Successor/Predecessor Rules in Binary Trees

Thomas A. Anastasio

July 7, 2003

Introduction

Binary tree traversals are commonly made in one of three patterns, *inorder*, *preorder*, and *postorder*. These traversals are easy to describe recursively as follows, for a subtree rooted at `n`:

```
void inorder(node * n)
{
  inorder(leftChild(n);
  visit(n);
  inorder(rightChild(n);
}
```

```
void preorder(node * n)
{
  visit(n);
  preorder(leftChild(n));
  preorder(rightChild(n));
}
```

```
void postorder(node * n)
{
  postorder(leftChild(n));
  postorder(rightChild(n));
  visit(n);
}
```

Unfortunately, these easy recursive functions are not useful for building iterators for binary trees. We need functions that can take one step at a time through the traversal. Each step can be thought of as finding the next or previous node for forward-moving or backwards-moving iterators, respectively.

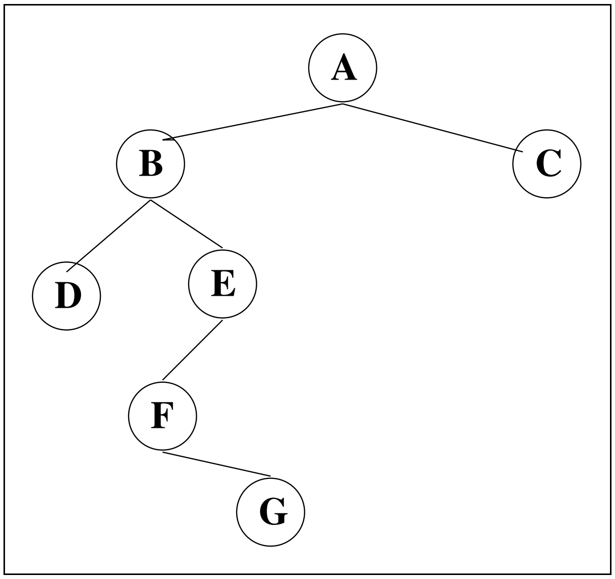
This paper describes the "rules" for determining the next (successor) or previous (predecessor) nodes for any node in a binary tree for each of the traversal patterns.

Inorder Successor

To find the inorder successor of node u:
If u has a right child, r, then succ(u) is the leftmost descendent of r
Otherwise, succ(u) is the closest ancestor, v, of u (if any) such that u is descended from the left child of v. If there is no such ancestor, then succ(u) is undefined.

An iterator would start with the leftmost node.

For example, an inorder traversal of the following binary tree yields the sequence DBFGEAC.



Taking the nodes one at a time and applying the rule:

node D: Does not have a right child. Its successor is the closest ancestor, v such that node-D is descended from the left child of v. Node-D is descended from the left child of node-B, so succ(D) is node-B.

node B: Has a right child (node-E), so successor is the leftmost descendent of node-E, namely node-F.

node F: Has a right child (node-G), so successor is the leftmost descendent of node-G, namely node-G itself.

node G: Does not have a right child. Its successor is the closest ancestor, v such that node-G is descended from the left child of v. Node-G is descended from the left child of node-E, so succ(G) is node-E.

node E: Does not have a right child. Its successor is the closest ancestor, v such that node-E is descended from the left child of v. Node-E is descended from the left child of node-A, so succ(E) is node-A.

node A: Has a right child (node-`C`), so successor is the leftmost descendent of node-`C`, namely node-`C` itself.

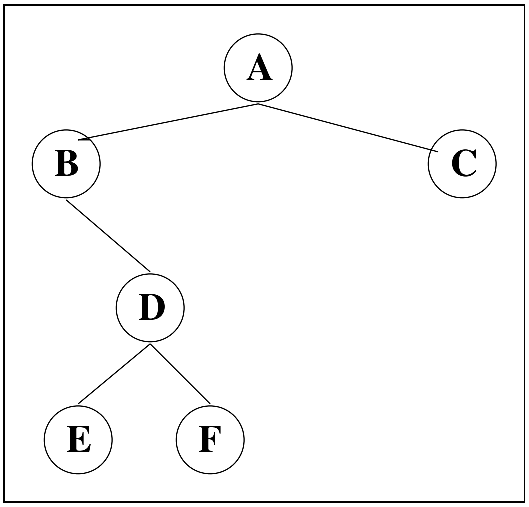
node C: Does not have a right child. Its successor would be the closest ancestor, `v` such that node-`C` is descended from the left child of `v`. However, there is no such ancestor, so `succ(C)` is undefined (node-`C` has no successor).

