April 9 2022 11:30 PM

CS 2223 D22 Term. Homework 3

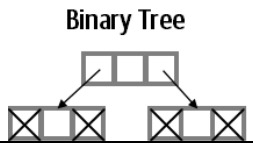
This homework covers material that extends back to before the midterm.

Homework Instructions

- This homework is to be completed individually. If you have any questions as to what constitutes improper behavior, review the examples as I have posted online http://web.cs.wpi.edu/~heineman/html/teaching_/cs2223/d22/#policies.
- Due Date for this assignment is **6PM April 19th**. Homeworks received after 6PM will receive a 25% penalty if submitted within 48 hours, otherwise zero points.
- Submit your assignments electronically using the canvas site for CS2223. Submit your homework under "HW3". You must submit a single ZIP file that contains all of your code as well as the written answers to the assignment.
- All of your Java classes must be defined in a packager USERID.hw3 where USERID is your CCC user id (i.e., your email address without the @wpi.edu).

Homework Context

This homework introduces students to the Binary Tree data structure. When used as the basis for Binary Search Trees, this structure offers the ability to dynamically insert and remove values, while supporting efficient search and traversals.

Recursive Data Structure	

April 9 2022 11:30 PM

Q1. Evaluating Sorting Algorithms (TBD pts)

When evaluating sorting algorithms, we have so far only been concerned with the number of times two elements are compared with each other. It's time to do a deeper dive into the number of times two values are exchanged. After all, with comparison-based sorting, we would like to minimize the number of times two values are exchanged.

An Observation

The array [1, 2, 3, 4, 5, 0] contains SIX values, and requires five pair-wise exchanges to sort in ascending order. Try it out yourself. So you can always construct an array of size N that requires N-1 exchanges as the worst case. Given a random array of integers, how should we proceed?

I have created an `algs.hw3.sort.Exchangeable` class that stores an Integer value and which can be used to record the number of times an object was exchanged during a sort. **Note that for this to happen, the `static void exch(Exchangeable[] a, int i, int j)` method has been instrumented to increment the exchange count for a[i] and a[j]** for four different sorting algorithms (INSERTION SORT, SELECTION SORT, QUICKSORT, HEAP SORT and MERGE SORT). I have done this for you.

Q1.1 Trial Number One: Run 100 Trials for each size N

For N in the range from 128 to 16,384, run 100 trials of each algorithm sorting a different random array of `Exchangeable` objects (using the `Exchangeable.create(N)` method). You know you are doing this right when you see that for each problem N, you ultimately call `Exchangeable.create` 500 times because there are FIVE sorting algorithms and ONE HUNDRED trials.

- For each trial, inspect the `Exchangeable` objects themselves to count the total number of times all objects were exchanged. Note that if you simply accumulate the total increment counts of all `Exchangeable` objects in the array, the resulting total is TWICE the number of times the `exch()` method was called (do you see why?), so be sure to cut it in half.
- For these 100 trials of sorting arrays of length N, record the largest number of `exch()` invocations you recorded, which will reflect "the random trial that required the most number of exchanges."

Your output should look like the following (although because these are random trial, some of the numbers in the columns may be different). You should order the columns from left to right in ascending order BASED ON THE FINAL ROW OF 16384. Note that for one of the algorithms, the total number of exchanges becomes unmanageable, so for that algorithm (you will eventually see which one) do not run it for values of N ≥ 4096, but instead output "*" for its number of exchanges.

N	Sort1	Sort2	Sort3	Sort4	Sort5
128	128	364	807	896	4647
256	256	785	1851	2048	17736
512	512	1663	4202	4608	69183
1024	1024	3557	9370	10240	271760
2048	2048	7556	20797	22528	1097560
4096	4096	16012	45611	49152	*
8192	8192	33949	99340	106496	*
16384	16384	71571	215011	229376	*

Deleted: 807
Formatted: Highlight
Deleted: 2048
Deleted: 1851
Deleted: 4608
Deleted: 4202
Deleted: 10240
Deleted: 9370
Deleted: 22528
Deleted: 20797

April 9 2022 11:30 PM

Naturally, you need to replace the column headings with the corresponding label of “Heap”, “Insert”, “Merge”, “Quick”, “Select”.

