Algorithms in Systems Engineering
IE170

Lecture 10

Dr. Ted Ralphs
References for Today’s Lecture

• Required reading
  – CLRS Chapter 12

• References
Selection

• Recall that the selection problem is that of finding the $k^{th}$ element in an ordered list.

• Selection can be done using an algorithm similar to the quicksort algorithm from Lab 2 (notice the connection again).

• However, we need an additional data member count in the node class that tracks the size of the subtree rooted at each node.

• With this additional data member, we can recursively search for the $k^{th}$ element.
  
  – Starting at the root, if the size of the left subtree is $k - 1$, return a pointer to the root.
  – If the size of the left subtree is more than $k - 1$, recursively search for the $k^{th}$ element of the left subtree.
  – Otherwise, recursively search for the $(k - t - 1)^{th}$ element of the right subtree, where $t$ is the size of the left subtree.

• Note that maintaining the count data member can be expensive.
Rotation and Balancing

- To guard against poor performance, we would like to have a scheme for keeping the tree balanced.
- There are many schemes for automatically maintaining balance.
- We describe here a method of manually rebalancing the tree.
- The basic operation that we’ll need is that of rotation.
- Rotating the tree means changing the root from the current root to one of its children, while maintaining the BST structure.
- To change the right child of the current root into the new root.
  - Make the current root the left child of the new root.
  - Make the left child of the new root the right child of the old root.
- Note that we can make any node the root of the BST through a sequence of rotations.
Partitioning and Rebalancing

• To partition the list around the $k^{\text{th}}$ item, select the $k^{\text{th}}$ item and rotate it to the root.

• This can be implemented easily in a recursive fashion.

• The left and right subtrees form the desired partition.

• To (re)balance a BST
  – Partition around the middle node.
  – Recursively balance the left and right subtrees.

• This operation can be called periodically

• What is the running time of this operation?
Another Implementation of Delete

• Using the `partition` operation, we can implement delete in a slightly different way.
  – Partition the right subtree of the node to be deleted around its smallest element $x$.
  – Make the root of the left subtree the left child of $x$. 
Root Insertion and Joining

- Often it is useful to be able to insert a node as the root of the BST.
- This can be done easily by inserting it as usual and then rotating it to the root, i.e., partition around it.
- With root insertion, we can define a recursive method to join two BSTs.
  - Insert the root of the first tree as the root of the second.
  - Recursively join the pairs of left and right subtrees.
Randomized BSTs

• Recall that we used randomization to guard against the worst case behavior of quicksort.

• We can do the same here.

• The procedure for randomly inserting into a BST of size $n$ is as follows.
  
  – With probability $1/(n + 1)$, perform root insertion.
  – Otherwise, recursively insert into the right or left subtree, as appropriate, using the same method.

• One can prove mathematically that this is the same as randomly ordering the elements first and then inserting them as usual.

• Hence, this should guard against common worst-case inputs.