Homework 9 – Due Monday, March 26, 2012

Reminders  Collaboration is permitted, but you must write the solutions by yourself without assistance, and be ready to explain them orally to a member of the course staff if asked. You must also identify your collaborators. Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

Exercises  These should not be handed in, but the material they cover may appear on exams.

- Show how to augment a red-black search tree to maintain, for each node $x$,
  - fields $min$ and $max$, which point to the nodes with minimum and maximum keys in the subtree rooted at $x$
  - fields $pred$ and $next$ which point to the predecessor and successor nodes of $x$ in the current tree.

- Exercises from Chapter 13 and 14 of the textbook. In particular, Problem 14-1 (point of maximum overlap) and Problem 13-3 (AVL trees).

- What are the (exact) minimum and maximum numbers of nodes one can have in a red-black tree of height $h$?

- These exercises: [http://www.bowdoin.edu/~ltoma/teaching/cs231/fall07/Problems/augRBtrees.pdf](http://www.bowdoin.edu/~ltoma/teaching/cs231/fall07/Problems/augRBtrees.pdf)

Problems to be handed in. Please submit each problem on a separate sheet of paper.

1. **Spreading Gossip on Trees** (from Erickson’s lecture notes, Chapter 3, Problem 17) Suppose we need to distribute a message to all the nodes in a rooted tree. Initially, only the root node knows the message. In a single round, any node that knows the message can forward it to at most one of its children. Design an algorithm to compute the minimum number of rounds required for the message to be delivered to all nodes.

![A message being distributed through a tree in five rounds.](image)
2. **(Programming problem)** You maintain a database of financial transactions. For each transaction, you store the time and amount of the transaction (both real numbers) as well as a transaction ID.

Because the business you work for uses highly creative accounting practices, your employer wishes to be able to insert and delete transactions quickly from the database, as well as to support the following operations:

- **Insert (ID, t, a):** inserts a new transaction with id ID, time t and amount a. Should return an error if a transaction with that ID already exists.
- **Delete (ID):** deletes the transaction with id ID if one exists. Should return an error if the transaction does not exist.
- **Average (t_a, t_b):** returns the average amount of transactions that occurred between times t_a and t_b.

Your goal is to implement, in Python, a data structure supporting the operations above. The operations should all run in time $O(\log n)$, where $n$ is the number of transactions currently in the set.

In order to do this, you should implement a red-black tree and augment it appropriately.

You do not have to implement everything from scratch. To get you started, the page below links to a basic binary search tree implementation as well as several examples of augmentation: [http://6.006.scripts.mit.edu/~6.006/spring08/wiki/index.php?title=Binary_Search_Trees](http://6.006.scripts.mit.edu/~6.006/spring08/wiki/index.php?title=Binary_Search_Trees)

The basic `bst.py` package there provides, in particular, code for printing a picture of your tree in ASCII (useful for testing, and for convincing others, such as the TA, that your code does what it claims).

You should submit:

- An description of your data structure(s).
- (Readable, documented) Python code. This should contain clear instruction on how to use your program.
- Documented test results for your code. It is up to you to determine an adequate set of tests. Consider using the “print” functionality provided by the bst.py package to illustrate some tests on smaller data sets.

You will get significant partial credit for a working implementation that does not support Delete. You will also get significant partial credit for implementation that does not do rebalancing but supports all the operations in time $O(\text{height})$ where $\text{height}$ is the current height of the tree. Buggy implementations will not get significant credit.

Submissions should be electronic, via Angel.