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

Iterate through data structure

Iterate: process elements of data structure
- Sum all elements
- Print each element
- ...

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Order to iterate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>Forwards: 2, 1, 3, 0, Backwards: 0, 3, 1, 2</td>
</tr>
<tr>
<td>Linked List</td>
<td>Forwards: 2, 1, 3, 0</td>
</tr>
<tr>
<td>Binary Tree</td>
<td>???</td>
</tr>
</tbody>
</table>
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?

Syntax Trees

- Trees can represent (Java) expressions
- Expression: $2 \times 1 - (1 + 0)$
- Tree:

  - Traversals of expression tree

    * Preorder traversal
      1. Visit the root
      2. Visit the left subtree
      3. Visit the right subtree

    * Postorder traversal
      2 $1 + 0 -$

    * Inorder traversal
      1. Visit the left subtree
      2. Visit the root
      3. Visit the right subtree

Example: Syntax Trees

Syntax Trees can represent (Java) expressions. For example, the expression $2 \times 1 - (1 + 0)$ can be represented as a tree:

- Preorder traversal
  1. Visit the root
  2. Visit the left subtree
  3. Visit the right subtree

- Postorder traversal
  $2 \times 1 + 0 -$

- Inorder traversal
  1. Visit the left subtree
  2. Visit the root
  3. Visit the right subtree
Traversals of expression tree

<table>
<thead>
<tr>
<th>Method</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preorder traversal</td>
<td>$- * 2 1 + 0$</td>
</tr>
<tr>
<td>Postorder traversal</td>
<td>$2 1 + 0 + -$</td>
</tr>
<tr>
<td>Inorder traversal</td>
<td>$2 * 1 - 1 + 0$</td>
</tr>
</tbody>
</table>

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.

```latex
\text{(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)

```latex
2 1 mul 1 0 add sub
```

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

Postfix notation

- 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.

Java syntax

- Java compiler:
  - translates your text file (list of characters) into a syntax tree
  - decides whether program is legal
- Grammar for legal programs:
  - You could use it to generate every possible Java program. (That would take forever.)

Syntax trees: in code

```java
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)
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 traversal

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 traversal

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
  - Before: B C
  - After: E D

Right:
- Inorder is E D
  - Before: E D
  - After: nothing
### Recover tree from traversals

Suppose inorder is B C A E D
preorder is A B C D E

Tree is

```
   A
  / \  
 B   C
   \   
    D   E
```