SIENAcollege
Computer Science

Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2018

Lab 8: Search Trees

Due: Start of your next lab session

You will be assigned a partner to work with on this lab. Only one submission per group is needed.

Group members: _____

Learning goals:

1. to review and extend understanding of binary tree algorithms
2. to gain experience working with a binary search tree implementation
3. to understand the details of AVL trees and 2-3 trees
4. to see a first example of presorting as a solution technique

---

Getting Set Up

You will receive an email with the link to follow to set up your GitHub repository, which will be named `searchtrees-lab-yourgitname`, for this Lab. One member of the group should follow the link to set up the repository on GitHub, then that person should email the instructor with the other group members' GitHub usernames so they can be granted access. This will allow all members of the group to clone the repository and commit and push changes to the origin on GitHub.

Submitting

Once all written items are initialed to indicate completion, turn in one copy of this handout. Be sure names of all group members are clearly on the first page.

Your submission requires that the code deliverables are committed and pushed to the master for your repository's origin on GitHub. That's it! If you see everything you intend to submit when you visit your repository's page on GitHub, you're set.

Grading

Score: ____ / 100

Your first tasks are mostly a reminder about binary search trees as you saw them in data structures. The `BinarySearchTree` class you will find in your GitHub starter repository has a few methods implemented only as stubs that you'll need to fill in. You'll also answer a few questions along the way.

For testing, the `main` method of that class creates two binary search trees and calls the methods you will be completing to test their functionality. The smaller tree has just 5 nodes, and you can find an ASCII-art representation of it in a comment in the constructor. The larger tree has 1000 nodes with values chosen randomly, but we will all get the same tree every time because the `Random` object is seeded with a specific value (10).

Before you begin, familiarize yourself with the data structure and the provided methods.

Question 1:
The `numDistK(int k)` method is intended to return the number of nodes in the tree that are a distance of exactly `k` from the root. What values would you expect to get for the small BST for all `k` values below? (2 points)

```
small.numDistK(0) =

small.numDistK(1) =

small.numDistK(2) =

small.numDistK(3) =

small.numDistK(4) =
```

Question 2:
Complete the implementation of `recursiveNumDistK(int k, Node v)`, which is a recursive helper method used by `numDistK(int k)`. Once the answers for the small BST test cases match what you expect from the previous question, show your instructor your completed code to check if the answers are correct for the large BST. (6 points)

Question 3:
For a BST with n nodes and a given value of k, describe the worst case running time and its efficiency class, and briefly describe how you arrived at your answer. (2 points)

Question 4:

The next method you will be working with is `numWithinRange(int s, int e)`. This method returns the number of values in the BST which are in the range from `s` to `e`, inclusive. For the small BST, what answers would you expect from each call below? (2 points)

`small.numWithinRange(0, 10) =`

`small.numWithinRange(1, 4) =`

`small.numWithinRange(5, 7) =`

`small.numWithinRange(8, 10) =`

Question 5:

You will be completing the recursive helper method `recursiveNumWithinRange(int s, int e, Node v)`. Obviously one place you would look for a value in the desired range is in `v`'s data field. For a given value of `s`, `e`, and `v`'s data field, under what circumstances would you need to search in each subtree? (3 points)

Question 6:

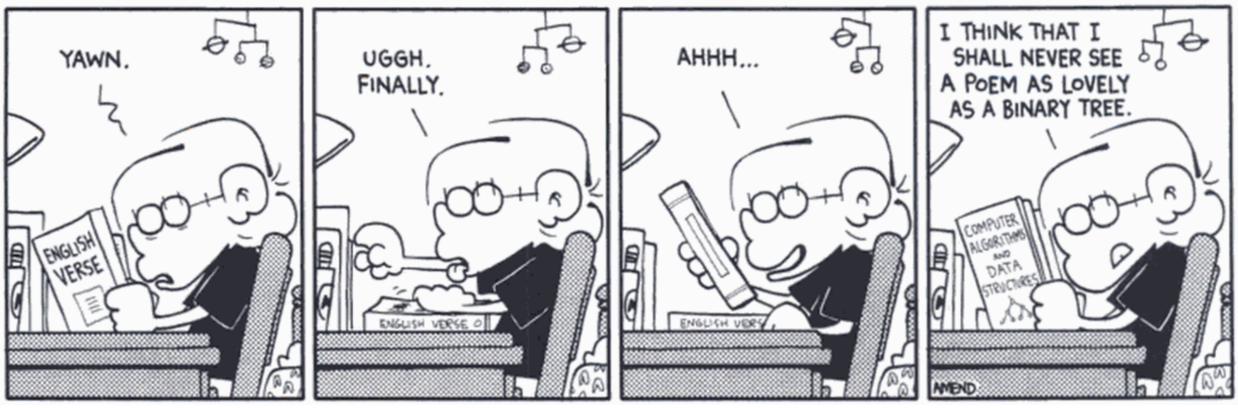
Complete the implementation of `recursiveNumWithinRange`. Once the answers for the small BST test cases match what you expect from the earlier question, show your instructor your completed code to check if the answers are correct for the large BST. For full credit, your code must be as efficient as possible, given your answer to the previous question. (6 points)

Question 7:

