CS 6301.502. Implementation of advanced data structures and algorithms

Fall 2017

Short Project 7: Binary search trees

Wed, Oct 4, 2017

Version 1.0: Initial description (Wed, Oct 4).

Due: 11:59 PM, Sun, Oct 22.

Solve as many problems as you wish. Maximum score: 100 (at most 50 from Q6-Q8).

Up to 5 EC for the best 15 projects. Try to write high-quality code,

following OOP techniques, coding standards, and good design of code that

allows maximum code reuse in inherited classes from base class BST.

Starter code is provided for Q1-Q5. Do not add parent link to Entry class.

Q6-Q8 can be solved with Java's TreeMap/TreeSet or using your own BST class.

Do not change the signatures of the public methods of the classes provided.

Place the sources of classes BST, BSTMap, AVLTree, RedBlackTree, and SplayTree

in cs6301/gXX, where gXX is your group name. Do not place them in subfolders

or rename the classes.

1. [50 points, 1EC]

Implement binary search trees (BST), with the following public methods:

contains, add, remove, iterator. Concentrate on elegance of code

(especially, iterator), and reusability of code in extended BST classes

(AVL, Red Black, and, Splay Trees).

2. [50 points, 1EC]

Extend BST to AVL trees (AVLTree).

3. [50 points, 1EC]

Extend BST to Red Black Trees (RedBlacktree). Important: elegance of code,

code reuse from BST, and implementation of single pass algorithms.

4. [50 points, 1EC]

Extend BST to Splay Trees (SplayTree). Implement bottom-up splaying.

Include in your submission, code that generates a random sequence of

add/remove/contains operations, with skewed distributions, that can be

used as input to the Splay tree driver. Experiment and find probability

distributions for which the splay tree is faster than BST or Java's TreeMap

for a sequence of n operations by at least 10%, for large n (say, 10M+).

The probability distribution need not be uniform, but every element should

have a non-zero probability of being chosen for contains/remove.

5. [50 points, 1EC]

Implement BSTMap (like a TreeMap), on top of one of the BST classes.

6. [30 points]

Given an array A of integers, and an integer X, find how many pairs of

elements of A sum to X:

static int howMany(int[] A, int X) { // RT = O(nlogn).

// How many indexes i,j (with i != j) are there with A[i] + A[j] = X?

// A is not sorted, and may contain duplicate elements

// If A = {3,3,4,5,3,5} then howMany(A,8) returns 6

}

7. [30 points]

Given an array A, return an array B that has those elements of A that

occur exactly once, in the same order in which they appear in A:

static<T extends Comparable<? super T>> T[] exactlyOnce(T[] A) { // RT = O(nlogn).

// Ex: A = {6,3,4,5,3,5}. exactlyOnce(A) returns {6,4}

}

8. [30 points]

Given an array A of integers, find the length of a longest streak of

consecutive integers that occur in A (not necessarily contiguously):

static int longestStreak(int[] A) { // RT = O(nlogn).

// Ex: A = {1,7,9,4,1,7,4,8,7,1}. longestStreak(A) return 3,

// corresponding to the streak {7,8,9} of consecutive integers

// that occur somewhere in A.

}