Note in my original code, I had defect that swapped two columns by mistake. The above is now accurate (again, within the noise that you will get with randomly generated runs).

Q1.2 Classifying Number of Exchanges

For each of these algorithms, you must classify the # of exchanges using the Big O notation, to capture the order of growth.

April 9 2022 11:30 PM

Q2. Working with Binary Search Trees (TBD pts).

There are three kinds of methods you can envision for Binary Search Trees:

- Structural – just inspects .left and .right references, like computing the height of a tree
- Read Only – traverses a tree by inspecting the keys but makes no changes to the structure. Like the `get()` method.
- Modifying – like the `put()` method.

Your task is to copy the **`algs.hw3.BST`** class (and **`TestBST`** class) into your **`USERID.hw3`** package and complete the methods at the end of the class.

Note that all references below to BST refer to `USERID.hw3.BST`.

- Delete the key from the BST that is the maximum key from the BST
- Locate the key whose count is largest among all keys in the BST
- Return a copy of the BST

Once you are done, you can execute the `TestBST` class and if it doesn't throw any exceptions, you have successfully implemented BST.

Q2.1. BONUS QUESTION (1 pt.)

Create in your project a package `USERID.hw3.bonus` and copy the classes from `algs.hw3.bonus` into your package (this includes `algs.hw3.bonus.BST` and `algs.hw3.bonus.SimpleRangeList`). Yes, RangeList is BACK!

For this bonus question, you are to complete the implementation of `SimpleRangeList` and the BST to provide the `public SimpleRangeList ranges()` method, which returns a `SimpleRangeList`.

Formatted: Font: Consolas
Formatted: Font: Consolas
Formatted: Font: Consolas
Formatted: Font: Consolas
Formatted: Font: Consolas
Formatted: Font: Consolas
Formatted: Font: Consolas

April 9 2022 11:30 PM

Q3. Benfords Law (TBD pts).

Benford's law, also known as the Newcomb–Benford law, the law of anomalous numbers, or the first-digit law, is an observation that in many real-life sets of numerical data, the leading digit is likely to be small.

Examining a list of the heights of the 62 tallest structures in the world by category shows that 1 is by far the most common leading digit, irrespective of the unit of measurement (whether in feet or meters):

I have created a `algs.hw3.Table` class that contains the heights of 62 tallest structures in the world by category. Process the data from these building heights to represent the table (by leading digit)

Reproduce the **values** in the table shown below (including the percentages for actual (meters and feet) and predicted (per Benford's law). Of course, your output will be in plain text, but you can see what I am looking for.

Leading digit	m		ft		Per Benford's law
	Count	%	Count	%	
1	24	41.4 %	16	27.6 %	30.1 %
2	9	15.5 %	8	13.8 %	17.6 %
3	7	12.1 %	5	8.6 %	12.5 %
4	6	10.3 %	7	12.1 %	9.7 %
5	1	1.7 %	10	17.2 %	7.9 %
6	5	8.6 %	4	6.9 %	6.7 %
7	1	1.7 %	2	3.4 %	5.8 %
8	4	6.9 %	5	8.6 %	5.1 %
9	1	1.7 %	1	1.7 %	4.6 %

Q3.1. BONUS QUESTION (1 pt.)

The original Wikipedia entry in Benford's law refers to "58 tallest structures by category", but the revised table has 62 tallest structure categories. (a) what are the four newest categories that were added? (b) did any category change its height, based on the stats from the Wikipedia table?

April 9 2022 11:30 PM

Q4. Using a BST as a Symbol Table (TBD pts).

