April 22 2021 3:05 PM

CS 2223 D21 Term. Homework 3

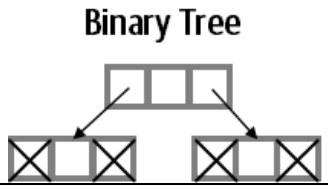
This homework covers material that extends back to before the midterm.

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/d21/#policies.
- Due Date for this assignment is **6PM April 27th**. Homeworks received after 6PM will receive zero credit.
- 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	

Q1 Evaluating Sorting Algorithms (**34 pts**) ← Rubric posted to show breakdown of points
Q2 Working with BSTs (**26 pts**)
Q3 Shakespeare (**20 pts**)
Q3.1 Bonus (1pt)
Q3.2 Bonus (2pt)
Q4 AVL Trees (**20 pts**)
Q4.1 Bonus (1pt)
Q5 Fibonacci Tree BONUS (1pt)

April 22 2021 3:05 PM

Q1. Evaluating Sorting Algorithms (34 pts)

When evaluating sorting algorithms, we have so far only been concerned with runtime performance. There is another consideration: Is the sorting algorithm *stable*? A stable sorting algorithm is concerned with the relative placement of equal values in the original array being sorted. It is easiest to explain visually. Assume the following five values are to be sorted. As you can see, the number 7 appears twice. The notation 7_{left} refers to the 7 on the left side, while 7_{right} refers to the right one.

[2, 5, 7_{left}, 3, 7_{right}, 4] → [2_0, 5_1, 7_2, 3, 7_4, 4_5]

When the array is sorted, what happens to the relative positioning of these equal values? In the original unordered array, 7_{left} was to the left of 7_{right}.

- In a stable sort, the sorted array must be [2, 3, 4, 5, 7_{left}, 7_{right}] where the values maintain their relative positions with regards to each other
- An unstable sorting algorithm makes no such guarantee, and could produce [2, 3, 4, 5, 7_{right}, 7_{left}]

I have created an `algs.hw3.CountedItem<E>` class that records a counter with each `CountedItem()` object that is created. This class can be used to determine unstable sorts.

```
CountedItem<Integer> one = new CountedItem(7);
CountedItem<Integer> two = new CountedItem(7);
System.out.println(one.equals(two));            // <-- will be TRUE
System.out.println(one.earlier(two));           // <-- will be TRUE
System.out.println(two.earlier(one));           // <-- will be FALSE
```

You only need to use `CountedItem<E>` objects for Q1.1.

Q1.1 Trial Number One: Verify which sorting algorithms are stable

I provide you with a `trial1_1()` method that creates an array of 4,096 random integers using `StdRandom.uniform(128)`. This array contains a good number of duplicate values and provides a template to evaluate each of five sorting techniques (**Heap Sort**, **Insertion Sort**, **Selection Sort,** **Merge Sort**, **Quick Sort**, and **Tim Sort**, both Primitive and Optimized). From this random array, create **seven** different `CountedItem<Integer>[]` arrays, using the `toCountedArray()` method, and evaluate the result using the `isSortedArrayStable()` method which you must complete.

Your output should look like this, where "XXXX" is replaced by True or False based on your analysis.

```
Algorithm           Stable Sort
HeapSort:           XXXX
InsertionSort:      XXXX
MergeSort:          XXXX
QuickSort:          XXXX
SelectionSort:      XXXX
TimSort Primitive:  XXXX        ← algs.hw3.PrimitiveTimSort
TimSort Optimized:  XXXX        ← algs.days.day16.ComparableTimSort
```

April 22 2021 3:05 PM

Q1.2 Trials Number Two: sorting with low likelihood of duplicate values

Over the range of N from 1,048,576 to 16,777,216, create an unordered array, A, of N random integers using `StdRandom.uniform(N)`. This array provides the template for launching a trial for each of four efficient sorting techniques (**Merge Sort**, **Heap Sort**, **Quick Sort**, and **Tim Sort**, both Primitive and Optimized). Copy **`algs.hw3.submission.Q1`** into your **`USERID.hw3`** package.

