Problem Set 5

Assigned: June 25
Due: July 2

Problem 1

(It is of course OK to use a calculator or computer for this problem.)

Suppose that you have a B-tree where B = 99 and the height is 5 (that is, there are 5 levels of nodes, including the leaves but not the root). For simplicity, assume that each leaf can also hold up to 99 data items. Assume that every item is in the leaves (the B-trees discussed in class and in Siegel).

A. What is the maximum number of data items that this tree can hold?

B. What is the height of the deepest B-tree with the same value of B that holds the number of items in (A)?

C. Suppose you start with the B-tree in part A and add an item. Necessarily, the B-tree has to grow in height. For each level L of the tree and each value of K, say how many nodes there are at level L that have K children or data items. (E.g. your answer might have the form "The root has 35 children; at level 1 there are 30 nodes with 99 children, 4 with 72 children, and 1 with 50 children; ..."). Note that the number of nodes at level L has to equal the total number of children of the nodes at level $L-1$.

Problem 2

Assume that you have an effective hash function $h(S)$ for ASCII strings; that is, the hash function h runs in time $|S|$ and does not tend to create collisions among closely related strings.

Consider trees whose nodes are labelled with alphanumeric strings (just letters and numbers and no other characters). A node in a tree can have arbitrarily many children; these are ordered left to right. Define the size $|T|$ of such a tree to be the sum of the lengths of the labels plus the number of nodes (e.g. if all the nodes are labelled with the empty string, $|T|$ is just the number of nodes.) Define $|n(T)|$ to be just the number of nodes in tree T, independent of the labels.

A. Consider the task of storing a set S of such trees in a hash table. That is, you want to support three operations: `S = Null();` `Add(T,S);` and `Element(T,S)`. The operations `Add(T,S)` and `Element(T,S)` should run in time O($|T|$). Design a hash function for labeled trees that will have this running time and that will not tend to create collisions among related trees.

B. Suppose that you additionally want to be able to support the operation `Subtree(T,S)`. This will return **true** if `T` is a subtree of any tree in `S` and **false** if it is not. Describe an implementation that will execute `Add(T,S)` in time $O(|T| \cdot n(T))$, and will execute both `Element(T,S)` and `Subtree(T,S)` in time $O(|T|)$.