CSI 3334, Data Structures

Lecture 9: More on Trees

Date: 2012-09-18
Author(s): Sam Itskin, Steven Simon
Lecturer: Fredrik Niemelä

This lecture explains how trees can be implemented and why you would implement them this way.

1 Array Implementations of Trees: Pros and Cons

The advantages of having a tree implemented in an array format is that it results in a faster search time. Along with a quicker search time, you can easily turn the array into a binary tree. The greatest advantage of having a tree in an array format is you save memory by storing fewer pointers. The disadvantages of having an array implementation of a tree is that it has a fixed size, which makes it harder to grow and, depending on the size of the array, can take a long time to grow. The main disadvantage of this implementation is the memory wasted in an unbalanced tree. With the tree being unbalanced in the array, parts of the memory that the array is stored in don't get used.

2 Another Way to Implement a Tree

As a vector was already discussed in the last lecture note this one will focus on another way to implement a tree. This will be a linked structure implementation of a tree. This type of tree contains a set of nodes with three pointers each: one for the parent, one for the first child of that node, and the third for the next sibling of that node. This way the nodes of the tree always have the exact same number of pointers and no space is wasted if the tree is not a full tree unlike what would happen to a vector or array implementation of a tree. While the memory required for the tree may be more overall, the memory per node is usually much less and much easier to calculate. If you take one node away, then you take three pointers away; if you add one node, then you add three pointers. This allows allocation and deallocation of nodes much faster as the number of pointers needed for each node is a constant, no calculation needed.

3 Speed of the BST

The binary search tree is formed where the left node is less than the parent node itself which is less than the node on the right. This allows the find function to preform in logarithmic time because every time you travel to a new node you cut out half of the tree. This allows insert to also be completed in logarithmic time because the most taxing part of insert is trying to find where the number should go. After the position is found, then the function is completed in constant time. Delete is slightly trickier as you must first find the number you wish to delete, then you must grab a node from either the greatest of the least or the least of the greatest to substitute for the node that was just deleted. depending on how balanced the tree is this could take anywhere from logarithmic to linear time.