Analyzing How Priority Transformations Affect the Shape and Operation Speed of Self-Balancing Binary Search Trees

Alexis VanderWilt

Background of the Problem and Motivations:

Storage and fast retrieval of data is a fundamental problem in Computer Science. As more and more data is created, it becomes harder to search through quickly. This creates the need for a data structure that can efficiently store the data, allowing for quick information retrieval.

Self-balancing binary search trees, also known as treaps, have become a popular option for data storage because of their search efficiency. Self-balancing binary search trees use a random 'priority' value, sorted with a heap to keep the binary search tree balanced, which optimizes the searching speed.

Another example is B-trees, which are the most frequently used data structure for one-dimensional database indexes. B-trees are different from binary search trees in that keys and pointers are clustered in memory, which allows for better cache behavior, making it faster to read from. If a treap can be created from a B-tree and a heap and the location of items could be modified in a way that allows commonly searched items to be retrieved quickly, this could create a new data structure that speeds up search time for common objects, giving most users a faster response time. This is especially important in large databases that are searched through often.

The building blocks for this proposed data structure, which retrieves popular items faster than less popular items, are there. For instance, there are data structures already that move items around in a data structure to make them faster to access. For example, splay trees are designed so that every time an element is searched for, that element is moved to the root of the tree, meaning the time to retrieve that data again immediately is O(1) or constant time. Splay trees help organize data based on the chronological ordering of searches, meaning recently searched for data is much faster to retrieve. However, this is not the same as making commonly searched items faster to find, which potentially could be more effective.

Objectives:

In this work, the goal is to design and develop a data structure based on self-balancing binary search trees, or treaps, that prioritizes more commonly searched items by applying mathematical transformations to their priority values, which will move them up the data structure and make them faster to retrieve. What this type of priority transformation does to the shape, balance, insertion speed, deletion speed, and search speed of self-balancing binary search trees will be analyzed.

Time allowing, treaps that are created using other kinds of trees and heaps will also be assessed. For example, is priority transformation more effective in B-treaps or Binary Search treaps? Is there a difference in shape or operation speeds when using a minimum heap versus a maximum heap to sort the priority values?