Lecture Eleven

Introduction to Binary Search Trees
Recursive algorithms on BSTs: member, insert
Problems with BSTs: worst case trees

Lab: Performance of BSTs: average case trees

Next Time:
Recursive algorithms on BSTs: delete
Recursive Tree Traversals
Binary Trees

Linked lists have a single pointer to the next item in a linear sequence:

Recall that ordering a LL does not help much, so we can’t use binary search! The next data structure is an attempt to fix this problem.

Binary Trees add an additional pointer, so that the linear sequence becomes an upside-down tree where each node has 0, 1, or 2 “children”:

```java
class Node {
    int item;
    Node left;
    Node right;
}

Node root;
```
Binary Trees

Binary Trees are an inherently recursive data structure and best manipulated by recursive algorithms:

Recursive Definition:
A Binary Tree is either
- Null (empty tree); or
- A node containing data, with pointers left and right to two Binary Trees
Some basic notions:

- A **path** is a sequence of nodes connected by links; the length of the path is the number of links;
- Trees under a node are called **subtrees** of that node;
- The nodes B, C immediately under another node A are (left and right) **children** of A; B and C are **siblings** of each other; A is the **parent** of B and C (similarly: **descendant**, **ancestor**);
- The **size** of a tree is the number of nodes in it;
- The **depth** of a node in a tree is the number of links on the path from the root;
- The **height** of a tree is the maximum depth among any of its nodes (or: the length of the longest path).

Root is A  
Leaves: B, F, G, E  
Size = 7  
Depth of D = 2  
Depth of G = 3  
Height = 3  
Path from A to F in red.  
D is descendant of A  
G is ancestor of C
In order to use Binary Trees for search, we need to simulate binary search! This requires a different definition (still recursive):

A Binary Search Tree is either
- Null (empty tree); or
- A node containing a key K, with pointers left and right to two Binary Search Trees; all the keys in the left subtree are less than K, and all the keys in the right subtree are greater than K:
We will look at the following algorithms for BSTs:

Size()
Height()
Lookup/member()
Insert()

[next time]
Delete()

Traversal()

Go to:  www.cs.bu.edu/fac/snyder/cs112/CourseMaterials/BinaryTreeCode.html