Preorder Successor

To find the preorder successor of node `u`:
If `u` has a left child, `l`, then `succ(u)` is `l`.
Otherwise, if `u` has a right child, `r`, then `succ(u)` is `r`.
Otherwise, `u` is a leaf and the following rules apply:
if `u` is a left child and has a right sibling, `rs`, then `succ(u)` is `rs`.
otherwise, if `u` has an ancestor, `v`, which is a left-child and `v` has a right sibling, `vrs`, then `succ(u)` is `vrs`
If there is no such ancestor, then `succ(u)` is undefined.

An iterator would start with the root of the tree.

For example, a preorder traversal of the following binary tree yields the sequence `ABDEFC`.



Taking the nodes one at a time and applying the rule:

node A: Has a left child, node-`B`, so successor is node-`B`.

node B: Has a right child, node-`D`, so successor is node-`D`.

node D: Has a left child, node-`E`, so successor is node-`E`.

node E: Is a leaf. It is a left child and has a right sibling, node-`F`, so successor is node-`F`.

node F: Is a leaf. Is not a left child. It has an ancestor, node-`B`, that is a left child and that has a right sibling, node-`C`, so successor of node-`F` is node-`C`.

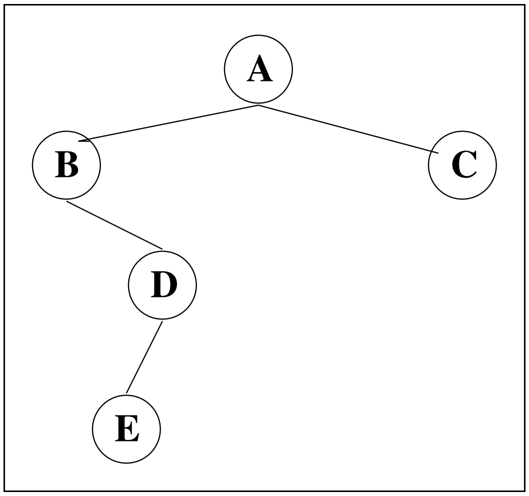
node C: Is a leaf. Is not a left child. Does not have an ancestor that is a left child. Therefore, the successor of node-`C` is undefined.

Postorder successor

To find the postorder successor of node `u`:
If `u` is the root of the tree, `succ(u)` is undefined.
Otherwise, if `u` is a right child, `succ(u)` is `parent(u)`.
Otherwise `u` is a left child and the following applies:
  if `u` has a right sibling, `r`, `succ(u)` is the leftmost *leaf* in `r`'s subtree
  otherwise `succ(u)` is `parent(u)`.

An iterator would start with the leftmost *leaf* (not necessarily the leftmost node).

For example, a postorder traversal of the following binary tree yields the sequence `EDBCA`. Notice that it starts with the leftmost *leaf*, node-`E`, not the leftmost node, node-`B`.



Taking the nodes one at a time and applying the rule:

node E: Is a left child and does not have a right sibling. Therefore, the successor of node-`E` is its parent, node-`D`.

node D: Is a right child. The successor of node-`D` is its parent, node-`B`.

node B: Is a left child and does have a right sibling, node-`C`. Therefore the successor of node-`B` is node-`C`.

node C: Is a right child. The successor of node-`C` is its parent, node-`A`.

node A: Is the root of the tree, so its successor is undefined.

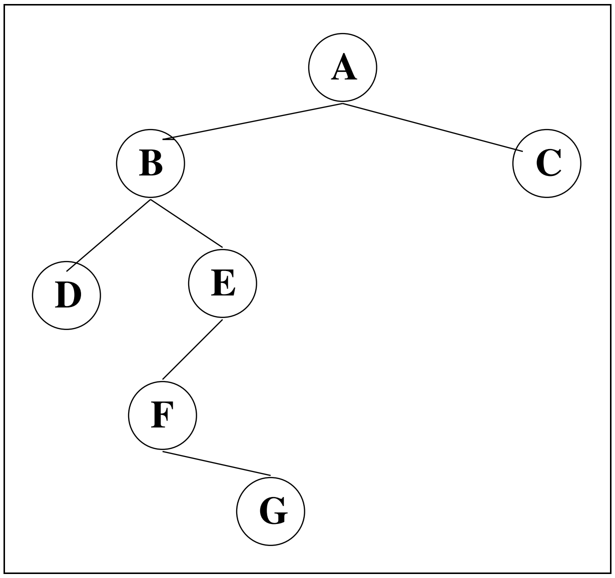
Inorder Predecessor

To find the inorder predecessor of node `u`
If `u` has a left child, `l`, then `pred(u)` is the rightmost descendent of `l`
Otherwise, `pred(u)` is the closest ancestor, `v`, of `u` (if any) such that `u` is descended from the right child of `v`.

If there is no such ancestor, then `pred(u)` is undefined.

An iterator would start with the rightmost node.

For example, a reverse inorder traversal of the following binary tree yields the sequence `CAEGFBD`. Notice that it starts with the rightmost node-`C`.



