CSE 143 Java

Applications of Trees

Overview

- Applications of traversals
- Syntax trees
- Expression trees
- Postfix expression evaluation
- Infix expression conversion and evaluation

Traversals (Review)

- **Preorder** traversal:
  - "Visit" the (current) node first
    i.e., do what ever processing is to be done
  - Then, (recursively) do preorder traversal on its children, left to right
- **Postorder** traversal:
  - First, (recursively) do postorder traversals of children, left to right
  - Visit the node itself last
- **Inorder** traversal:
  - (Recursively) do inorder traversal of left child
  - Then visit the (current) node
  - Then (recursively) do inorder traversal of right child
    Footnote: pre- and postorder make sense for all trees; inorder only for binary trees

Two Traversals for Printing

```
public void printInOrder(BTreeNode t) {
  if (t != null) {
    printInOrder(t.left);
    system.out.println(t.data + " ");
    printInOrder(t.right);
  }
}
```

```
public void printPreOrder(BTreeNode t) {
  if (t != null) {
      system.out.println(t.data + " ");
      printPreOrder(t.left);
       printPreOrder(t.right);
  }
}
```

Traversing to Delete

- Use a postorder traversal to delete all the nodes in a tree

```
// delete binary tree with root t
void deleteTree(BTreeNode t) {
  if (t != null) {
    deleteTree(t.left);
    deleteTree(t.right);
    t=null;
  }
}
```

- Puzzler: Would inorder or preorder work just as well??

Analysis of Tree Traversal

- How many recursive calls?
  - Two for every node in tree (plus one initial call);
  - `O(N)` in total for `N` nodes
- How much time per call?
  - Depends on complexity `O(V)` of the visit
  - For printing and many other types of traversal, visit is `O(1)` time
- Multiply to get total
  - `O(N)*O(V) = O(N*V)`
- Does tree shape matter?

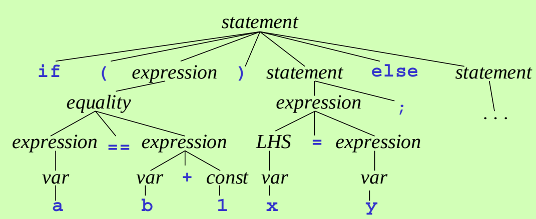
Syntax and Expression Trees

- Computer programs have a hierarchical structure
  - All statements have a fixed form
  - Statements can be ordered and nested almost arbitrarily (nested `if-then-else`)
- Can use a structure known as a *syntax tree* to represent programs
  - Trees capture hierarchical structure

A Syntax Tree

Consider the Java statement:

`if ( a == b + 1 ) x = y; else ...`



Syntax Trees

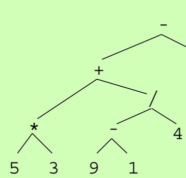
- An entire .java file can be viewed as a tree
- Compilers build syntax trees when compiling programs
  - Can apply simple rules to check program for syntax errors
  - Easier for compiler to translate and optimize than text file
- Process of building a syntax tree is called *parsing*

Binary Expression Trees

- A *binary expression tree* is a syntax tree used to represent meaning of a mathematical expression
  - Normal mathematical operators like `+`, `-`, `*`, `/`
- Structure of tree defines result
- Easy to evaluate expressions from their binary expression tree (as we shall see)

Example

```
5 * 3 + (9 - 1) / 4 - 1
```



Infix, Prefix, Postfix Expressions

```
5 * 3
```

- **Infix: binary operators are written between operands**
- **Postfix: operator after the operands**
- **Prefix: operator before the operands**

Expression Tree Magic

- Traverse in <u>postorder</u> to get <u>postfix</u> notation!

  `5 3 * 9 1 - 4 / + 1 -`
- Traverse in <u>preorder</u> to get <u>prefix</u> notation

  `- + * 5 3 / - 9 1 4 1`
- Traverse in <u>inorder</u> to get <u>infix</u> notation

  `5 * 3 + 9 - 1 / 4 - 1`
  - Note that infix operator precedence may be wrong! Correction: add parentheses at every step

  `(((5*3) + ((9 - 1) / 4)) - 1)`

More on Postfix

- **3 4 5 * -** means same as (3 (4 5 *) -)
  - infix: **3 - (4 * 5)**
- Parentheses aren't needed!
  - When you see an operator:
    - both operands must already be available.
    - Stop and apply the operator, then go on
- Precedence is implicit
  - Do the operators in the order found, period!
- Practice converting and evaluating:
  - 1 2 + 7 * 2 %
  - (3 + (5 / 3) * 6) - 4

Why Postfix?

- Does not require parentheses!
- Some calculators make you type in that way
- Easy to process by a program
  - simple and efficient algorithm

Postfix Evaluation Algorithm

- Create an empty stack
  - Will hold tokens
- Read in the next "token" (operator or data)
  - If data, push it on the data stack
  - If (binary) operator:
    - call it "op"
    - Pop off the most recent data (B) and next most recent (A) from the stack
    - Perform the operation R = *A op B*
    - Push R on the stack
- Continue with the next token
- When finished, the answer is the stack top.
- Simple, but works like magic!

Check Your Understanding

- According to the algorithm, 3 5 - means
  - **3 - 5 ? or**
  - **5 - 3 ?**
- If data stack is ever empty when data is needed for an operation:
  - Then the original expression was bad
  - Why? Give an example
- If the data stack is not empty after the last token has been processed and the stack popped:
  - Then the original expression was bad
  - Why? Give an example

Example: 3 4 5 - *

Draw the stack at each step!

- Read 3. Push it (because it's data)
- Read 4. Push it.
- Read 5. Push it.
- Read -. Pop 5, pop 4, perform 4 - 5. Push -1
- Read *. Pop -1, pop 3, perform 3 * -1. Push -3.
- No more tokens. Final answer: pop the -3.
  - note that stack is now empty

Algorithm: converting in- to post-

- Create an empty stack to hold operators
- Main loop:
  - Read a token
  - If operand, output it immediately
  - If '(', push the '(' on stack
  - If operator
    - hold it aside temporarily
    - if stack top is an op of => precedence: pop and output
      - repeat until '(' is on top or stack is empty
    - push the new operator
  - If ')', pop and output until '(' has been popped
- Repeat until end of input
- Pop and output rest of stack

Magic Trick

- Suppose you had a bunch of numbers, and inserted them all into an initially empty BST.
- Then suppose you traversed the tree in-order.
- The nodes would be visited in order of their values. In other words, the numbers would come out sorted!
- Try it!
- This algorithm is called **TreeSort**

Tree Sort

- O(N log N) most of the time
  - Time to build the tree, plus time to traverse
  - When is it not O(N log N)?
- Trivial to program if you already have a binary search tree class
- Note: not an "in-place" sort
  - The original tree is left in as-is, plus there is a new sorted list of equal size
  - Is this good or bad?
  - Is this true or not true of other sorts we know?

Preview of CSE326/373: Balanced Search Trees

- Cost of basic binary search operations
  - Dependent on tree height
  - `O(log N)` for `N` nodes if tree is balanced
  - `O(N)` if tree is very unbalanced
- Can we ensure tree is always balanced?
  - Yes: `insert` and `delete` can be modified to keep the tree pretty well balanced
    - Several algorithms and data structures exist
    - Details are complicated
  - Results in `O(log N)` "find" operations, even in worst case