
CSC 143 Java

Trees

Overview

- Topics
 - Trees: Definitions and terminology
 - Binary trees
 - Tree traversals
 - Binary search trees
 - Applications of BSTs



Trees

- Most of the structures we've looked at so far are linear
 - Arrays
 - Linked lists
- There are many examples of structures that are not linear, e.g. hierarchical structures
 - Organization charts
 - Book contents (chapters, sections, paragraphs)
 - Class inheritance diagrams
- *Trees* can be used to represent hierarchical structures

Looking Ahead To An Old Goal

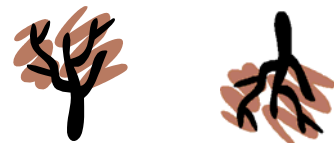
- Finding algorithms and data structures for fast searching
 - A key goal
 - Sorted arrays are faster than unsorted arrays, for searching
 - Can use binary search algorithm
 - Not so easy to keep the array in order
 - `LinkedLists` were faster than arrays (or `ArrayLists`), for insertion and removal operations
 - The extra flexibility of the "next" pointers avoided the cost of sliding
 - But... `LinkedLists` are hard to search, even if sorted
- Is there an analogue of `LinkedLists` for sorted collections??
- The answer will be... Yes: a particular type of *tree*!

Tree Definitions

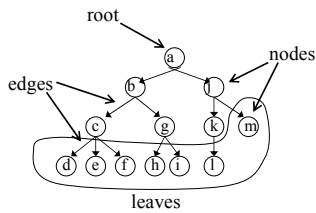
- A *tree* is a collection of *nodes* connected by *edges*
- A *node* contains
 - Data (e.g. an `Object`)
 - References (edges) to two or more *subtrees* or *children*
- Trees are hierarchical
 - A node is said to be the *parent* of its *children* (subtrees)
 - There is a single unique *root* node that has no parent
 - Nodes with no children are called *leaf nodes*
 - A tree with no nodes is said to be *empty*

Drawing Trees

- For whatever reason, computer sciences trees are normally drawn upside down: root at the top

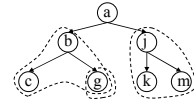


Tree Terminology



Subtrees

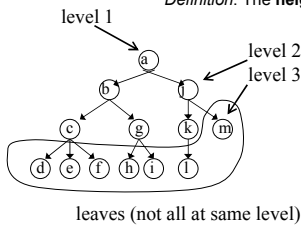
- A *subtree* in a tree is any node in the tree together with all of its descendants (its children, and their children, recursively)



- Note: note every subset is a subtree!

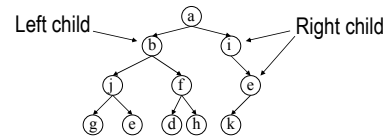
Level and Height

- Definition:** The root has **level 1**
Children have level 1 greater than their parent
- Definition:** The **height** is the highest level of a tree.



Binary Trees

- A *binary tree* is a tree each of whose nodes has no more than two children
 - The two children are called the *left child* and *right child*
 - The subtrees belonging to those children are called the *left subtree* and the *right subtree*



Binary Tree Implementation

- A node for a binary tree holds the item and references to its subtrees

```
public class BTreeNode {
    public Object item;           // data item in this node
    public BTreeNode left;        // left subtree, or null if none
    public BTreeNode right;       // right subtree, or null if none
    public BTreeNode(Object item, BTreeNode left, BTreeNode right) { ... }
}
```

- The whole tree can be represented just by a pointer to the root node, or null if the tree is empty

```
public class BinTree {
    private BTreeNode root;       // root of tree, or null if empty
    public BinTree() { this.root = null; }
    ...
}
```

Tree Algorithms

- The definition of a tree is naturally recursive:
 - A tree is either null,
 - or data + left (sub-)tree + right (sub-)tree
 - Base case(s)?
 - Recursive case(s)?
- Given a recursively defined data structure, recursion is often a very natural technique for algorithms on that data structure
 - Don't fight it!

A Typical Tree Algorithm: size()

```
public class BinTree {
    ...
    /** Return the number of items in this tree */
    public int size() {
        return subtreeSize(root);
    }
    // Return the number of nodes in the (sub-)tree with root n
    private int subtreeSize(BTNode n) {
        if (n == null) {
            return 0;
        } else {
            return 1 + subtreeSize(n.left) + subtreeSize(n.right);
        }
    }
}
```

(c) 1997-2003 University of Washington

20-13

Tree Traversal

- Functions like subtreeSize systematically "visit" each node in a tree
 - This is called a *traversal*
 - We also used this word in connection with lists
- Traversal is a common pattern in many algorithms
 - The processing done during the "visit" varies with the algorithm
- What order should nodes be visited in?
 - Many are possible
 - Three have been singled out as particularly useful for binary trees: *preorder*, *postorder*, and *inorder*

(c) 1997-2003 University of Washington

20-14

Traversals

- Preorder** traversal:
 - "Visit" the (current) node first
i.e., do what ever processing is to be done
 - Then, (recursively) do preorder traversal on its children, left to right
- Postorder** traversal:
 - First, (recursively) do postorder traversals of children, left to right
 - Visit the node itself last
- Inorder** traversal:
 - (Recursively) do inorder traversal of left child
 - Then visit the (current) node
 - Then (recursively) do inorder traversal of right child

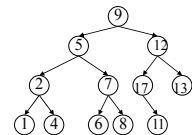
Footnote: pre- and postorder make sense for all trees; inorder only for binary trees

(c) 1997-2003 University of Washington

20-15

Example of Tree Traversal

In what order are the nodes visited, if we start the process at the root?



Preorder:

Inorder:

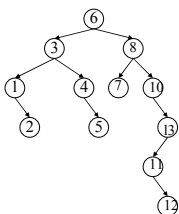
Postorder:

(c) 1997-2003 University of Washington

20-16

More Practice

What about this tree?



Preorder:

Inorder:

Postorder:

(c) 1997-2003 University of Washington

20-17

New Algorithm: contains

- Return whether or not a value is an item in the tree

```
public class BinTree {
    ...
    /** Return whether elem is in tree */
    public boolean contains(Object elem) {
        return subtreeContains(root, elem);
    }
    // Return whether elem is in (sub-)tree with root n
    private boolean subtreeContains(BTNode n, Object elem) {
        if (n == null) {
            return false;
        } else if (n.item.equals(elem)) {
            return true;
        } else {
            return subtreeContains(n.left, elem) || subtreeContains(n.right, elem);
        }
    }
}
```

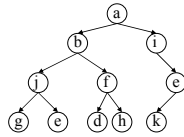
(c) 1997-2003 University of Washington

20-18

Test

contains(d)

contains(c)



Cost of *contains*

- Work done at each node:
- Number of nodes visited:
- Total cost: