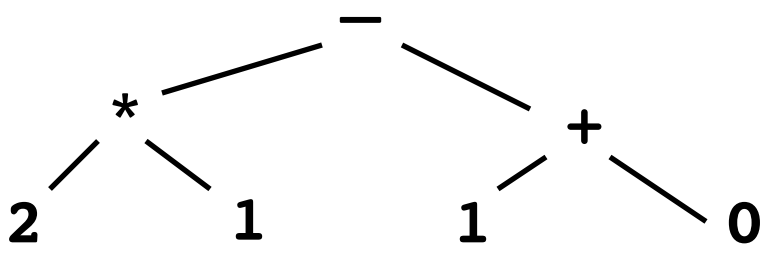
Which traversal would print out this BST in ascending order?



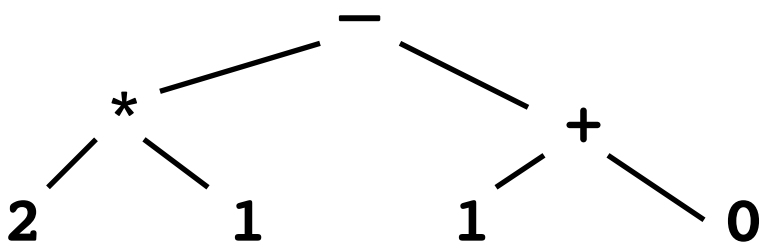
Example: Syntax Trees

Syntax Trees

- Trees can represent (Java) expressions
- Expression: `2 * 1 – (1 + 0)`
- Tree:



Traversals of expression tree

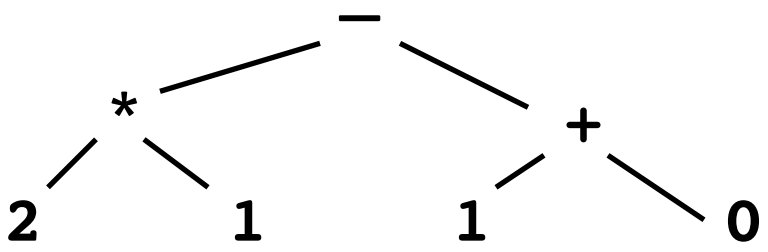


Preorder traversal

1. Visit the root
2. Visit the left subtree
3. Visit the right subtree

– * 2 1 + 1 0

Traversals of expression tree



Preorder traversal

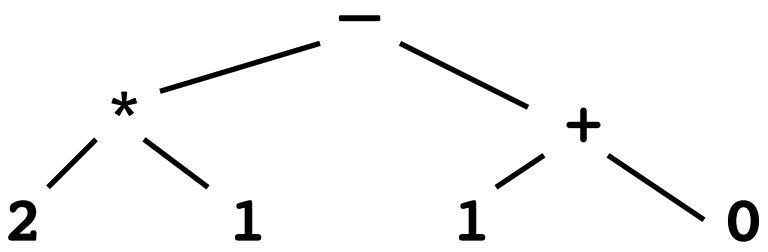
– * 2 1 + 1 0

Postorder traversal

2 1 * 1 0 + –

1. Visit the left subtree
2. Visit the right subtree
3. Visit the root

Traversals of expression tree



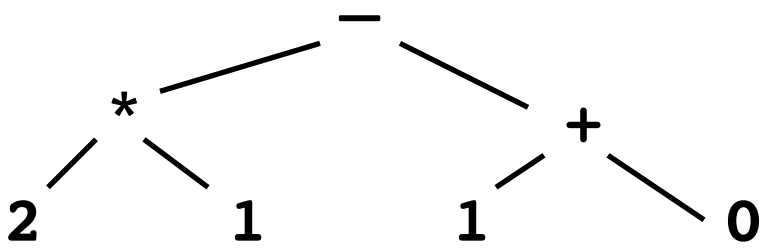
Preorder traversal — – * 2 1 + 1 0

Postorder traversal — 2 1 * 1 0 + –

Inorder traversal — 2 * 1 – 1 + 0

1. Visit the left subtree
2. Visit the root
3. Visit the right subtree

Traversals of expression tree



Preorder traversal — – * 2 1 + 1 0

Postorder traversal — 2 1 * 1 0 + –

Inorder traversal — 2 * 1 – 1 + 0

Original expression, except for parens

Prefix notation

- Function calls in most programming languages use prefix notation: e.g., **`add(37, 5)`**.
- Aka Polish notation (PN) in honor of inventor, Polish logician Jan Łukasiewicz
- Some languages (Lisp, Scheme, Racket) use prefix notation for *everything* to make the syntax uniform.

```
(- (* 2 1) (+ 1 0))

(define (fib n)
  (if (<= n 2)
      1
      (+ (fib (- n 1)) (fib (- n 2)))))
```

Postfix notation

- Some languages (Forth, PostScript, HP calculators) use postfix notation
- Aka reverse Polish notation (RPN)

```
2 1 mul 1 0 add sub

/fib { dup
       3 lt
         { pop 1 }
         { dup 1 sub fib exch 2 sub fib add }
       ifelse
     } def
```

Back to Trees

Recover tree from traversal

Suppose inorder is B C A E D.

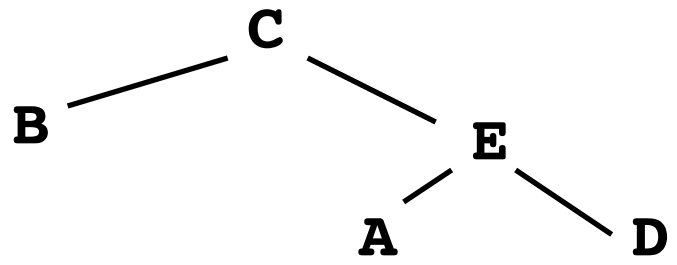
Can we recover the tree uniquely?

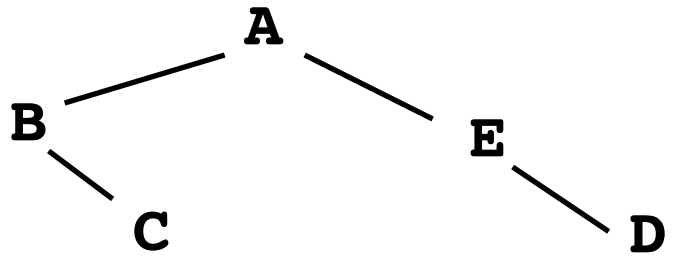
Discuss.

Recover tree from traversal

Suppose inorder is B C A E D.

Can we recover the tree uniquely? No!





Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Can we determine the tree uniquely?

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Can we determine the tree uniquely? Yes!

- What is root? Preorder tells us: A
- What comes before/after root A? Inorder tells us:
  - Before: B C
  - After: E D
- Now recurse! Figure out left/right subtrees using same technique.

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

How can we determine the tree uniquely?

Discuss.

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Root is A; left subtree contains B C; right contains E D

Left:
Inorder is B C
Preorder is B C

- What is root? Preorder: B
- What is before/after B? Inorder:
  - Before: nothing
  - After: C

Right:
Inorder is E D
Preorder is D E

- What is root? Preorder: D
- What is before/after D? Inorder:
  - Before: E
  - After: nothing

Recover tree from traversals

Suppose inorder is B C A E D

preorder is A B C D E

Tree is