CSC 220 Data Structures

Trees II - Traversals
Parkland College Spring 2016
20160224
Traversals

• How do we actually move through a tree?
• Sequential data structures only had two choices
  1. Forwards
  2. Backwards
• Trees, more options
• Pre-, In-, Post-order traversals
• Breadth-first
Preorder traversal

1. “visit” root of current subtree
2. recursively preorder traverse left subtree
3. recursively preorder traverse right subtree
   - (At this point, left-first vs. right-first doesn’t matter as much as consistency. We’ll see exceptions when we get to Binary Search Trees in a few weeks.)
Preorder example

prints ‘A’
preorder(A.left())
prints ‘B’
preorder(B.left())
prints ‘D’
preorder(D.left())
preorder(D.right())
preorder(B.right())
preorder(A.right())
prints ‘C’
preorder(C.left())
prints ‘E’
preorder(E.left())
preorder(E.right())
prints ‘G’
preorder(G.left())
preorder(G.right())
preorder(C.right())
prints ‘F’
preorder(F.left())
prints ‘H’
preorder(H.left())
preorder(H.right())
preorder(F.right())
prints ‘J’
preorder(J.left())
preorder(J.right())
(return from F.right)
(return from C.right)
(return from A.right)
Inorder traversal

1. recursively inorder traverse left subtree
2. visit node
3. recursively inorder traverse right subtree
Inorder example

```plaintext
inorder(A.left)
inorder(B.left)
inorder(D.left)
visit D
inorder(D.right)
visit B
inorder(B.right)
visit A
inorder(A.right)
inorder(C.left)
inorder(E.left)
visit E
inorder(E.right)
inorder(G.left)
visit G
inorder(G.right)
inorder(C.right)
inorder(F.left)
inorder(H.left)
visit H
inorder(H.right)
visit F
inorder(F.right)
inorder(J.left)
visit J
inorder(J.right)
(return from F.right)
(return from C.right)
(return from A.right)
```
Postorder traversal

1. recursively postorder traverse left subtree
2. recursively postorder travers right subtree
3. visit node
postorder example

```
postorder(A.left)
postorder(B.left)
postorder(D.left)
postorder(D.right)
visit D
postorder(B.right)
visit B
postorder(A.right)
postorder(C.left)
postorder(E.left)
postorder(E.right)
postorder(G.left)
postorder(G.right)
visit G
visit E
postorder(C.right)
postorder(F.left)
postorder(H.left)
postorder(H.right)
visit H
postorder(F.right)
postorder(J.left)
postorder(J.right)
visit J
visit F
visit C
visit A
```
pre-, in-, post- for general trees

- if siblings can be ordered

- pre:
  1. visit
  2. recursively traverse children in order

- in:
  1. recursively traverse “left-most”
  2. visit
  3. recursively traverse remaining children in order

- post:
  1. recursively traverse children in order
  2. visit
Pre-, in-, post- complexities

- Work at a node depends on number of children
  - $O(1)$ for the visit
  - $O(\text{num children})$ for the recursions
  - $O(\sum c_p + 1)$ total

- Overall, that accumulates for each node in the tree
  - $O(\sum p(\sum c_p + 1)) = O(\sum p\sum c_p + \sum p1)$ by distributive law
  - $\sum p1$ is just $n$
  - $\sum p\sum c_p$ is just $n - 1$ (since all but root are children)
  - $O(n - 1 + n) = O(2n - 1)$ or $O(n)$, linear.
Breadth-first traversal

• sibling-sibling or “per generation”
• not recursive (would have lots of needless work)
• use an auxiliary sequential data structure
• used in game trees for all possible moves at a particular time
Implementing traversal

• `Tree.__iter__`
• `Tree.positions()`
• `Tree.preorder`
• `Tree._subtree_preorder`