TREES, PART 2

Lecture 13
CS2110 – Spring 2019

Finish Lec12

Announcements

- Prelim conflict quiz was due last night. Too late now to make changes. We won't be sending confirmations about time swaps (5:30 vs 7:30); if you requested it, you got it.
- Room assignments for the prelim (including SDS accommodations) will be announced by Monday. Please be patient.

JavaHyperText topics

- Tree traversals (preorder, inorder, postorder)
- Stack machines

…will be added by the end of this weekend

Trees, re-implemented

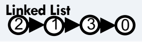
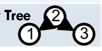
- Last time: lots of **`null`** comparisons to handle empty trees
- A more OO design:
  - Interface to represent operations on trees
  - Classes to represent behavior of empty vs. non-empty trees

Demo

Iterate through data structure

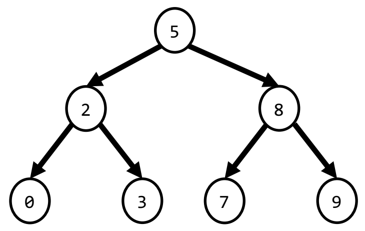
Iterate: process elements of data structure

- Sum all elements
- Print each element
- …

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

Iterate through data structure

7



Discuss: What would a reasonable order be?

Demo

Tree Traversals

8

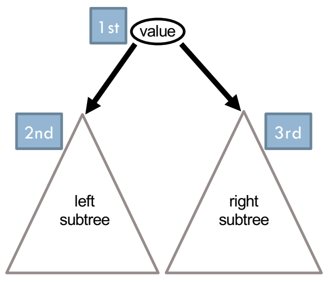
Tree traversals

9

- Iterating through tree is aka tree traversal
- Well-known recursive tree traversal algorithms:
  - Preorder
  - Inorder
  - Postorder
- Another, non-recursive: level order (later in semester)

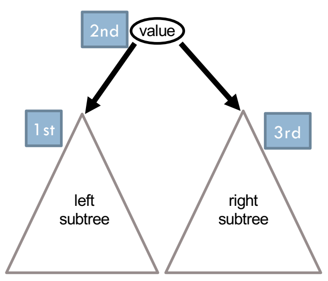
Preorder

"Pre:" process root before subtrees



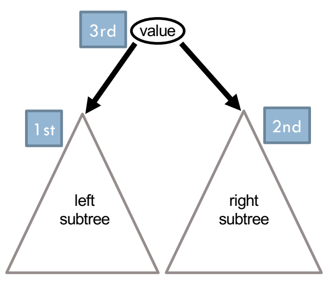
Inorder

"In:" process root in-between subtrees



Postorder

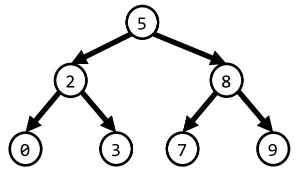
"Post:" process root after subtrees



Poll

13

Which traversal would print out this BST in ascending order?



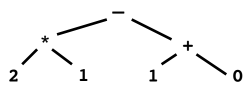
14

Example: Syntax Trees

Syntax Trees

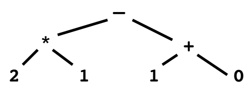
15

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



Traversals of expression tree

16

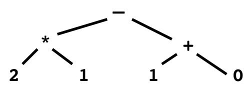


Preorder traversal — `– * 2 1 + 1 0`

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

Traversals of expression tree

17



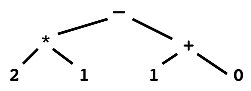
Preorder traversal — `– * 2 1 + 1 0`

Postorder traversal — `2 1 * 1 0 + –`

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

Traversals of expression tree

18



Preorder traversal — `– * 2 1 + 1 0`

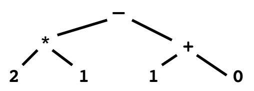
Postorder traversal — `2 1 * 1 0 + –`

Inorder traversal — `2 * 1 – 1 + 0`

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

Traversals of expression tree

19



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

20

- 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

21

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

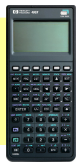
```
2 1 mul 1 0 add sub

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

Postfix notation

22

In about 1974, Gries paid $300 for an HP calculator, which had some memory and used postfix notation. Still works.

In about 1993, Clarkson paid $150 for an HP calculator with more memory, buttons, and screen.



Mac Calculator also does RPN

Demo

Syntax trees: in code

23

```
public interface Expr {
  int eval();
  String inorder();
}

public class Int implements Expr {
  private int v;
  public int eval() { return v; }
  public String inorder() { return " " + v + " "; }
}

public class Add implements Expr {
  private Expr left, right;
  public int eval() { return left.eval() + right.eval(); }
  public String inorder() {
    return "(" + left.infix() + "+" + right.infix() + ")";
  }
}
```

(see website for full code)

Java syntax

24

- Java compiler:
  - translates your text file (list of characters) into a syntax tree
  - decides whether program is legal
- *Grammar* for legal programs: https://docs.oracle.com/javase/specs/jls/se8/html/jls-19.html
  - You could use it to generate every possible Java program. (That would take forever.)

25

Back to Trees

Recover tree from traversal

26

Suppose inorder is B C A E D.

Can we recover the tree uniquely?

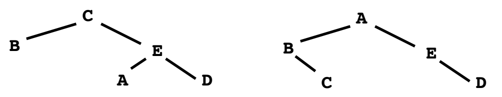
Discuss.

Recover tree from traversal

27

Suppose inorder is B C A E D.

Can we recover the tree uniquely? No!



Recover tree from traversals

28

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

29

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

30

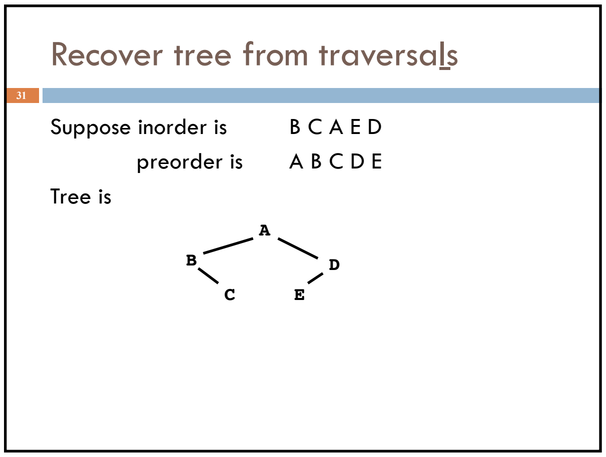
Suppose inorder is B C A E D

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

31

Suppose inorder is B C A E D

preorder is A B C D E

Tree is