Trees (Chapter 4)

Notation: n is number of nodes in Tree.

Trees: operations of $O(\log n)$ time on **average**.

In this Chapter, we will see the following:

- Worst-case $O(\log n)$ running time.
- Binary search trees.
- Trees used to implement file systems.
- Trees used to evaluate arithmetic expressions.

Skip Sections 4.3.6, 4.5, 4.7.

Tree Terminology

A tree is a directed acyclic graph with a distinguished node called its **root**.

When there is an edge from node P to node C, P is called a **parent** and C is called a **child**.

Nodes with the same parent are **siblings**.

Nodes with no children are **leaves**.

The depth of a node is the length of a unique path from the root to that node.

The length of a path from a node to itself is 0, so the depth of the root is ?

How do we implement the tree data structure? See Figures 4.3 and 4.4.

Tree Traversals

The Unix file system is organized as a tree. (See Figure 4.5)

It's natural to print or list the entire contents of a tree. To do this, we traverse the tree recursively in one of three ways:

- **pre-order:** first print the parent, and then recursively traverse the left and right children.
- **post-order:** first recursively visit the left and right children, and then print the parent.
- **in-order:** visit the left child, then the parent, then the right child.

Example:

```
     P
    / \
   /   \
  L     R
```

Binary Trees

A binary tree is a tree in which each node has at most two children. (See Figures 4.11 and 4.12)
A binary tree where each node is either a leaf or has exactly two children is called **perfect**.

What is the number of nodes in a perfect binary tree of depth D?

How does the number of leaves compare with the number of non-leaf nodes?

What is the depth of a perfect binary tree as a function of n (the number of nodes)?
What is the depth of a binary tree in the worst case?

A binary tree node is often represented as a triple: left-pointer, right-pointer, data.

Expression Trees (Application)

We can represent an arithmetic expression such as $(d + e) * (f + g)$ as a tree.

Leaves are operands, such as numbers, constants, or variables.

Internal nodes are operators, such as $+$ or $*$. (See, for example, Figure 4.14)

- How does a compiler construct an expression tree when reading an arithmetic expression?
- How do we evaluate an expression tree?

Constructing an Expression Tree

First, convert the expression from infix to postfix.

```
REPEAT

        S := read(E);
        IF operand(S) THEN
              PUSH(S, stack)
        ELSE  POP(T1, stack)
              POP(T2, stack)
              MAKE_TREE(S, T1, T2)
              PUSH(tree, stack)

UNTIL E IS EMPTY
```

Binary Search Trees

Binary trees are often used to organize a search. Each node in the tree is assigned a **key** value. We assume keys are distinct integers.

In a binary search tree rooted in node X, the values of the key in the left subtree are smaller than the value of X. And the value of the key in the right subtree are larger than the value of X. (See Fig. 4.15)

The main operations on the binary search tree ADT are: FIND(x), INSERT, and DELETE.

If an element is not found, FIND returns nil. For an example of INSERT, see Fig. 4.21. DELETE has three cases corresponding to the number of children of the deleted node. See Figs. 4.23 and 4.24.

Balanced Trees

Problem: in the worst case, a tree has depth D. What invariant can we state that insures that the depth of a tree is $O(\log D)$?

Review: the height of a node, D, is the longest path from D to a leaf. Thus, all leaves have height 0. And the height of a tree is equal to the height of a root.

An **AVL tree** is a tree where the height of a left and right subtree can differ by at most 1. (See Figure 4.29)

Which node in tree is rotated?

When is a single rotation used as opposed to a double rotation?

Can an insertion at the bottom of a large tree cause a rotation at the root?