CSE 326: Data Structures
Assignment #4
April 29, 1999
Due: Monday, May 10

Splay Trees

For this assignment you will be implementing a Dictionary class using splay trees, and experimenting with it.

Splay Tree Implementation

Implement the procedures `Rotate`, `Splay`, `Lookup`, `Insert`, `Concat`, `Delete`, `FindMin` and `FindMax` as described in the handout and lecture from scratch. With the exception of `Splay`, these procedures are very short and easy, so don't be concerned about the number of procedures you have to implement. The procedures `Lookup`, `Insert` and `Delete` should all be public members of your Dictionary class. You are free to modify the interface and implementation of the private members, as long as they accomplish their task by the same algorithms given in the handout.

For the experiments you will be performing, the data type for the `Key` field will be `int` and the `Info` field will be unused (but should be available). In this case, `Lookup` should simply return a boolean that is true if and only if the key was found.

Your implementation of `Splay` should be iterative (non-recursive) and should not use an explicit stack. It will be convenient to have each tree node contain an extra field which contains a pointer to its parent in the tree.

Experiments

Your main program should consist of an empirical evaluation of the performance of your Dictionary ADT.

Generate at least three different long sequences of insertion, deletion and lookup operations. Each sequence should consist of at least 500,000 operations, starting from an empty tree, and should result in trees with at least 100,000 nodes (at maximum size). For each sequence of operations, compute the total number of rotations your implementation performs to execute all the operations, and the ratio between this number and $m \log n$, where m is the number of operations in the sequence and n is the maximum size the tree ever gets over the course of executing the sequence.

Electronic turnin

Turn in an electronic copy of your source code, along with your `Makefile`. You program must be able to be run by executing `make` and then `splay-trees`. You can do this by running the `turnin` program on `orcas` as:

```
turnin -v -c cse326 Makefile *.cpp *.h
```

or a whole directory with:

```
turnin -v -c cse326 hw4/
```

Feel free to turn in a `README` file as well. It is very important that you turn in all of your files at once. **If you forget some files the first time then turn everything in again. The `turnin` program only keeps your last turn in.** Do not turn in object files `.o` or your compiled main programs.

Paper turnin

The paper turnin should contain the following:

1. A hardcopy of all your source code.
2. A description of the sequences generated for the empirical evaluation and a brief discussion of the results.
3. A separate test program and output that demonstrates that every splay tree operation does what it is supposed to. It might be helpful to have a `Print` method that prints out everything in the tree.

Extra Credit

For extra credit, you can do the following:

- **Strongly encouraged!** Use templates in your splay tree implementation so that your Dictionary class can be used to store Keys of types other than `int`.
- Implement (in addition to the non-recursive version described above) a recursive version of `Splay` that does not use parent pointers. Instead, the recursion stack will be used to do the job of remembering the path back to the root.

  Hint: Because the type of rotation done depends on the path to P from the grandparent of P, you need some ancestral context after returning from a recursive call. Design your procedure so that when the recursive call returns, P is either at depth 1 or depth 2 from the current root. If it is at depth 1, then simply return without doing any rotation; if at depth 2, then do the appropriate two rotations before returning. If P is at depth 1, you may want your recursive call to return the direction (left or right) to P, to help the recursive invocation find P from its current root. Notice that when all the recursion ends, P may be left at depth 1 of the entire tree, so some Case I cleanup may be necessary.

Warning: It may occur to you that each recursive call could leap down 2 levels rather than 1, and then do the obvious Case II or Case III rotation when the recursive call returns. This won't work. The problem is that, if the path length to the splayed node is odd, then you will do the Case I rotation as the very first rotation rather than the very last. This results in entirely different splay behavior than the algorithm handed out, and we cannot give you any guarantee that the amortized analysis holds anymore.

- Perform a more rigorous empirical evaluation of the performance of splay trees. In particular, try to empirically explore the dynamic optimality conjecture. (Find long sequences of operations that the optimal offline strategy could perform in linear time, and verify that the splay tree algorithm also achieves linear time on these sequences. Keep in mind that any binary search tree T_1 can be transformed into another binary search tree T_2 with the same set of items in $O(n)$ single rotations, where n is the number of nodes in the tree.)