Data Structures and Algorithms

Spring 2008-2009

Outline

Outline

Implementation

Weiss' implementation of a BST:

`declaration`

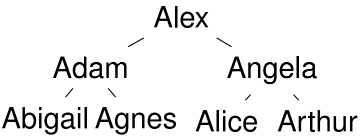
`definition`

`test harness` (always important)

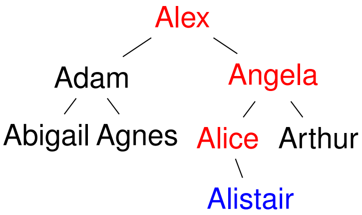
Things to watch out for:

- difference between `public` and `private` versions of each of the interface functions

Inserting a BST Node



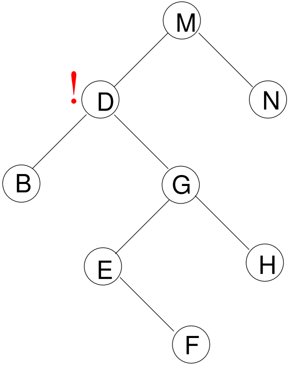
We want to insert 'Alistair' in this tree. Starting at root node we compare the 'to-insert' to node's key; the result of this comparison tells us whether to insert 'to-insert' in the left tree or the right tree.



The red nodes indicate the nodes visited and the descent into the tree.

Removing a BST Node

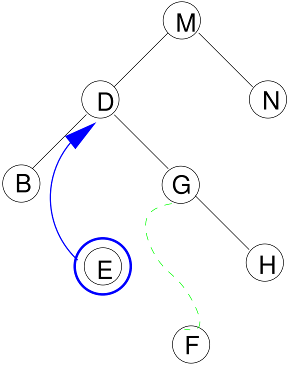
- Suppose we want to delete node **D** in the tree below



$Inorder$:BDEFGHMN

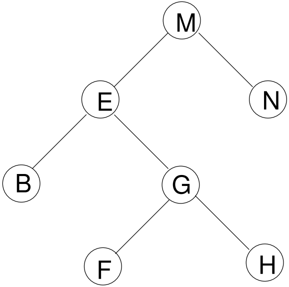
- The node that should replace it is the next largest one *lexicographically* in the tree, **E**

Removing a BST Node



- But what about **E**'s right subtree?
- To maintain the inorderedness of the BST, we need to hang all of this subtree from **E**'s parent's left

Removing a BST Node

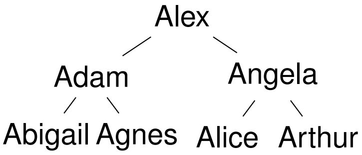


Inorder: BEFGHMN

see *internal* methods `insert()`, `remove()`

Completely Balanced BS Trees: Average-Case Analysis

- In a completely balanced binary search tree (CBBST), what is the average search time in terms of no. of probes?
- We will assume that all searches are successful since the 'time' for every unsuccessful search is $k+1$ probes (k is height of tree)
- Here's a CBBST of height 2



- If a CBBST has height k then it has levels $0, \ldots, k$ and it has $n = \sum_{i=0}^{k} 2^i = 2^{k+1} - 1$ nodes
  ($n = 2^{k+1} - 1 \Rightarrow k = \log(n+1) - 1$) (then round up if necessary)

CBBST: Average-Case Analysis (contd.)

The average time for a search in an n-node CBBST will be, S_n, the sum of the search times of each node divided by n

$$\begin{aligned} S_n &= \sum_{i=0}^{k}(i+1)2^i \\ &= 1 \times 2^0 + 2 \times 2^1 + 3 \times 2^2 + \cdots + (k+1)2^k \\ &= 1 + \sum_{i=1}^{k}(i+1)2^i \end{aligned} \tag{1}$$

CBBST: Average-Case Analysis (contd.)

Now, using the trick from Q 1.8 of *Weiss*

$$
\begin{aligned}
2S_n &= \sum_{i=0}^{k}(i+1)2^{i+1} \\
&= 1 \times 2^1 + 2 \times 2^2 + 3 \times 2^3 + \cdots + (k+1)2^{k+1} \\
&= (k+1)2^{k+1} + \sum_{i=1}^{k} i2^i \qquad (2)
\end{aligned}
$$

and subtracting equation 1 from 2 we get

CBBST: Average-Case Analysis (contd.)

$$\begin{aligned} S_n &= (k+1)2^{k+1} + \sum_{i=1}^{k} i2^i - \sum_{i=1}^{k}(i+1)2^i - 1 \\ &= (k+1)2^{k+1} - \sum_{i=0}^{k} 2^i \\ &= (k+1)2^{k+1} - (2^{k+1} - 1) \\ &= k2^{k+1} + 1 \end{aligned}$$

Then the *average* time for a successful search is

$$\begin{aligned} \frac{S_n}{n} &= \frac{k2^{k+1} + 1}{2^{k+1} - 1} \\ &\approx k, \qquad \text{for large } k \ (2^k \gg 1) \\ &\approx \log n \end{aligned}$$

BST: Asymptotic Analysis

- Interesting to note that as $n \to \infty$ it can be shown that the number of nodes at the very lowest level does *not* tend to ∞
- Even more curious, the number of nodes on the lowest level appear to oscillate in a periodic manner
- See Drmota, Michael, “On Robson’s convergence and boundedness conjectures concerning the height of binary search trees”, *Theoretical Computer Science*, Vol. 329, Issues 1 - 3, pp. 47 - 70