Alexander Swanson

March 7, 2018

Professor Lisa Dion

CS 124 – Data Structures and Algorithms

Project 3

## *Project 3 Report*

Dear CS 124 Grader,

For over seven months now, I have been working on a project fundamentally aimed at evaluating and influencing online public opinion. There are many platforms to work on this problem – *Facebook*, *Twitter*, *Instagram*, etc. However, attempting to access all of these platforms immediately is not a practical approach. Thus, working with the social, information aggregation forum *Reddit*, a large amount of data has been collected to begin evaluating public opinion. A portion of this data has been selected for use with this project in order to complete fundamental analysis.

In this third portion of work for *CS 124 – Data Structures and Algorithms*, a class was designed to represent a row of data from the original dataset. The data is composed of general metadata of Reddit comments posted to submissions discussing world news events. An instance of this class, "*RedditElement*," includes the following fields:

*ID*

The unique ID of the Reddit *comment* object.

*Subreddit Name: Prefixed*

The common identifier of a categorical forum on Reddit.

*Ups*

The comment's total resulting *upvotes*.

*Downs*

The comment's total resulting *downvotes*.

*Score*

The overall score, a function of the comment's upvotes and downvotes.

*Category*

The general topic category of the comment's text body.

*Sentiment Score*

The score of the sentiment, rational to the sentiment magnitude. That is, the type of opinion expressed in the comment text body.

*Sentiment Magnitude*

The magnitude of the sentiment, rational to the sentiment score. That is, the weight of the opinion expressed in the comment text body.

*Created*

The UTC unit measure of the date the comment was created.

This newer project has been completed with the dataset used for the *Project 1* – however, some data fields determined to be irrelevant were removed.

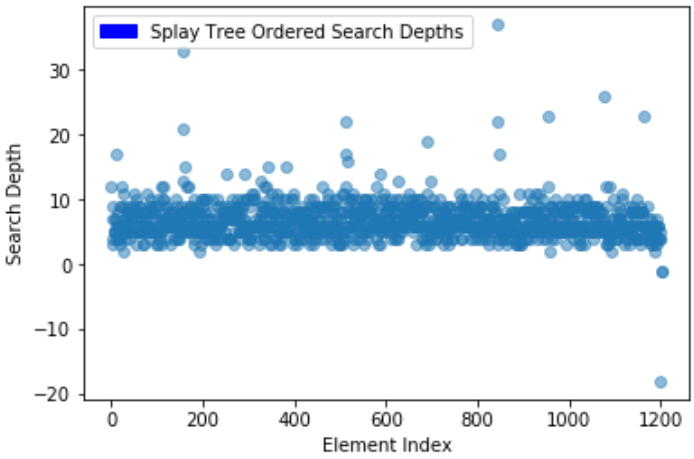
Beside functionality for setting and getting data field values for an instance of RedditElement, the class also allows for the formatted output of all the statistics of the object with the *info*() method. Moreover, using the *sentiment_score* data field, the *measure_happiness*() function performs a sum of every instance of *sentiment_score* for each RedditElement object and calculates the arithmetic mean, yielding a naïve measurement of the "happiness." These calculations are not, however, the most complex for the program. Such is, rather, the data input function, which exhibits a complexity of $O(N^2)$, as it contains a nested *while* loop.

