CS 466/666, Fall 2006 due Friday October 20

A. Lubiw

ASSIGNMENT 2

Do not copy from others, and acknowledge your sources.

1. Consider the Union/Find problem, but with an additional operation **Remove(x)** that removes x from its current set and places it in a set by itself. Show how to modify the data structure given in class so that a sequence of m union, find, and remove operations takes $O(n \cdot \alpha(m, n))$ time.

2. (Easy) Describe how to implement a partially persistent stack with time and space $O(1)$ per operation. Update operations are Push and Pop. Query operation is Top (returns, but does not remove, the top element of the Stack). Assume that the times are discrete, i.e. the first update occurs at time $t = 1$, the second at $t = 2$, etc., and that a query specifies a time $t \in N$.

3. Using range trees we can list the points in a given query rectangle in time $O(\log^2 n + k)$ where k is the number of points. Suppose that instead of listing the points, we only need to answer how many points are in the query rectangle. Show how to modify the range tree structure to achieve $O(\log^2 n)$ time per query. Hint: think about the 1-D case first.

4. The range trees we discussed in class were *static*, i.e. we assumed all the items were given at the beginning, and we did not allow insertions or deletions. This question is about range trees with insertions. Recall that a range tree has all its leaves at almost the same level and has height $O(\log n)$. Recall that each node of the primary tree has a secondary, y-ordered search structure attached to it. For the static case we used arrays for the secondary structures, but now we will use balanced binary search trees. It still takes time $O(n \log n)$ to build a range tree on n items. The main idea is to do "lazy" insertions in the primary tree These may unbalance the tree, but after enough insertions in a subtree, we will rebuild that subtree from scratch.
   (i) Show how to insert a new item into a range tree of n items and height $O(\log n)$ in time $O(\log^2 n)$. Do not rebalance the primary tree. Give details and an analysis of your method.
   (ii) This part of the question is only about the primary tree. Let $n(v)$ be the number of descendants of node v, not counting v. After we rebuild a subtree, every node in the subtree will have half its descendants to the right and half to the left (with an extra node on one side in case $n(v)$ is odd). Associate with each node v two counters, $n\text{-}bal(v)$ and $inc(v)$, where $n\text{-}bal(v)$ is the number of descendants of v (not counting v) just after the last rebuild that affected v, and $inc(v)$ is the number of descendants added below v since the last rebuild. Prove that if we do rebuilds to maintain $inc(v) \leq \frac{1}{4} n\text{-}bal(v)$ for every node v, then the tree stays balanced in the sense that $n(v_L) \leq \frac{3}{4} n(v)$, for every node v, where v_L is the left child of v. Conclude that the height of the tree is $O(\log n)$.

   The last step in showing that this kind of lazy insertion in range trees behaves well is to show that the amortized cost of rebuilding is $O(\log^2 n)$. You don't need to do that for this assignment.