CMPT 280

Topic 9: Tree Traversals

Mark G. Eramian

University of Saskatchewan

References

- Textbook, Chapter 9

Depth First Traversals

Recall the general algorithm for depth-first traversals:

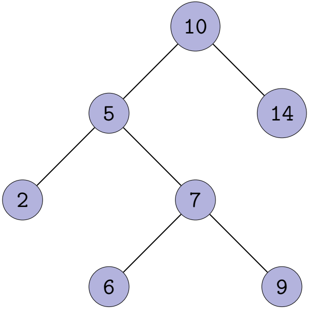
```
Algorithm depthFirstTraversal(N)
Parameters:
    N is a tree node

visit node N
for each child T of N
    depthFirstTraversal(T)
```

Exercise 1

- Write a method for `LinkedSimpleTree280<I>` that performs a depth-first traversal that prints out the contents of each node.
- Besides being a depth-first traversal, what kind of binary tree traversal is your solution?

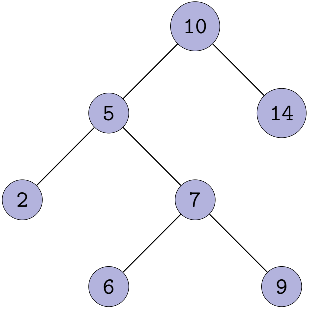
In-order Traversal of Binary Search Trees

- Recall the Binary Search Trees (from CMPT 145).
- What special property does the in-order traversal of a binary search tree have?



Exercise 2: In-order traversals

- Modify the method from Exercise 1 to create an in-order traversal.



Exercise 3: Post-order traversals

- Modify the method from Exercise 1 to create a post-order traversal.
- The sequence generated by a post-order traversal has no special significance for binary search trees.
- BUT... we can do some fairly interesting things with post-order traversal that don't involve printing the node contents.

Post-order traversals

We can accomplish the following with depth-first traversals:

- Count number of nodes in the tree.
- Determine hight of the tree
- Evaluate expression tree.

Exercise 4: Counting number of nodes.

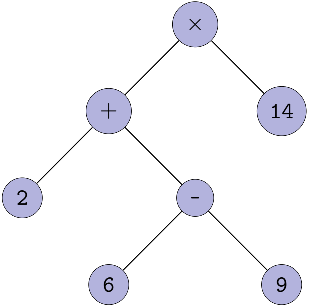
- Write a post-order traversal for `LinkedSimpleTree280<I>` that returns the number of nodes in the tree.
- Hint: If you know how many nodes are in the left and right subtrees, then the number of nodes in the tree is 1 + # nodes in left subtree + # nodes in right subtree.
- What was the "visit" operation in the solution?

Exercise 5: Computing tree height.

- Modify the method from Exercise 4 to compute the height of the tree. It shouldn't take much since both are post-order traversals!
- Hint: If you know the height of the left and right subtrees, then the height of the tree is 1 + max(left subtree height, right subtree height)
- What was the "visit" operation in the solution?

Postorder Traversal: Evaluate Expression Tree

- Why is evaluating an expression tree a post-order traversal?



Breadth-First Traversals

- Algorithm for BFT requires a queue.
- Idea:
  - After we visit a node, put all of it's children on a queue.
  - Keep visiting nodes as long as the queue is not empty.

```
Algorithm breathFirstTraversal(T)
T is a tree.

Let Q be a queue.
Q.add(T.rootNode)

while(Q is not empty)
    p = Q.remove();
    visit p
    for each child s of p
     Q.add(s)
```

Exercise 6

- Write a method for LinkedSimpleTree280 that prints the contents of the nodes in level-order. The method should follow the BFT algorithm below with an appropriate "visit" operation.
- Hint: for a queue we could use any of java.util.LinkedList, LinkedList280<I>, or LinkedQueue280<I>.

```
Algorithm breathFirstTraversal(T)
T is a tree.

Let Q be a queue.
Q.add(T.rootNode)

while(Q is not empty)
    p = Q.remove();
    visit p
    for each child s of p
     Q.add(s)
```

Deep Clone: A depth-first algorithm.

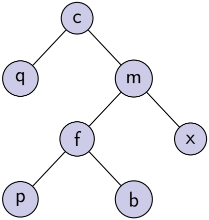
We can now see that deep cloning of an object is a depth-first traversal of the reference fields in an object. Note that the references in the object X and the objects those objects reference (etc.) form a tree (whether X is a actually a tree ADT or not)!

```
Algorithm deepClone(X)
X is an object

Y = shallowClone(X)  // "Visit" operation is a shallow clone of
                     // the current object.

for each non-null reference X.r in X
    Y.r = deepClone(X.r);

return Y
```

Can we re-write this algorithm so that it is a bit more specific to cloning a LinkedSimpleTree280<I>?

Printing the Structure of Binary Trees

- Trees are hard to draw with root at the top.
- Trees are easy to draw with root at the left and descendants to the right:




```
          3: x
     2: m
               4: b
          3: f
               4: p
1: c
     2: q
```

- Right subtree of a node is above and indented; left subtree is below and indented.
- This can be achieved with a non-standard traversal: right, root, left

Printing the Structure of Trees

(This is in LinkedSimpleTree280)

```
/** Form a string representation that includes level numbers when the tree
 *  is at level i.
 *  @param i  the level of this tree in the main tree, i.e., number of
 *            indentations minus 1
 *  @timing  Time = O(n), where n = number of items in this tree  */
 protected String toStringByLevel(int i)
{
     StringBuffer blanks = new StringBuffer((i - 1) * 5);
     for (int j = 1; j < i; j++)
          blanks.append("       ");
     if (isEmpty())
          return   "\n" + blanks + i + ": -";
     else
     {
          boolean printSubtrees  = !rootLeftSubtree().isEmpty() ||
                                   !rootRightSubtree().isEmpty();

          String result = "";
          if (printSubtrees)
               result += rootRightSubtree().toStringByLevel(i+1);

          result += "\n" + blanks + i + ": " + rootItem();

          if (printSubtrees)
               result += rootLeftSubtree().toStringByLevel(i+1);

          return result;
     }
}
```

Next Class

- Next class reading: Chapter 10 — Dispensers