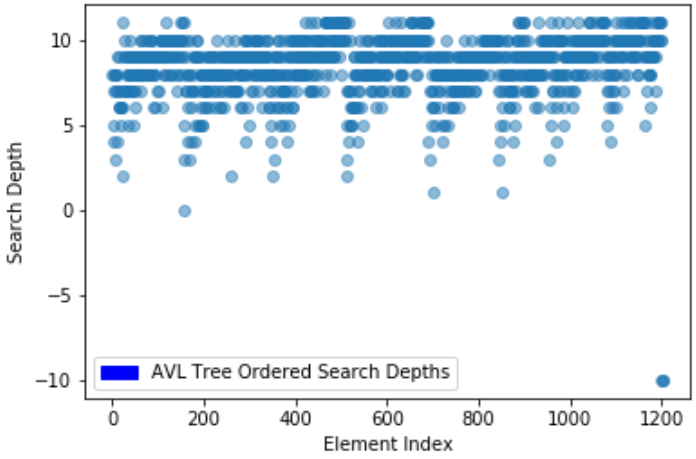
Moreover, for this new expansion of the project, we apply an implementation of *Binary Search* (BST), *AVL*, and *Splay* Trees. The collection of RedditElements, contained in a Standard Library Vector, is stored in each of the afore mentioned data structures. In order to do so, the standard Boolean operators have been overloaded to compare the *ID* field of a RedditElement.

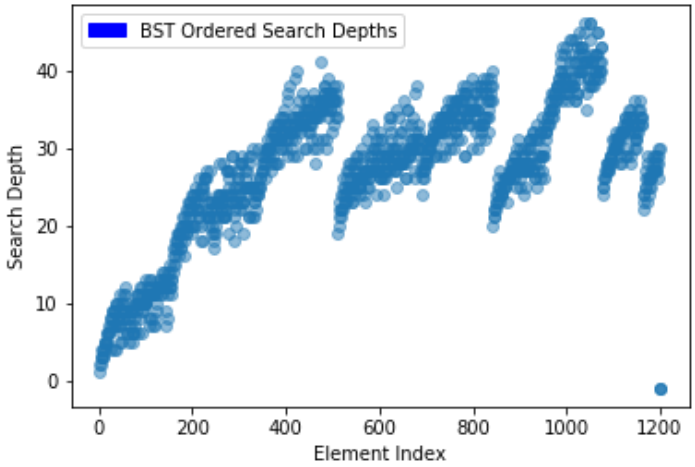
Upon applying the data structure organization, we work to test the abilities of each. Using each data structure's *contains* method, we observe the depths of each of the RedditElements within each Tree. This, however, is more interestingly observed by searching for each RedditElement in three different ways:

1. An ordered search, applying "contains" for every element in the range of the RedditElement vector for 0 to *N*.
2. A randomized search. The order of vector indexes is randomized for the range 0 to *N*.
3. A repeating search. For every RedditElement, for 0 to $\frac{N}{5}$, the index is repeated five times.

These distinct search orderings yield very interesting results. Observe below the resuls for searching the BST:

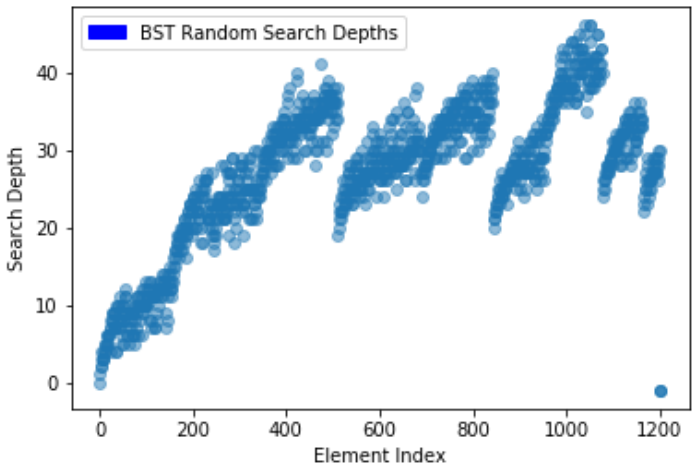

Splay Tree Ordered Search Depths
Search Depth
Element Index
30
20
10
0
−10
−20
0
200
400
600
800
1000
1200

