ADVANCED DATA STRUCTURES - ASSIGNMENT 2

SPLAY TREES

April 6, 2018

Kymry Burwell

Work Done

For this assignment, I decided to implement a splay tree in c++ and compare its performance with a normal binary search tree (BST). I chose splay trees because they have many applications in practice and I think they will be beneficial to know on a detailed level. The steps I took in completing this assignment are as follows:

1. Implement both a splay tree and a binary search tree in c++
2. Perform an average height comparison between the two tree types
3. Perform a search performance comparison between the two tree types

Splay Tree Overview

A splay tree is a binary search tree that is self-adjusting, but not necessarily balanced. In fact, it is possible that the height of a splay tree reaches O(n) in the worst case, but is O(log(n)) in the average case, the same as a binary search tree. In splay trees, search, insert, and delete all have average case O(log(n)) and worst case amortized O(log(n)).

A splay tree is self-adjusting because it "splays" (i.e. moves) the inserted or queried element to the root of the tree. If the element isn't found during a search operation, the last accessed element is moved to the root. This is the main advantage splay trees have over normal binary search trees. When performing multiple queries, where most of the elements queried are biased and comprise only a small percentage of the overall data, splay trees will oftentimes perform better than binary search trees.

BST and Splay Tree Height Comparison

As a slay tree is self-adjusting and a normal binary search tree is not, I thought it would be interesting to compare the heights of the two trees. In order to do this, I used the c++ built in random number generator to generate 20,000 random integers and used them to create both a BST and a splay tree. After each insertion, I recorded the height of the tree. The graph comparing the heights over 20,000 iterations is displayed in figured 1 below. The y-axis shows the height of the tree and the x-axis shows the number of elements currently in the tree. It's interesting to see that the height of the BST is almost always shorter than the splay tree. It's also worth pointing out that the height of the BST is monotonically increasing, while the height of the splay tree varies greatly after each insertion. This is due to the effects of the "splay" operation.

