CS 361 Data Structures and Algorithms I Fall 2001

AVL Trees — Do They Really Work

This last phase the term project is designed to test whether the excellent performance of AVL trees suggested by the theory is reflected in what happens in practice—while the theory cannot be wrong, there is the overhead of building the stack of pointers on the way down the tree and a potentially long walk up the tree where you have to figure out which case applies (and when you have to perform rotations they do take a bit more time). In this phase of the project you are to design an experiment that will test how well AVL trees perform when compared to some other data structure we have discussed this semester (vanilla-flavored binary search trees, heaps [since any binary search tree can be used as a heap by adding a member function DeleteMin that just deletes the leftmost leaf in the tree, returning its key and associated data], or hash tables [which provide less functionality than a binary search tree]). You can also use a data structure with similar capabilities that we haven't discussed (there are other balanced tree structures, like 2-3 trees and red-black trees, as well as something called splay trees, which have $O(\log n)$ performance in an "amortized sense," something you will learn about in CS 461 [if you are going to do splay trees, talk to me about it first]).

The simplest experiment you could perform would be to compare the running time of vanilla-flavored binary search trees and AVL trees under a regimen of random finds, inserts, and deletes. In fact, all you would have to do to accomplish this is clean up my driver by removing the calls to `Check` and `Iterate` and then time both your BST and AVL packages. Needless to say, this is not an adequate experiment—you need to do something more substantial and interesting.

Here are some suggestions, some being better than others (and also involving more work).

1. Do the experiment just described, but don't restrict yourself to the regimen of random inserts and deletes. Construct sequences of operations that aren't random, in an attempt to make the vanilla-flavored binary search trees, which have good average-case performance, but poor worst-case performance, perform badly; confirming that AVL trees do not show a performance degradation. Don't just do the extreme worst case.

2. Determine where the AVL tree algorithm is spending its time. While each each insert can do at most one rotation, a delete can do many rotations. How many are actually performed (on average)? How many times do you have to propagate up the tree?

3. How well do binary-search trees perform as priority queues when compared to heaps? While vanilla-flavored binary search trees might get unbalanced with this regimen, AVL trees always stay balanced.

4. How well do AVL trees compare to hashing? The theory says that hashing should, if the table is not close to full, wipe out AVL trees. But if the keys are strings, most of the comparisons can be resolved in one or two characters and hashing has to use the whole key to form the hash. Is enough time spent computing the hash that the advantage of hashing is dissipated?

5. How well do AVL trees (the oldest of the balanced tree structures) compare in performance to a newer type (2-3, red-black, splay).

The only ground rule for your experiment is that it has to involve AVL trees.

Note that the information I mailed back to you when I tested your programs might suggest that your implementation of AVL trees is doing excess element comparisons. This might affect the running time you report for some experiments, since for some data types key comparisons can take a relatively long time. If you are one of these people (there weren't very many) you will need to do some clean up before starting to do your experiment.

As you can see, the choices are wide open and there are lots of interesting experiments you can perform (besides those listed above). You will be graded on the thoughtfulness of your experiment and the care with which you measure whatever it is you measure. That is, you will be graded on the design of your experiment and how well you carry it out. This is very important; *a simple, not-very-much-work experiment will not receive a good grade.* Don't delay getting started, or you simply won't have enough time to do a good project.

In addition to performing the experiment, you need to provide me with a written report that describes your experiment, the results you obtained, and (especially if the results seem strange) why they are the way they are. This report should be 5–10 pages (this is only a suggestion, you might need more or [rather unlikely] fewer pages). The report will very likely contain diagrams, tables, and graphs. I won't know about the details of your experiment if you do not describe it well in your report. A poorly written report might well result in a poor grade, even though you did a careful and interesting experiment—if you don't communicate it to me, how am I to know what you did? Your report will be graded for the quality of the presentation as well as the quality of the underlying work.

Some ground rules: You can code up the other data structure you use (that is good, because both programs being tested will have the same coding style, so one won't be faster than the other just because somebody is a hotshot programmer). This is the natural thing to do for heaps or hashing, because the

data structures aren't very complex and the code for heaps is actually in the text (in Pascal). If you are going to use a more complex data structure, unless you are a hotshot programmer yourself, you won't have time to learn about it and code it and do the experiment and analyze the results in the six weeks remaining in the semester (red-black trees and splay trees are in the text; I can point you to other books for 2-3 trees). So *you can go out on the web and find a code for these data structures*, as long as you give credit where credit is due (you may need to modify the code a little to fit into the style used for our AVL assignment). You can also get the code for your non-AVL data structure from a book or research paper (though you'll have to type it in). There are some neat things out there to be found.

Due: Wednesday, December 5, 2001 at 9:00AM. Note that *late days may not be used on this phase of the project* (it said that in the policy statement). Late reports will not be accepted. You should probably have completed your experimental work by December 1 so as to give yourself enough time to write the report properly.

What to turn in. You should turn in a notebook that contains the following sections, in this order:

1. Your report. This is primarily what I will be grading.
2. Any code you wrote to do the project. This includes driver code, code for the data structure(s) you compared AVL trees to, including any code you got off the Web and modified (or didn't modify). Don't forget to say where the code came from.
3. Your completed AVL tree package. Turn in a clean listing that includes any upgrades and corrections you made after I turned it back.
4. All the old graded work for AVL and BST trees.

Note: If your AVL tree program does not successfully run through my secret driver you cannot get a passing grade for this phase of the project. I will be happy to run your BST and AVL tree programs against the secret driver at any time. Just send me mail, making sure I know where your files are and making sure that your files are accessible.