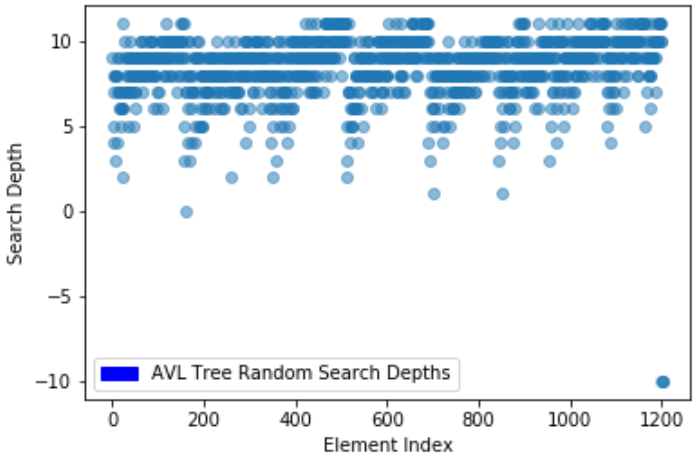

AVL Tree Ordered Search Depths
Search Depth
Element Index
10
5
0
−5
−10
0
200
400
600
800
1000
1200

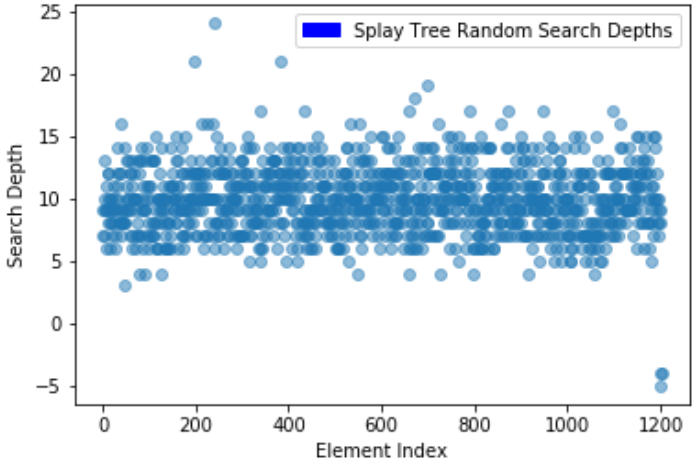

BST Ordered Search Depths
Search Depth
Element Index
40
30
20
10
0
0
200
400
600
800
1000
1200

Before searching, three “dummy objects” were added to the RedditElement vector, in order to verify that each could correctly apply the “contains” functionality – the negative values in the graphs represent the depths of the searches for these dummy objects. This depth is as such because for objects that are not contained in the trees, a search for them yields a search depth converted to a negative number, indicative of the inexistence of the object in the tree.

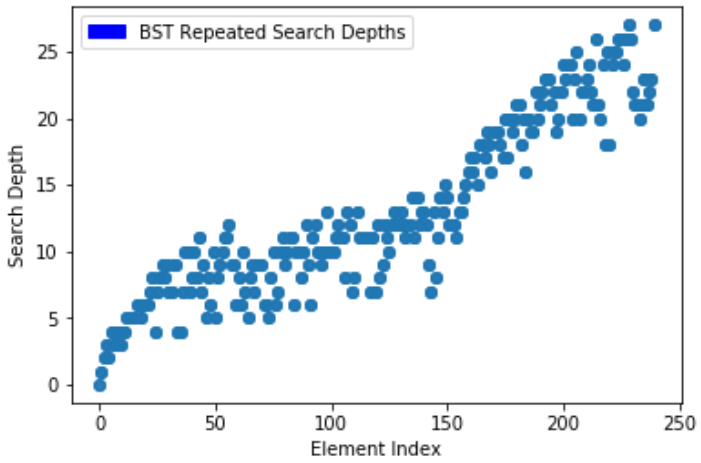
From the ordinary, ordered index search, we can clearly see and differentiate the efficiencies of each Tree. The BST returned very different depths. Moreover, although the AVL Tree’s search depths were all under twelve, the Splay Tree yields a much more similar set of depths. The following graphs represent the searches for a random ordering of indexes and a repeated ordering, respectively:

