CS 466/666 Fall 2004
Naomi Nishimura

Assignment 1
Due October 5 at noon

For all problems you are expected to justify your answers, by showing your work or stating arguments, as is appropriate.

The solutions you hand in must be entirely your own work. Do not look up either full or partial solutions in the literature or on the Internet.

Please read course policies and the course Web page for more information.

1. *[20 marks]* [Adapted from text exercise] Consider an ordinary binary heap data structure with n elements that supports the instructions `Insert` and `Extract-Min` in $O(\log n)$ worst-case time (that is, a heap that stores the minimum element at the root). Demonstrate a potential function Φ such that the amortized cost of `Insert` is $O(\log n)$ and the amortized cost of `Extract-Min` is $O(1)$ over a sequence of n operations on an initially empty heap, and prove that it meets these specifications.

2. *[20 marks]* Consider a perfect binary tree with n leaves, labelled ℓ_1 through ℓ_n in order from left to right. Suppose that n `Find-Set` operations are executed, namely `Find-Set`(ℓ_1), `Find-Set`(ℓ_2), ..., `Find-Set`(ℓ_n), in order. Express the time complexity of this sequence of operations in asymptotic notation, assuming that path compression is used.

3. *[20 marks]* Give a sequence of n `Make-Set`, `Union`, and `Find-Set` operations that takes $\Omega(n \log n)$ time when we use path compression but unweighted union of roots (that is, we specify which root is made the child of the other root, with no restriction on their relative ranks).

4. *[20 marks]* In this problem, we consider a variant of path compression, *path rearrangement*, where the operation `Rearrange`(v) results in v becoming the parent of every node (except v itself) on the path from v to the root. The cost of this operation is the number of edges on the path. Prove an upper bound of $O(m \log n)$ on the cost of m path rearrangement operations, starting from an n-node tree that consists of a root with $n-1$ children; use the potential function $\Phi(T) = \frac{1}{2} \sum_{x \in T} \log \mathsf{size}(x)$, where $\mathsf{size}(x)$ is defined to be the number of descendants of x, including x itself. You may find it handy to use the following inequality: for any $c > 1$, $1 + [\log(c-1)]/2 \leq \log c$.

5. *[20 marks]* In this question we consider the amortized analysis of a tree data structure that supports the operations of insertion and deletion.

   Our structure is a search tree, not necessarily binary, where data items are stored in the leaves (possibly more than one item per leaf) and key values are used in internal nodes to indicate ranges stored in each subtree. For example, a node storing the keys 3, 35, and 100 would have four children, one with descendants storing data items with key values up to 3, one for values greater than 3 and no more than 35, one for values greater than 35 but no more than 100, and one for values greater than 100.

The tree supports insertion by searching for and then modifying a leaf to store the new data item, but handles deletion by marking data items for later deletion, and then periodically rebuilding a tree of unmarked nodes only. At each node v is a stored value $\mathsf{old}(v)$ and a counter, incremented whenever an insertion occurs or a deletion is marked in the subtree rooted at that node (that is, for an insertion or a deletion we increment the counters of all nodes on the path from the altered leaf up to the root).

More precisely, if ever the counter at any node v reaches more than 1/4 of $\mathsf{old}(v)$, the entire subtree T_v is rebuilt in a more efficient manner, with marked data items omitted. A rebuilding operation resets to zero all counters of nodes in T_v and resets all values old for nodes in T_v. The cost of rebuilding is $\mathsf{all}(v)$, the total number of data items, marked or unmarked, stored in T_v. The value of $\mathsf{old}(v)$ is the number of data items stored in the subtree T_v rooted at v right after the last rebuilding of a subtree containing T_v (that is, a rebuilding due to an overflow of a counter at an ancestor of v).

Prove that in a tree storing at most n data items, the amortized cost of an insertion or deletion, not counting the time to search for the leaf into which the data item is inserted or from which the data item is deleted, is in $O(\log n)$. We assume that each of the following tasks costs one unit: inserting a data item in a leaf, marking a data item for deletion, or updating a single counter.

Many details have been omitted from this description, but they are not needed for the solution to the problem. You may assume that the maximum possible depth of a tree storing n unmarked data items is in $O(\log n)$.