Computer Science 331

Binary Tree Traversals

Mike Jacobson

Department of Computer Science
University of Calgary

Supplemental Material

Outline

The Java Iterator Interface

An *iterator* is a program component that enables you to step through (traverse) a collection of data sequentially

- each item is considered exactly once
- typically does *not* permit the data to be modified

Java's `Iterator<T>` interface defines the following functions:

- `boolean hasNext()` — true if there is another entry to return
- `T next()` — returns the next entry (type `T`) in the iteration and advances the iterator by one position
- `void remove()` — (optional) removes last item returned

`next` throws a `NoSuchElementException` if there are no items left to return (enter collection has been traversed)

The Iterable Interface

Java's `Iterable` interface specifies one function:

- `Iterator<T> iterator()` — returns an iterator object for the data structure implementing this interface

Idea: data structures that implement `Iterable` provide an easy mechanism to traverse all currently-stored objects

Why this is useful:

- not all data structures are easily traversed (eg. with a for-loop)
- different types (orders) of traversals may be possible
- provides an identical interface for traversing any data structure that implements `Iterable`

Example: Using a BST Iterator

```
public class BST<E,V> implements Iterable<V> {
 public Iterator<E> iterator()
   { return new BSTIterator<E,V>(); }

 private class BSTIterator<E,V> implements Iterator<E>
   {  }
}

BST<E,V> myTree;
Iterator<E> myIterator = myTree.iterator();
while (myIterator.hasNext()) {
  E nextKey = (E) myIterator.next();
  // do something with nextKey
}
```


Types of Traversals

Two main strategies, total of four variations

- each visits tree nodes in a different order

Depth-first:

- includes preorder, inorder, and postorder traversals
- visit a tree's components (root, left subtree, right subtree) in some specific order

Breadth-first:

- includes level-order traversal
- visit all nodes on the same level before going deeper in the tree

Depth-first Traversals

Preorder (parents before children):

- order of visitation: root, left subtree (recursively), right subtree (recursively)

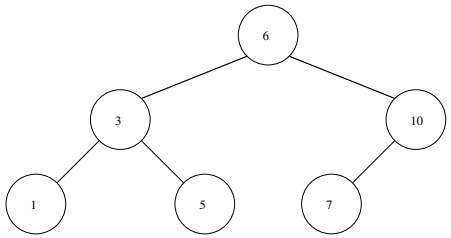
Inorder:

- order of visitation: left subtree (recursively), root, right subtree (recursively)

Postorder (children before parents):

- order of visitation: left subtree (recursively), right subtree (recursively), root

Example



Results of traversals:

- Preorder: 6,3,1,5,10,7
- Inorder: 1,3,5,6,7,10 (sorted if T is a BST)
- Postorder: 1,5,3,7,10,6
- Level order: 6,3,10,1,5,7

Recursive Inorder Traversal

```
public void printInorder() {
  printInorder(root);
}

private void printInorder(BSTnode<E,V> T) {
  if (isEmpty())  return;

  printInorder(T.left);
  System.print(T.value);
  printInorder(T.right);
}
```

Preorder and postorder traversals are analogous

A Binary Search Tree Inorder Iterator

Problem: iterator must maintain state, whereas the recursive function traverses the entire tree in one call

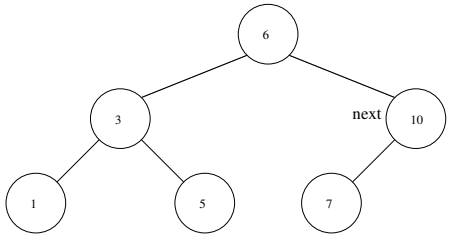
Solution: simulate recursion using a stack

- pop the stack when you have to go back up the tree

Eg. inorder traversal:

- start at root, move to left-most node and push each node traversed on the stack
- pop the stack and return this value
- begin next iteration with the right child of the returned node
- terminates when the stack is empty and the current node is null

Inorder Traversal Example



S:

Traversal: 1,3,5,6,7,10

Iterative Versions of Other Traversals

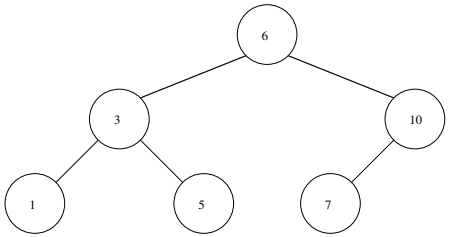
Preorder traversal:

- process current node (initially root) and push on the stack
- set current node to left child (if non-empty)
- otherwise, pop the stack and set current node to right child until current is non-null or stack is empty and current is null
- terminates when current node is null and the stack is empty

Level-order traversal (similar, but use a queue)

- enqueue the root of the tree to start
- dequeue node at the head of the queue and process it
- enqueue the left and right children, repeat
- terminates when the queue is empty

Level-order Traversal Example



Q:

Traversal: 6, 3, 10, 1, 5, 7