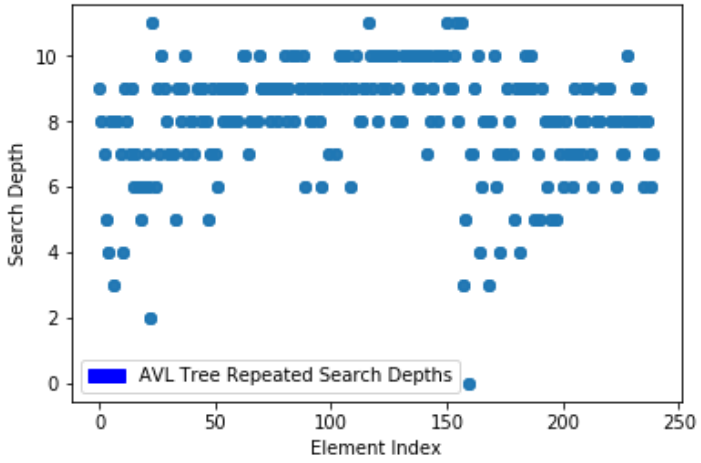


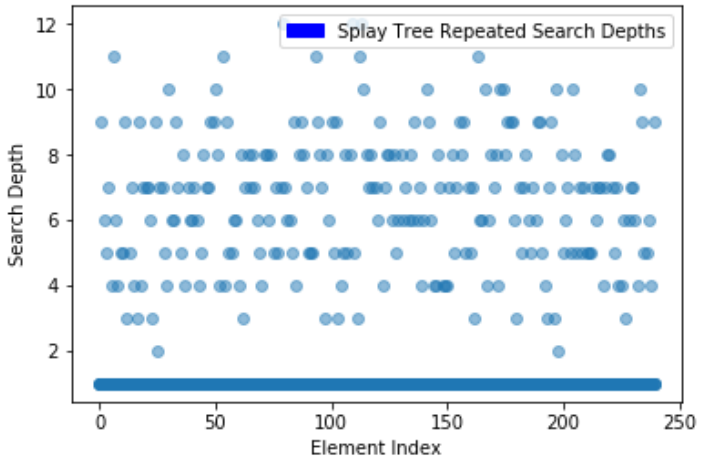



For these randomized index searches, we can observe that although there is significant distinction with respect to ordered index search, the Trees maintain visible consistency. Finally, we observe the depths for repeated index search:




As expected, these search depth plots exhibit very consistent repetition. Interestingly, their shapes reflect that of the plots of the previous index orderings, suggesting a correlation

between the different algorithm process instances. Moreover, these graphs show even more similarity with the first ones observed: we observe that elements at identical indexes frequently produce similar, if not exact, search depths.

Perhaps more significant, however, are the maxima and minima for these several distinct search depths. For each Tree, the absolute minimum is clearly 0. However, the absolute maximums differ. For Binary Search Trees, the absolute maximum – 46 – is clearly greater than the other examples, while AVL Trees show the smallest absolute maximum of 12 – these measures revealing which of the Trees performs most efficiently.

This experiment is important because we are able to observe how these incredibly useful structures perform in comparison with each other. In the future, we may observe that these structures exhibit distinct behaviors when applying different searching algorithms. Binary Search Trees, of search complexity $O(N)$, and both AVL and Splay Trees with search complexities of $O(log\ N)$, may very well exhibit more efficient, or more expensive, computation with different searching algorithms.

Sincerely,

Alexander Swanson

University of Vermont

References

*C++* Code for Binary Search, AVL, and Splay Trees:

Weiss, Mark Allen. *Data Structures and Algorithms in C++* (*Fourth Edition*). Florida International University. 2014. http://users.cs.fiu.edu/~weiss/dsaa_c++4/code/