Compare the efficiency of using a BST as a Symbol Table. I have created a stripped-down Binary Search Tree, which you should use "as is" without copying/modifying. Find this class as `algs.hw3.BST_SymbolTable`.

Compare the performance of using this class as a symbol table against the `edu.princeton.cs.algs4.SeparateChainingHashST` and `edu.princeton.cs.algs4.LinearProbingHashST` implementations, provided by Sedgewick.

For each value of N from 16384 to 1048576, complete 10 different trials. In each trial, you will generate N/2 random integers using `StdRandom.uniform(N*8)`. For each of these N/2 random numbers, check if the number is contained in the Symbol table – if not, put the number there with a count of 1, otherwise increment the count that is associated with it. The essential idea is that you are trying to count the number of times that you see each of the random numbers. You will need to use **`StopwatchCPU`** to time the total time it takes to complete all ten trials (I run ten trials just to be sure I have something to measure – computers are just so fast these days!). As you can see, for each value N, you will perform a get() and a put() invocation, thus the total number of operations will be 2 * 10 * N/2 = 10*N.

Deleted: 0

Once done, compute the AVERAGE cost of a put() or get() operation, by dividing the total running time by 10*N, and output the result.

Q4.1 Generate Output Table

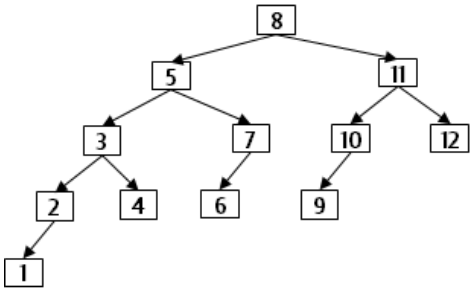
N	Avg.BST	Avg.List	Avg.OA
16384	#	#	#
32768	#	#	#
65536	#	#	#
131072	#	#	#
262144	#	#	#
524288	#	#	#
1048576	#	#	#

Q4.2 What is O() classification for the average operation for each of these implementations?

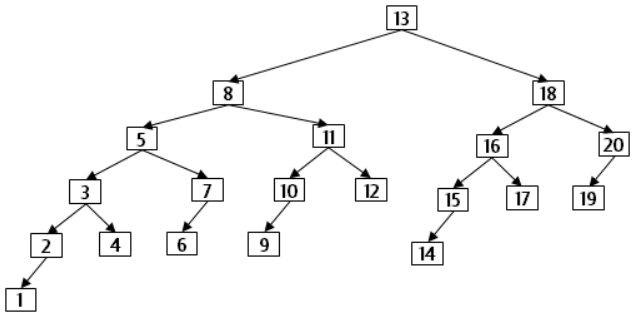
April 9 2022 11:30 PM

Q5. Bonus Questions (1 pt)

A complete binary tree with $N=2^k - 1$ nodes is the most compact representation for storing N nodes. This bonus question asks what is the "least compact" AVL trees you can construct. A **Fibonacci tree** is an AVL tree such that in every node, the height of its left subtree is bigger (by just 1) than the height of its right subtree. Think of this as an AVL tree that would need to perform the most number of rotations when deleting its largest value. Complete the implementation of `FibonacciTree(N)` that returns a Fibonacci AVL tree whose root value is the N^{th} Fibonacci number. For example, `FibonacciTree(7)` returns a simple `FibonacciTree` object (composed of `algs.hw3.Node` objects) as depicted below:



To confirm you have the right structure, implement a postorder traversal of this tree and output the values in the traversal, as 1 2 4 3 6 7 5 9 10 12 11 8. The following is the next-larger Fibonacci Tree:



April 9 2022 11:30 PM

Change Log

1. Clarification that Q4 generates random numbers and then counts the number of times it sees each of the numbers between 0 and N*8.
2. Added Bonus Question Q2.1
3. Table for sorting had some switched columns once I started printing "*" for the final three rows.

Formatted: List Paragraph, Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Indent at: 0.5"