CMPSCI 187, Spring 2015 Discussion #10: Traversing Binary Trees Individual Handout

Marc Liberatore and John Ridgway

06 April 2015

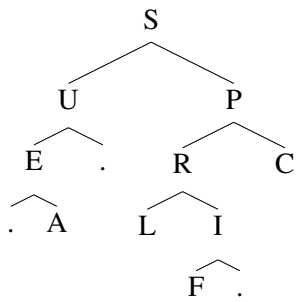
We've seen three ways to traverse a binary tree: inorder, preorder, postorder. A natural fourth way is level order, where you start with the root, then visit all the level 1 nodes in left to right order, then all the level 2 nodes in left to right order, and so on. The first three have elegant recursive definitions, though the fourth does not. In this discussion we will study these four traversals in a different way — for each one we will write a method that will return the next node in the given order, using the pointers to navigate. This will give us some good experience navigating binary trees, though our trees may have nodes with only a single child.

To simplify this assignment, we won't concern ourselves with the data stored at each nodes or parent/child getters. We will use the following simple classes:

```
1  public class Tree {
2    public Node root;
3    // other methods of Tree would go here
4  }
```

```
1  public class Node {
2    public Node up, left, right;
3
4    // assume the existence of a reasonable constructor, etc.
5
6    public boolean isLeft() {
7      return (up != null && this == up.left);
8    }
9
10   public boolean isRight() {
11     return (up != null && this == up.right);
12   }
13 }
```

As a warmup, list the order of the ten nodes of this tree in each of the four traversals. S has children U and P, U has left child E which has right child A, P has children R and C, R has children L and I, I has left child F. Dots ('.') signify null pointers in cases where nodes have only one child.



Here is one of the four methods, for an inorder traversal:

```
public Node nextInorder() {
  // if there is a right subtree,
  // return inorder–first descendant of right subtree
  if (right != null) {
    Node cur = right;
    while (cur.left != null)
      cur = cur.left;
    return cur;
  }

  // otherwise, find the parent of lowest ancestor that
  // is a left child, or null if there is no such ancestor
  Node anc = this;
  while (anc.isRight())
    anc = anc.up;
  return anc.up;
}
```

From the current node, this method returns the *next* node in an inorder traversal of the tree. Trace the operation of the method, and note that if you want to perform a correct inorder traversal using it, you'll need to start with the correct first node of an inorder traversal.

Question 1: Write a method `public Node nextPreorder()` for the `Node` class. This method should return the next node in preorder, or `null` if the calling node is the last one in the tree.

Question 2: Write a method `public Node nextPostorder()` for the `Node` class. This method should return the next node in postorder, or `null` if the calling node is the last one in the tree.

Question 3: Write a method `public Node nextLevelOrder()` for the `Node` class. This method should return the next node in level order, or `null` if the calling node is the last one in the tree. You may find it useful to also write a method `public Node firstDesc(int depth)`. If the calling node has any descendants at the given `depth` (that many levels below it), `firstDesc` returns the leftmost one. This method is most easily coded recursively.

Remember: Each of these methods returns the *next* node in the given traversal order. If you expect the method to generate the entire traversal, it must start at the correct first node, which depends upon the traversal order!