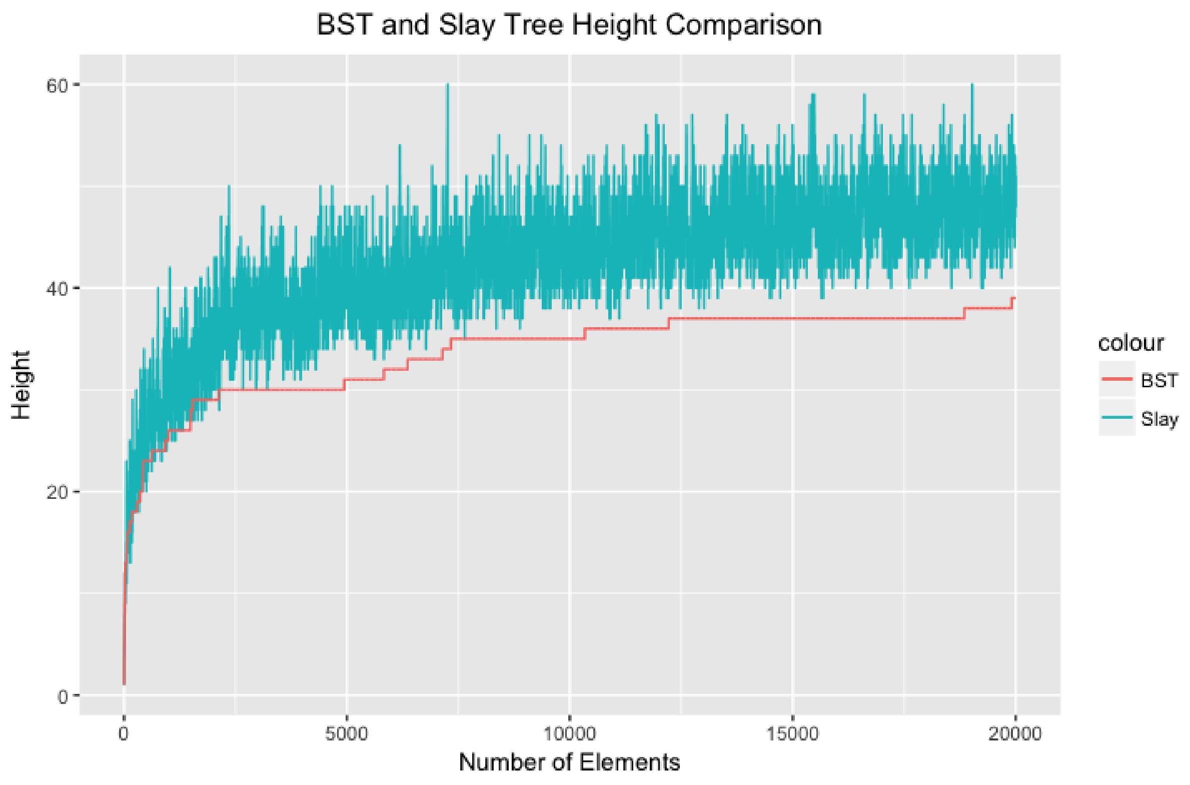

Figure 1: Height Comparison

BST and Splay Tree Search Comparison

The main benefit of a splay tree is better overall search performance when the majority of the queries are comprised of only a small subset of the data. Thus, I performed a comparison of search performance between a BST and splay tree. I did this in two different ways. For both, I searched for 10,000 elements, recording the number of nodes the search visited before either finding the element or exhausting the search and determining the element was not present.

For the first approach, I used purely random numbers for all 10,000 searches. The results are displayed in graph 2 below. The y-axis is the number of nodes that were visited during the search and they x-axis is simply the search performed (10,000 in total). We can see that the performance of both trees were roughly the same. This is expected as we were searching for random elements each time.

For the second approach, I first extracted a subset of roughly 10 percent of the elements in the tree and added them to a vector. Next, I performed a search that pulled from the aforementioned vector with a 90 percent probability and used a purely random number with 10 percent probability. I did this in order to simulate a real-world situation where 90 percent of the queries are for the same subset of elements. The results are displayed in graph 3 below. In this graph, we can see that the slay tree performed better, with many search operations visiting fewer nodes than the BST tree. This helps demonstrate the power of splay trees in certain contexts.

