CS41 Final Project Online Algorithms
Initial Report and Proposed Action Plan
Tahmid Rahman, Dylan Jeffers

Initial Report

Splay trees are an interesting binary search tree variant that self-adjusts to minimize the lookup time for recently accessed items by moving frequently used nodes nearer to its root, taking advantage of locality of reference. In industry, this tree provides practical advantages for important cache and garbage collection algorithms.

Splay trees are part of a field known as online algorithms because the algorithms that govern their behavior dont generally know the entire input beforehand (how can a tree know exactly what items will be searched for?). When analyzing online algorithms, its common to look at how the offline counterpart to the algorithm compares. Such a comparison is maintained via the competitive ratio. Splay trees have an interesting and yet unproven fact regarding their competitive ratio to binary search trees. Known as the Dynamic Optimality conjecture and formalized by Sleator and Tarjan, splay trees are thought to be O(1) competitive to any offline rotational-based search trees for long enough search sequences.

Our final project does not necessarily involve proving this conjecture. Instead, we will work to provide experimental data backing up this conjecture by creating our own version of a splay tree. We will then theoretically analyze our splay tree and give an argument more akin to the kinds of arguments weve been making in class so far as to why our data shows what it shows.

Detailed motivation for our plan:

Splay trees, as mentioned above, have many advantages. For example, they require little bookkeeping data, and it's possible to use past versions after an update, requiring amortized $O(\log n)$ space per update. However, they also come with the significant disadvantage of having a worst case linear height, which can occur if all n elements are accessed in non-decreasing order. Our primary objective for this final project is to design and implement a new splay tree that solves this issue by decreasing its overall height to some factor of $\log n$. To do so, we plan to incorporate an AVL-Tree implementation into our splay tree to help balance the lower sections of our tree, minimizing the total height.

Once implemented, we will run tests to determine the query speed between a traditional Splay tree, a traditional AVL Tree, and our Splay/AVL Tree. To test each tree's performance, we plan to systematically vary each tree's input size and it's level of randomness. Finally we will plot these performances and determine the sets of input that each tree performs best. We hope to see our algorithm smooth the edges between the positives of a traditional splay tree and AVL tree, giving an overall runtime faster than most AVL trees that can work over a larger set of inputs than a traditional Splay tree.

Further Research:

For our project, we designed a splay tree that minimizes worst case height using an AVL tree implementation. We also will show how different binary trees perform better given different input. Such data could determine just how beneficial (or feasible) it might be to have a preprocessing pattern recognition algorithm. If theres a way to efficiently run a pattern recognition algorithm as queries are made, then it might be possible to implement a tree that goes back and forth between activating splaying options (for example, if were accessing all the nodes, one after the other, theres really no point in moving the previously accessed nodes further up). In essence, someone could create a data structure containing multiple tree algorithms that could alternate control to take advantage of this variance.

Proposed Action Plan

I Week 1: April 10th - April 17th

(a) Read one to two academic papers on online algorithms: *"Dyamic Optimality–Almost"*

(b) Create outline for our paper

(c) Implement the foundation of splay tree data structure.

II Week 2: April 17th - April 24th

(a) Read another one to two academic papers.

(b) Write rough draft of main sections of paper.

(c) Understand MIT professor Erik Demaines Online BST implementation.

(d) Finish standard splay tree implementation, including testing.

(e) Consider ways of making our splay tree faster.

(f) (We might go to you with a lot of questions/ seek advice around this time)

III Week 3: April 24th - May 1st

(a) Read another one academic paper.

(b) Test our new splay/ALV tree, comparing it against other BSTs

(c) Compile and analyze our data

(d) Final Draft including out testing conclusions.

(e) Commented splay tree code