Report the time of each sorting algorithm as N doubles, using the provided code in Q1. Your output should look like the following:

```
      N         XXXX    XXXX    XXXX    XXXX    XXXX
 1048576        0.234   0.266   0.266   0.250   0.500
 2097152        0.531   0.469   0.516   0.578   1.031
 4194304        1.094   0.969   1.234   1.406   2.641
 8388608        2.516   2.219   2.719   3.297   6.297
16777216        5.859   5.219   6.188   7.625   14.984
```

Be sure to label your output columns to declare which algorithm is presented. **The columns should be organized from left to right in terms of most efficient to least efficient** in the final row.

Use the following labels for the columns:

- Heap
- Merge
- PrimTS ← `algs.hw3.PrimitiveTimSort`
- Quick
- TimSort ← `algs.days.day16.ComparableTimSort`

Q1.3 Trials Number Three: sorting with high likelihood of duplicate values

Over the range of N from 1,048,576 to 16,777,216, create an unordered array, A, of N random integers using `StdRandom.uniform(N/512)` which increases the likelihood of duplicate values occurring. This array provides the template for launching a trial for each of four efficient sorting techniques (**Merge Sort**, **Heap Sort**, **Quick Sort**, and **Tim Sort**, both Primitive and Optimized).

```
    N           XXXX    XXXX    XXXX    XXXX    XXXX
 1048576        0.203   0.250   0.219   0.234   0.375
 2097152        0.438   0.297   0.453   0.500   0.906
 4194304        0.938   0.641   1.141   1.281   2.422
 8388608        2.141   1.391   2.922   3.125   5.672
16777216        5.063   3.203   5.625   7.156   13.391
```

Report the time for each algorithm and problem instance size. Keep the columns in the same order as for Q1.2 to see if there is a change (and there should be).

Q1.4 Trials Number Four: Sort Timing with reversed order input

Over the range of N from 1,048,576 to 16,777,216, create an array, A, of N integers in reverse order, which typically is the **worst case** input. This array provides the template for launching a trial for each of four efficient sorting techniques (**Merge Sort**, **Heap Sort**, **Quick Sort**, and **Tim Sort**, both Primitive and Optimized). Provide the table output. Are you surprised at the results?

April 22 2021 3:05 PM

Q2. Working with Binary Search Trees (26 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.submission.BST`** class (and **`TestBST`** class) into your **`USERID.hw3`** package and complete the methods at the end of the class:

- Make an exact copy of a BST
- Return the number of nodes in the BST that exist at a given depth (where the root has a depth of 0, and its children have a depth of 1, and so on…)
- Truncate the BST at a given depth, d, so all nodes with depth d+1 and greater are removed from the tree. Be sure to update the count, N, associated with each node, since that will change.
- Return the **`String`** that has the greatest associated integer count in 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.

April 22 2021 3:05 PM

Q3. Did William Shakespeare write all of his 38 plays? (20 pts).

This is a question that has plagued amateur sleuths for centuries. Can you do some statistical analysis to find any clues?

In particular, you are to find MOSTCOMMON, the most common word in all of the Shakespeare plays, using the `BST<String,Integer>` data structure.

Once you have MOSTCOMMON, your next task is to find the top five most frequently used words in each of the 38 plays individually (do this by constructing 38 different BST trees, one at a time, to find the five most frequently used words in each play).

Hint: Once you construct a BST, you can delete nodes at will if that will help you with your computations.

For example, given the play "All's Well that ends Well" (play 1.txt) the most common five words are (in order):

i	the	and	to	you

Go through all 38 plays and (a) print out the title of the play that does not include MOSTCOMMON as one of the top five words in the play; and (b) print out the top five words of that play.

It might be helpful to simply print out the title for each play, together with its five most common words. Then after this table, you can print out which one is the play that I am looking for…. The first two lines of this output would look like this:

```
i       the     and     to      you     All's Well That Ends Well
and     the     i       to      you     A Midsummer Night's Dream
…
```

You will find the **`algs.hw3.ShakespearePlay`** class quite useful. Here is a small snippet:

```
public static void main(String[] args) throws IOException {
  ShakespearePlay sp = new ShakespearePlay(7);
  System.out.println(sp.getTitle() + " has " + sp.size() + " words.");
  for (String s : sp) {
    if (s.equals("weasel")) {
      System.out.println(s);
    }
  }
}
```

The above code tells you that the **Hamlet** uses the word "weasel" twice. Who knew?