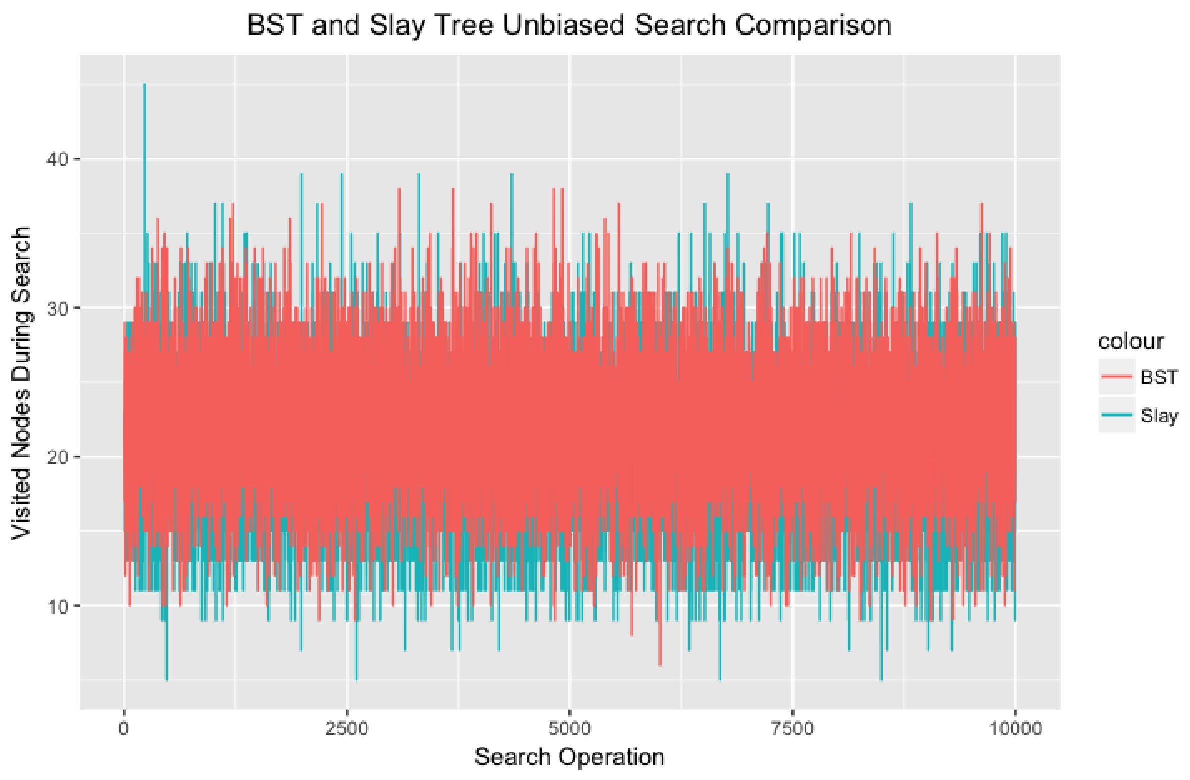

Figure 2: Unbiased Search Comparison

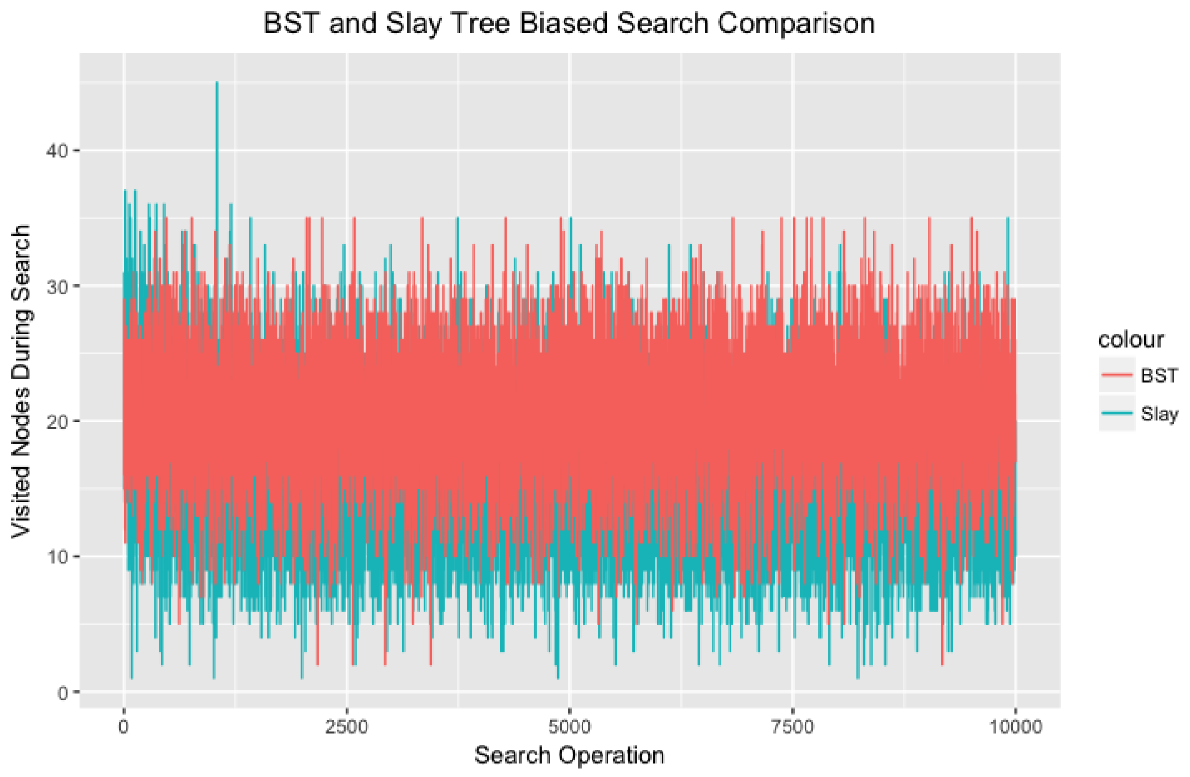

Figure 3: Biased Search Comparison

REFERENCES

Cormen, Thomas H. Introduction to algorithms. MIT press, 2009.

Goodrich, Michael T., and Roberto Tamassia. Algorithm design: foundation, analysis and internet examples. John Wiley and Sons, 2006.

Sedgewick, Robert, and Kevin Wayne. Algorithms. Addison-Wesley Professional, 2011.