15-122: Principles of Imperative Computation, Fall 2014
Lab 9: Binary Search Trees

Tom Cortina (tcortina@cs) and Rob Simmons (rjsimmon@cs)

Monday, October 27, 2014

For this lab, you will show your TA your answers once you complete your activities. Autolab is not used for this lab. Make sure you use appropriate contracts throughout your code.

SETUP: Make a directory `lab9` in your private `15122` directory and copy the required lab files to your `lab9` directory:

```
cd private/15122
mkdir lab9
cp /afs/andrew.cmu.edu/usr9/tcortina/public/15122-f14/bst-*.c0 lab9
```

1 Counting Nodes in a Binary Search Tree (BST)

Recall that we implemented a binary search tree as a doubly-linked set of nodes, with link fields `left` and `right`. Review the code in `bst-lab.c0` to make sure you understand the representation for a binary search tree as a `bst_header` that has a pointer to the root of a binary search tree made up of nodes of type `tree`. In addition to the structures used for a binary search tree, the operations are generally implemented using a function that the client will call to start the operation, and an internal helper function that actually does the work, node by node, hiding the internal representation of the tree. Often, these tree operations are implemented recursively due to their naturally recursive nature.

Exercise 1. Complete the `bst_count` client function that has one parameter: the binary search tree `B` (of type `bst`). The function returns the total number of elements currently stored in the binary search tree. For this exercise, do not modify the internal data structure. Instead, traverse through the tree recursively, counting the nodes one by one, until you have visited all of the nodes. *HINT*: In general, the number of nodes in a binary search tree is equal to 1 + the number of nodes in its left subtree + the number of nodes in its right subtree. (What is the base case here?)

To complete this function, you will need a helper function, `tree_count` which has one parameter: `T`, a pointer to a tree node (of type `tree*`). This function will be recursive and will return the number of nodes in the tree starting at the given tree node `T`. The `bst_count` client function will then call this helper function with a pointer to the root of the binary search tree to return the total number of nodes in the tree.

To test your function, use the provided test file `bst-testcount.c0`. (Read through the file to see how it works.) You should compile and run your code as follows:

```
cc0 -d bst-client.c0 bst-lab.c0 bst-testcount.c0
./a.out
```

Once you have your code completed and tested well, show your work to your TA for review.

Exercise 2. In the previous exercise, we didn't care what elements were stored in the binary search tree. We just counted how many there were. In this exercise, you will count only those elements that are strictly less than a specific element `e` (which may or may not be in the tree). In our design, the client is required to provide an `elem_compare` function to allow us to compare two elements together to determine their relationship: less than, equal, or greater than. See the client code for an example for this lab.

Complete the `bst_countifless` client function that has two parameters: the binary search tree `B` (of type `bst`) and an element `e` (of type `elem`). The function returns the total number of elements currently stored in the binary search tree that are strictly "less than" the given element, where "less than" is defined by the client-side `elem_compare` function. You will need a recursive helper function again. *HINT*: The functions you write will be very similar to the previous exercise, except that when you count nodes, you don't always add 1 to the total number of nodes in the two subtrees. Write your own `main` function in a new file `bst-testcountifless.c0`. For comparisons, check the client code to determine which field of the element is being used as the key. Once you have your code completed and tested well, show your work to your TA for review.

2 Height of a Binary Search Tree

Exercise 3. CHALLENGE Complete the `bst_height` client function that has one parameter: the binary search tree `B`. This function returns the height of the given binary search tree. You will need a recursive helper function for this operation as well. Test your solution by writing a main function in another file that computes the heights of a number of binary search trees, including the empty tree. *HINT*: The height of a binary search tree is 0 if the tree is empty. Otherwise it is 1 + the height of its largest-height subtree.