Fundamental Algorithms, Spring 1998

Assignment 7. **Midterm practice.**

Given March 11, due Monday, March 30 (March 16 is an NYU holiday).

You should think of this homework assignment as a midterm exam. Sit down by yourself and try to do it in a few hours. This should give you an idea how well you are doing so far and what will be expected on the final exam. This assignment will not count more than the others.

1. We have $100,000$ credit card numbers with 16 digits each. Design an algorithm using a 10−branching trie that can determine whether a given number 16 digit number (thought of as a string of 16 digits) differs from one in the list by a single digit. Assume that the ones in the list have already been inserted into the trie. Generalize this to the case of n digit numbers and up to k differences. What is the cost for a check, as a function of n and k?

2. We want to compute the "total depth" of a tree. This is the sum of the depths (i.e. the distances from the root) of the elements. We use two routines

```
int depth( node x) {
   if ( x.parent == NIL ) return 0;
   return depth( *(x.parent) ) + 1;  \\ x.parent is a pointer.
                                     \\ *(x.parent) is the node it points to ...
                                     \\ I think.
  }

int total_depth( node x) {  \\ Make a depth first traversal of the subtree
                            \\ rooted at x, adding up the depths of the
                            \\ nodes.
   int this_depth;
   this_depth = depth( x );
   if ( x.left_child != NIL ) \\
      this_depth = this_depth + total_depth( *(x.left_child) );
   if ( x.right_child != NIL )
      this_depth = this_depth + total_depth( *(x.right_child) );
  }
```

a. What is the work for computing the total depth of a complete tree of height h?

b. What is the work for computing the total depth of a linear tree of height h with h elements in it?

c. Suppose we replace total depth by total height. The height of node x is the height of the subtree rooted at x. We compute the height and total height by routines similar to those above. What is the work for a complete tree of height h with $n = 2^{h+1} - 1$ nodes?

Some algorithms for performing M operations have the property that the work for all M is $O(M)$ even though some of the individual operations may take $O(M)$ operations themselves. For example, if the first $M - 1$ operation take 1 second but the last takes N seconds, then the total time for all the operations is $2M - 1 = O(M)$. That is, the work per operation is $O(1)$. Analysis of algorithms that attempts to understand the time per operations for a long series of operations is *amortized* analysis. The algorithm just described might be considered good in the amortized sense even though a rare operation can be very slow.

3. Consider a hash function that uses closed hashing and a "perfect" hash function and collision resolution strategy. Perfect means that the probability of a collision at a given probe is equal to the fraction of the table that is occupied (either containing an element or marked as deleted). This has table is going to hold the names of students currently enrolled at NYU and will be kept up to date for the next 10 years. As a matter of university policy there are never more than $N = 50,000$ students enrolled at any one time and the enrollment is usually usually close to that. We are going to make a table of size $3N$. If the fraction of occupied sites is more than $2/3$ (because many sites are marked deleted) a new table of size $3N$ is created and all the elements are copied to it. Show that this is efficient in the amortized sense. That is, no matter how long we run the algorithm and how many inserts, finds, and deletes we do, the work per operation is $O(1)$.

4. An $\alpha - \omega$ heap (my own terminology) is a double ended priority queue. It supports the operations `insert()`, `delete_min()`, and `delete_max()`. Again we promise that there will never be more than N elements in the system at any time. Find an implementation so that the work per operation (any of the three) is $O(\log(N))$ in the amortized sense. This implementation should use something like a heap (or pair of heaps) rather than a tree. It may do rebalancings from time to time, but these must be counted in the total work count.