Taking the nodes one at a time and applying the rule:

node C: Does not have a left child. Closest ancestor such that node-`C` is descended from the right child is node-`A`. Therefore, the predecessor of node-`C` is node-`A`.

node A: Has a left child, node-`B`. Rightmost descendent of node-`B` is node-`E`.

node E: Has a left child, node-`F`. Rightmost descendent of node-`F` is node-`G`.

node G: Does not have a left child. Closest ancestor such that node-`G` is descended from the right child is node-`F`.

node F: Does not have a left child. Closest ancestor such that node-`F` is descended from the right child is node-`B`.

node B: Has a left child, node-`D`. Rightmost descendent of node-`D` is node-`D` itself.

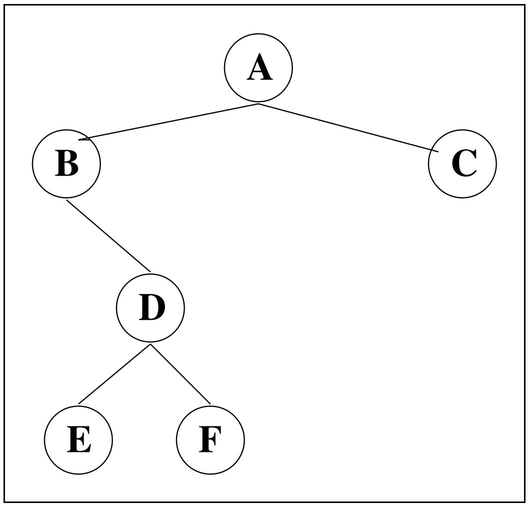
node D: Does not have a left child. There is no ancestor such that node-`D` is descended from the right child. Therefore, the predecessor of node-`D` is undefined.

Preorder Predecessor

To find the preorder predecessor of node `u`:
If `u` is the root of the tree, then `pred(u)` is undefined
If `u` has a left sibling, `ls`, then `pred(u)` is the rightmost descendent of `ls`
Otherwise, `pred(u)` is `parent(u)`.

An iterator would start with the rightmost node.

For example, a reverse preorder traversal of the following binary tree yields the sequence `CFEDBA`. Notice that it starts with the rightmost node-`C`.



Taking the nodes one at a time and applying the rule:

node C: Has left sibling, node-`B`. Predecessor of node-`C` is rightmost descendent of node-`B`, namely node-`F`.

node F: Has left sibling, node-`E`. Rightmost descendent of node-`E` is node-`E` itself.

node E: Does not have left sibling, so predecessor is the parent of node-`E`, namely node-`D`.

node D: Does not have left sibling, so predecessor is its parent, namely node-`B`.

node B: Does not have left sibling, so predecessor is its parent, namely node-`A`.

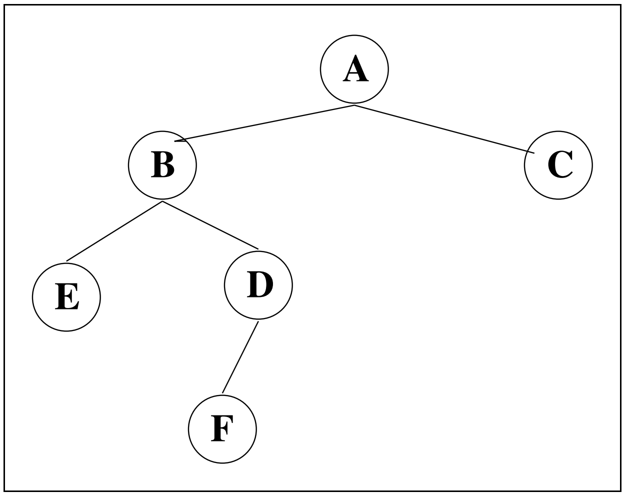
node A: Root, so predecessor is undefined.

Postorder Predecessor

To find the postorder predecessor of node `u`:
If `u` has a right child, `r`, then `pred(u)` is `r`.
Otherwise If `u` has a left child, `l`, then `pred(u)` is `l`.
Otherwise if `u` has a left sibling, `ls`, then `pred(u)` is `ls`
Otherwise if `u` has an ancestor, `v`, which is a right child and has a left sibling, `vls`, then `pred(u)` is `vls`
Otherwise, `pred(u)` is undefined.

An iterator would start with the root of the tree.

For example, a reverse postorder traversal of the following binary tree yields the sequence `ACBDFE`. Notice that it starts with the root node-`A`.



Taking the nodes one at a time and applying the rule:

node A: Has a right child, node-`C`.

node C: Has a left sibling, node-`B`.

node B: Has a right child, node-`D`.

node D: Has a left child, node-`F`.

node F: Has an ancestor, node-`D`, that is a right child and that has a left sibling, node-`E`. Therefore, the postorder predecessor of node-`F` is node-`E`.

node E: No right child, no left child, no suitable ancestor. Therefore, the postorder predecessor of node-`E` is undefined.