Peter Collins + Kristin Cox

1. An optimal binary search tree is a static binary search tree constructed so that the expected cost of looking up an item is minimized. To construct the tree, you must know the frequency of each item being accessed.
2. The goal of the project was to implement both a splay tree and an optimal binary search tree and thoroughly test both to see if the splay tree could be competitive with the optimal binary search tree. We created various tests and applied them to both trees to find sequences which were easy and hard for both structures. The open problem is whether or not splay trees are competitive with optimal binary search trees.
3. This problem is described as the most important open problem because the idea that an amortized dynamic data structure is competitive with a static optimal a priori data structure is surprising.
4. A splay tree would be preferred if you do not know the query sequence ahead of time, or if there are a lot of items that are looked up redundantly.
5. The measure of complexity was the sum of the number of operations required for each lookup in the sequence. We did three different types of tests, each with 1 – 250 items: in-order lookups, random lookups, and redundant lookups. We chose this test data because, for the two different data structures, we hypothesized that these tests would produce different efficiency results between them. For the random test we did 30 samples and took the average number of operations.
6. As we see in the first graph, the splay tree is less efficient than the optimal binary search tree when N <= 50 but is more efficient with larger N values. For larger numbers, the splay operation will cause the tree for in-order numbers to have the next value very close to the recently splayed root, resulting in a quicker lookup. We found this surprising because we assumed that the splay tree would be a consistent line with a linear lookup time and worse than the optimal binary search tree. In the random lookups graph, the optimal binary search tree performed better than the splay tree for the average of all N values. What we found in this test case was that knowing the frequency of lookups beforehand was better than the self-adjusting splay tree mechanism. Finally, in the redundant lookups graph, the optimal binary search tree performed far worse than the splay tree. This is because, for the splay tree, most of the lookups occur in constant time even though the frequency of lookups for all the items is the same, as seen by the distribution.
7. We already implemented a comparison of a self-balancing red-black tree and a self-adjusting splay tree. As another extension we would attempt to add weights to each splay node so when a node is splayed its weight is increased. Also, it isn’t splayed up to the root but rather to a position whose parent nodes have greater weights. We would test this against the random lookups to see if it would improve performance for the splay tree on this test. This could have real-life applications such as trending tweets on Twitter, where recently posted and searched terms would show up near the top of the tree.
8. Peter Collins: Code and Algorithm Design

Kristin Cox: Analysis and Questions