Q3.1. BONUS QUESTION (1 pt.) Shakespeare Longissimum

What is the longest non-hyphenated word that Shakespeare used in any of his plays? And what does it mean? Make sure you filter out those hyphenated words like "`tragical-comical-historical-pastoral`" which appears in Hamlet.

April 22 2021 3:05 PM

Q3.2 BONUS QUESTION (1 pt.) Shakespeare Numerorum Mysteria

How many times does Shakespeare use five consecutive words of the length 3, 1, 4, 1, 5 in his plays? In fact, there are other patterns worth exploring:

- Sqrt(2) – 1, 4, 1, 4, 2, 1
- Increasing – 1, 2, 3, 4, 5, 6
- e – 2, 7, 1, 8, 2, 8, 1

Write a method that takes in an array of values to search for, and outputs the words found (and the titled of these plays).

April 22 2021 3:05 PM

Q4. AVL Binary Search Trees (20 pts).

AVL trees are self-balancing to maintain their efficiency. This question asks you to generate 10,000 random AVL trees of size N (**for N from 1 to 40**) and record the greatest height in any of these trees. Because AVL trees are effective at compactly storing values, this question is really testing the limits of this efficiency. The following table contains the first five rows that your program should output:

N	Largest Height	Number Found
1	0	10000
2	1	10000
4	2	10000
7	3	5656 *(* yours might vary *)*
12	4	1952 *(* yours might vary *)*

This table shows that after generating 10,000 AVL trees by inserting 4 random values into an empty AVL tree, each of these trees has a height of 2. There is no entry for the 10,000 AVL trees generated for 5 or 6 random values, since none of these random AVL trees had a height greater than 2. In fact, you need 7 values before some AVL has a height of 3 (how many you ask? Well it appears to be a little more than half of the random trees).

When you complete this assignment, the first two columns should be identical to the table above, and your output will have two additional rows.

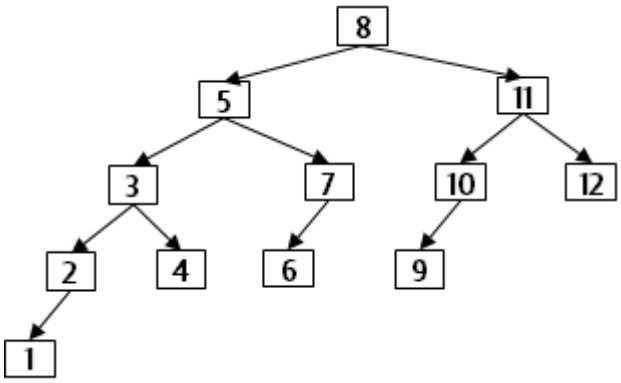
Q4.1. (1 pt.) BONUS question

What is the pattern of the N values in the first column? What are the next five values that will print?

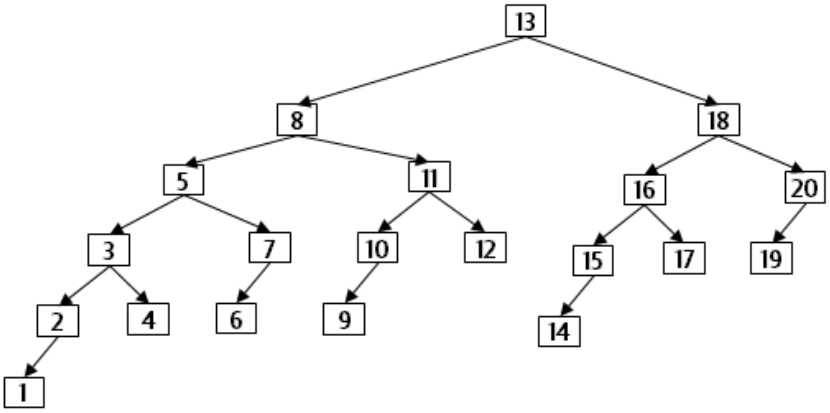
April 22 2021 3:05 PM

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 22 2021 3:05 PM

Change Log

1. **Selection Sort is needed for Q1.1 – this means SEVEN entries to print out.**
2. **Updated points for questions to align with rubric (which has been posted).**
3. **Clarified in Q3 that the questions Q3.1 and Q3.2 are bonus questions only.**