Great Oaks From Little Acorns Grow

Robin Dawes

October 4, 2021

You will implement a Binary Search Tree class, build a lot of trees, and analyse their height.

.

Define a Binary Search Tree Class

In the language of your choice create a complete implementation of a binary search tree class for storing sets of integers. Your class's interface must provide five functions:

as long as it's one of Python, Java, C or C++

- a constructor that returns a reference to a new, empty Binary Search Tree
- an instance function **insert**(**x**) that adds integer x to the tree (duplicates allowed)
- an instance function **search**(**x**) that returns a reference to a vertex in the tree that contains x, or a null (nil, none) reference if x is not in the tree
- an instance function **remove**(**x**) that removes integer x from the tree. If the tree contains multiple instances of x, only one instance of x is removed. If the tree contains no instances of x, the tree is unchanged
- an instance function **height**() that returns the number of levels in the tree

EXAMPLE OF USE: The pseudo-code shown here illustrates the use of these functions.

```
...
my_tree = BST() # calling the constructor ... syntax will depend on your language
my_tree.add(7)
my_tree.add(7)
my_tree.add(10)
my_tree.add(8)
my_tree.add(9)
h = my_tree.height()
my_tree.remove(10)
my_tree.add(4)
# etc.
```

The details of the implementation are up to you. You may define the functions iteratively, recursively, or with a hybrid of the two. You are welcome to make use of the pseudo-code provided in the class notes.

Experiment

THE REASON BALANCED BINARY TREES were created is to allow us to ensure that the height of a binary search tree holding n values will always be in $O(\log n)$. In fact, Red-Black trees guarantee a height of $\leq 2 * \log n$.

such as the Red-Black Trees we will cover in class

The need for this guarantee is debatable. It is certainly true that some binary search trees can have $O(n)$ levels, but it seems plausible that for large values of n these pathological trees may be quite rare in practice.

In this experiment you will generate a set of trees on n vertices, for increasing values of n. Within each set, you will compute the average height and collect other height-related information. You will then analyse your collected data to answer some research questions.

Experimental Procedure

```
for n in {1000, 2000, 4000, 8000, 16000}:
     for i = 1 to 500:
          construct a BST containing the values {1, ..., n*1.5}, inserted in a randomly chosen order
          perform a sequence of n/2 remove operations
                  (removing randomly selected values that actually are in the tree)
          # this results in a tree with exactly n vertices
          compute the height of this tree and add it to a collection of heights of all your
               trees on n vertices
     compute the minimum height, the maximum height, and the average height
          of all the generated trees on n vertices
     express the average height as k*log n for some k
     express the average height as t*n for some t
     compute the percentage of trees on n vertices whose height is >= n/2
```

Presentation of Results

Create a table or graph showing the k value, the t value, and the percentage of tall trees for each of the specified values of n, as calculated by your experiment.

Research Questions

1. As n increases, does the average height appear to grow at a logarithmic rate, a linear rate, or something else?
2. Does the percentage of tall trees (height >= n/2) grow, shrink, or remain fairly constant as n increases?

What Do I Submit?

Please submit your source code for the Class and the experiment, your tabular or graphical presentation of your experimental results, and your answers to the two research questions.

Due Date

The assignment was originally due on 20211008. This date has been changed. The assignment is now due at 11:59 PM on 20211018.