Bill Davis
605.421

1. 12.3-6 Implement TREE-DELETE to give equal priority to predecessor and successor.

   We can modify TREE-DELETE to randomly select either the TREE-SUCCESSOR or TREE-PREDECESSOR to splice. Such a function could begin:

   TREE-DELETE(T,z)
   If $left[z] = NIL$ or $right[z] = NIL$
   then $y \leftarrow z$
   else if RANDOM < 0.5
   $y \leftarrow$ TREE-SUCCESSOR(z)
   else $y \leftarrow$ TREE-PREDECESSOR

   Where random generates a number 0<RANDOM<1, and the rest of the function is the same as the original.

2. 12.4(a) Let b_n denote the number of different binary trees with n nodes. Show that $b_0 = 1$ and that , for $n \geq 1$,

   $$b_n = \sum_{i=0}^{n} b_k b_{n-1-k}$$

   We know that $b_0 = 1$ since there is only 1 tree with no nodes, i.e. $\emptyset$. Also $b_1 = 1$, since there is only 1 tree with 1 node. $b_2 = 2$ since there are two trees with 2 nodes, ie a node with either a right or left child.

   So inductively we know that if we determine b_n then b_{n+1} can be determined by putting the new node at the head of the tree and then counting the size of trees made by connecting b_n to the left child + b_{n-1} to the left child $\times$ b_1 to the right child $+b_{n-2}$ to the left child $\times$ b_2 to the right child ...

   $$b_{n+1} = b_n b_0 + b_{n-1} b_1 + b_{n-2} b_2 ... b_0 b_n$$

   This counts all of the possibly binary trees that can be generated from $n + 1$ vertices using the number only b_n.

3. 18.2-6 B-TREE-SEARCH with binary search rather then linear search.

   If we can replace the linear search in each node with a binary search, then the time takes with each node is $\mathbf{O}(\lg t)$, and the total CPU times is $\mathbf{O}(h \lg t) = \mathbf{O}(\lg t^h)$. And since, regardless of the choice of t, $t^h < \mathbf{O}(n)$ the total CPU time is $\mathbf{O}(\lg n)$

4. 18-2

   (a) Show how to maintain for every node x of a 2-3-4 tree, the height of the subtree rooted at x as a field of height[x].

   We can maintain the height of the subtree rooted at x by keeping track of when the root node splits. Since the only way to increase the height of a B-TREE is to split the root, we know that if after an insertion the root node has changes we need to set the root node to have the height of its child node + 1. On deletions, since the root is removed, all the other nodes remain correct. This doesn't change the run time of deletions, and searches, and only changes the runtime of the insertions by a constant factor.