CSE 326 Spring 1995

Data Structures

Project 4

May 10, 1995

DUE: Wednesday, May 31, 1995 at 12:01am

1 Objectives

The goals of this assignment are

- To give you experience in implementing a moderately complex data structure and benchmarking its performance, and
- To demonstrate the effectiveness of formal analysis in predicting an algorithm's behavior.

2 Project Materials

All of the materials you will need are in `/cse/courses/cse326/project4`. For this project you will be required to develop virtually all of the code rather than having the majority given to you. There will be a subdirectory of the project directory in which on-line versions of the figures from the book relating to splay trees have been placed. These can be used as a starting point for your work, but I strongly advise against copying them directly into your code because they will not be directly compatible.

3 What to Do

- *Start early!!!* This project relies less upon thinking and more upon coding than any of the other three. You have been warned.
- In this assignment you will implement the splay tree ADT and verify the performance claims that we made in class. As usual, copy the materials from the project directory. Don't bother copying the figures from the book unless you intend to use them.

  In the project directory you'll find five files. `Timer.C` and `Timer.h` are provided for your convenience, and you should not have to modify them or turn them in. You *will* need to modify `SplayTree.h`, `SplayTree.C`, and probably the makefile.
- In `SplayTree.h` you will find a skeleton class declaration for the SplayTree class. You should not modify the method declarations which are already there. However, you will need to "flesh out" the class by adding various private and protected members. You should also add to the declaration of the SplayTreeNode class as necessary.
- In `SplayTree.C` you should implement all of the methods that appear in `SplayTree.h`.

- After you have implemented the splay tree ADT, you should test its performance. To do this, you should write some code that constructs a tree of size n and then measures the length of time it takes to do n random finds upon it. Repeatedly increase the size of n (you'll have to start with n relatively large in order to get accurate timings, since the UNIX clock has a resolution of about 10 ms.) and graph the resulting curve relating n to the amount of time taken. Hopefully, you'll get something that looks like the function $n \log n$. You should do at least 5 repetitions for each value of n and average the results will help to remove "noise" from your data.

  NOTE: To figure out how to generate random numbers in UNIX, check out the manpages for `gettimeofday()` (useful for generating the random number seed), `srandom()`, and `random()`.

- Besides the $n \log n$ amortized cost for n random operations, one of the nice properties of splay trees is their locality. This property causes series of searches for objects which are "near" each other in the tree to be more efficient than series of random searches. Test this property by timing how long n finds take when each object searched for is constrained to be within some fixed distance of the target of the previous find. Plot your results on the same graph as the random finds.

- Just in case you missed it before: START EARLY!!!

4 What to Turn In

You will use the makefile to submit `SplayTree.h` and `SplayTree.C` with the turnin program. You should also turn in a hardcopy of the timing graphs described above in class on Wednesday.