CSE 326: Data Structures
Assignment #4
November 1, 2004
due: Friday, November 12

Implement the Dictionary class using splay trees. Here are the details:

1. Implement the procedures Rotate, Splay, LookUp, Insert, Concat, and Delete as described in Algorithm 7.3 and Section 7.3. Also implement a procedure Display that displays the splay tree after each dictionary operation, so that you can watch its shape change. (The outline form as in Figure 4.8(b) will suffice if you don't want to attempt anything more picturesque.) With the exception of Splay and your possibly picturesque Display procedure, 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, Delete, and Display 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 as given in the text.

   The data type for the Key field should be `int`. There is no need to have an Info field for this assignment; just keep in mind that in a real application there would be one. Since there is no Info field, LookUp should simply return a `bool` that is `true` if and only if the key was found.

   For full credit, do not use parent pointers. Instead, design Splay recursively, so that the recursion stack will 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.)

   (Antihint: 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 in the book, and I cannot give you any guarantee that the amortized analysis holds anymore.)

   To implement Concat, you can use any convenient key from T_2 in place of $+\infty$. If T_2 is empty, it is not necessary to splay T_1 at all.

   How you display trees is up to you: make it as fancy or as plain as you like. (There will be a couple of extra credit points available for especially nice trees, but not enough that it's worth spending time on unless you're interested.) Be sure that the tree is unambiguously recoverable from your display; if a node does not have a sibling, it should be made clear whether it is a left or right child. For instance, you could print out a "$-$" to indicate an empty child if you are using the outline form.

2. Run some experiments with your Dictionary package, trying to find a sequence of operations from an initially empty tree that causes some of its operations to take $\Omega(n)$ time (even though the average time must be $O(\log n)$). What happens to the shape of the tree after a few such expensive operations? Once the tree is balanced, can you find some operations that cause it to become quite unbalanced? Include a short report on these experiments in your `README` file.