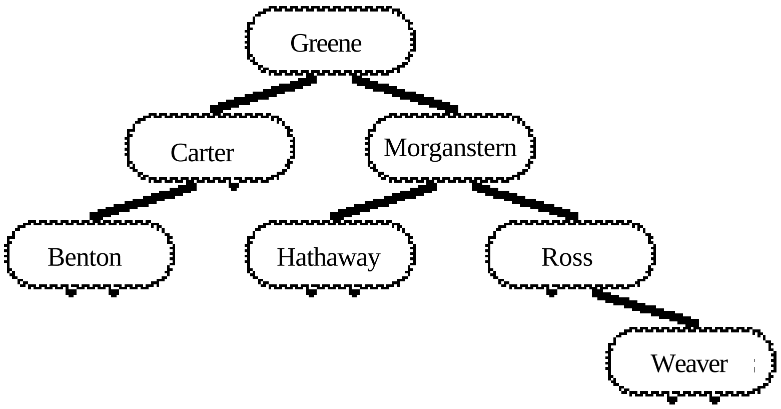
CS106B
Spring 99-00

Handout #28
May 1, 2000

Section Handout #5

Problem 1: Given the following binary search tree:



a) What is the height of this tree? What is the depth of the node 'Ross'? Who are the descendants of the node 'Morganstern'?

b) What is the sequence of nodes visited on a pre-order tree traversal? in-order? post-order?

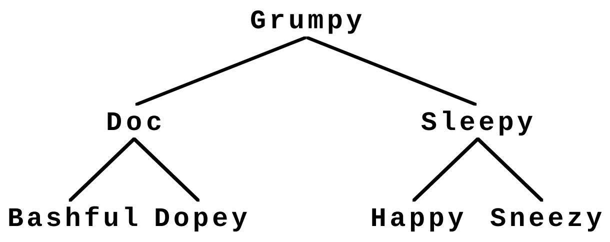
c) If we entered the node 'Del Amico', where would it end up in this tree?

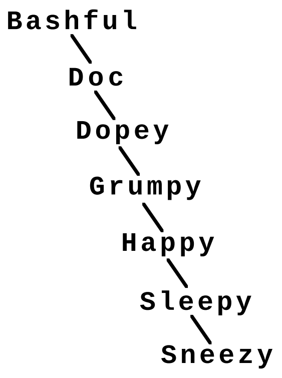
d) What was the first node inserted into this tree?

e) If the node 'Ross' is removed from the tree, will people still watch the show?

Problem 2: Binary Trees

As indicated in class, the definition of a binary search tree does not in fact guarantee that the root node falls in the middle of the complete list of keys. For example, both of the following trees are binary search trees containing the names of the seven dwarves:



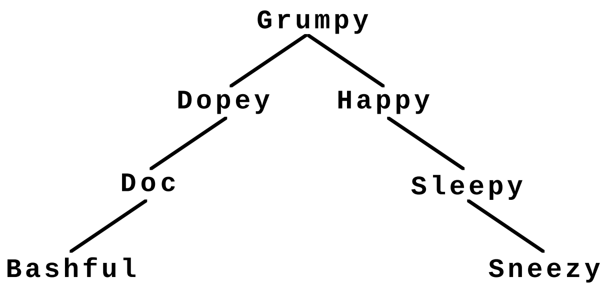


Only the tree on the left guarantees *O*(log *N*) search performance. The tree on the right is extremely unbalanced—it's essentially a linked list—and requires *O*(*N*) time to search.

A binary tree is **balanced** if each of the two following conditions is met:

1. The height of its left and right subtree differ by no more than one, where the **height** of a tree is defined to be the length of the longest path from the root to a leaf. Thus, the height of the tree appearing at the left of the example on the previous page is 3 while the height of the example at the right is 7.
2. Both of its left and right subtrees are themselves balanced.

Both conditions are important. For example, the tree



is a legal binary search tree that meets the first condition for being balanced but not the second.

Write recursive implementations of the functions

```
int TreeHeight(treeT t);

bool IsBalanced(treeT t);
```

that return the height of a tree and whether it is balanced, respectively.

In your code, assume that the following type definition provides the underlying structure for the tree:

```
typedef struct nodeT {
    string key;
    struct nodeT *before, *after;
} nodeT, *treeT;
```