**SECTION ON ONLINE ALGORITHMS**

Text…

**Overview of Splay Trees**

Binary search trees are data structures that show up all over the world of Computer Science. Many different kinds of binary search trees exist, each with their own set of advantages and disadvantages. For example, balanced binary search trees can guarantee a worst-case runtime bound of O(log(n)) per operation on a tree with n nodes. But they require more memory to store balancing information and may not be the most efficient data structure under various conditions (Sleator and Tarjan, 1985).

Developing an efficient binary search tree guided Daniel Sleator and Robert Tarjan to develop the splay tree. The splay tree is a self-adjusting tree that applies a heuristic known as splaying. To splay a node means to rotate it up to the root. Whenever a node is accessed, it is splayed (Sleator and Tarjan, 1985).

Under Sleator and Tarjan’s implementation, there are three cases to consider when splaying an element up to the root. In case 1, if p, the parent of x, is the tree root, then we rotate the edge joining x to p. In case 2, if p, the parent of x, is not the tree root and p and x are either both left or right children of their respective parent nodes, then we rotate the edge joining p with its parent g (the grandparent of x) first before rotating the edge joining x with p. In case 3, if p, the parent of x, is not the root and x is a left child and p is a right child, or x is a right child and p is a left child, then we rotate the edge joining x with p first before rotating the edge joining p with g (again, the grandparent of x). These sets of two rotations when the parent is not the root distinguishes Sleator and Tarjan’s work from that of Allen, Munro, and Bitner, who proposed rotating nodes up to the root using single rotations. The big win using Sleator and Tarjan’s method of splaying is that not only is the accessed element brought to the root, but also all the elements along the access path are brought up roughly half their original depth (Sleator and Tarjan, 1985).

Sleator and Tarjan also proved what is now known as the Static Optimality Theorem; that if every item is accessed at least once, then the total access time is

O(m + \sum\_{i=1}^{n} q(i)log(\frac{m}{q(i)}))

**see proof – should probably include here**

**link to paper is here:** https://www.cs.cmu.edu/~sleator/papers/self-adjusting.pdf

Where n is the number of nodes in the tree, m is the number of accesses on an n node tree, and q(i) is the total number of times that item i is accessed (Sleator and Tarjan, 1985).

The also showed that if f represented any fixed item, then the total access time for a sequence of m accesses on an n node tree is O(nlogn + m + \sum\_{j=1}^{m} log(|i\_j-f|+1)) (Sleator and Tarjan, 1985).

However, Sleator and Tarjan left their paper with a series of conjectures that they could not prove. The Dynamic Optimality Conjecture, which remains yet unproven, states that given an arbitrary sequence of successful accesses on a search tree of size n, given an algorithm A that performs each of the accesses by traversing from the root to the node to be accessed at a cost of one plus the depth of the node, and given that A performs any number of rotations anywhere in the tree at a cost of only one per rotation, then the total access time by splaying is bounded above by O(n) plus a constant times the time required by algorithm A. In the language of online algorithms, this means a splay tree is O(1) competitive with any offline algorithm when executing successful sequences of node accesses. Sleator and Tarjan could only prove that splay trees were O(log(n)) competitive.

In 2005, Erik Demaine introduced a new online BST called Tango Trees that he argued was O(log(log(n))) competitive with offline BSTS. He used the interleave lower bound as a source of inspiration for developing the tango tree. Take a search sequence X. Consider a balanced reference tree P, containing nodes from 1 to n. Each node in the tree that is not a leaf has one of its children designated as preferred. The preferred child is selected based on which subtree of the parent node had the most recent look-up. Given an arbitrary node, a preferred path is a path in this reference tree P that starts from the arbitrary node and goes down until a leaf node is reached. We can say that the size of any preferred path has to be O(log(n)) simply because the P is a balanced tree. A Tango tree stores each preferred path in a balanced binary search tree of height O(log(log(n))) (Iacono, 2013).

A few other papers on the dynamic optimality conjecture, and online self-adjusting tree structures have since come out. In our research project, we tried to tackle a possible route for improving online trees by creating hybrids between AVL trees and splay trees.

**Further Research**

For our SPLAVL tree, we used the heuristics provided by Allen, Munro, and Bitner to rotate a node up to the root using single rotations. We used this heuristic primarily because it made implementing the partial balancing easier. However, as noted earlier in the paper, splaying helps roughly halve the depths of nodes along the accessed path, which is more than what these single rotations do. Now that we’ve observed that a hybridized version of a splay tree and an AVL tree can yield performance boosts over normal splay trees under the right conditions, we would like to see how a hybridized tree would perform using the more traditional splay mechanics that Sleator and Tarjan introduced (Sleator and Tarjan, 1985).

MENTION THAT IT MIGHT BE COOL TO HYBRIDIZE TANGO