The method `removeLeaves` is intended to remove every leaf node from a given BST. How many nodes should be removed the first time this method is called on the small BST? The second time? The third time? The fourth time? (2 points)

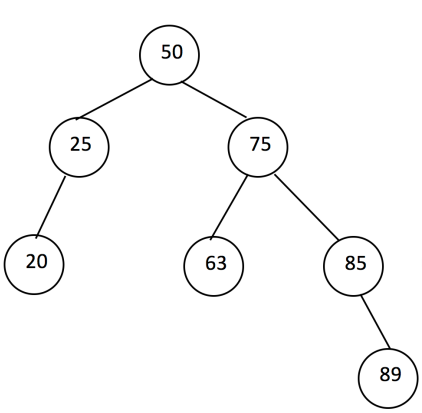
Question 8:

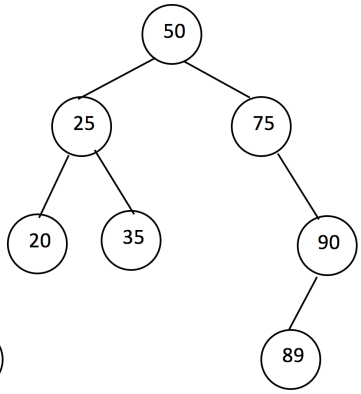
Complete the implementation of the recursive helper method `recursiveRemoveLeaves(Node v)`, which should remove all of the leaves in the subtree rooted at `v`. Once the answers for the small BST test cases match what you expect from the earlier question, show your instructor your completed code to check if the answers are correct for the large BST. (6 points)

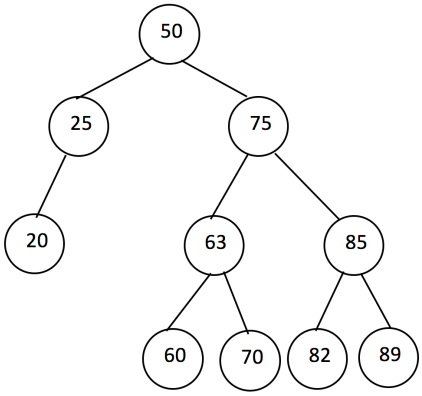


Now, on to balanced search trees. First, a little review of AVL trees.

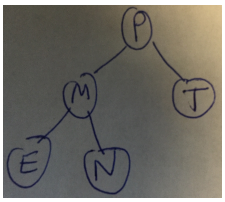
Question 9:
For the binary trees below, indicate which are valid AVL trees. For ones that are not, place a star by each node at which the AVL condition is violated. (2 points)



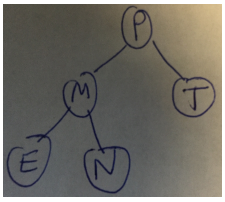




Question 10:
Insert the value `O` into the given AVL Tree. Show all steps. (2 points)



Question 11:
Insert the value `B` into the (same) given AVL Tree. Show all steps. (2 points)



For the next several questions, you will be working with three possible search tree approaches: a straightforward BST, an AVL tree, and a 2-3 tree. Please note that for questions that as you to construct a tree, you should count the number of comparisons being made as values are being inserted.

Question 12:
Insert the values 1, 2, 3, ..., 10 into a straightforward BST. (5 points)

Question 13:
How many comparisons were needed during the entire tree construction? (2 points)

Question 14:

Insert the values 1, 2, 3, ..., 10 into an AVL tree. Show the steps needed and label each necessary rotation as either a single rotation or a double rotation. (8 points)

Question 15:

How many comparisons were needed during the entire tree construction? (2 points)

Question 16:

Insert the values 1, 2, 3, ..., 10 into a 2-3 tree. Show the steps needed, including places where a "4 node" is temporarily created. (8 points)

Question 17:

How many comparisons were needed during the entire tree construction? (2 points)

Now suppose you were inserting the numbers 1, 2, 3, ..., 1000, into search trees.

Question 18:
Describe the shape of the straightforward BST that would result and the number of comparisons that would be made during its construction. (4 points)

Question 19:
Describe the shape of the AVL tree that would result and estimate the number of comparisons that would be made during its construction. (4 points)

Question 20:
Describe the shape of the 2-3 tree that would result and estimate the number of comparisons that would be made during its construction. (4 points)

Suppose we were first to shuffle the numbers 1-1000 before adding them instead of adding them in ascending order.

Question 21:
How do you expect that would affect the properties of the resulting straightforward BST? (4 points)

Question 22:
How do you expect that would affect the properties of the resulting AVL tree? (4 points)

Question 23:
How do you expect that would affect the properties of the resulting 2-3 tree? (4 points)

Question 24:

Under what circumstances would you recommend such a "pre-shuffle" step for the construction of these kinds of trees? (4 points)

Question 25:

Suppose you want to determine if all the elements in an array are unique. For example, the elements in array `A = [6, 9, 2, 1, 0, 3, 10, 9, 5, 4]` are not all unique because the number 9 appears more than once. You could easily solve this problem with brute force in $\Theta(n^2)$ time. Come up with an asymptotically more efficient algorithm that solves this problem. Write pseudocode for your solution below. (10 points)

ALGORITHM UNIQUEELEMENTS(A)
//Input: an array $A[0..n-1]$
//Output: true if all values are unique, false if any duplicates