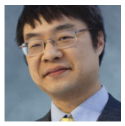
Tree-walking algorithms: Incrementally enumerating leaf nodes of an N-ary tree

devblogs.microsoft.com/oldnewthing/20200106-00

January 6, 2020



Raymond Chen

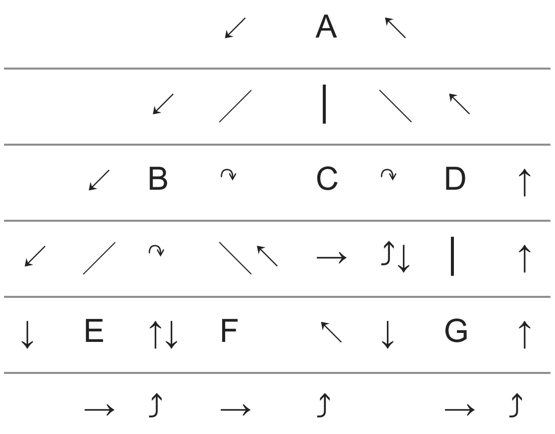
Suppose you have an *N*-ary tree, in which the node operations are

- Get first child.
- Get next sibling.
- Get parent.

For example, this type of tree structure may represent a window hierarchy. You also see it in a TreeView control.

Enumerating the nodes of this tree with a recursive algorithm is relatively straightforward. Doing it incrementally is trickier.

The idea is that we want to walk through the tree following the red arrows, as if we are walking along the outside of the tree with our left hand touching it.



The various types of tree walks differ primarily in where you stop to rest. And therefore, the differences are in the state that needs to be preserved when we reach the stopping point, so that we know how to resume our tree walk.

Let’s assume that we have a cursor class that can move through the tree.

```
class TreeCursor
{
  TreeCursor(TreeNode node);
  bool TryMoveToFirstChild();
  bool TryMoveToNextSibling();
  bool TryMoveToParent();
  TreeNode Current { get; };
};
```

Our first algorithm is to walk through the tree, stopping at each leaf node. A leaf node is a node with no children.

I choose this as our first algorithm because it has only one state: You’re at a leaf node and need to find the next leaf node. Being at a leaf node means that you’re at the “curve around the bottom of a node ” part of the path around the tree.

To find the first leaf node, we keep moving to the first child until we find a node with no children.

To find the next leaf node, we first realize that since we are at a leaf node, we have no children of our own. So we move up to the parent, and then back down to the next child. That next child is our starting node’s next sibling. Once there, we keep moving to the first child until we find a node with no children.

The last case is where we are at the last sibling. In that case, we move up to the parent node and try again. If we are at the root (no parent node), then we’re done.

Capturing this algorithm results in the following:

```
class LeafWalker
{
  private TreeCursor cursor;

  public LeafWalker(TreeNode node)
  {
    cursor = new TreeCursor(node);
    GoDeep();
  }

  public bool MoveNext()
  {
    do {
      if (cursor.TryMoveToNextSibling()) {
        GoDeep();
        return true;
      }
    } while (cursor.TryMoveToParent());
    return false;
  }

  public TreeNode Current => cursor.Current;

  private void GoDeep()
  {
    while (cursor.TryMoveToFirstChild()) { }
  }
}
```

That was a nice warm-up. We'll try something a little harder next time.

Raymond Chen

Follow